

Anthony Brabazon  
Michael O'Neill  
Dietmar G. Maringer (Eds.)

# Natural Computing in Computational Finance

Volume 3

Anthony Brabazon, Michael O'Neill, and Dietmar G. Maringer (Eds.)

---

Natural Computing in Computational Finance

# Studies in Computational Intelligence, Volume 293

## Editor-in-Chief

Prof. Janusz Kacprzyk  
Systems Research Institute  
Polish Academy of Sciences  
ul. Newelska 6  
01-447 Warsaw  
Poland  
E-mail: kacprzyk@ibspan.waw.pl

---

Further volumes of this series can be found on our homepage: [springer.com](http://springer.com)

Vol. 272. Carlos A. Coello Coello, Clarisse Dhaenens, and Laetitia Jourdan (Eds.)  
*Advances in Multi-Objective Nature Inspired Computing*, 2009  
ISBN 978-3-642-11217-1

Vol. 273. Fatos Xhafa, Santi Caballé, Ajith Abraham, Thanasis Daradoumis, and Angel Alejandro Juan Perez (Eds.)  
*Computational Intelligence for Technology Enhanced Learning*, 2010  
ISBN 978-3-642-11223-2

Vol. 274. Zbigniew W. Raś and Alicja Wieczorkowska (Eds.)  
*Advances in Music Information Retrieval*, 2010  
ISBN 978-3-642-11673-5

Vol. 275. Dilip Kumar Pratihari and Lakhmi C. Jain (Eds.)  
*Intelligent Autonomous Systems*, 2010  
ISBN 978-3-642-11675-9

Vol. 276. Jacek Mańdziuk  
*Knowledge-Free and Learning-Based Methods in Intelligent Game Playing*, 2010  
ISBN 978-3-642-11677-3

Vol. 277. Filippo Spagnolo and Benedetto Di Paola (Eds.)  
*European and Chinese Cognitive Styles and their Impact on Teaching Mathematics*, 2010  
ISBN 978-3-642-11679-7

Vol. 278. Radomir S. Stankovic and Jaakko Astola  
*From Boolean Logic to Switching Circuits and Automata*, 2010  
ISBN 978-3-642-11681-0

Vol. 279. Manolis Wallace, Ioannis E. Anagnostopoulos, Phivos Mylonas, and Maria Bielikova (Eds.)  
*Semantics in Adaptive and Personalized Services*, 2010  
ISBN 978-3-642-11683-4

Vol. 280. Chang Wen Chen, Zhu Li, and Shiguo Lian (Eds.)  
*Intelligent Multimedia Communication: Techniques and Applications*, 2010  
ISBN 978-3-642-11685-8

Vol. 281. Robert Babuska and Frans C.A. Groen (Eds.)  
*Interactive Collaborative Information Systems*, 2010  
ISBN 978-3-642-11687-2

Vol. 282. Husrev Taha Sencar, Sergio Velastin, Nikolaos Nikolaidis, and Shiguo Lian (Eds.)  
*Intelligent Multimedia Analysis for Security Applications*, 2010  
ISBN 978-3-642-11754-1

Vol. 283. Ngoc Thanh Nguyen, Radoslaw Katarzyniak, and Shyi-Ming Chen (Eds.)  
*Advances in Intelligent Information and Database Systems*, 2010  
ISBN 978-3-642-12089-3

Vol. 284. Juan R. González, David Alejandro Pelta, Carlos Cruz, Germán Terrazas, and Natalio Krasnogor (Eds.)  
*Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, 2010  
ISBN 978-3-642-12537-9

Vol. 285. Roberto Cipolla, Sebastiano Battiato, and Giovanni Maria Farinella (Eds.)  
*Computer Vision*, 2010  
ISBN 978-3-642-12847-9

Vol. 286. Zeev Volkovich, Alexander Bolshoy, Valery Kirzhner, and Zeev Barzilay  
*Genome Clustering*, 2010  
ISBN 978-3-642-12951-3

Vol. 287. Dan Schonfeld, Caifeng Shan, Dacheng Tao, and Liang Wang (Eds.)  
*Video Search and Mining*, 2010  
ISBN 978-3-642-12899-8

Vol. 288. I-Hsien Ting, Hui-Ju Wu, Tien-Hwa Ho (Eds.)  
*Mining and Analyzing Social Networks*, 2010  
ISBN 978-3-642-13421-0

Vol. 289. Anne Hökansson, Ronald Hartung, and Ngoc Thanh Nguyen (Eds.)  
*Agent and Multi-agent Technology for Internet and Enterprise Systems*, 2010  
ISBN 978-3-642-13525-5

Vol. 290. Weiliang Xu and John E. Bronlund  
*Mastication Robots*, 2010  
ISBN 978-3-540-93902-3

Vol. 291. Shimon Whiteson  
*Adaptive Representations for Reinforcement Learning*, 2010  
ISBN 978-3-642-13931-4

Vol. 292. Fabrice Guillet, Gilbert Ritschard, Djamel A. Zighed, and Henri Briand (Eds.)  
*Advances in Knowledge Discovery and Management*, 2010  
ISBN 978-3-642-00579-4

Vol. 293. Anthony Brabazon, Michael O'Neill, and Dietmar G. Maringer (Eds.)  
*Natural Computing in Computational Finance*, 2010  
ISBN 978-3-642-13949-9

Anthony Brabazon, Michael O'Neill,  
and Dietmar G. Maringer (Eds.)

# Natural Computing in Computational Finance

Volume 3

Prof. Anthony Brabazon  
Quinn School of Business  
University College Dublin, Belfield  
Dublin 4  
Ireland  
E-mail: anthony.brabazon@ucd.ie

Dr. Michael O'Neill  
UCD CASL  
8 Belfield Office Park  
Beaver Row, Clonskeagh  
Dublin 4  
Ireland  
E-mail: m.oneill@ucd.ie

Prof. Dietmar Maringer  
University of Basel  
Wirtschaftswissenschaftliches  
Zentrum (WWZ)  
Abteilung Quantitative Methoden  
Peter Merian-Weg 6  
4002 Basel  
Switzerland  
E-mail: dietmar.maringer@unibas.ch

ISBN 978-3-642-13949-9

e-ISBN 978-3-642-13950-5

DOI 10.1007/978-3-642-13950-5

Studies in Computational Intelligence

ISSN 1860-949X

Library of Congress Control Number: Applied for

© 2010 Springer-Verlag Berlin Heidelberg

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

*Typeset & Cover Design:* Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

*To Maria  
Tony*

*To Gráinne, Aoife and Michael  
Michael*

*To Klaus  
Dietmar*

---

# Preface

Recent years have seen the application of various Natural Computing algorithms for the purposes of financial modelling. In this context Natural Computing algorithms can be broadly defined as computer algorithms whose design draws inspiration from phenomena in the natural world. Particular features of financial markets, including their dynamic and interconnected characteristics, bear parallels with processes in the natural world and prima facie, this makes Natural Computing methods *'interesting'* for financial modelling applications. In addition to the problem-solving potential of natural processes which Natural computing seeks to embody in its algorithms, we can also consider Natural Computing in terms of its potential to understand the natural processes which themselves serve as inspiration. For example, financial and biological systems exhibit the phenomenon of emergence, or the activities of multiple individual agents combining to co-evolve their own environment, and a stream of work has emerged which applies learning mechanisms drawn from Natural Computing algorithms for the purposes of agent-based modelling in finance and economics.

This book consists of eleven chapters each of which was selected following a rigorous, peer-reviewed, selection process. The chapters illustrate the application of a range of cutting-edge natural computing and agent-based methodologies in computational finance and economics. While describing cutting edge applications, the chapters are written so that they are accessible to a wide audience. Hence, they should be of interest to academics, students and practitioners in the fields of computational finance and economics.

The inspiration for this book was due in part to the success of EvoFIN 2009, the 3<sup>rd</sup> European Workshop on Evolutionary Computation in Finance and Economics. EvoFIN 2009 took place in conjunction with Evo\* 2009 in Tübingen, Germany (15-17 April 2009). Evo\* is an annual collection of European conferences and workshops broadly focused on Evolutionary Computation, and is the largest European event dedicated to this field of research. A number of the chapters presented in this book are extended versions of papers presented at EvoFIN 2009 and these have undergone the same rigorous, peer-reviewed, selection process as the other chapters.

This book follows on from **Natural Computing in Computational Finance Volumes I and II**.

We would like to thank all the authors for their high-quality contributions, the reviewers who generously gave of their time to peer-review all submissions, and Wei Cui who helped with the preparation of the final manuscript. We would also like to thank Dr. Thomas Ditzinger of Springer-Verlag and Professor Janusz Kacprzyk, editor of this book series, for their encouragement of, and their support during, the preparation of this book. Finally, Anthony Brabazon and Michael O'Neill would like to acknowledge the support of their research activities provided by Science Foundation Ireland (Grant number 08/SRC/FM1389).

Dublin and Basel  
February 2010

Anthony Brabazon  
Michael O'Neill  
Dietmar Maringer



---

# Contents

<b>1 Natural Computing in Computational Finance (Volume 3): Introduction</b> <i>Anthony Brabazon, Michael O’Neill, Dietmar Maringer</i> .....	1
--	---

---

## Part I: Financial and Agent-Based Models

---

<b>2 Robust Regression with Optimisation Heuristics</b> <i>Manfred Gilli, Enrico Schumann</i> .....	9
<b>3 Evolutionary Estimation of a Coupled Markov Chain Credit Risk Model</b> <i>Ronald Hochreiter, David Wozabal</i> .....	31
<b>4 Evolutionary Computation and Trade Execution</b> <i>Wei Cui, Anthony Brabazon, Michael O’Neill</i> .....	45
<b>5 Agent-Based Co-operative Co-evolutionary Algorithms for Multi-objective Portfolio Optimization</b> <i>Rafał Dreżewski, Krystian Obrocki, Leszek Siwik</i> .....	63
<b>6 Inferring Trader’s Behavior from Prices</b> <i>Louis Charbonneau, Nawwaf Kharma</i> .....	85

---

## Part II: Dynamic Strategies and Algorithmic Trading

---

<b>7 Index Mutual Fund Replication</b> <i>Jin Zhang, Dietmar Maringer</i> .....	109
<b>8 Frequent Knowledge Patterns in Evolutionary Decision Support Systems for Financial Time Series Analysis</b> <i>Piotr Lipinski</i> .....	131

**9 Modeling Turning Points in Financial Markets with Soft Computing Techniques**  
*Antonia Azzini, Célia da Costa Pereira, Andrea G.B. Tettamanzi* . . . . . 147

**10 Evolutionary Money Management**  
*Philip Saks, Dietmar Maringer* . . . . . 169

**11 Interday and Intraday Stock Trading Using Probabilistic Adaptive Mapping Developmental Genetic Programming and Linear Genetic Programming**  
*Garnett Wilson, Wolfgang Banzhaf* . . . . . 191

**Index** . . . . . 213

---

# Natural Computing in Computational Finance (Volume 3): Introduction

Anthony Brabazon<sup>1,2</sup>, Michael O'Neill<sup>1,3</sup>, and Dietmar Maringer<sup>4</sup>

<sup>1</sup> Natural Computing Research & Applications Group,  
Complex & Adaptive Systems Laboratory,  
University College Dublin, Ireland  
{anthony.brabazon,m.oneill}@ucd.ie

<sup>2</sup> School of Business, University College Dublin, Ireland

<sup>3</sup> School of Computer Science and Informatics, University College Dublin, Ireland

<sup>4</sup> Business and Economics Faculty, University of Basel, Switzerland  
dietmar.maringer@unibas.ch

## 1.1 Introduction

Computational Finance covers a wide and still growing array of topics and methods within quantitative economics. The core focus has long been on efficient methods, models and algorithms for numerically demanding problems. The advent of new computational methods, together with the advances in available hardware, has pushed the boundaries of this field outwards. Not only can the complexity of investigated problems be increased, one can even approach problems that defy traditional analytical examination all together. One major contributor of such methods is natural computing.

Natural computing draws its inspiration from phenomena and systems in nature, converts them into computer programs and algorithms which can then be applied in many real-world application areas, including computational finance. The rationale behind this approach comes from the parallels between economic systems and the real world: Financial environments are typically extremely complex with high levels of uncertainty, noise and dynamics. This is equally true for real environments, yet nature has produced many mechanisms to deal with these requirements. Successful strategies from the natural world could therefore serve as blueprints for solving problems in all sort of areas. Some of these methods are valuable supplements to existing techniques: optimization heuristics, e.g., can be used where traditional concepts failed or required unreasonable simplifications in the problem statements. Typical examples would be evolution and natural selection: individuals with superior features have a higher chance of outperforming their competitors and, eventually, passing these features on to their offspring. Analogously, evolutionary search and optimization techniques have populations of candidate solutions to the (financial) problem at hand, and the better suitable one of them is, the better its chances that it replaces a weaker one and forms the basis for new (and further improved) ones. Akin to natural evolution, superior offspring will replace their not-so-good competitors and ancestors. Several chapters in this book use evolutionary

principles in this manner to develop populations that fit a pre-defined objective function as well as possible.

More recently a new variety has been added to this area, co-evolution. In simple evolutionary systems, one population searches for the optimum location or strategy within a given environment. In certain situations, however, the population shapes and changes this environment, mutually affecting the development of other species or populations. Natural examples range from “arms races” between predators and prey to symbiotic “partnerships” between species. Natural computing builds on these ideas and utilizes them for both competitive and cooperative co-evolution. This book series includes some of the first financial applications of such methods.

Nature inspired methods can be used not only for previously unapproachable problems, they have also created whole new avenues of research. Artificial life is concerned with the creation of artificial creatures or systems that have properties similar to those of their natural counterparts. Such artificial systems can be used to study complexity and self-organizing properties. In economics, agent-based models do exactly that: To emulate realistic properties of macro structures, behavior and interaction is modelled on an individual micro level. The decision-making agents are autonomous, yet their behaviours are interdependent and interacting. They can be equipped with heterogeneous risk preferences, learning rules and behaviour. Even when agents follow rather simple rules and strategies, highly complex systems can emerge with properties very similar to those of real financial markets.

Another strand of natural computing borrows from linguistics and developmental processes. Examples for this are Genetic Programming and Grammatical Evolution which develop rules or formulae, using a given grammar which contain a set of variables and sub-structures. Early financial applications of these approaches included the derivation of models for pricing and estimation where analytical solutions are not feasible, such as implied volatility models. More recently, finding patterns and trade triggers for financial time series has become a fruitful area of research. This raised interest in the development of automated trading systems is also reflected in the real world where algorithmic trading contributes substantially to the traded volume in major equity markets (cf. [5]).

Other popular areas in natural computing include (but are not limited to) neuro-computing (which loosely draws its inspiration from the workings of the human brain); social computing (adopting swarming or communication behaviours of social animal such as birds, fish or insects); physical computing (mimicking chemical or physical processes); and immunocomputing (algorithms inspired by biological immune systems). Readers interested in a broader introduction to these methods are referred to [1, 2, 3, 4, 6, 7, 8] and the literature quoted therein.

As in the previous two volumes of this series, this volume covers a variety of financial applications. The first part of the book addresses topics that combine financial and agent-based models with optimization. The applications range from the use of natural computing techniques to calibrate financial econometric models, to agent-based models for optimal financial decision-making. The second part of the book collects applications of natural computing to the uncovering of dynamic and automated trading strategies. The following section provides a short introduction to the individual chapters.

## 1.2 The Chapters

### 1.2.1 Financial and Agent-Based Models

Optimization is one of the key areas in quantitative finance. The problems range from estimation and model calibration, to portfolio optimization, asset selection and development of trading strategies.

Chapter 2 (*Robust Regression with Optimisation Heuristics* by Manfred Gilli and Enrico Schumann) demonstrates how heuristic methods can be used in regression analysis. To obtain parameter estimations for linear models, the traditional way is to minimize the mean squared error. The advantage of a closed form solution, however, comes at a high price: the data ought to be normally distributed – a requirement rarely met by financial data. These are known to have fat tails, and extreme observations and outliers can make the estimated parameters unstable. Minimizing a quantile, e.g., the median, of squared residuals instead can remedy this problem: extreme values have substantially less impact and the estimates are more robust. Unfortunately, there exist no closed form solutions for this approach, and the rough search space makes traditional gradient based estimation methods unreliable. Heuristic methods can deal with demanding search spaces because they incorporate strategies to escape local optima and are more flexible in the generation of new candidate solutions. Gilli and Schumann test a variety of settings for Differential Evolution, Particle Swarm Optimisation (PSO) and Threshold Accepting for the estimation problem at hand. They find that PSO works best in their experiments, but the other methods also perform very well.

Another estimation problem is dealt with in Chapter 3 (*Evolutionary Estimation of a Coupled Markov Chain Credit Risk Model* by Ronald Hochreiter and David Wozabal). Markov Chain models are a popular approach to capture risk class transitions. In Coupled Markov Chains, defaults are directly modelled as Bernoulli events. Two major advantages of this approach are the ability to capture different default correlations for different sectors and rating classes, and that “closeness to default” is pictured more accurately than in the standard model. Empirical application of this model requires the maximization of a log likelihood function which, from an optimizer’s point of view, is not well behaved. Hochreiter and Wozabal therefore suggest evolutionary estimation. They, too, suggest Particle Swarm Optimization, but contrast it with Evolutionary Algorithms. Numerical experiments show that both methods are suitable candidates to solve the estimation problem.

The complexity of markets does not allow easy learning from mere observation for many reasons, including the inability of researchers to undertake ‘what - if’ experiments and the open rather than closed nature of market systems. An increasingly popular approach to boost our theoretical understanding of financial markets is to undertake “in silico” experiments, using computer simulations to investigate interesting aspects of financial markets. To capture market microstructure and the aggregate effects of individual decision makers, agent-based models can be employed. An agent represents an individual decision maker who acts and reacts within an artificial environment. Depending on the specifics of the model, agents can have different levels of intelligence and perception, their set of possible activities can be rather diverse, as can be the degree of interaction (or, sometimes, interference) with their fellow agents and / or the

environment. Equally diverse as the models are the problems they can be applied to: from investigating the price processes emerging in such markets to testing the stability and consequences of market frameworks and current or possible new regulations to understanding and optimizing individual behaviour.

Chapter 4 (*Evolutionary Computation and Trade Execution* by Wei Cui, Anthony Brabazon and Michael O'Neill) focuses mainly on the latter issues. Trade execution is the process of trading a particular instrument of interest. A practical issue in trade execution is how to trade a large order as efficiently as possible. For example, trading of a large order in one lot may produce significant market impact costs. Conversely, by dividing an order into smaller lots and spreading these over time, a trader can reduce market impact cost but increases the risk of suffering opportunity cost. An efficient trade execution strategy seeks to balance out these costs in order to minimise the total trade cost, order or a market order. Learning suitable strategies cannot be easily done using historical observations alone: as from past observed prices one cannot conclude how the market would have reacted to different decisions by the agents. In this chapter, the authors implement an agent-based artificial stock market that responds to the agents' decisions. The agents place orders which are carried out or not, depending on the orders of the other agents and according to market rules. The individual agents are equipped with some intelligence: using genetic algorithms, they can learn successful strategies and how to place optimal orders. In addition to results from numerical experiments, different opportunities for promising future research are suggested.

A different case of optimal decision making is considered in Chapter 5 (*Agent-Based Co-operative Co-evolutionary Algorithms for Multi-Objective Portfolio Optimization* by Rafał Dreżewski, Krystian Obrocki and Leszek Siwik). In their model, co-evolutionary techniques are combined with an agent-based framework. Several different settings are tested against each other. The results show that the agents find Pareto-efficient solutions and are therefore capable of solving a multi-objective optimization problem.

Louis Charbonneau and Nawwaf Kharma, on the other hand, focus on the price processes generated by agent-based models. In Chapter 6 (*Inferring Trader's Behavior From Prices*), they assume a market where a group of heterogeneous agents follows trading rules to make investment decisions which drive the price process. The agents are able to adopt their behaviour and change their rules, which in return will influence the subsequent price process, and so on. The resulting time series exhibit patterns and statistical signatures that reflect those of actual real-world data.

## 1.2.2 Dynamic Strategies and Algorithmic Trading

Real investment strategies are not just a sequence of single period investment decisions. Ideally, a dynamic strategy considers several future time periods in making each decision. Economic wisdom suggests that, in a perfect market, previous decisions should not affect current decisions. In practice, however, it is not always possible to pretend that one can start from scratch. When previous decisions have led to the current status quo and moving away from the current situation is too costly, optimal decision making is sometimes reduced to making the best of a given situation. This shows all the more, that in a dynamic framework, all current decisions should avoid future costly adjustments.

Dynamic financial decision making is not only concerned with *how*, but also *when* to react. Entering or leaving a position at the wrong moment can have devastating effects. The heavy tails and negative skewness of asset returns can potentially destroy a cumulation of past profits in a very short period of time. Hence, spotting potentially devastating events as well as profitable stretches of time in advance can be crucial.

Chapter 7 (*Index Mutual Fund Replication* by Jin Zhang and Dietmar Maringer) addresses a different approach. In the tradition of passive portfolio management strategies, index tracking aims to find a portfolio whose return process is as close to that of the benchmark index as possible. Typically, these approaches assume no specific parametric distribution for asset returns, but use historical simulation as a close enough estimation of possible future events. Consequently, they require no specific predictions of assets' returns or risks, nor do they target a prespecified return or risk limit. While simple in concept, practical application is, again, hampered by numerical issues in the optimization process, in particular when, as in this chapter, realistic constraints on different dynamic portfolio adjustment strategies are considered. Using Differential Evolution, numerical experiments for five mutual funds demonstrate that this method can identify suitable tracker portfolios with appropriate in sample and out of sample behaviour.

Chapter 8 (*Frequent Knowledge Patterns in Evolutionary Decision Support Systems for Financial Time Series Analysis* by Piotr Lipinski) addresses the issue of recognizing patterns in financial time series. These patterns can then be used for the successful implementation of trading rules. Given the vast range of potential trading rule candidates, the search space needs to be reduced to a realistic size. The authors suggest the use of frequent knowledge patterns. Evolutionary methods are then used to optimize the system by modeling trading experts who can choose any combination from a list of given trading rules. Numerical experiments compare different frequencies of updating knowledge patterns.

Chapter 9 (*Modeling Turning Points in Financial Markets with Soft Computing Techniques*, by A. Azzini, C. da Costa Pereira and A. Tettamanzi) is also about detecting patterns in financial time series. Here, the main objective is to spot as soon as possible when an upward trend changes into a downward trend and vice versa. In bullish markets, investors want to be long to benefit from the price increases, while in bearish markets, short position generate them profits. Identifying such turning points correctly can therefore be easily translated into profitable strategies. The authors suggest two different approaches, fuzzy logic and neural networks. Applied to different financial instruments, they find both methods can be used to predict some of the trend reversals.

Chapter 10 (*Evolutionary Money Management*, by Philip Saks and Dietmar Maringer) uses Genetic Programming to generate high-frequency trading rules for foreign exchange markets. Automatically derived trading rules sometimes suffer from a high level of complexity, making them difficult to interpret and even harder to manually evaluate. Also, human traders often follow a so-called money management strategy. The idea behind this reflects again the problem that what is a good decision can depend on the current status quo. If a decision maker has no clear opinion about the trend or predicts no substantive changes, then she might keep her current position. Hence, if staying long is optimal, then going long is not necessarily optimal, too. Optimal signals might therefore be contingent to the current position. This chapter introduces a multiple tree

structure with contingent trading rules. Numerical experiments show that these distinctions can lead to superior decisions, in particular when transaction costs and different forms of risk and loss aversion are considered.

Genetic Programming approaches are also employed in Chapter [11](#) (*Interday and Intraday Stock Trading Using Probabilistic Adaptive Mapping Developmental Genetic Programming and Linear Genetic Programming*, by Garnett Wilson and Wolfgang Banzhaf). In this implementation, trading rules are represented as binary strings that undergo instruction-dependent interpretation. Unlike traditional settings, however, co-evolution takes place, and a population of genotypes co-evolves cooperatively with a separate population of mappings. The two populations are linked via a probability table. This approach, and a linear genetic programming approach, are then tested on different technology stocks, both for interday as well as intraday data. The results show that both methods can produce profits and that relative advantages can be traced back to differences in the market situation.

## References

1. Brabazon, A., O'Neill, M.: *Biologically Inspired Algorithms for Financial Modelling*. Springer, Berlin (2006)
2. Brabazon, A., O'Neill, M. (eds.): *Natural Computing in Computational Finance*. Springer, Berlin (2008)
3. Brabazon, A., O'Neill, M. (eds.): *Natural Computing in Computational Finance*, vol. 2. Springer, Berlin (2009)
4. de Castro, L.N.: Fundamentals of natural computing: an overview. *Physics of Life Reviews* 4(1), 1–36 (2007)
5. Grant, J.: Computer-driven trading boom raises meltdown fears, p. 19. *Financial Times* (January 26, 2010)
6. Maringer, D.: *Portfolio Management with Heuristic Optimization*. Springer, Berlin (2005)
7. Tesfatsion, L., Judd, K. (eds.): *Handbook of Computational Economics. Agent-Based Computational Economics*, vol. 2. North-Holland, Amsterdam (2006)
8. Wong, B., Lai, V., et al.: A bibliography of neural network business applications research: 1994–1998. *Computers and Operations Research* 27, 1045–1076 (2000)



**Financial and Agent-Based Models**

---

# Robust Regression with Optimisation Heuristics

Manfred Gilli and Enrico Schumann

Department of Econometrics, University of Geneva,  
Bd du Pont d'Arve 40, 1211 Geneva 4, Switzerland  
Manfred.Gilli@unige.ch, Enrico.Schumann@unige.ch

**Summary.** Linear regression is widely-used in finance. While the standard method to obtain parameter estimates, Least Squares, has very appealing theoretical and numerical properties, obtained estimates are often unstable in the presence of extreme observations which are rather common in financial time series. One approach to deal with such extreme observations is the application of robust or resistant estimators, like Least Quantile of Squares estimators. Unfortunately, for many such alternative approaches, the estimation is much more difficult than in the Least Squares case, as the objective function is not convex and often has many local optima. We apply different heuristic methods like Differential Evolution, Particle Swarm and Threshold Accepting to obtain parameter estimates. Particular emphasis is put on the convergence properties of these techniques for fixed computational resources, and the techniques' sensitivity for different parameter settings.

## 2.1 Introduction

Linear regression is a widely-used tool in finance. A common practice is, for instance, to model the returns of single assets as a linear combination of the returns of various types of 'factors'. Such regressions can then be used to explain past returns, or in attempts to forecast future returns. In financial economics, such factor models are the main tools for asset pricing, for instance in the Capital Asset Pricing Model (CAPM), or in the Arbitrage Pricing Theory (APT). Even if these models, when interpreted as equilibrium models, do not hold in practice, the underlying regressions are still valuable. A main area of application is risk management, where the regression estimates can be used to construct variance–covariance matrices. There is considerable evidence of the usefulness of such models in this context [6].

Regression models may not only be used to inform financial decisions by analysing assets, but may be more explicitly used when constructing portfolios. For instance, a possible approach to replicate a portfolio or an index is to find investable assets whose returns 'explain' the chosen regressand (eg, the index); see for instance [26]. Assume we have  $p$  assets, and let the symbol  $x_i$  stand for the return of asset  $i$  at some point in time; we use  $x_i^*$  for the excess return over a constant riskfree rate. If a riskfree asset exists, mean–variance portfolio optimisation reduces to finding the portfolio with the maximum Sharpe ratio. This optimisation problem can be rewritten as

$$1 = \theta_1 x_1^* + \theta_2 x_2^* + \dots + \theta_p x_p^* + \epsilon$$

where  $\theta_i$  are the coefficients to be estimated, and  $\epsilon$  holds the errors. Estimating the  $\theta_i$  with Least Squares and rescaling them to conform with the budget constraint is equivalent to solving a mean–variance problem for the tangency portfolio weights, see [4]. The approach is outlined in the Appendix.

We can also find the global minimum-variance portfolio by running a regression [19]. We write the portfolio return as the sum of its expectation  $\mu$  and an error  $\epsilon$ , hence

$$\mu + \epsilon = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_p x_p.$$

Imposing the budget constraint  $\sum \theta = 1$  and rearranging we get

$$x_p = \mu + \theta_1(x_p - x_1) + \theta_2(x_p - x_2) + \dots + \theta_{p-1}(x_p - x_{p-1}) + \epsilon.$$

We can directly read off the portfolio weights from the regression; the weight of the  $p$ th position is determined via the budget constraint.

Finally, linear models are used to evaluate the ex post performance of investment managers: since [28], ‘style analysis’ has become a building block in performance measurement and evaluation. The regression coefficients are then interpreted as portfolio weights and the residuals as managerial skill (or luck).

The standard method to obtain parameter estimates for a linear regression model is Least Squares (LS). LS has very appealing theoretical and numerical properties, but the resulting estimates are often unstable if there exist extreme observations which are common in financial time series [5, 21, 11]. In fact, a few or even a single extreme data point can heavily influence the resulting estimates. A much-studied example is the estimation of  $\beta$ -coefficients for the CAPM, where small changes in the data (resulting, for instance, from a moving-window scheme) often lead to large changes in the estimated  $\beta$ -values. Earlier contributions in the finance literature suggested some form of shrinkage of extreme coefficients towards more reasonable levels, with different theoretical justifications (see for example [2, 32, 20]). An alternative approach, which we will deal with in this Chapter, is the application of robust or resistant estimation methods [5, 22].

There is of course a conceptual question as to what constitutes an extreme observation or outlier in financial time series. Extreme returns may occur rather regularly, and completely disregarding such returns by dropping or winsorising them could mean to throw away information. Errors in the data, though, for example stock splits that have not been accounted for, are clearly outliers. Such data errors occur on a wide scale, even with commercial data providers [18]. Hence in particular if data are processed automatically, alternative techniques like robust estimation methods may be advisable.

In this Chapter, we will discuss the application of robust estimators. Such estimators were specially designed not to be influenced too heavily by outliers, even though this characteristic often comes at the price of low efficiency if the data actually contain no outliers. Robust estimators are often characterised by their breakdown value. In words, the breakdown point is the smallest percentage of contaminated (outlying) data that may cause the estimator to be affected by an arbitrary bias [25]. While LS has a breakdown point of 0%, other estimators have breakdown points of up to 50%. Unfortunately, the estimation becomes much more difficult, and for many models only approximative solutions exist. We will describe the application of heuristics to such optimisation problems. More precisely, we will compare different optimisation methods, namely Differential

Evolution, Particle Swarm, and Threshold Accepting. All three methods are general-purpose heuristics and have been successfully applied to a wide range of problems, see for instance [23], [33].

The remaining Chapter is structured as follows: Section 2.2 will introduce the linear regression model and several alternative optimisation criteria for parameter estimation. Section 2.3 will discuss numerical estimation strategies, ie, we will discuss different optimisation procedures. In Section 2.4 then, we use the Monte-Carlo setup from [27] to test the convergence behaviour of the different optimisation methods when used for a specific estimator, Least Median of Squares. Section 2.5 concludes.

## 2.2 The Linear Regression Model

We consider the linear regression model

$$y = \begin{bmatrix} x_1 & \cdots & x_p \end{bmatrix} \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_p \end{bmatrix} + \epsilon.$$

Here,  $y$  is a vector of  $n$  observations of the independent variable; there are  $p$  regressors whose observations are stored in the column vectors  $x_j$ . We will usually collect the regressors in a matrix  $X = [x_1 \cdots x_p]$ , and write  $\theta$  for the vector of all coefficients. The  $j$ th coefficient is denoted by  $\theta_j$ . We will normally include a constant as a regressor, hence  $x_1$  will be a vector of ones. The residuals  $r$  (ie, the estimates for the  $\epsilon$ ), are computed as

$$r = y - X\hat{\theta}$$

where  $\hat{\theta}$  is an estimate for  $\theta$ . Least Squares (LS) requires to minimise the sum or, equivalently, the mean of the squared residuals, hence the estimator is defined as

$$\hat{\theta}_{\text{LS}} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n r_i^2.$$

The advantage of this estimator is its computational tractability: the LS solution is found by solving the system of normal equations

$$(X'X)\theta = X'y$$

for  $\theta$ .

Rousseeuw [24] suggested to replace the mean of the squared residuals with their median. The resulting Least Median of Squares (LMS) estimator can be shown to be less sensitive to outliers than LS; in fact, LMS's breakdown point is almost 50%. More formally, LMS is defined as

$$\hat{\theta}_{\text{LMS}} = \underset{\theta}{\operatorname{argmin}} \operatorname{median}(r^2).$$

LMS can be generalised to the Least Quantile of Squares (LQS) estimator. Let  $Q_q$  be the  $q$ th quantile of the squared residuals, that is

$$Q_q = \text{CDF}^{-1}(q) = \min\{r_i^2 \mid \text{CDF}(r_i^2) \geq q\}, \tag{2.1}$$

where  $q$  may range from 0% to 100% (we drop the %-sign in subscripts). Hence the LMS estimator becomes

$$\hat{\theta}_{\text{LMS}} = \underset{\theta}{\text{argmin}} Q_{50}(r^2),$$

and more generally we have

$$\hat{\theta}_{\text{LQS}} = \underset{\theta}{\text{argmin}} Q_q(r^2).$$

For a given sample, several numbers satisfy definition (2.1), see (17). A convenient approach is to work directly with the order statistics  $[r_{[1]}^2 \ r_{[2]}^2 \ \dots \ r_{[n]}^2]'$ . For LMS, for instance, the maximum breakdown point is achieved not by minimising  $Q_{50}(r^2)$ , but by defining

$$h = \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{p+1}{2} \right\rfloor \tag{2.2}$$

and minimising  $r_{[h]}^2$  (24)[p. 873].

The Least Trimmed Squares (LTS) estimator requires to minimise the order statistics of  $r^2$  up to some maximum order  $k$ . Formally,

$$\hat{\theta}_{\text{LTS}} = \underset{\theta}{\text{argmin}} \frac{1}{k} \sum_{i=1}^k r_{[i]}^2.$$

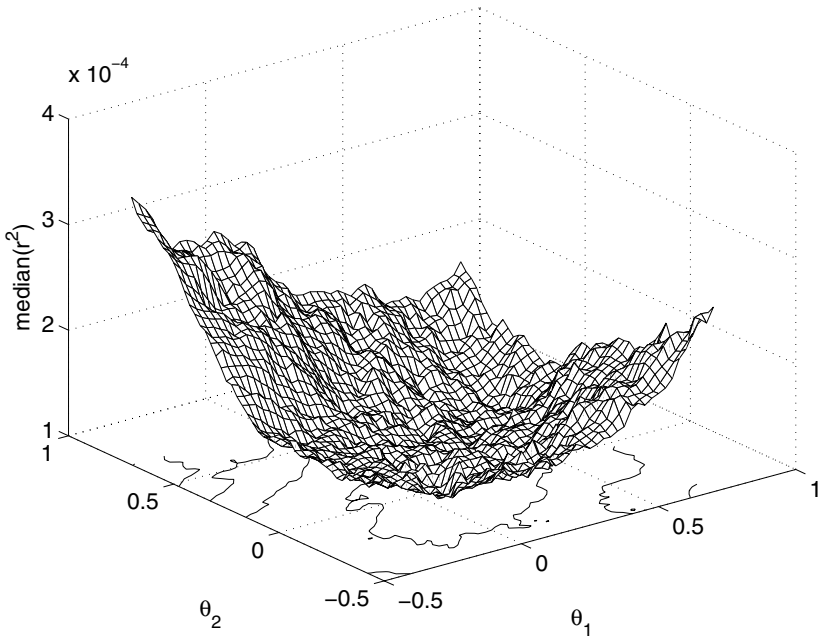
To achieve a high breakdown value, the number  $k$  is set to roughly  $\lfloor 1/2(n + p + 1) \rfloor$ , or the order statistic defined in Equation (2.2).

LQS and LTS estimators are sometimes called ‘resistant’ estimators, since they do not just reduce the weighting of outlying points, but essentially ignore them. This property in turn results in a low efficiency if there are no outliers. However, we can sometimes exploit this characteristic when we implement specific estimators.

## 2.3 Estimation

### 2.3.1 Strategies

Robust estimation is computationally more difficult than LS estimation. A straightforward estimation strategy is to directly map the coefficients of a model into the objective function values, and then to evolve the coefficients according to a given optimisation method until a ‘good’ solution is found. For LMS, for instance, we may start with a ‘guess’ of the parameters  $\theta$  and then change  $\theta$  iteratively until the median squared residual cannot be reduced any further. We will refer to this strategy as the ‘direct approach’.



**Fig. 2.1.** Search space for LMS.

The difficulty with the direct approach arises from the many local minima that the objective function exhibits. This is illustrated in Figure 2.1 which shows the mapping from a given set of coefficients into the median squared residual (ie, the search space) for a CAPM regression  $y = \theta_1 + \theta_2 x + \epsilon$  (here  $y$  is the excess return of a specific asset over the riskfree rate, and  $x$  the excess return of the market).

Heuristic methods deploy different strategies to overcome such local minima. We will compare three different techniques – Differential Evolution, Particle Swarm, and Threshold Accepting – for the direct approach.

Since many resistant estimators essentially fit models on only a subset of the data, we may also associate such subsets with particular objective function values – hence transform the estimation into a combinatorial problem. An intuitive example is the LTS estimator: since the objective is to minimise the sum of the  $k$  smallest squared residuals, we could also, for every subset of size  $k$ , estimate LS-coefficients. The subset with the minimum objective function will give us the exact solution to the problem. Since such a complete enumeration strategy is clearly infeasible for even moderately-sized models, we will investigate an alternative search strategy based on Threshold Accepting. We refer to this estimation strategy as the ‘subset approach’.

In the remainder of this Chapter, we will limit ourselves to LMS estimation. The direct approach is, however, applicable to any estimation criterion that allows to directly connect the coefficients to the residual vector  $r$ . The subset approach presented later is applicable to LQS estimation; it could easily be modified (in fact, simplified) for LRS. Next we outline the different algorithms.

### 2.3.2 Differential Evolution

Differential Evolution (DE) was developed for continuous optimisation problems [29], we outline the procedure in Algorithm 2.1. DE evolves a population of  $n_p$  solutions, stored in real-valued vectors of length  $p$  (ie, the number of coefficients of the regression model). The population  $P$  may be visualised as a matrix of size  $p \times n_p$ , where each column holds one candidate solution. In every iteration (or ‘generation’), the algorithm goes through the columns of this matrix and creates a new candidate solution for each existing solution  $P_{:,i}^{(0)}$ . This candidate solution is constructed by taking the difference between two other solutions, weighting this difference by a parameter  $F$ , and adding it to a third solution. Then an element-wise crossover takes place with probability  $CR$  between this auxiliary solution  $P_{:,i}^{(v)}$  and the existing solution  $P_{:,i}^{(0)}$  (the symbol  $\zeta$  represents a random variable that is uniformly distributed between zero and one). If this final candidate solution  $P_{:,i}^{(u)}$  is better than  $P_{:,i}^{(0)}$ , it replaces it; if not, the old solution  $P_{:,i}^{(0)}$  is kept.

---

#### Algorithm 2.1. Differential Evolution.

---

```

initialise parameters  $n_p, n_G, F$  and  $CR$ ;
initialise population  $P_{j,i}^{(1)}, j = 1, \dots, p, i = 1, \dots, n_p$ ;
for  $k = 1$  to  $n_G$  do
     $P^{(0)} = P^{(1)}$ ;
    for  $i = 1$  to  $n_p$  do
        generate  $\ell_1, \ell_2, \ell_3 \in \{1, \dots, n_p\}, \ell_1 \neq \ell_2 \neq \ell_3 \neq i$ ;
        compute  $P_{:,i}^{(v)} = P_{:,i}^{(0)} + F \times (P_{:, \ell_2}^{(0)} - P_{:, \ell_3}^{(0)})$ ;
        for  $j = 1$  to  $p$  do
            if  $\zeta < CR$  then  $P_{j,i}^{(u)} = P_{j,i}^{(v)}$  else  $P_{j,i}^{(u)} = P_{j,i}^{(0)}$ ;
        end
        if  $\Phi(P_{:,i}^{(u)}) < \Phi(P_{:,i}^{(0)})$  then  $P_{:,i}^{(1)} = P_{:,i}^{(u)}$  else  $P_{:,i}^{(1)} = P_{:,i}^{(0)}$ ;
    end
end

```

---

### 2.3.3 Particle Swarm Optimisation

The narrative for Particle Swarm Optimisation (PS) is based on swarms of animals like birds or fish that look for food [9]. Like DE, PS is applicable to continuous problems; Algorithm 2.2 details the procedure. We have, again, a population that comprises  $n_p$  solutions, stored in real-valued vectors. In every generation, a solution is updated by adding another vector called velocity  $v_i$ . We may think of a solution as a position in the search space, and of velocity as a direction into which the solution is moved. Velocity changes over the course of the optimisation, the magnitude of change is the sum of two components: the direction towards the best solution found so far by the particular solution,  $Pbest_i$ , and the direction towards the best solution of the whole population,  $Pbest_{gbest}$ . These two directions are perturbed via multiplication with a uniform random variable  $\zeta$  and constants  $c_{(.)}$ , and summed, see Statement 2.2. The vector so obtained is added to the previous  $v_i$ , the resulting updated velocity is added to the respective

**Algorithm 2.2.** Particle Swarm.

---

```

initialise parameters  $n_p, n_G, \delta, c_1$  and  $c_2$ ;
initialise particles  $P_i^{(0)}$  and velocity  $v_i^{(0)}, i = 1, \dots, n_p$ ;
evaluate objective function  $F_i = \Phi(P_i^{(0)}), i = 1, \dots, n_p$ ;
 $Pbest = P^{(0)}, Fbest = F, Gbest = \min_i(F_i), gbest = \operatorname{argmin}_i(F_i)$ ;
for  $k = 1$  to  $n_G$  do
  for  $i = 1$  to  $n_p$  do
     $\Delta v_i = c_1 \times \zeta_1 \times (Pbest_i - P_i^{(k-1)}) + c_2 \times \zeta_2 \times (Pbest_{gbest} - P_i^{(k-1)})$ ;
     $v_i^{(k)} = \delta v_i^{(k-1)} + \Delta v_i$ ;
     $P_i^{(k)} = P_i^{(k-1)} + v_i^{(k)}$ ;
  end
  evaluate objective function  $F_i = \Phi(P_i^{(k)}), i = 1, \dots, n_p$ ;
  for  $i = 1$  to  $n_p$  do
    if  $F_i < Fbest_i$  then  $Pbest_i = P_i^{(k)}$  and  $Fbest_i = F_i$ ;
    if  $F_i < Gbest$  then  $Gbest = F_i$  and  $gbest = i$ ;
  end
end

```

---

solution. In some implementations, the velocities are reduced in every generation by setting the parameter  $\delta$  to a value smaller than unity.

### 2.3.4 Threshold Accepting (Direct Approach)

Threshold Accepting (TA) is a descendant of Simulated Annealing and was introduced by [8]. Other than DE and PS, TA is a so-called trajectory method and evolves only a single solution. It is based on a local search [14] but may, like Simulated Annealing, also move ‘uphill’ in the search space. More specifically, it accepts new solutions that are inferior when compared with the current solution, as long as the deterioration does not exceed a specified threshold, thus the method’s name. Over time, this threshold decreases to zero, and so TA turns into a classical local search. Algorithm 2.3 describes the procedure; for an in-depth description see [33].

**Algorithm 2.3.** Threshold Accepting.

---

```

initialise  $n_{Rounds}$  and  $n_{Steps}$ ;
compute threshold sequence  $\tau$ ;
randomly generate current solution  $\theta^c$ ;
for  $r = 1 : n_{Rounds}$  do
  for  $i = 1 : n_{Steps}$  do
    generate  $\theta^n \in \mathcal{N}(\theta^c)$  and compute  $\Delta = \Phi(\theta^n) - \Phi(\theta^c)$ ;
    if  $\Delta < \tau_r$  then  $\theta^c = \theta^n$ ;
  end
end
 $\theta^{sol} = \theta^c$ ;

```

---



Here,  $\theta^c$  denotes the current solution, and  $\theta^n$  is the ‘new’ (or neighbour) solution. For each of the  $n_{\text{Rounds}}$  thresholds, stored in the vector  $\tau$ , the algorithm performs  $n_{\text{Steps}}$  iterations, so the number of objective function evaluations is  $n_{\text{Rounds}} \times n_{\text{Steps}}$ .

---

**Algorithm 2.4.** Threshold Accepting – Neighbourhood definition.

---

$\theta^n = \theta^c$ ;  
 randomly select  $j \in \{1, \dots, p\}$ ;  
 randomly generate  $\zeta \in [-z, z]$ ;  
 $\theta_j^n = \theta_j^c + \zeta \times (1 + |\theta_j^c|)$ ;

---

### Neighbourhood Definition

While  $\tau_A$  was originally introduced for combinatorial (ie, discrete) problems, it can easily be modified for continuous functions. We implement the neighbourhood function  $\mathcal{N}$  as a small perturbation of the current coefficients vector. We use a random step size that is proportional to the respective coefficient (see Algorithm 2.4). Variations are possible; [35] for example suggest to shrink the step size over time.

The constant 1 is added in Statement 2.4 to make a sign-change for the given parameter more probable: without such a constant, when a coefficient gets closer to zero in absolute terms, its variation also goes to zero.

### Threshold Sequence

To compute the threshold sequence we take a random walk through the solution space under the specified neighbourhood function and record the changes in the objective function. The thresholds are then equidistant quantiles of the distribution of the absolute values of the changes. For the rationale of this approach see [34, 15].

This procedure requires the number of thresholds  $n_{\text{Rounds}}$  to be set in advance. We set  $n_{\text{Rounds}}$  to 10, even though  $\tau_A$  is robust for other choices. There is some evidence, though, that for very small numbers of thresholds, for instance 2 or 3, the performance of the algorithm deteriorates [13].

---

**Algorithm 2.5.** Computing the threshold sequence.

---

randomly choose  $\theta^c$ ;  
**for**  $i = 1 : n_{\text{Deltas}}$  **do**  
 | compute  $\theta^n \in \mathcal{N}(\theta^c)$  and  $\Delta_i = |\Phi(\theta^c) - \Phi(\theta^n)|$ ;  
 |  $\theta^c = \theta^n$ ;  
**end**  
 compute empirical distribution CDF of  $\Delta_i, i = 1, \dots, n_{\text{Deltas}}$ ;  
 compute threshold sequence  $\tau_r = \text{CDF}^{-1}\left(\frac{n_{\text{Rounds}} - r}{n_{\text{Rounds}}}\right), r = 1, \dots, n_{\text{Rounds}}$ ;

---

**Algorithm 2.6.** Chebyshev regression for  $p + 1$  subset.

---

solve  $(X'_s X_s)\theta = X'_s y_s$  for  $\theta$ ;  
 compute  $r_s = y_s - X_s \theta$ ;  
 compute  $\omega = \sum r_s^2 / \sum |r_s|$ ;  
 compute  $\sigma = \text{sign}(r_s)$ ;  
 compute  $y_s^* = y_s - \omega \sigma$ ;  
 solve  $(X'_s X_s)\theta = X'_s y_s^*$  for  $\theta$ ;  
 $\theta_c = \theta$ ;

---

**2.3.5 Threshold Accepting (Subset Approach)**

Let  $r_{[h]}^2$  denote the median order statistic of the squared residuals. [30] noted that an estimator that minimises  $r_{[h]}^2$  is equivalent to an estimator that minimises the largest squared residual for a subset of size  $h$ . This is almost equivalent to the so-called Chebyshev estimator for this subset, defined by

$$\hat{\theta}_c = \underset{\theta}{\operatorname{argmin}} \max |r_i|.$$

(Only ‘almost equivalent’ because of using the absolute value instead of squaring the residuals.) A convenient fact about  $\hat{\theta}_c$  is that there exists also a subset of size  $p + 1$  that yields the same fit as a  $h$ -subset. More generally, the LQS estimator for any order statistic  $h$  (not just LMS) corresponds to a Chebyshev estimate of some subset of size  $p + 1$ . Thus a solution with this approach is identified by  $p + 1$  indices, pointing to specific rows in  $[y \ X]$ . Then, by computing  $\hat{\theta}_c$  for this subset, we obtain a link from the subset into an objective function value for the total data set. [30] suggested to examine all subsets of size  $p + 1$ , which is infeasible even for small models. We will thus apply TA to this subset selection problem. Algorithms 2.3 and 2.5 remain valid; the neighbourhood is implemented as an exchange of one element from the solution against an index that is currently not in the solution. (A similar approach is taken in [10] for quantile regression.)

We thus need to solve two nested optimisation problems: the outer loop moves through different subsets, while the inner loop needs to find  $\hat{\theta}_c$  for the given subset. Fortunately, for a subset of size  $p + 1$ , there exists an exact and fast method to compute the Chebyshev-fit. Let  $X_s$  be a subset of  $X$  of size  $(p + 1) \times p$ , the corresponding entries of  $y$  are stored in the vector  $y_s$ . Then Algorithm 2.6 describes a method, based on LS, to obtain the Chebyshev-fit [30, 11].

**2.4 Numerical Experiments**

All the considered optimisation techniques are stochastic algorithms, so restarting the same algorithm several times for the same data will result in different solutions. We characterise a solution  $\theta$  by its associated objective function value. We may now

describe the solution obtained from one optimisation run as the realisation of a random variable with an unknown distribution  $\mathcal{F}$ . For a given data set and a model to estimate (LMS in our case), the shape of  $\mathcal{F}$  will depend on the particular optimisation technique, and on the amount of computational resources spent on an optimisation run. Heuristic methods are specially designed such that they can move away from local optima, hence if we allow more iterations, we would expect the method to produce better results on average. In fact, for an ever increasing number of iterations, we would finally expect  $\mathcal{F}$  to degenerate to a single point, the global minimum. In practice, we cannot let an algorithm run forever, hence we are interested in the convergence of specific algorithms for finite amounts of computational resources. ‘Convergence’ here means the change in the shape of  $\mathcal{F}$  when we increase the number of iterations. Fortunately, it is straightforward to investigate  $\mathcal{F}$ : fix the settings for a method (data, parameters, numbers of iterations) and repeatedly restart the algorithm. Thus we obtain a sample of draws from  $\mathcal{F}$ , from which we can compute an empirical distribution function as an estimate for  $\mathcal{F}$ .

Since we deal with different heuristics – population-based techniques and trajectory methods – we define computational resources as the number of objective function evaluations. For `DE` and `PS`, this is equal to the number of generations times the population size, for `TA` it is the number thresholds times the steps per threshold. This is justified for LMS regression since the overhead incurred from evolving solutions is small compared with the run time necessary to compute the median of the squared residuals (which requires at least a partial sorting of the squared residuals). Fixing the number of function evaluations has the advantage of allowing us to compare the performance of different methods for a given amount of computational resources. However, we cannot directly compare the subset approach with the direct approach, since in the former the objective function is much more expensive.

We use the experimental setting described in [27], thus we consider the regression model

$$y = X\theta + \epsilon, \quad (2.3)$$

where  $X$  is of size  $n \times p$ ,  $\theta$  is the  $p$ -vector of coefficients, and  $\epsilon$  is Gaussian noise, ie,  $\epsilon \sim N(0, 1)$ . We always include a constant, so the first column of  $X$  is a vector of ones. The remaining elements of  $X$  and  $y$  are normally distributed with a mean of zero and a variance of one. Thus, the true  $\theta$ -values are all zero, and the estimated values should be close to zero. We replace, however, about 10% of the observations with outliers. More precisely, if a row in  $[y \ X]$  is contaminated with an outlier, it is replaced by

$$[M \ 1 \ 100 \ 0 \ \dots \ 0]$$

where  $M$  is a value between 90 and 200. This setting results in a region of local minima in the search space where  $\theta_2$  will be approximately  $M/100$ . In their paper, [27] analyse how often a given estimator converges to this wrong solution. This analysis, however, confounds two issues: the ability of a given estimator to identify the outliers on the one hand, and the numerical optimisation on the other. Since we are interested in the optimisation, we will not compare coefficients, but look at the value of the objective function.

We set  $M$  to 150, and vary the number of regressors  $p$  between 2 and 20. The number of observations  $n$  is fixed at 400.

### 2.4.1 Results: Direct Approach

All the methods employed require us to set a number of parameters. We start with ‘typical’ parameter values: for DE, we set the population size  $n_p$  to 10 times the number of coefficients; CR and F are set to 0.9 and 0.75, respectively. Thus we stay closely with the recommendations of K. Price and R. Storn (see <http://www.icsi.berkeley.edu/~storn/code.html>). For PS, we set  $n_p$  to 200,  $c_1$  to 1 and  $c_2$  to 2. Inertia ( $\delta$ ) is set to 1, hence velocity is not reduced systematically. For TA, there are no typical parameter choices, in particular since the neighbourhood function (Algorithm 2.4) is problem-specific. The variable  $z$ , which controls the size of the step, was initially set to 0.2.

Figures 2.2, 2.3 and 2.4 give the results for models with 5, 10, and 20 coefficients, respectively. We estimated the distributions (ie,  $\mathcal{F}$ ) by restarting the algorithms 1 000 times. The three panels (top to bottom) in every graphic show the resulting objective function values for 10 000, 20 000, and 30 000 function evaluations.

For the model with 5 coefficients (which is, in practice, a reasonably-sized model), DE gives the best results. With more function evaluations, the DE runs converge on a small number of local minima. The performance of DE deteriorates, though, with more

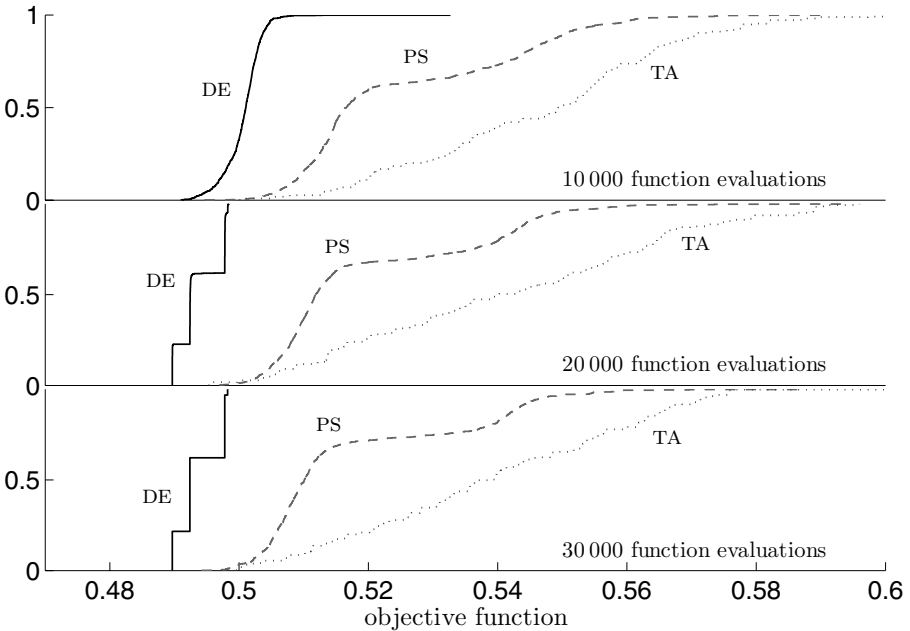
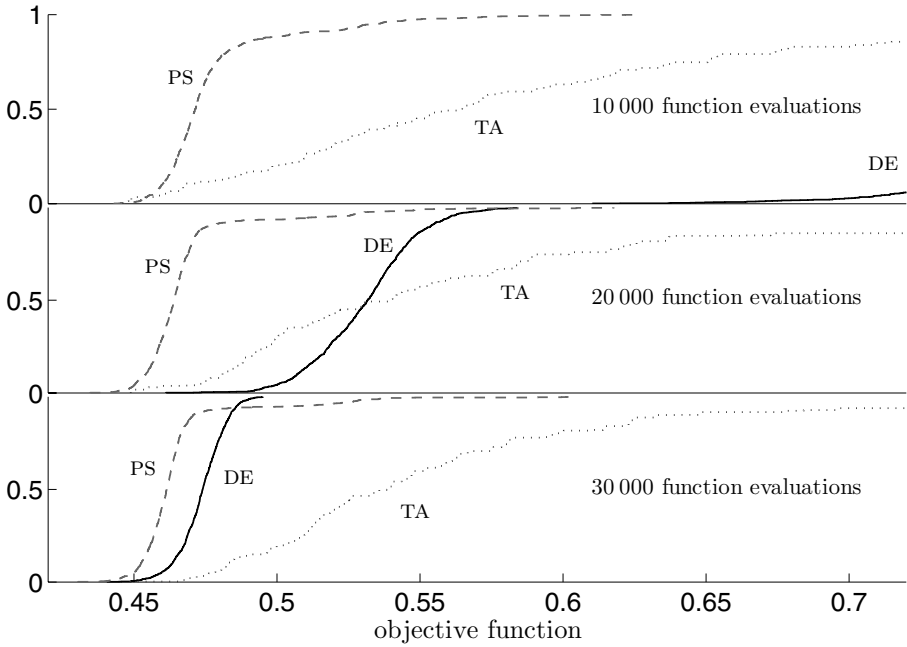


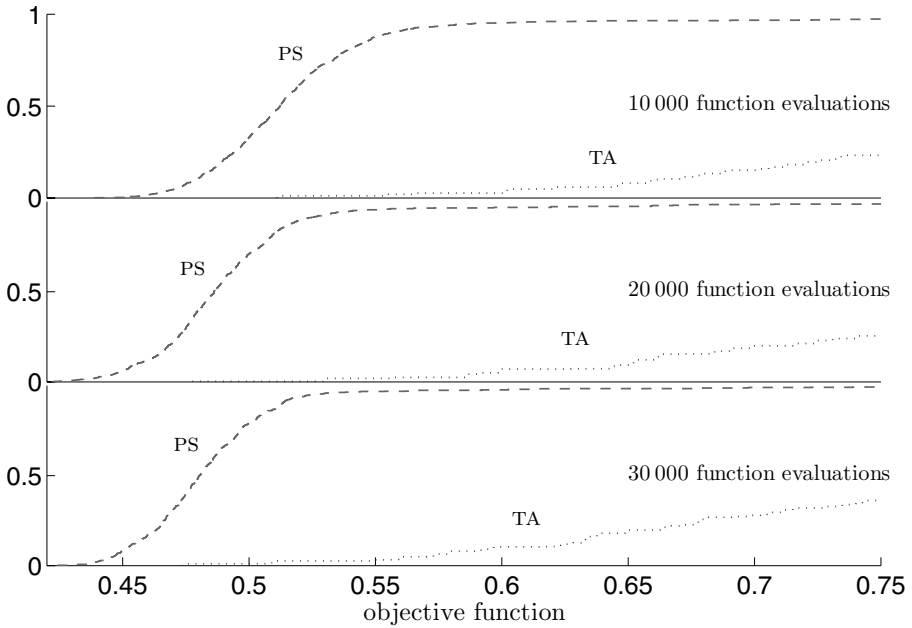
Fig. 2.2. Estimated distributions  $\mathcal{F}$ : direct approach with  $p = 5$ .



**Fig. 2.3.** Estimated distributions  $\mathcal{F}$ : direct approach with  $p = 10$ .

coefficients, ie, larger models. For  $p = 20$  no solution for DE is visible any more in Figure 2.4 the distribution is too far to the right. PS performs best for such larger models, even though the distribution is skewed to the right. In other words, the method occasionally converges on a comparatively bad solution. TA gives reasonable solutions, though generally either DE or PS give better results. In particular, the distribution of solutions for TA is rather dispersed. Take for instance the model with  $p = 20$  and 30 000 function evaluations: the probability of reaching the median solution of PS with TA is only about 1%.

These results are conditional on the chosen values for the method’s parameters. An important part of implementing heuristics is hence the ‘tuning’ of the algorithm, ie, finding ‘good’ parameter values. This search is again an optimisation problem: find those parameter values that lead to optimal (or ‘good’) results in every restart, ie, parameter values that lead to a ‘good’  $\mathcal{F}$ . Since all methods need several parameters to be set, this optimisation problem is not trivial, in particular since the objective function has to be evaluated from simulation and thus will be noisy. Though this is an (interesting) problem to be investigated, for our purposes here, we do not need such an optimisation – quite the opposite actually. Parameter setting is sometimes portrayed as an advantage, for it allows to adapt methods to different problems. True. But at the same time it requires the analyst who wishes to apply the method to have a much deeper understanding of the respective method. In other words, the analyst will have to be a specialist in optimisation, rather than in finance or econometrics.



**Fig. 2.4.** Estimated distributions  $\mathcal{F}$ : direct approach with  $p = 20$ .

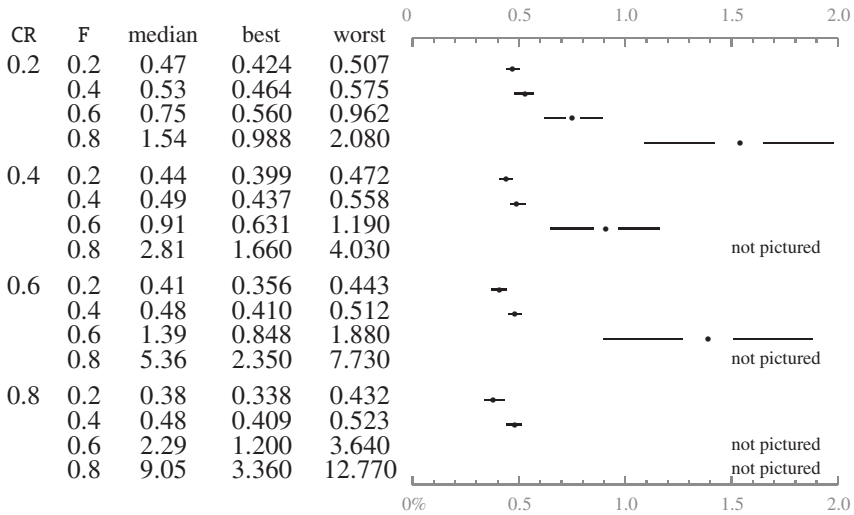
Boyd and Vandenberghe [3] [p. 5] call a method for solving a particular problem ‘a (mature) *technology*, [if it] can be reliably used by many people who do not know, and do not need to know, the details.’ (Their example is, fittingly, LS.) If heuristics are to become a technology in this sense, the more pressing question is not whether we have used the ‘optimal’ parameters, but how sensitive our method’s solutions are to specific parameter settings. Since this is a volume on computational finance, let us give a financial analogy: while parameter optimisation may be regarded equivalent to the trading side of a business, we are more interested in risk management.

To illustrate this point, we look at the model with  $p = 20$  which proved the most difficult, and solve it with different settings for the parameters. The number of function evaluations was set to 30 000. For every parameter setting we conducted 1 000 restarts. All calculations are based on the same data, hence the results in the following tables are directly comparable for different methods.

### Parameter Sensitivity for Differential Evolution

Table 2.1 shows the results when we vary  $F$  and  $CR$ . We include the median, best, and worst value of the obtained solutions. Furthermore we include quartile plots [31, 12] of the distributions. A quartile plot is constructed like a boxplot, but without the box: it only shows the median (the dot in the middle) and the ‘whiskers’.

**Table 2.1.** Parameter sensitivity DE.



The solutions returned by DE improve drastically when we set F to low values while different choices for CR have less influence. This suggests that for LMS-regression, using DE needs to be accompanied by testing of the robustness of the solutions. With small F, we evolve the solutions by adding small changes at several dimensions of the solution. In a sense, then, we have a population of local searches, or at least of slowly-moving individuals.

**Parameter Sensitivity for Particle Swarm Optimisation**

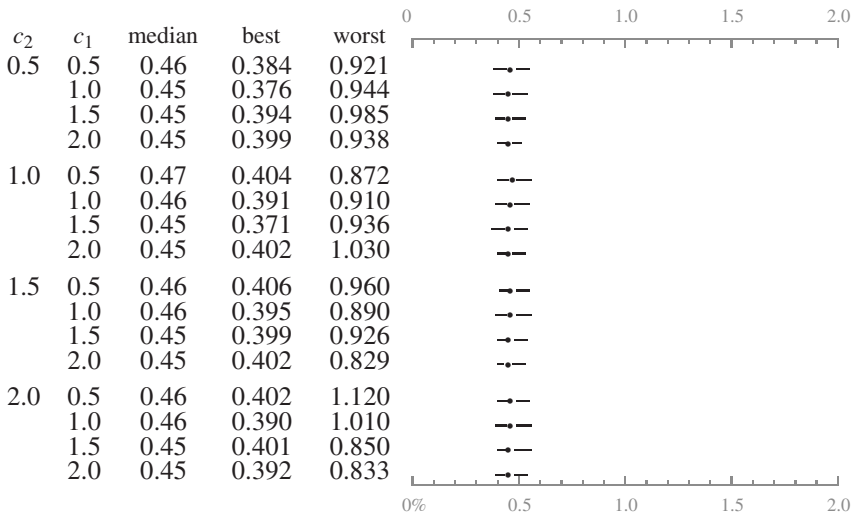
Tables 2.2-2.5 give the result for PS; here the picture is different. While there are differences in the results for different settings of the parameters, the results are more stable when we vary  $\delta$ ,  $c_1$  and  $c_2$ . Each table gives results for different values of  $c_1$  and  $c_2$ , with  $\delta$  fixed for the whole table. The most salient result is that velocity should not be reduced too fast, hence  $\delta$  should be below but close to one.

Though not reported here, we also reran our initial tests (Figures 2.2, 2.3 and 2.4). With ‘improved’ parameter values for both DE and PS, both methods performed equally well for small models, but PS still was superior for large models.

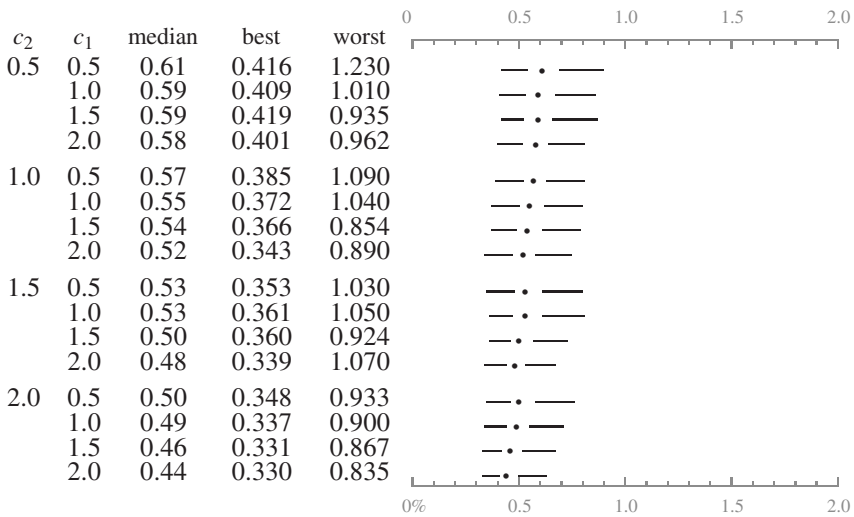
**Parameter Sensitivity for Threshold Accepting**

We ran TA with different values for  $z$  (see Algorithm 2.4): 0.05, 0.10, and 0.20. Table 2.6 gives the results. The results indicate that  $z$  should be small; in our setting 0.05 performed best on average. At the same time, reducing  $z$  deteriorated the worst solution. Thus for too small step sizes, TA more often seemed to get stuck in local, but globally suboptimal, minima.

**Table 2.2.** Parameter sensitivity  $\rho_s$  for  $\delta = 1$ .



**Table 2.3.** Parameter sensitivity  $\rho_s$  for  $\delta = 0.5$ .

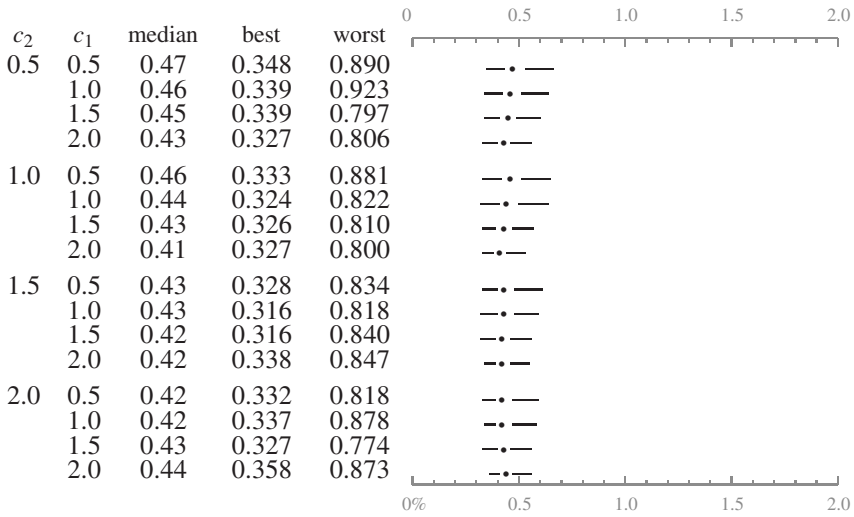


### 2.4.2 Results: Subset Approach

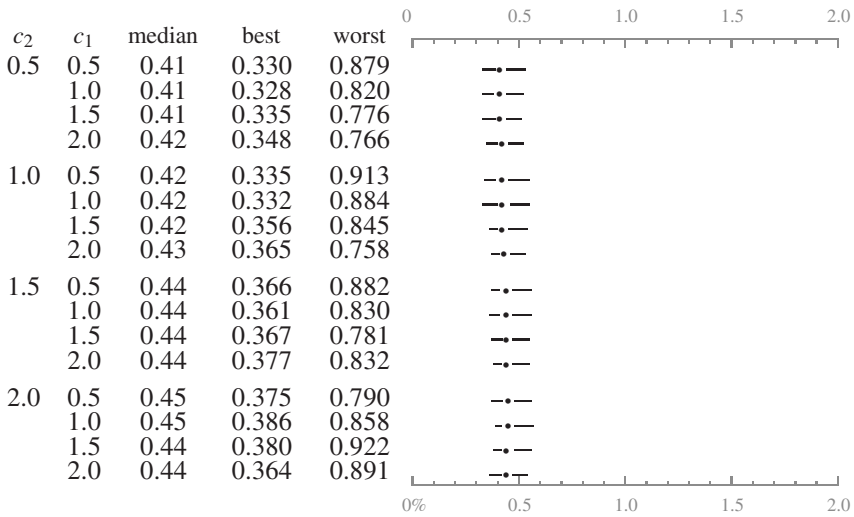
As a first benchmark for our algorithm we ran a greedy search, described in Algorithm 2.7. That is, for some random initial solution we check all neighbours, and always move to the best one, given it improves the current solution. For any given solution, there are  $(p + 1)(n - p - 1)$  neighbours, hence visiting them all is time-consuming but still feasible. If, at some point, no improvement can be found any more, the search stops.



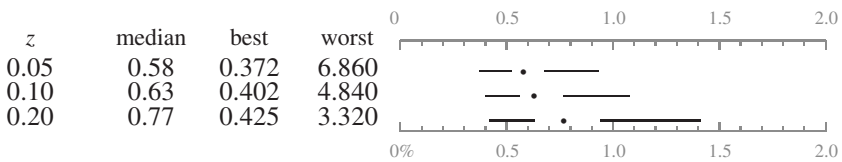
**Table 2.4.** Parameter sensitivity  $\rho_s$  for  $\delta = 0.75$ .



**Table 2.5.** Parameter sensitivity  $\rho_s$  for  $\delta = 0.9$ .



**Table 2.6.** Parameter sensitivity  $\tau_A$ .



**Algorithm 2.7.** Greedy search for subset selection.

---

```

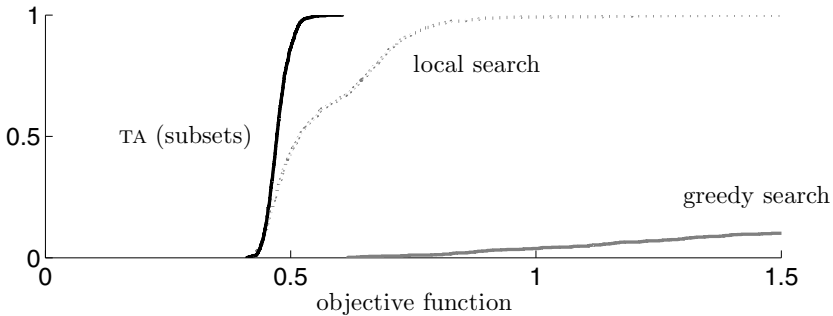
select random initial solution  $\theta^c$ ;
set converged = false;
while not converged do
    choose best neighbour  $\theta^{\text{best}} = \operatorname{argmin}_{\theta^n \in \mathcal{N}(\theta^c)} \Phi(\theta^n)$ ;
    if  $\Phi(\theta^{\text{best}}) < \Phi(\theta^c)$  then
        |  $\theta^c = \theta^{\text{best}}$ ;
    end
    converged = true;
end
 $\theta^{\text{sol}} = \theta^c$ ;

```

---

A second benchmark is a classical local search: we start with a random solution and choose a neighbour randomly. If the neighbour is better than the current solution, we move to this new solution. This is equivalent to  $\text{TA}$  with just one zero-threshold. Results for both searches are shown in Figure 2.5 ( $p = 10$ ), again the distributions are computed from 1 000 restarts. We also add the results for a subset-selection  $\text{TA}$  with 10 000 function evaluations. Local search performs already much better than the greedy search, and even reaches solutions as good as the  $\text{TA}$ . The  $\text{TA}$  runs result in a very steep distribution, thus giving consistently better solutions than the benchmarks.

To illustrate the quality of the solutions obtained with the subset approach, we next plot results for all methods (direct approach and subset approach) for 10 000 function evaluations. It needs to be stressed, though, that the objective function for the subset approach is computationally much more expensive than for the direct approach (one restart needs about 5 times the computing time). We set the parameters of the direct approach techniques to ‘good’ values (DE: F is 0.2, CR is 0.8; PS:  $\delta$  is 0.75,  $c_1$  is 2 and  $c_2$  is 1;  $\text{TA}$ :  $z$  is 0.05.) We give just selected results to outline the general findings: with a low level of contamination (10%), for small models, the subset approach gives very good solutions, but lacks behind PS once the model grows. The subset-selection  $\text{TA}$  is,



**Fig. 2.5.** Estimated distributions  $\mathcal{F}$ : greedy search, local search, and  $\text{TA}$  (subsets).

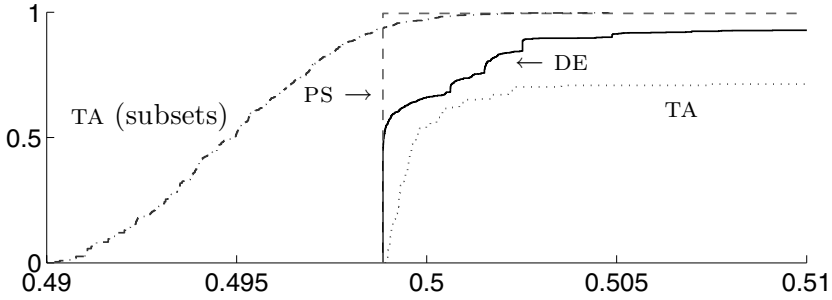


Fig. 2.6. Comparison of  $\mathcal{F}$  for models with 10 000 function evaluations ( $p = 2$ ).

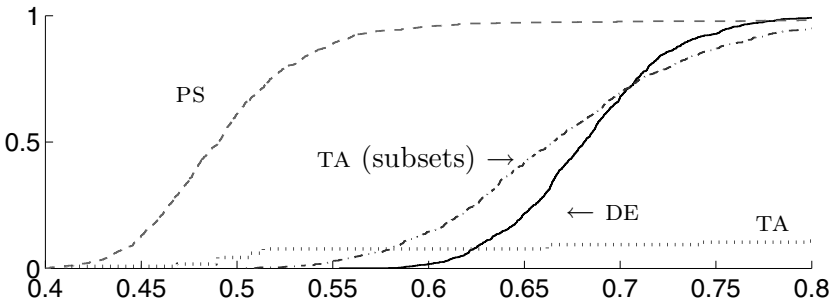


Fig. 2.7. Comparison of  $\mathcal{F}$  for models with 10 000 function evaluations ( $p = 20$ ).

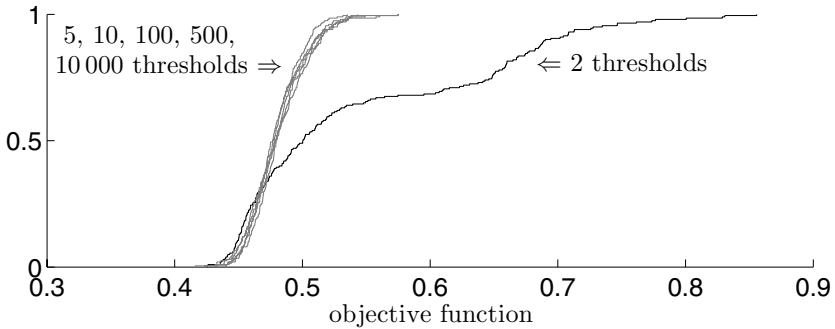


Fig. 2.8. TA (subsets): distributions  $\mathcal{F}$  for different numbers of thresholds.

however, very robust when the degree of contamination increases, ie, when the number of outliers increases.

Figure 2.6 and 2.7 show results for  $p = 2$  and  $p = 20$  with 10% outliers. The distributions are obtained from 1 000 restarts.

## Parameter Sensitivity for Threshold Accepting

We ran tests where we fixed the number of function evaluations, but varied the distribution between thresholds ( $n_{\text{Rounds}}$ ) and steps per thresholds ( $n_{\text{Steps}}$ ) (see Algorithm 2.3). Figure 2.8 shows the resulting distributions for 10 000 function evaluations.

The algorithm performs worse for very small numbers of thresholds, but once more than about five thresholds are used, performance becomes stable.

## 2.5 Conclusion

In this Chapter we described how optimisation heuristics can be used for robust regression. More precisely, we investigated whether Differential Evolution, Particle Swarm Optimisation, and Threshold Accepting are able to minimise the median squared residual of a linear model.

While all the tested methods seem capable of giving ‘good’ solutions to the LMS-problem, the computational resources (ie, number of function evaluations) would have to be increased drastically to make the distribution of outcomes collapse to a narrow support. In other words, there always remains stochasticity in the solutions. It is difficult to judge the importance of this remaining randomness without a particular application.

For the direct approach we found that while DE performed well for small models, the obtained results were very sensitive to the specific parameter settings once we estimated models with more coefficients. PS showed a much more robust performance. When using good parameter values for both DE and PS, the latter method always dominated DE in our tests. The TA implementations were less efficient in the sense of having much more variable distributions of solutions. The subset approach was more expensive in terms of computing time, but had the advantage of being very robust for different models, in particular for high levels of contamination.

Given its speed and robustness, PS would certainly be our first choice for LMS-estimation. But there are several points to be kept in mind. Firstly, all results are conditional on our model setup. The study of [27] uses one specific data setup; for alternative data the results do not have to be similar. Furthermore, while PS performed well on average, some restarts returned low-quality solutions. It is difficult to judge the relevance of such outcomes: the errors that may occur from the optimisation have to be weighted in light of the actual application, eg, a portfolio construction process. Our suggestion for actual implementations is thus to diversify, that is to implement several methods for the problem given, at least as benchmarks or test cases.

## Acknowledgement

The authors gratefully acknowledge financial support from the EU Commission through MRTN-CT-2006-034270 COMISEF. The chapter partially builds on work with Alfio Marazzi whom the authors would like to thank.

## A Maximising the Sharpe Ratio

Assume there are  $p$  assets, with expected excess returns (over the riskfree rate) collected in a vector  $\bar{x}$ . The variance–covariance matrix of the assets' returns is  $\Omega$ . Maximising the Sharpe ratio can be formalised as

$$\max_{\theta} \frac{\theta' \bar{x}}{\sqrt{\theta' \Omega \theta}}.$$

The first-order conditions of this problem lead to the system of linear equations

$$\bar{x} = \Omega \theta,$$

see for instance [7] [ch. 6]. Solving the system and rescaling  $\theta$  to sum to unity gives the optimal weights.

Assume now that we have  $T$  observations; we define  $\bar{x}$  to be the sample mean, and collect the  $p$  return series in a matrix  $X$  of size  $T \times p$ . For the regression representation as proposed in [4], we need to solve

$$\iota = X\theta^*$$

(which is an LS problem here), where  $\iota$  is the unit vector and the superscript  $*$  only serves to differentiate between  $\theta$  and  $\theta^*$ .

This can be rewritten as

$$\begin{aligned} \frac{1}{T} X' \iota &= \frac{1}{T} X' X \theta^*, \\ \bar{x} &= \frac{1}{T} X' X \theta^*, \\ \bar{x} &= \frac{1}{T} (\Omega + \bar{x} \bar{x}') \theta^*. \end{aligned}$$

Applying the Sherman–Morrison formula [16] [ch. 2] allows to show that  $\theta^*$  will be proportional to  $\theta$ , and hence after rescaling we have  $\theta^* = \theta$ .

## References

1. Agulló, J.: Exact Algorithms for Computing the Least Median of Squares Estimate in Multiple Linear Regression. In: Dodge, Y. (ed.) *L<sub>1</sub>-Statistical Procedures and Related Topics*. IMS Lecture Notes–Monograph Series, vol. 31, pp. 133–146. IMS (1997)
2. Blume, M.: On the Assessment of Risk. *Journal of Finance* 26(1), 1–10 (1971)
3. Boyd, S., Vandenberghe, L.: *Convex Optimization*. Cambridge University Press, Cambridge (2004)
4. Britten-Jones, M.: The Sampling Error in Estimates of Mean–Variance Efficient Portfolio Weights. *Journal of Finance* 54(2), 655–671 (1999)
5. Chan, L., Lakonishok, J.: Robust Measurement of Beta Risk. *Journal of Financial and Quantitative Analysis* 27(2), 265–282 (1992)
6. Chan, L., Karceski, J., Lakonishok, J.: On Portfolio Optimization: Forecasting Covariances and Choosing the Risk Model. *Review of Financial Studies* 12(5), 937–974 (1999)

7. Cuthbertson, K., Nitzsche, D.: *Quantitative Financial Economics*, 2nd edn. Wiley, Chichester (2005)
8. Dueck, G., Scheuer, T.: Threshold Accepting. A General Purpose Optimization Algorithm Superior to Simulated Annealing. *Journal of Computational Physics* 90(1), 161–175 (1990)
9. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, Nagoya, Japan, pp. 39–43 (1995)
10. Fitzenberger, B., Winker, P.: Improving the computation of censored quantile regressions. *Computational Statistics & Data Analysis* 52(1), 88–108 (2007)
11. Genton, M., Elvezio Ronchetti, E.: Robust Prediction of Beta. In: Kontoghiorghes, E., Rustem, B., Winker, P. (eds.) *Computational Methods in Financial Engineering – Essays in Honour of Manfred Gilli*. Springer, Heidelberg (2008)
12. Gilli, M., Schumann, E.: An Empirical Analysis of Alternative Portfolio Selection Criteria. *Swiss Finance Institute Research Paper No. 09-06* (2009)
13. Gilli, M., Schumann, E.: Distributed Optimisation of a Portfolio's Omega. *Parallel Computing* (forthcoming)
14. Manfred Gilli, M., Winker, P.: Heuristic optimization methods in econometrics. In: Belsley, D., Kontoghiorghes, E. (eds.) *Handbook of Computational Econometrics*. Wiley, Chichester (2009)
15. Gilli, M., Këllezzi, E., Hysi, H.: A data-driven optimization heuristic for downside risk minimization. *Journal of Risk* 8(3), 1–18 (2006)
16. Golub, G., Van Loan, C.: *Matrix Computations*. John Hopkins University Press, Baltimore (1989)
17. Hyndman, R., Fan, Y.: Sample quantiles in statistical packages. *The American Statistician* 50(4), 361–365 (1996)
18. Ince, O., Porter, R.B.: Individual Equity Return Data from Thomson Datastream: Handle with Care! *Journal of Financial Research* 29(4), 463–479 (2006)
19. Kempf, A., Memmel, C.: Estimating the Global Minimum Variance Portfolio. *Schmalenbach Business Review* 58(4), 332–348 (2006)
20. Klemkosky, R., Martin, J.: The Adjustment of Beta Forecasts. *Journal of Finance* 30(4), 1123–1128 (1975)
21. Knez, P., Ready, M.: On the Robustness of Size and Book-to-Market in Cross-Sectional Regressions. *Journal of Finance* 52(4), 1355–1382 (1997)
22. Martin, R.D., Simin, T.: Outlier-Resistant Estimates of Beta. *Financial Analysts Journal* 59(5), 56–69 (2003)
23. Price, K., Storn, R., Lampinen, J.: *Differential Evolution – A practical approach to global optimization*. Springer, Heidelberg (2005)
24. Rousseeuw, P.: Least median of squares regression. *Journal of the American Statistical Association* 79(388), 871–880 (1984)
25. Rousseeuw, P.: Introduction to Positive-Breakdown Methods. In: Maddala, G.S., Rao, C.R. (eds.) *Handbook of Statistics*, vol. 15, ch. 5. Elsevier, Amsterdam (1997)
26. Rudolf, M., Wolter, H., Zimmermann, H.: A linear model for tracking error minimization. *Journal of Banking & Finance* 23(1), 85–103 (1999)
27. Salibian-Barrera, M., Yohai, V.: A Fast Algorithm for S-Regression Estimates. *Journal of Computational and Graphical Statistics* 15(2), 414–427 (2006)
28. Sharpe, W.: Asset Allocation: Management Style and Performance Measurement. *Journal of Portfolio Management* 18(2), 7–19 (1992)
29. Storn, R., Price, K.: Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* 11(4), 341–359 (1997)

30. Stromberg, A.: Computing the Exact Least Median of Squares Estimate and Stability Diagnostics in Multiple Linear Regression. *SIAM Journal on Scientific Computing* 14(6), 1289–1299 (1993)
31. Tufte, E.: *The Visual Display of Quantitative Information*, 2nd edn. Graphics Press (2001)
32. Vasicek, O.: A Note on the Cross-Sectional Information in Bayesian Estimation of Security Betas. *Journal of Finance* 28(5), 1233–1239 (1973)
33. Winker, P.: *Optimization Heuristics in Econometrics: Applications of Threshold Accepting*. Wiley, Chichester (2001)
34. Winker, P., Fang, K.-T.: Application of threshold-accepting to the evaluation of the discrepancy of a set of points. *SIAM Journal on Numerical Analysis* 34(5), 2028–2042 (1997)
35. Winker, P., Lyra, M., Sharpe, C.: Least Median of Squares Estimation by Optimization Heuristics with an Application to the CAPM and a Multi Factor Model. *Journal of Computational Management Science* (2009) (forthcoming)

---

# Evolutionary Estimation of a Coupled Markov Chain Credit Risk Model

Ronald Hochreiter<sup>1</sup> and David Wozabal<sup>2</sup>

<sup>1</sup> Department of Statistics and Mathematics, WU Vienna University of Economics and Business, Augasse 2-6, A-1090 Vienna, Austria

ronald.hochreiter@wu.ac.at

<sup>2</sup> Department of Business Administration, University of Vienna, Brünner Straße 72, A-1210 Vienna, Austria

david.wozabal@univie.ac.at

**Summary.** There exists a range of different models for estimating and simulating credit risk transitions to optimally manage credit risk portfolios and products. In this chapter we present a Coupled Markov Chain approach to model rating transitions and thereby default probabilities of companies. As the likelihood of the model turns out to be a non-convex function of the parameters to be estimated, we apply heuristics to find the ML estimators. To this end, we outline the model and its likelihood function, and present both a Particle Swarm Optimization algorithm, as well as an Evolutionary Optimization algorithm to maximize the likelihood function. Numerical results are shown which suggest a further application of evolutionary optimization techniques for credit risk management.

## 3.1 Introduction

Credit risk is one of the most important risk categories managed by banks. Since the seminal work of [13] a lot of research efforts have been put into the development of both sophisticated and applicable models. Furthermore, de facto standards like CreditMetrics and CreditRisk<sup>+</sup> exist. Numerous textbooks provide an overview of the set of available methods, see e.g. [4], [12], and [14]. Evolutionary techniques have not yet been applied extensively in the area of credit risk management – see e.g. [5] for credit portfolio dependence structure derivations or [15] for optimization of transition probability matrices. In this chapter, we apply the Coupled Markov Chain approach introduced by [9] and provide extensions to the methods presented in [8]. Section 3.2 briefly describes the Coupled Markov Chain model and its properties, and outlines the data we used for subsequent sampling. The likelihood function, which is to be maximized, is discussed in Section 3.3. A non-trivial method to sample from the space of feasible points for the parameters is outlined in Section 3.4. Two different evolutionary approaches to optimize the maximum likelihood function are presented: in Section 3.5 a Particle Swarm Algorithm is shown, and Section 3.6 introduces an Evolutionary Optimization approach. Section 3.7 provides numerical results for both algorithmic approaches, while Section 3.8 concludes the chapter.



## 3.2 Coupled Markov Chain Model

### 3.2.1 Model Description

In the Coupled Markov Chain model proposed in [9] company defaults are modeled directly as Bernoulli events. This is in contrast to standard models used in the literature where indirect reasoning via asset prices is used to model default events of companies. The advantage of the proposed approach is that there are no heavy model assumptions necessary (normality of asset returns, no transaction costs, complete markets, continuous trading ...).

Portfolio effects in structured credit products are captured via correlations in default events. Companies are characterized by their current rating class and a second characteristic which can be freely chosen (industry sector, geographic area, ...). This classification scheme is subsequently used to model joint rating transitions of companies. We keep the basic idea of the standard Gaussian Copula model

$$X = \rho\tau + (1 - \rho)\phi,$$

where  $\tau$  is the idiosyncratic part and  $\phi$  is the systematic part determining the rating transition, while  $0 \leq \rho \leq 1$  is a relative weighting factor. More specifically the Coupled Markov Chain model can be described as follows: A company  $n$  belongs to a sector  $s(n)$  and is assigned to a rating class  $X_n^t$  at time  $t$  with  $X_n^t \in \{0, \dots, M + 1\}$  and  $t: 1 \leq t \leq T$ , with the credit quality decreasing with rating classes, i.e.  $(M + 1)$  being the default class, while 1 is the rating class corresponding to the best credit quality. The ratings of company  $n$  are modeled as Markov Chains  $X_n^t$ . The rating process of company  $n$  is determined by

- an idiosyncratic Markov Chain  $\xi_n^t$ .
- a component  $\eta_n^t$  which links  $n$  to other companies of the same rating class.
- Bernoulli switching variables  $\delta_n^t$  which decide which of the two factors determines the rating, with  $\mathbb{P}(\delta_n^{t+1} = 1) = q_{s(n), X_n^t}$ , i.e. the probability of success depends on sector and rating.

All the  $\xi_n^t$  and  $\delta_n^t$  are independent of everything else, while the  $\eta_n^t$  have a non-trivial joint distribution modeled by common Bernoulli tendency variables  $\chi_i$ ,  $i: 1 \leq i \leq M$ , such that

$$\mathbb{P}(\eta_n^t \leq X_n^t) = \mathbb{P}(\chi_{X_n^{t-1}} = 1) \text{ and } \mathbb{P}(\eta_n^t > X_n^t) = \mathbb{P}(\chi_{X_n^{t-1}} = 0),$$

i.e. the variables  $\chi_i$  are indicators for a (common) non-deteriorating move of all the companies in rating class  $i$ . The rating changes of companies in different rating classes are made dependent by the non-trivial probability mass function  $P_\chi: \{0, 1\}^M \rightarrow \mathbb{R}$  of the vector  $\chi = (\chi_1, \dots, \chi_M)$ .

The Coupled Markov Chain model is of the form:

$$X_n^t = \delta_n^t \xi_n^t + (1 - \delta_n^t) \eta_n^t$$

and exhibits properties, which are interesting for practical application. It takes a transition matrix  $P = (p_{i,j})$  as input which governs the probability of transitions for  $\xi_n^t$  and  $\eta_i^t$ , i.e.

$$\mathbb{P}(\xi_n^t = j) = p_{m(n),j} \text{ and } \mathbb{P}(\eta_i^t = j) = p_{i,j}.$$

The model is capable of capturing different default correlations for different sectors and rating classes, and is able to give a more accurate picture of closeness to default than the standard model by including more than two states. The overall transition probabilities of  $X_n$  again follow  $P$ , i.e.

$$\mathbb{P}(X_n = j) = p_{m(n),j}.$$

### 3.2.2 Data

Rating data from Standard & Poors has been used, whereby 10166 companies from all over the world have been considered. The data consists of yearly rating changes of these companies over a time horizon of 23 years up to the end of 2007. In total a number of 87.296 data points was used. The second characteristic is the SIC industry classification code. Sectoral information has been condensed to six categories: Mining and Construction (1), Manufacturing (2), Transportation, Technology and Utility (3), Trade (4), Finance (5), Services (6). Likewise, rating classes are merged in the following way: AAA, AA  $\rightarrow$  1, A  $\rightarrow$  2, BBB  $\rightarrow$  3, BB, B  $\rightarrow$  4, CCC, CC, C  $\rightarrow$  5, D  $\rightarrow$  6. These clusters allow for a more tractable model by preserving a high degree of detail. The estimated rating transition probabilities from the data are shown in Table [3.1](#).

**Table 3.1.** Estimated rating transition probabilities

$$P = \begin{pmatrix} 0.9191 & 0.0753 & 0.0044 & 0.0009 & 0.0001 & 0.0001 \\ 0.0335 & 0.8958 & 0.0657 & 0.0036 & 0.0006 & 0.0009 \\ 0.0080 & 0.0674 & 0.8554 & 0.0665 & 0.0011 & 0.0016 \\ 0.0039 & 0.0092 & 0.0794 & 0.8678 & 0.0244 & 0.0153 \\ 0.0023 & 0.0034 & 0.0045 & 0.1759 & 0.6009 & 0.2131 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

## 3.3 Maximum Likelihood Function

The approach proposed by [\[9\]](#) takes a Markov transition matrix  $P = (p_{m_1, m_2})_{1 \leq m_1, m_2 \leq (M+1)}$  as an input, i.e.

$$\sum_{i=1}^{M+1} p_{i,m} = \sum_{i=1}^{M+1} p_{m,i} = 1, \quad \forall m : 1 \leq m \leq (M+1).$$

For  $(M+1)$  rating classes,  $N$  companies and  $S$  industry sectors the parameters of the model are a matrix  $Q = (q_{m,s})_{1 \leq s \leq S, 1 \leq m \leq M}$  and a probability measure  $P_\chi$  on  $\{0, 1\}^M$  satisfying some constraints dependent on  $P$  (see problem [\(3.1\)](#)). Given rating transition data  $X$  ranging over  $T$  time periods we maximize the following monotone transformation of the likelihood function of the model

$$L(X; Q, P_\chi) = \sum_{t=2}^T \log \left( \sum_{\bar{\chi} \in \{0,1\}^M} P_\chi(\chi^t = \bar{\chi}) \prod_{s, m_1, m_2} f(x^{t-1}, s, m_1, m_2, ; Q, P_\chi) \right)$$

with

$$f(x^{t-1}, s, m_1, m_2, ; Q, P_\chi) = \begin{cases} \left( \frac{q_{m_1, s} (p_{m_1}^+ - 1) + 1}{p_{m_1}^+} \right)^{I^t}, & m_1 \geq m_2, \bar{\chi}_{m_1} = 1 \\ \left( \frac{q_{m_1, s} (p_{m_1}^- - 1) + 1}{p_{m_1}^-} \right)^{I^t}, & m_1 < m_2, \bar{\chi}_{m_1} = 0 \\ q_{m_1, s}^{I^t}, & \text{otherwise.} \end{cases}$$

where  $I^t \equiv I^t(m_1, m_2, s)$  is a function dependent on the data  $X$  which takes values in  $\mathbb{N}$ ,  $p_m^+ = \sum_{i=1}^m p_{m,i}$  and  $p_m^- = 1 - p_m^+$ .

The above function is clearly non-convex and since it consists of a mix of sums and products this problem can also not be overcome by a logarithmic transform. Maximizing the above likelihood for given data  $X$  in the parameters  $P_\chi$  and  $Q$  amounts to solving the following constrained optimization problem

$$\begin{aligned} \max_{Q, P_\chi} \quad & L(X; Q, P_\chi) \\ \text{s.t.} \quad & q_{m,s} \in [0, 1] \\ & \sum_{\bar{\chi}: \bar{\chi}_i=1} P_\chi(\bar{\chi}) = p_{m_i}^+, \quad \forall i: 1 \leq i \leq M \\ & \sum_{\bar{\chi}: \bar{\chi}_i=0} P_\chi(\bar{\chi}) = 1 - p_{m_i}^+. \end{aligned} \tag{3.1}$$

### 3.4 Sampling Feasible Points

To sample from the space of feasible joint distributions for  $\chi$  (i.e. the distributions whose marginals meet the requirements in (3.1)), first note that the distributions  $P_\chi$  of the random variable  $\chi$  are distributions on the space  $\{0, 1\}^M$  and therefore can be modeled as vectors in  $\mathbb{R}^{2^M}$ . To obtain samples we proceed as follows.

1. To get a central point in the feasible region, we solve the following problem in dependence of a linear functional  $\Psi : \mathbb{R}^{2^M} \rightarrow \mathbb{R}$ .

$$\begin{aligned} \max_{P_\chi} \quad & \Psi(P_\chi) \\ \text{s.t.} \quad & q_{m,s} \in [0, 1] \\ & \sum_{\bar{\chi}: \bar{\chi}_i=1} P_\chi(\bar{\chi}) = p_{m_i}^+, \quad \forall i: 1 \leq i \leq M \\ & \sum_{\bar{\chi}: \bar{\chi}_i=0} P_\chi(\bar{\chi}) = 1 - p_{m_i}^+. \end{aligned} \tag{3.2}$$

and call the solution set  $\mathcal{S}(\Psi)$ . By generating linear  $\Psi$  functionals with random coefficients and picking  $x^+ \in \mathcal{S}(\Psi)$  and  $x^- \in \mathcal{S}(-\Psi)$  we get vertices of the feasible set of distributions for  $\chi$  modeled as a polyhedron in  $\mathbb{R}^{2^M}$ . In this way we generate a set of vertices  $V$  for the feasible region  $\mathcal{Q}$  of the above problem. Note that to enforce all the constraints in (3.2) we need  $M + 1$  linear equality constraints which describe a  $2^M - (M + 1)$  dimensional affine subspace in  $\mathbb{R}^{2^M}$ .

2. Get a central point in  $c \in \mathcal{Q}$  by defining

$$c = \frac{1}{|V|} \sum_{v \in V} v.$$

3. Sample  $K \in \mathbb{N}$  directions of unit length from a spherical distribution (like the multivariate standard normal with independent components) in  $\mathbb{R}^{2^M - M - 1}$  to get uniformly distributed directions. Map these directions to  $\mathbb{R}^{2^M}$  using an orthogonal basis of the affine subspace  $A$  of  $\mathbb{R}^{2^M}$  described by the last set of constraints in (3.2) to obtain a set of directions  $\mathcal{D}$  in  $A$ .
4. For every  $d \in \mathcal{D}$  determine where the line  $c + \lambda d$  meets the boundary of the feasible set of (3.2). Call the length of the found line segment  $l_d$ .
5. Fix a number  $L \in \mathbb{N}$  and sample

$$\left[ \frac{l_d}{\sum_{d \in \mathcal{D}} \bar{l}_d} L \right]$$

points on the line  $l_d$ . In this way we get approximately  $KL$  samples for  $P_\chi$ .

Contrary to obtaining samples from  $P_\chi$ , getting suitable samples for  $Q$  is fairly easy, since all the components are in  $[0, 1]$  and independent of each other. Note that both the set of feasible matrices  $Q$  as well as the set of feasible measures  $P_\chi$  are convex sets in the respective spaces.

### 3.5 Particle Swarm Algorithm

In the following we give a brief description of the Particle Swarm Algorithm (PSA), which follows the ideas in [10].

1. Choose  $\delta > 0$  and  $S \in \mathbb{N}$ .
2. Generate  $S$  permissible random samples  $x_k = (Q^k, P_\chi^k)$  for  $k = 1, \dots, S$  as described above, i.e.  $q_{s,m}^k \in [0, 1]$  and  $P_\chi$  is consistent with the constraints in (3.2). Each sample is a particle in the algorithm. Set  $\hat{x}_k = x_k$  and  $v_k = 0$  for all  $k = 1, \dots, S$ .
3. Set  $\hat{g} \leftarrow \arg \min_k L(x_k)$ .
4. For all particles  $x_k$ 
  - a) Let the particles fly by first computing a velocity for the  $k$ -th particle

$$v_k \leftarrow c_0 v_k + c_1 r_1 \circ (\hat{x}_k - x_k) + c_2 r_2 \circ (\hat{g} - x_k) \quad (3.3)$$

where  $c_0, c_1, c_2$  are fixed constants,  $r_1$  and  $r_2$  are random matrices (component-wise uniform) of the appropriate dimension and  $\circ$  is the Hadamard matrix multiplication. Then a new position for the particle is found by the following assignment

$$x_k \leftarrow x_k + v_k.$$

- b) If  $L(x_k) > L(\hat{x}_k)$  then  $\hat{x}_k \leftarrow x_k$ .
5.  $L(x_k) > L(\hat{g})$  for some  $x_k$ , then  $\hat{g} \leftarrow x_k$ .
6. If  $\text{var}(L(x_k)) < \delta$  terminate the algorithm, otherwise go to step 3.

The main idea is that each particle  $k$  knows its best position  $\hat{x}_k$  as of yet (in terms of the likelihood function) and every particle knows the best position ever seen by any particle  $\hat{g}$ . The velocity of the particle changes in such a way that it is drawn to these positions

(to a random degree). Eventually all the particles will end up close to one another and near to a (local) optimum.

Note that in step 4(a) of the above algorithm, a particle may leave the feasible region either by violating the constraints on  $P_\chi$  or  $Q$ . In this case the particle *bounces off the border* and completes its move in the modified direction. To be more precise: the particles can only leave the feasible region by violating the constraints that either elements of  $Q$  or probabilities assigned by  $P_\chi$  are no longer in  $[0, 1]$  (the last two constraints in (3.2) can not be violated since the particles only move in the affine subspace of  $\mathbb{R}^{2^M}$  where these constraints are fulfilled).

If  $x_k^i + v_k^i > 1$  for some  $1 \leq i \leq 2^{2^M} + MS$  (the case where  $x_k^i + v_k^i < 0$  works analogously), determine the maximum distance  $\lambda$  that the particle can fly without constraint violation, i.e. set

$$\lambda = \frac{(1 - x_k^i)}{v_k^i}$$

and set  $x^k \leftarrow x^k + \lambda v^k$ . Now set  $\bar{v}_k$  such that the new velocity makes the particle bounce off the constraint *as would be expected* and make the rest of the move, i.e. set

$$x^k \leftarrow x^k + (1 - \lambda)\bar{v}^k.$$

In the case that the violation concerns an element of  $Q$  the modification only concerns a change of sign, i.e.  $\bar{v}_k^i \leftarrow -v_k^i$  and  $\bar{v}_k^j \leftarrow v_k^j$  for all  $j \neq i$ .

In the following we describe how to find a *bounce off* direction, if a constraint on an element of  $P_\chi$  is violated: first determine the hyperplane  $H$  in  $\mathbb{R}^{2^M}$  that represents the constraint. Notice that  $H = \{x \in \mathbb{R}^{2^M} : x_i = 1\}$  for some  $i$ . We use the following Lemma to get a representation of the hyperplane  $H$  in the affine subspace  $A$ .

**Lemma 3.1.** *Let  $A$  be an affine subspace of  $\mathbb{R}^D$  with orthonormal basis  $e_1, \dots, e_d$  with  $D < d$  and  $H$  a hyperplane with normal vector  $n$ . The normal vector of the hyperplane  $A \cap H$  in  $A$  is*

$$\bar{n} = \sum_{i=1}^d \langle e_i, n \rangle e_i.$$

*Proof.* Let  $H = \{x \in \mathbb{R}^D : \langle x, n \rangle = c\}$  for some  $c \in \mathbb{R}$ , then

$$\begin{aligned} c = \langle y, n \rangle &= \left\langle \sum_{i=1}^d \langle y, e_i \rangle e_i, n \right\rangle = \sum_{i=1}^d \langle y, e_i \rangle \langle e_i, n \rangle \\ &= \left\langle y, \sum_{i=1}^d \langle e_i, n \rangle e_i \right\rangle = \langle y, \bar{n} \rangle. \end{aligned}$$

This implies that the point  $y \in A \cap H$ , iff  $\langle y, \bar{n} \rangle = c$ .

Using the above Lemma we identify the normal vector  $\bar{n} \in A$  of the hyperplane  $\bar{H} = H \cap A$  in  $A$ . Without loss of generality we assume that  $\|\bar{n}\| = 1$ . Now use Gram-Schmidt to identify a orthonormal system  $\bar{n}, y_2, \dots, y_{2^{2^M} - (M+1)}$  in  $A$  and represent  $v_k$  as

$$v_k = \langle \bar{n}, v_k \rangle \bar{n} + \sum_{i \geq 2} \langle y_i, v_k \rangle y_i.$$

The transformed velocity can now be found as

$$\bar{v}_k = -\langle \bar{n}, v_k \rangle \bar{n} + \sum_{i \geq 2} \langle y_i, v_k \rangle y_i.$$

Obviously an implementation of the algorithm has to be able to handle multiple such bounces in one move (i.e. situation where the new direction  $\bar{v}_k$  again leads to a constraint violation). Since the details are straightforward and to avoid too complicated notation, we omit them here for the sake of brevity.

### 3.6 Evolutionary Algorithm

Evolutionary algorithms are well suited to handle many financial and econometric applications, see especially [2], [3], and [1] for a plethora of examples.

Each chromosome consists of a matrix  $Q$  and a vector  $P_\chi$ . While the parameters in  $Q$  can be varied freely between 0 and 1, and the parameters  $P_\chi$  do need to fulfill constraints (see above), the genetic operators involving randomness are mainly focused around the matrix  $Q$ . Therefore, four different genetic operators are used:

- **Elitist selection.** A number  $e$  of the best chromosomes are added to the new population.
- **Intermediate crossover.**  $c$  intermediate crossovers (linear interpolation) between the matrices  $Q_1$  and  $Q_2$  of two randomly selected parents are created using a random parameter  $\lambda$  between 0 and 1, i.e. two children  $Q_3, P_{\chi,3}$  and  $Q_4, P_{\chi,4}$  are calculated as follows:

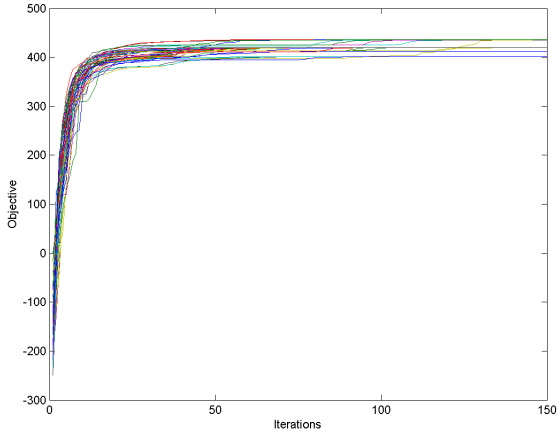
$$Q_3 = \lambda Q_1 + (1 - \lambda) Q_2, P_{\chi,3} = P_{\chi,1},$$

$$Q_4 = (1 - \lambda) Q_1 + \lambda Q_2, P_{\chi,4} = P_{\chi,2}.$$

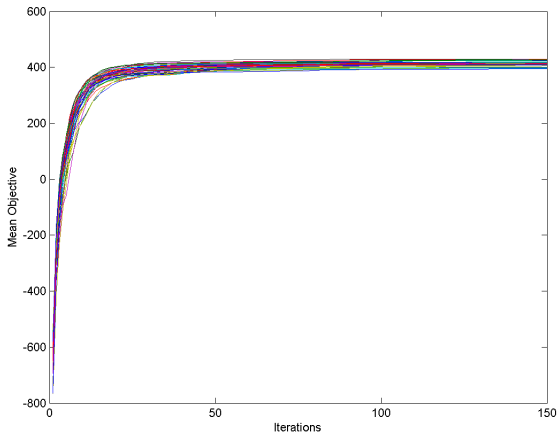
- **Mutation.**  $m$  new chromosomes are added by mutating a randomly selected chromosome from the parent population, and adding a factor  $\phi$  in the range  $[-0.5, 0.5]$  to the matrix  $Q$ . The values are truncated to values between 0 and 1 after the mutation.
- **Random additions.**  $r$  random chromosomes are added with a random matrix  $Q$  and a randomly selected vector  $P_\chi$  from the parent population.

### 3.7 Numerical Results

Both algorithms were developed in MatLab R2007a, while the linear problems (3.2) were solved using MOSEK 5. A stability test has been conducted to validate the results of both optimization algorithms: the maximum (pointwise) differences of parameter estimates  $P_\chi$  and  $Q$  between the different optimization runs is used to verify that these important parameters, which are e.g. used for a credit portfolio optimization procedure, do not differ significantly.



**Fig. 3.1.** Objective function of the PSA algorithm: maximum per iteration.

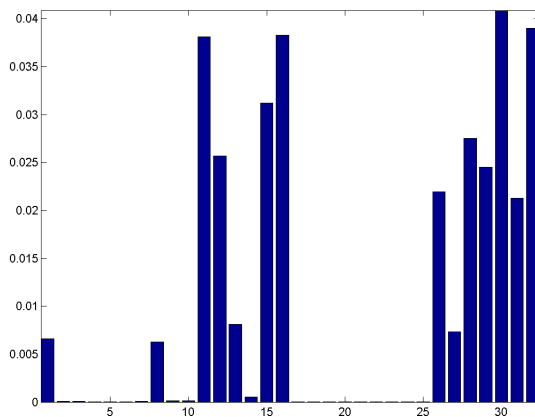


**Fig. 3.2.** Objective function of the PSA algorithm: population mean.

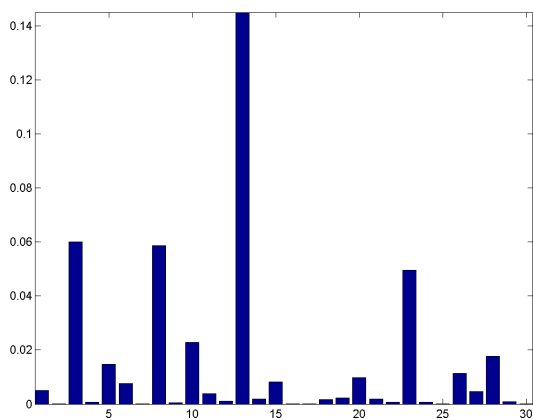
### 3.7.1 Particle Swarm Algorithm

The parameters in (3.3) were set to  $c_0 = 0.5$ ,  $c_1 = 1.5$  and  $c_2 = 1.5$ . The algorithm was made to complete 150 iterations with around 200 initial samples (where the  $\chi$  are simulated on 40 lines with approximately 5 samples each). To test the stability of the algorithm, 50 runs of the algorithm were performed. As can be clearly observed in Fig. 3.1 and 3.2, the likelihood of the best particle as well as the mean likelihood of the swarm converges nicely and stabilizes after around 25 iterations. Each run took around 1 hour to complete 150 iterations. Stability results are shown in Fig. 3.3 and 3.4.

The variances of the populations in every iteration are plotted in Fig. 3.5. Since the variances sharply decrease from very high initial values and in most cases drop



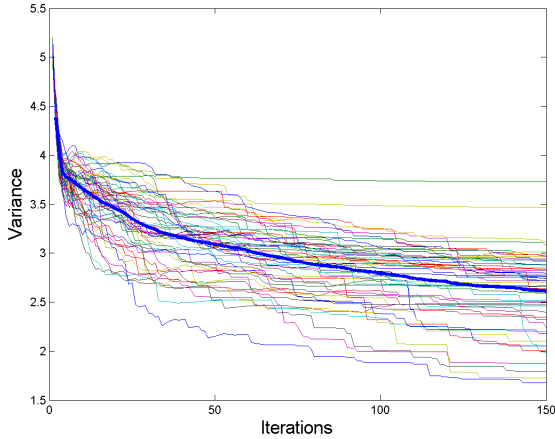
**Fig. 3.3.** Maximum (pointwise) differences of parameter estimates  $P_\chi$  for different runs for the PSA.



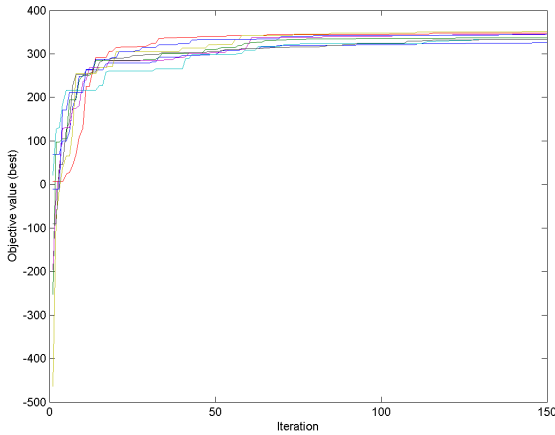
**Fig. 3.4.** Maximum (pointwise) differences of parameter estimates  $Q$  for different runs for the PSA.

to rather low values quickly, the plot depicts the variance after applying a logarithmic transformation (base 10) as well as the mean variance over all the runs. While in most runs the variances decreases from values of the magnitude  $10^5$  to the range of  $10^3$ , some of the runs end up with significantly lower and higher variances. The latter being a sign that the PSA sometimes fails to converge, i.e. the particles do not concentrate at one point after the performed number of iterations. However, this is not problematic since we are only interested in the likelihood of the best particle which seems to be pretty stable at around 400. The results depicted in Fig. 3.3 and 3.4 confirm, that despite the high variance in some of the runs the likelihood of the best particle remains stable for all the runs.





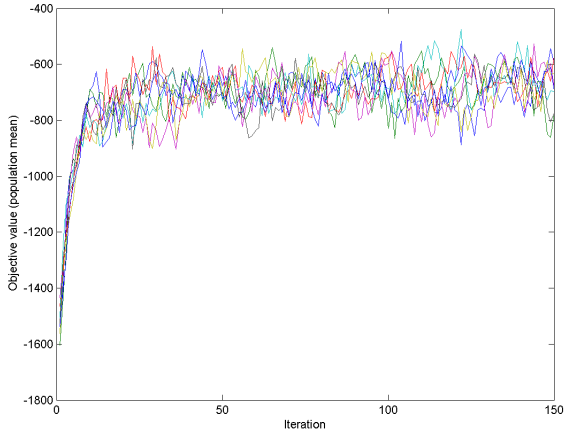
**Fig. 3.5.** Variances of the objective values of the swarm (variances are transformed with  $x \mapsto \log_{10}(x)$  for better interpretability of the results. The mean (logarithmic) variance is depicted by the bold line.



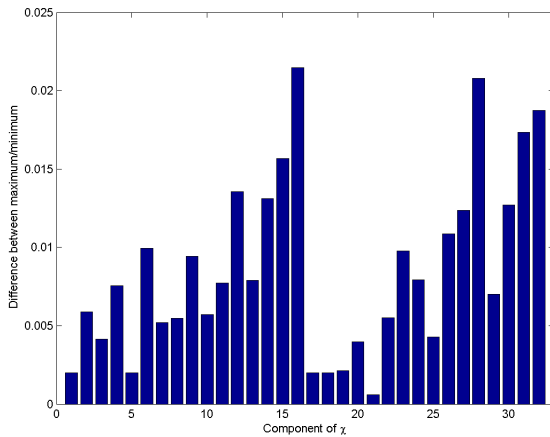
**Fig. 3.6.** Objective function of the EA: maximum per iteration.

### 3.7.2 Evolutionary Algorithm

The following parameters have been used to calculate results: The number of maximum iterations has been set to 150. Each new population consists of  $e = 30$  elitist chromosomes,  $c = 50$  intermediate crossovers,  $m = 100$  mutations, and  $r = 50$  random additions. 50 runs have been calculated, and 10 tries out of these are shown in Fig. 3.6 and 3.7 – both the maximum objective value per iteration (3.6) as well as the population mean (3.7). Due to the high number of random additions, mutations and crossovers, the mean is relatively low and does not significantly change over the iterations, which does not influence the results. The initial population were 750 randomly sampled chromosomes,



**Fig. 3.7.** Objective function of the EA: population mean.



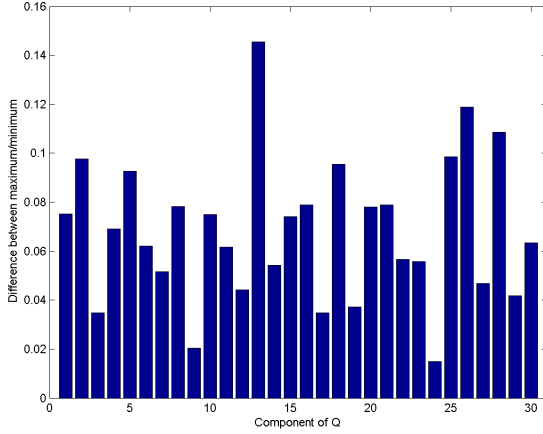
**Fig. 3.8.** Maximum (pointwise) differences of parameter estimates  $P_\chi$  for different runs for the EA.

independently sampled for each try. It can be clearly seen, that due to the different algorithmic approach, the convergence is different from the PSA.

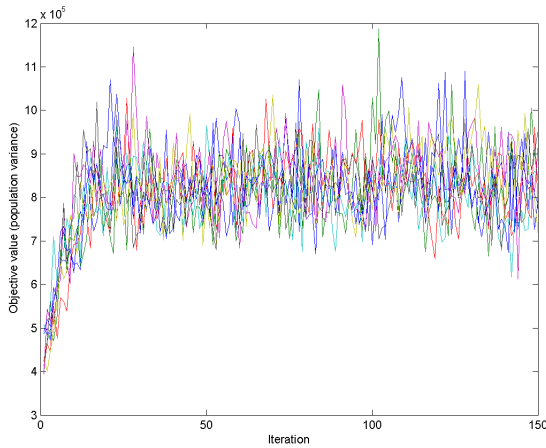
Each run took approximately 70 minutes to complete the 150 iterations. Stability results are shown in Fig. 3.8. The population variance is shown in Fig. 3.10, and clearly exhibits a different behavior than the PSA algorithm as expected.

### 3.7.3 Comparison of Methods

Comparing the results of the current implementations of the two optimization heuristics, the results found by the PSA consistently yield a higher objective value than the



**Fig. 3.9.** Maximum (pointwise) differences of parameter estimates  $Q$  for different runs for the EA.



**Fig. 3.10.** Population variance of the EA.

solutions obtained with the EA (for the best particle/chromosome as well as for the mean). The computing time for the two methods is similar and is mainly used for the expensive objective function evaluations. Furthermore the presented computational evidence shows the typical behavior of the variance given the two heuristic optimization techniques. While the PSA generally performs slightly better than the EA, it might well be that it gets stuck in a local optimum, which might be avoided using the EA. One can see from the figures that the maximum difference between the estimated parameters for different runs are smaller on average for the PSA. However, the analysis of the distribution of these differences reveals the interesting fact, that while for the EA the differences are more uniform in magnitude and the highest as well as the lowest deviations can be

observed for the PSA. With the realistically sized data set both methodologies are well suited and the final choice is up to the bank or company which implements and extends the presented method, i.e. has to be based on the expertise available.

### 3.7.4 Application of the Model

Once the Coupled Markov Chain model has been estimated using evolutionary techniques shown above, it can be used to simulate rating transition scenarios for different sets of companies, which allows for pricing and optimization of various structured credit contracts like specific CDX tranches, e.g. a Mean-Risk optimization approach in the sense of [11] can be conducted for which evolutionary techniques can be used again as shown by e.g. [6] and [7], such that a whole credit risk management framework based on evolutionary techniques can be successfully implemented.

## 3.8 Conclusion

In this chapter, we presented the likelihood function for a Coupled Markov Chain model for contemporary credit portfolio risk management. We presented two different heuristic approaches for estimating the parameter of the likelihood function. Both are structurally different, i.e. the population mean of each method differs significantly. However, both are valid approaches to estimate parameters. Once the parameters are estimated, many applications are possible. One prominent example is to generate scenarios for future payment streams implied by an existing portfolio of Credit Default Swap Indices (CDX) by Monte Carlo simulation. This allows for assessing the risk of the current position and price products which might be added to the portfolio in the future and thereby determine their impact on the overall exposure.

## Acknowledgement

This research was partly supported by the Austrian National Bank Jubiläumsfond Project 12306.

## References

1. Brabazon, A., O'Neill, M.: Biologically inspired algorithms for financial modelling. Natural Computing Series. Springer, Berlin (2006)
2. Brabazon, A., O'Neill, M. (eds.): Natural Computing in Computational Finance, Studies in Computational Intelligence, vol. 100. Springer, Heidelberg (2008)
3. Brabazon, A., O'Neill, M. (eds.): Natural Computing in Computational Finance, vol. 2. Studies in Computational Intelligence, vol. 185. Springer, Heidelberg (2009)
4. Duffie, D., Singleton, K.J.: Credit Risk: Pricing, Measurement, and Management. Princeton University Press, Princeton (2003)
5. Hager, S., Schöbel, R.: Deriving the dependence structure of portfolio credit derivatives using evolutionary algorithms. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006, Part IV. LNCS, vol. 3994, pp. 340–347. Springer, Heidelberg (2006)

6. Hochreiter, R.: An evolutionary computation approach to scenario-based risk-return portfolio optimization for general risk measures. In: Giacobini, M. (ed.) *EvoWorkshops 2007*. LNCS, vol. 4448, pp. 199–207. Springer, Heidelberg (2007)
7. Hochreiter, R.: Evolutionary stochastic portfolio optimization. In: Brabazon, A., O’Neill, M. (eds.) *Natural Computing in Computational Finance*. Studies in Computational Intelligence, vol. 100, pp. 67–87. Springer, Heidelberg (2008)
8. Hochreiter, R., Wozabal, D.: Evolutionary approaches for estimating a coupled markov chain model for credit portfolio risk management. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (eds.) *EvoCOMNET*. LNCS, vol. 5484, pp. 193–202. Springer, Heidelberg (2009)
9. Kaniovski, Y.M., Pflug, G.C.: Risk assessment for credit portfolios: A coupled markov chain model. *Journal of Banking and Finance* 31(8), 2303–2323 (2007)
10. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948. IEEE Computer Society, Los Alamitos (1995), doi:10.1109/ICNN.1995.488968
11. Markowitz, H.M.: *Mean-variance analysis in portfolio choice and capital markets*. Basil Blackwell, Oxford (1987)
12. McNeil, A.J., Frey, R., Embrechts, P.: *Quantitative risk management*. Princeton University Press, Princeton (2005)
13. Merton, R.: On the pricing of corporate debt: the risk structure of interest rates. *Journal of Finance* 29, 449–470 (1974)
14. Schönbucher, P.J.: *Credit Derivatives Pricing Models: Models, Pricing, Implementation*. Wiley Finance, Chichester (2003)
15. Zhang, J., Avasarala, V., Subbu, R.: Evolutionary optimization of transition probability matrices for credit decision-making. *European Journal of Operational Research* 200(2), 557–567 (2010)

# Evolutionary Computation and Trade Execution

Wei Cui<sup>1,2</sup>, Anthony Brabazon<sup>1,2</sup>, and Michael O'Neill<sup>1,3</sup>

<sup>1</sup> Natural Computing Research and Applications Group,  
University College Dublin, Ireland

will.weicui@gmail.com, anthony.brabazon@ucd.ie, m.oneill@ucd.ie

<sup>2</sup> School of Business, University College Dublin, Ireland,

<sup>3</sup> School of Computer Science and Informatics, University College Dublin, Ireland

**Summary.** Although there is a plentiful literature on the use of evolutionary methodologies for the trading of financial assets, little attention has been paid to the issue of efficient trade execution. Trade execution is concerned with the actual mechanics of buying or selling the desired amount of a financial instrument of interest. This chapter introduces the concept of trade execution and outlines the limited prior work applying evolutionary computing methods for this task. Furthermore, we build an Agent-based Artificial Stock Market and apply a Genetic Algorithm to evolve an efficient trade execution strategy. Finally, we suggest a number of opportunities for future research.

## 4.1 Introduction

Algorithmic trading (AT) can be broadly defined as the use of computers to automate aspects of the investment process. Hence, AT can encompass the automation of decisions ranging from stock selection for investment, to the management of the actual purchase or sale of that stock. A significant proportion of all financial asset trading is now undertaken by AT systems with this form of trading accounting for approximately 20-25% of total US market volume in 2005. Boston-based research firm Aite Group predicts that AT will account for more than half of all shares traded in the U.S. by the end of 2010 [21]. AT is also common in European financial markets with approximately 50% of trading volumes being accounted for by algorithmic trading programs [15]. Significant volumes in Asian markets are similarly traded [14]. Algorithmic trading is seen in multiple financial markets ranging from equities to FX (foreign exchange), to derivative (futures, options etc.) markets.

In this chapter we restrict attention to one aspect of financial trading to which AT can be applied, namely efficient trade execution. A practical issue that arises for investors is how they can buy or sell large quantities of a share (or some other financial asset) as efficiently as possible in order to minimize market impact. Typically, orders to buy or sell a share can be either *market orders* (the transaction is undertaken immediately in the market at current prices) or *limit orders* (the purchase (sale) must occur at a price which is no greater than (or less than) a pre-specified price). So for example, if a customer places a limit order to buy a stock at \$125 per share the transaction will only take place if the market price falls to \$125 or less. Hence, when a market order is placed, the customer does not have control over the final price(s) at which the order

**Table 4.1.** Sample order book for a share with volume and price information for bid and ask

<b>Bid</b>		<b>Ask</b>	
Vol	Price	Price	Vol
300	132.9	133.2	200
200	132.8	133.3	300
400	132.7	133.4	100
500	132.6	133.5	300
300	132.5	133.6	200
100	132.4	133.7	400

will be filled, and in a limit order, while the customer has some price control, there is no guarantee that the order will actually be executed.

Most major financial markets now are limit order markets which operate based on an electronic order book, where participants can see the current unfilled buy and sell orders. Table 4.1 illustrates a sample order book, showing the quantities that investors are willing to buy (bid side) and sell (ask side) at each price. We can see that 200 shares are currently available for sale at a price of 133.2 (or higher), and buyers are seeking 300 shares at a price of 132.9 (or lower). The order book also illustrates that there are limits to the quantity of shares available for purchase / sale at each price point. Of course, the order book is highly dynamic, with the quantities of shares offered at each price changing constantly as trades are executed, as investors add new limit orders on the bid and ask sides, or as investors cancel limit orders they have previously placed.

When trading financial assets, particularly when an investor is looking to buy or sell a large quantity of the asset, the problem of *market impact* arises. Market impact arises when the actions of an investor start to move the price adversely against themselves. Hence, market impact is the difference between a transaction price and what the market price would have been in the absence of the transaction. For example, if an investor wished to buy 400 shares given the above order book, he would end up driving up the price paid for some shares to 133.3. The obvious strategy to minimize market impact is to break the order up into smaller lots and spread them out over time. While this may reduce the market impact, it incurs the risk of suffering *opportunity cost*, that market prices may start moving against you during the multiple purchases. Hence, the design of trade execution strategies when trading large blocks of financial assets is intended to balance out these factors.

The task in devising an efficient execution strategy is complex as it entails multiple sub-decisions including how best to split up the large order, what execution *style* to adopt in executing each element of the order (aggressive or passive), what type of order to use, when to submit the order, and how execution performance is to be measured. In addition, the electronic order book(s) faced by the investor are constantly changing.

In the past the task of designing an execution strategy was undertaken by human experts but it is amenable to automation. In this chapter we apply a Genetic Algorithm (GA) to evolve an efficient trade execution strategy and highlight other possible Evolutionary Computation (EC) applications for this issue.

To test the performance of any trade execution strategy, we need highly detailed transaction level data. An ordinary way is to obtain the data from the exchange. However, this only provides us with a single sample path of order book data over time. Another approach is to consider the output data from an *Artificial Stock Market* (ASM), a simulation of the real stock market. An advantage of a simulation-based approach is that many sample paths can be generated and utility of a trade execution strategy can be tested over all of these paths. Most ASM models are built by a computer technique called *Agent-based Modeling* (ABM). Novelty, this chapter evaluates the strategy employing the data from an ASM.

This chapter is organized as follows: Section 4.2 gives the necessary microstructure background relevant to the trade execution from two aspects: trading cost and price formation. Section 4.3 discusses trade execution strategy and corresponding performance evaluation. Section 4.4 provides concise introduction to the EC methodologies, and related work with application in trade execution. Section 4.5 explains agent-based modeling and simulates an artificial stock market. Section 4.6 demonstrates the use of GA to evolve a quality execution strategy. Section 4.7 conclude this chapter by giving a number of avenues for future work.

## 4.2 Background

The finance literature on market microstructure is vast and consequently, we only discuss a limited subset of concepts from it, trading cost and price formation, which are most relevant to this chapter.

### 4.2.1 Trading Cost

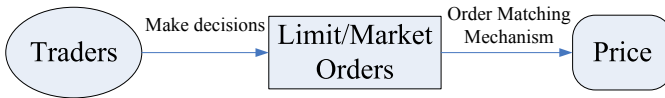
Trading cost can be decomposed into direct cost and indirect cost. Direct cost are observable directly, such as commissions, fees and taxes. Indirect costs are more difficult to observe. Indirect costs can be divided into three main components: market impact cost, opportunity cost and bid-ask spread cost. Early studies of indirect costs focused on the bid-ask spread [20]. Lately, market impact cost and opportunity cost have received more attention.

Execution needs to balance all of these factors [1]. If trading costs are not properly managed throughout trading it could cause a superior opportunity to become only marginally profitable and a normally profitable opportunity to turn into a loss [22]. Factors which influence transaction cost are trade size, market capacitation, time windows, trading time, order imbalance, volume of time horizon etc.

### Market Impact

As investors transact shares in the market they cause market impact (price impact) in the stock. Buy orders are associated with increasing prices and sell orders are associated with decreasing prices. The market impact is typically decomposed into permanent and transitory components which provide estimates of the information and liquidity costs of the trade [20].





**Fig. 4.1.** Price Formation

Due to its importance, there is much research on the causes of market impact [22]. Empirical evidence showed that block price impacts are a concave function of order size and a decreasing function of market capitalization (liquidity). Bikker [4] found that average market impact costs equal 20 basis points for buys and 30 basis points for sells, and market impact costs are influenced by timing of the trades, such as the day of the week, the period of the month and the month of the year at which the stock is traded. Stocks with high capitalization yield lower price impact cost than stocks with low capitalization [28]. Price impact costs increase as order imbalance increases [27].

## Opportunity Cost

There are two reasons why opportunity cost can arise [20]. One reason is that an order is only partially filled or is not executed at all. This often happens using passive trading strategies, such as a limit order strategy which trades only limit order in the market. For example, a trader who anticipates that the market price will move down, sets the limit price below the best available bid price. If the market price actually moves up during that day, he will suffer a high cost due to unexecuted order. The other reason is that some orders traded in the market are executed with a delay, in times of adverse price movement.

### 4.2.2 Price Formation

Many modern markets operate using an electronic limit order book as described above. In a limit order market, orders arrive randomly in time. The price limit of a newly arrived order is compared to those of orders already held in the system to ascertain if there is a match. If so, the trade occurs at the price set by the first order. The set of unexecuted limit orders held by the system constitutes the dynamic order book, where limit orders can be cancelled or modified at any time or executed in price priority and time priority sequence. According to the first rule, the buy (or sell) limit order with higher (or lower) price get executed prior to others with lower (or higher) price. The second rule means that where two or more limit buy (or sell) orders have the same limit price, the buy (or sell) limit order which arrives at the market earlier get executed prior to the others. A simple price formation process is shown in Figure 4.1.

## 4.3 Trade Execution Strategy

A dilemma facing traders is that trading too quickly reduces the risk of suffering high opportunity costs but is associated with high market impact cost. Trading too slowly minimizes market impact cost but can give rise to opportunity cost. These costs are

balanced out in a trade execution strategy, by splitting a large trade into lots of small pieces and spreading them over several hours using market or limit orders. For example, an algorithmic trading strategy may equally divide a 100,000-share order into 100 small orders, where each small order has 1,000 shares. These orders then may be sent to the market over the course of the day, in order to minimize market impact. Another advantage of doing this is that these orders can be disguised in the market and prevented from being detected by other participants in the market. This section presents important factors in a trade execution strategy and discusses how to evaluate performance of trading strategies.

### 4.3.1 Factors

Factors of a trading strategy include the number of orders, type of each order, size per order and submission time per order. Moreover, if a submitted order is a limit order, the strategy has to specify a limit price and a duration time.

If immediacy is desired, the market order is the appropriate instrument to use. However, market orders pay an implicit price for immediacy. Limit orders avoid impact cost but bear the risks of non-execution. In practice, traders submit both types of orders, in order to balance the opportunity cost of delaying execution against the price impact associated with immediate execution.

Duration time or lifetime is another important factor. The lifetime can range from zero to entire trading time of the day. However, longer lifetime is not always the better choice, since transacting on longer period also faces more risk.

The limit price of a limit order plays a significant role in order execution, which always floats around the best bid/ask price. When placing a limit order, it is simpler to just consider the relative limit price, which is the difference between the limit price of buy/sell order and the current best bid/ask price. Choosing a relative limit price is a strategic decision that involves a trade-off between patience and cost. For example, if a trader wants to submit a limit buy order when the current best ask price is  $a$ , an impatient buyer will submit a limit order with a limit price  $p$  well above  $a$ , which will immediately result in a transaction. A buyer of intermediate patience will submit an order with price  $p$  a little smaller than  $a$ ; this will not result in an immediate transaction, but will have high priority as new sell orders arrive. A very patient buyer will submit an order with  $p$  much smaller than  $a$ ; this order is unlikely to be executed soon, but it will trade a good price if it does. A lower price is clearly desirable to the buyer, but it comes at the cost of lowering the probability of trading. Usually, the lower the price to buy or the higher the price to sell, the lower the probability there will be a trade. However, the choice of limit price is a complex decision that depends on market environment.

### 4.3.2 Types of Execution Systems

Algorithmic Trading systems typically aim at achieving or beating a specified benchmark with their executions and may be distinguished by their underlying benchmark, their aggressiveness or trading style as well as their adaptation behavior [23].

In practice the most commonly used algorithms in the market place according to their benchmarks are: arrival price, time weighted average price (TWAP), volume weighted

average price (VWAP), market-on-close (MOC), and implementation shortfall (the difference between the share-weighted average execution price and decision price for a trade). Arrival price is the midpoint of the bid-ask spread at order-receipt time. VWAP is calculated as the ratio of the value traded and the volume traded within a specified time horizon. MOC measures the last price obtained by a trader at the end of the day against the last price reported by the exchange. Implementation shortfall is a model that weighs the urgency of executing a trade against the risk of moving the stock.

In terms of adaptation behavior, any algorithmic trading strategy can also be categorized into one of the two categories: static strategy and adaptive strategy. Static strategy pre-determines order trading schedule and will not change during the process of trading. This strategy can not adapt to changing environment. For example, an aggressive strategy that places only market orders to buy can not change its aggressiveness if the market price keeps moving up. On the other hand, adaptive strategy adapts to changing market conditions such as market price movements, volume profiles due to special events such as new announcements or fed indicators, as well as changes in volatility, by altering their aggressiveness of trading adequately. Intuitively, a more aggressive action can be represented either as raising (or lowering) the buy (or sell) limit order price or as increasing (or decreasing) the order volume. For example, in times of favorable price movement adaptive strategies are likely to become more aggressive by submitting more market orders or raising (or lowering) buy (sell) price or increasing order volume, and in times of adverse price movement adaptive strategies are more passive in order to avoid unnecessary market impact cost by submitting more limit orders or lowering (or raising) buy (sell) price or decreasing order volume.

Several researchers have made contributions to adaptive trading strategy. Almgren [2] showed evidence that strategies that are adaptive to market developments, i.e. that can for example vary their aggressiveness, are superior to static strategies. Nevmyvaka [30] proposed dynamic price adjustment strategy, where limit order's price is revised every 30 seconds adapting to the changing market state. Wang [35] proposed a dynamic focus strategy, which dynamically adjusts volume according to real-time update of state variables such as inventory and order book imbalance, and showed that dynamic focus strategy can outperform a static limit order strategy.

### 4.3.3 Performance Evaluation

Execution performance is assessed by comparing execution costs relative to a benchmark. The execution cost measure is a weighted sum of the difference between the transaction price and the benchmark price where the weights are simply the quantities traded [15]. The most used benchmark prices are VWAP, TWAP, arrival price, implementation shortfall, which have been introduced above.

The rationale here is that performance is considered good if the average execution price is more favorable than the benchmark price and bad if the average execution price is less favorable than the benchmark price. Take the VWAP as an example, which is an industry standard benchmark. The VWAP benchmark is calculated across the time horizon during which the trade was executed and is calculated as

$$\frac{\sum(\text{Volume} * \text{Price})}{\sum(\text{Volume})}$$

Hence, if the price of a buy trade is lower than VWAP, it is a good trade. If the price is higher, it is a bad trade. Although this is a simple metric, it largely filters out the effects of volatility, which composes market impact and price momentum during the trading period [1].

## 4.4 Evolutionary Computation in Trade Execution

Evolutionary computation is a subfield of artificial intelligence. The basic idea of an evolutionary algorithm is to mimic the evolutionary process, just as the name implies. The evolutionary process is operated on the solutions or the encodings of solutions. In financial markets, EC methodologies have been used for solving a broad selection of problems, ranging from predicting movements in current values to optimizing equity portfolio composition. An overview of EC applications in finance can be seen in [8]. Taking one of the best-known EC algorithms, the genetic algorithm, its key steps are [7]:

1. Initialization. Construct an initial population of encodings to potential solutions to a problem;
2. Calculation. Calculate the fitness of each potential solution in the population;
3. Selection. Select a pair of encodings (parents) corresponding to potential solutions from the existing population according to the fitness;
4. Crossover. Perform a crossover process on the encodings of the selected parent solutions;
5. Mutation. Apply a mutation process on the encodings of the two child solutions and then store them in the next population;
6. Repeat. Repeat steps 3-5 until  $n$  encodings of potential solutions have been created in the new population, and the old population are discarded;
7. Repeat Again. Go to step 2 and repeat until the desired fitness level has been reached or until a predefined number of populations have elapsed.

Another kind of EC is Genetic Programming (GP), an extension of GA. In GP, the evolutionary operators are applied directly to the solutions, thus the evolutionary search is operated on a space of computer programs. In financial application, this space can be a society of option pricing formulas, trading rules, or forecasting models. GP offers the potential to generate human-readable rules.

### 4.4.1 Related Work

Despite the importance of optimizing trade execution, there has been relatively little attention paid in the literature to the application of evolutionary methodologies for this task. One notable exception is Lim and Coggins [29] who applied a genetic algorithm to evolve a dynamic time strategy to optimize the trade execution performance using order book data from a fully electronic limit order market, the Australian Stock Exchange (ASX). In their study, the total volume of the order was divided into 10 slices and was traded within one day using limit orders. Each evolved chromosome had  $N$  genes where

each gene encoded the maximum lifetime that an individual order ( $1 \rightarrow N$ ) would remain on the order book (if it had not already been executed) before it was automatically ticked over the spread to close out the trade. The fitness function was the VWAP performance of that strategy relative to the benchmark daily VWAP. Each strategy was trained on three months' worth of transaction-level data using a market simulator. The results were tested out of sample on three highly liquid stocks and tested separately for sell side and buy side. The in sample and out of sample performances were better than pure limit / market order strategies.

## 4.5 Agent-Based Artificial Stock Market

In this chapter, the data used to test the execution strategies are derived from an artificial stock market. This section gives a brief introduction to the agent-based modeling technique.

### 4.5.1 Agent-Based Modeling

Agent-based modeling is a simulation technique to model non-linear systems consisting of heterogeneous interacting agents. The emergent properties of an agent-based model are the results of "bottom-up" processes, where the decisions of agents at a microscopic level determine the macroscopic behavior of the system. An 'agent' refers to a bundle of data and behavioral methods representing an entity constituting part of a computationally constructed world. The agents can vary from simple random zero-intelligence (ZI) agents as in [18] to sophisticated inductive learning agents. Even a simple agent-based model can exhibit complex behavior patterns and provide valuable information about the dynamics of the real-world system that it emulates [5].

The branch of agent-based modeling that deals with economic environments is sometimes referred to as agent-based computational economics (ACE), which naturally includes agent-based artificial markets [3]. They view financial markets as interacting groups of learning, boundedly-rational agents, by incorporating a well-defined price formation mechanism and a representation of market participants.

### 4.5.2 Artificial Market Models

Most artificial markets implement simplified market models, which omit some institutional details of trading, but serve their research needs sufficiently. One example is the clearing house model, where a number of agents only trade an asset at discrete time intervals. At the start of each time period, every agent forms his expectation for the future price in the next period, according to the available information, such as current market price and historical prices. Then, the trader can decide the proportion of the asset he will hold in the next period in order to maximize his profit. After collecting the accumulated buy and sell orders of all the agents, the market is cleared at a new market price. The renowned Santa Fe artificial stock market [26] is a such market, based on clearing house model. For general reviews, see for example [10, 24, 26, 25, 32, 34].

**Table 4.2.** Artificial Model Comparison

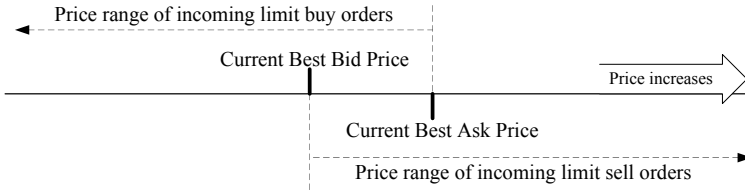
	Market Mechanism	Order Waiting Time	Order Size	Agent Type
Chiarella:2002	Double Auction	Discrete time steps	One unit of the stock	Fundamentalist, chartist and noise trader
Chiarella:2009	Double Auction	Discrete time steps	Generated using a specified demand function	Fundamentalist, chartist and noise trader
Raberto:2005	Double Auction	Exponential distributed	Depend on cash endowment	ZI agent
Chan:2001	Double Auction	Discrete time steps	One share	Artificial -intelligent trader
Yang:2003	Double Auction	Discrete time steps	A fixed number of shares	Neural learning agent
Daniel:2006	Double Auction	Exponential distributed	Normal distributed	ZI agent

However, the clearing house is only an approximate description of the way stock exchanges operate around the world. Nowadays, most financial markets are electronic markets, operating on an order book. Several researchers have made contributions to the models which implement the realistic trading market model, e.g. a limit order market, moving from the more stylized earlier financial market models toward more models incorporating explicit market microstructure [9, 31, 36].

It is difficult to design a market that perfectly reflects all the details of a real stock market. Therefore several choices, simplifications and assumptions are needed in order to make attempts to represent market structures and traders' behavior. In Chan's market [9], a submitted limit order price has to be better than the current price, for instance, any subsequent bid must be higher than the current bid to be posted, and subsequent ask is lower than the current ask to be posted. Yang's market [36] is similar to Chan's [9]. In Chiarella's markets [11, 12], traders set bids and asks and post market or limit orders according to exogenously fixed rules. One major drawback of the Chiarella's model comes from the assumption that there exists a constant fundamental value of the asset that all agents know, which is not realistic [19]. A comparison of these models can be seen in Table 4.2.

### 4.5.3 Simulation

In this chapter, our model is based on the zero-intelligence (ZI) model [13], which aims to generate a realistic aggregate order flow using very simple assumptions. The ZI agents are responsible for generating the order flow, by placing random orders to buy or sell. In this model, only one stock is traded, and dividends are ignored. Traders trade orders via a centralized limit order book, without the intermediacy of a market maker,



**Fig. 4.2.** Place Limit Price

**Table 4.3.** Initial Parameters for Order Book based ASM

Explanation	Value
Initial Price	$price^0 = 100$
Tick Price	$\delta = 0.01$
Probability of Cancellation Order	$\lambda_c = 0.07$
Probability of Market Order	$\lambda_m = 0.33$
Probability of Limit Order	$\lambda_l = 0.60$
Probability of Limit Order in Spread	$\lambda_{in} = 0.35$
Probability of Limit Order Out of Spread	$\lambda_{out} = 0.65$
Limit Price Tail Index	$1 + \alpha = 1.3$
Order Size	$(\mu, \sigma) \sim (4.5, 0.8) * 100$ shares
Waiting Time	$\tau = 6,90$

aiming to focus on the dynamics of a pure double auction. There are four aspects to design the ZI model, which are order sign, order type, limit order price and order size.

There are two order signs, buy or sell. The agents are equally probable to generate a buy order or a sell order. There are three types of orders in our model: market order, limit order and cancellation order (to delete an order from the order book). In London Stock Exchange, about 2/3 of the submitted order are limit orders and 1/3 are market orders [16], and roughly 10% of limits order in the order book are canceled before before being executed [6]. When an agent is active, she can try to issue a cancelation order with probability  $\lambda_c$  (oldest orders are canceled first), a market order with probability  $\lambda_m$ , a limit order with probability  $\lambda_l = 1 - \lambda_c - \lambda_m$ . Traders do not always place limit order at best bid/ask prices or inside the bid-ask spread. About 1/3 of limit orders fall outside the bid-ask spread and the density of placement falls off as a power law as a function of the distance from the best bid/ask price [17]. In our model, limit order price will be uniformly distributed in the spread with probability  $\lambda_{in}$ , and power-law distributed outside the spread with probability  $1 - \lambda_{in}$ . Limit order price ranges are illustrated in Figure 4.2. The parameters used in our simulation are presented in Table 4.3.

In this model, the order generation is modeled as a poisson process, which means that the time between orders follows an exponential distribution. In our simulation, we adopt a Swarm platform in JAVA [33], which is one of the most popular agent-based modeling platforms. The algorithm used in our simulation is described in Figure 4.1.

**Algorithm 4.1.** Behavior of ZI Agent

---

```

Simulator: generate  $t$  from EXPONENTIAL( $\tau$ );
current_time = current_time +  $t$ ;
Agent: generate  $P_{sign}$  from BERNOULLI(0.5);
generate independent  $P_{type}$  from UNIFORM(0,1];
if  $P_{type} \leq \lambda_c$  then
    /* a cancel order to be submitted */;
    Cancel oldest outstanding order;
end
else if  $P_{type} > (\lambda_c + \lambda_m)$  then
    /* a limit order to be submitted */;
    generate OrderSize  $\log(vol) \sim \text{NORMAL}(\mu, \sigma)$ ;
    generate independent  $P_{spread}$  from UNIFORM(0,1];
    if  $P_{spread} \leq \lambda_{in}$  then
        /* limit price to be in the spread */;
        generate LimitPrice  $price(t) \in \text{UNIFORM}(b(t), a(t))$ ;
    end
    /* limit price to be out of the spread */;
    generate LimitPrice  $price_i(\Delta) \sim \frac{1}{\Delta^{1+\alpha}}$ ;
    /* power-law distributed */;
    /* a market order to be submitted */;
    generate OrderSize  $vol(t) = [vol(a(t))|vol(b(t))]$ ;
    /* same size as best counterpart */;
end

```

---

## 4.6 Experiments

This section describes how to use a GA to uncover a quality trade execution strategy and evaluate it using the data generated from the artificial market described above.

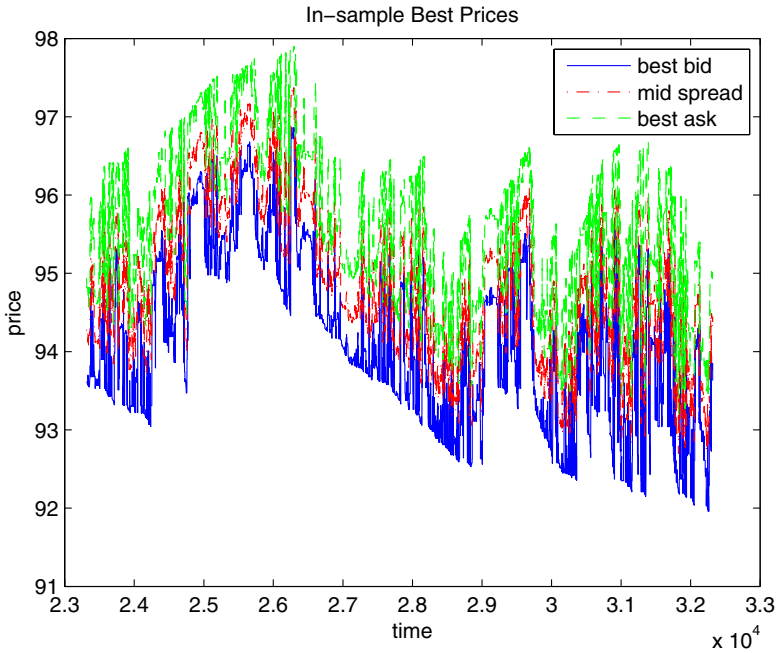
### 4.6.1 Data

This simulated market collects four kinds orders: market buy orders, limit buy orders, market sell orders and limit sell orders. The ASM simulation uses a database to store the details of each incoming order and best prices at each time point, which are limit buy orders, limit sell orders, market buy orders, market sell orders and best buy/sell orders.

The limit buy/sell order record contains each limit order's index number, arrival time, volume, submitting limit price, time when canceled or traded. The market buy/sell order record contains each market order's index number, arrival time, traded volume, traded price and the index number of corresponding traded limit order. The best price record contains the best bid and ask orders' index number, volume, price, and mid-spread price at each time when new order comes.

The ASM simulation was run for 30 virtual days. Each record in our dataset includes the following order-specific variables: size (in number of shares), side (buy or sell),





**Fig. 4.3.** In-sample Data

market or limit, limit price (if a limit order), starting time and ending time for the entire order.

#### 4.6.2 GA Strategies

Commonly, a trader may wish to trade an order over a specified period, if the order can not be filled at once. For the special characteristic of our artificial market, we assume that trading period is two and a half hours.

The design of an execution strategy can be considered of consisting of two steps. The first stage is to divide a big block of shares into multiple small orders, and the second step is to determine the parameters of each order, including order type (limit/market order), submission time, limit price (limit order) and lifetime (the time length when a limit order appears in the order book before it is canceled or changed).

How to divide a large trade depends on the order size and trading time. For simplicity, we divide our large trade into 30 smaller orders equally, and submit each smaller order into the market every 5 minutes (300 seconds).

Bear in mind that limit orders do not guarantee execution. When we are trading limit orders, we also need to consider how to deal with the unexecuted limit orders. They can either be executed as market orders at the end of their lifetimes to avoid unexecuted risk, or at the end of the whole trading period for better execution price.

In our experiment, we use both market and limit orders. The orders are submitted to the market every 5 minutes. As in the real market, divided orders can always be fully

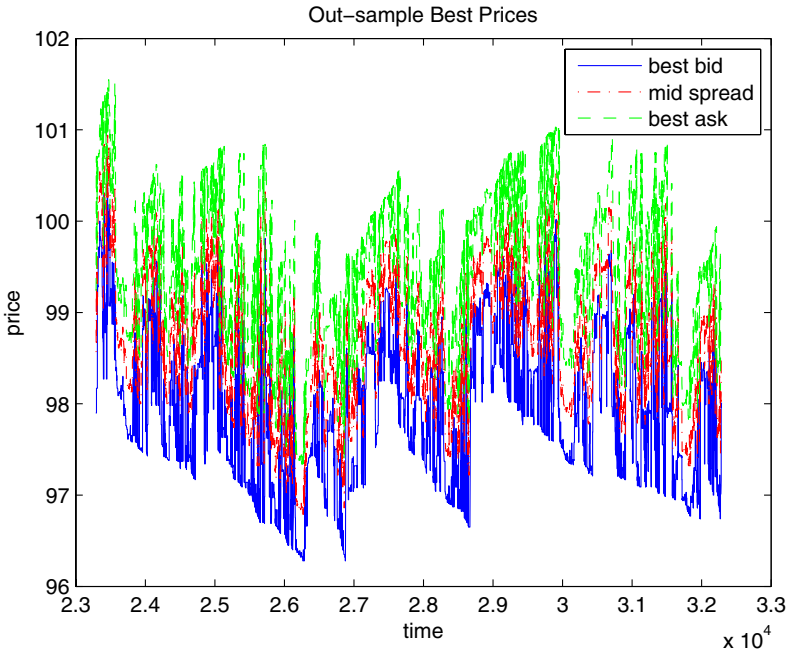


Fig. 4.4. Out-sample Data

$b_1$	.....	$b_{30}$	$c_1$	.....	$c_{30}$
-------	-------	----------	-------	-------	----------

$b_1 - b_{30}$ : lifetimes of 30 orders

$c_1 - c_{30}$ : relative limit prices of 30 orders

Fig. 4.5. Representation

traded if they are small enough. We assume that every market buy/sell order has the same size as the best limit sell/buy order in the order book, which is in accordance with the assumption in the ASM simulation. This means that one market order will cause only one limit order to be traded. So the parameters left for the 30 limit orders include limit order's price, lifetime in the order book before canceled if not executed by other market orders, which will be determined by the GA methodology. Figure 4.5 shows the representation of each GA individual or chromosome. These parameters of every GA individual form a GA strategy. The purpose of this experiment is to evolve an efficient execution strategy which has the best average execution price.

The objective function we used here is ratio of the difference between the VWAPs of the 30 orders and the entire executed orders generated from the ASM simulation to the entire executed orders' VWAP, which are  $VWAP_{30}$  and  $VWAP_{global}$  respectively. For both buy and sell orders, the smaller the VWAP Ratio, the better the strategy is.

$$VWAPRatio = \begin{cases} \frac{1000*(VWAP_{30}-VWAP_{global})}{VWAP_{global}} & \text{Buy Strategy} \\ \frac{1000*(VWAP_{global}-VWAP_{30})}{VWAP_{global}} & \text{Sell Strategy} \end{cases}$$

### 4.6.3 Parameter Settings

In each generation of GA computation, several new individuals are produced, each being a strategy which defines how to send the 30 orders into market. To test the performance of each strategy, we incorporate the 30 new orders into the order flow generated from ASM. The new order flow is simulated as a market, where the new order will be traded.

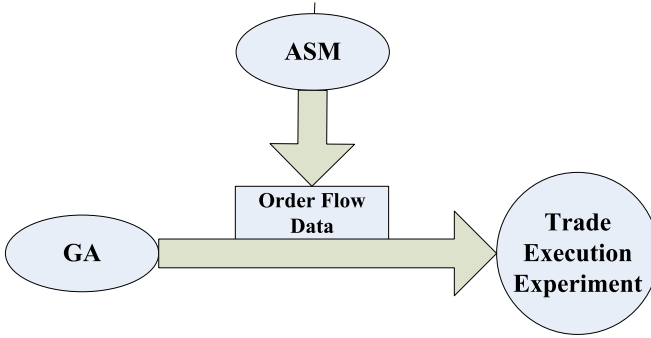


Fig. 4.6. Experiment

And we also assume that orders executed do not impact on the orders which arrive in the order book later. Figure 4.6 illustrates how the experiment works. In our experiment, orders can be executed in three different ways. The GA generates the limit price for each limit order. At the time when limit order is submitted to the order book, if the limit price crosses the best price in the opposite side of the order book, it will be executed immediately at the current best price as a marketable limit order (MLO). For instance, if the limit price of a buy order generated from GA is higher than the best ask price, this limit buy order is traded at the best ask price. If an order can not be executed during its lifetime, it will be automatically traded as a market order (MO) at the best price at the end of its lifetime. The last possibility is that the limit order (LO) is traded during its lifetime.

We used a population of 30 individuals, running for 100 generations, to evolve efficient GA strategies and we tested them with in-sample data and out-of-sample data separately. The parameters used in GA can be seen from Table 4.4. At the same time, we adopted another strategy to benchmark against our GA strategy, namely a pure market order strategy (MOS). It trades orders as market orders immediately on submission to the market.

### 4.6.4 Results and Discussion

Running both simulations for buy orders and sell orders over, we obtain the results shown in Tables 4.5 & 4.6.

**Table 4.4.** Parameters for Genetic Algorithm

Population size	30
Maximum number of generation	100
Generation gap	0.8
Crossover rate	0.75
Mutation rate	0.05
Selection method	Stochastic Universal Sampling
Crossover method	Single-Point

**Table 4.5.** Results of Buy Order.

	MOS	GA Strategy				
		VWAP Ratio ( $10^{-3}$ )	VWAP Ratio ( $10^{-3}$ )	TradedOrderType		
				MLO	LO	MO
In-sample	4.4474	-2.5899	4	20	6	
Out-of-sample	5.9748	0.5146	11	8	11	

**Table 4.6.** Results of Sell Order.

	MOS	GA Strategy				
		VWAP Ratio ( $10^{-3}$ )	VWAP Ratio ( $10^{-3}$ )	TradedOrderType		
				MLO	LO	MO
In-sample	2.7389	-5.8376	6	23	1	
Out-of-sample	3.2378	-1.8244	13	7	10	

The VWAP Ratio reveals the difference between the volume weighted execution price of GA orders and the average traded price of all orders during the whole simulation time. The better strategies have smaller VWAP ratios. The VWAP ratio of pure market order strategy, namely MOS, is also shown in Tables 4.5 & 4.6. In order to analyze the GA strategy, the execution types of the 30 orders are also calculated in our experiment. The three types are MLO, LO and MO.

From Tables 4.5 & 4.6, the GA strategy outperforms the MOS strategy significantly, both in-sample and out-of-sample, which is consistent with the results in [29]. The two tables show that the GA strategy, which has more orders executed in the way of LO, has a smaller VWAP ratio, meaning better performance. All the GA strategies with negative VWAP ratios have more orders executed in the way of LO than those executed in the two other ways, except the best out-of-sample strategy in Table 4.6. Also, GA strategies have achieved better VWAP than that of the whole simulation time for buy and sell in in-sample test, which is showed by the negative values of VWAP ratios. This is more significant for the sell order. These results suggest the applicability and potential of GA for trade execution.

## 4.7 Conclusion and Future Work

In this chapter, we present a problem in trade execution and emphasize an evolutionary approach to this problem. Initially, we built an order book using agent-based modeling. Using the order flow produced by the ASM, we applied a Genetic Algorithm to optimize the parameters of efficient trade execution strategies, in order to achieve a better execution price than the currently popular benchmark Volume Weighted Average Price (VWAP). In our experiments, GA evolved strategies provide satisfactory results for this trade execution problem, indicating Evolutionary Computation methodologies have potential applications in the domain of trade execution. The success of applying order book based ASM for trade execution experiment suggests an alternative way for testing trade execution strategies, instead of using backtesting strategies based on historical market data.

In future work, we intend to extend the application of EC to harder, dynamic, optimization problems in trade execution. For instance, if the price in market moves up or moves down, how should the trader change the limit price of limit order to get a better execution price? Kissell [23] proposed three adaptation tactics, which are Target Cost, Aggressive in the Money (AIM) and Passive in the Money (PIM), based on price adjustments to be consistent with investor's implementation goal during execution. Genetic Programming can be applied to this problem. Also, Agent-based Artificial Stock Market can be combined with GP. An agent with GP evolved strategy can be represented as an Algorithmic Trader in ASM, whose purpose is to evolve best execution strategy using GP. We also plan to relax some of the assumptions in our ASM, such as adding market impact into the current model.

## Acknowledgement

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/SRC/FM1389.

## References

1. Almgren, R.: Execution costs. In: Encyclopedia of Quantitative Finance. Wiley, Chichester (2008)
2. Almgren, R., Chriss, N.: Optimal execution of portfolio transactions. *Journal of Risk* 3(2), 5–39 (2000)
3. Berseus, P.: Creating an agent-based artificial market. Master's thesis, Lunds Tekniska Hogskola (2007)
4. Bikker, J., Spierdijk, L., Sluis, P.: Market impact costs of institutional equity trades. Technical Report 27, Netherlands Central Bank, Research Department (2007)
5. Bonabeau, E.: Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences* 93(3), 7280–7287 (2002)
6. Bouchaud, J., Mezard, M., Potters, M.: Statistical properties of stock order books: empirical results and models. *Quantitative Finance* 2(4), 251–256 (2002)
7. Brabazon, A., O'Neil, M.: *Biologically Inspired Algorithms for Financial Modeling*. Springer, Berlin (2006)

8. Brabazon, A., O'Neill, M., Dempsey, I.: An introduction to evolutionary computation in finance. *Computational Intelligence Magazine* 10(10), 1–12 (2008)
9. Chan, N., LeBaron, B., Lo, A., Poggio, T.: Agent-based models of financial markets: A comparison with experimental markets. Technical Report 4195-01, Massachusetts Institute of Technology (2001)
10. Chen, S.: Computationally intelligent agents in economics and finance. *Information Sciences* 177(5), 1153–1168 (2007)
11. Chiarella, C., Iori, G.: A simulation analysis of the microstructure of double auction markets. *Quantitative Finance* 2, 246–253 (2002)
12. Chiarella, C., Iori, G., Perello, J.: The impact of heterogeneous trading rules on the limit order book and order flows. *Journal of Economic Dynamics and Control* 33(3), 525–537 (2007)
13. Daniel, G.: Asynchronous simulations of a limit order book. PhD thesis, University of Manchester, U.K (2006)
14. Decovny, S.: Asian-pacific gears up for algorithmic trading. *Market View* 1, 13 (2008)
15. Engle, R., Ferstenberg, R., Russell, J.: Measuring and modeling execution cost and risk. Technical Report 08-09, Chicago GSB Research Paper (2008)
16. Farmer, J., Gerig, A., Lillo, F., Mike, S.: Market efficiency and the long-memory of supply and demand: Is price impact variable and permanent or fixed and temporary? *Quantitative Finance* 6(2), 107–112 (2006)
17. Farmer, J., Patelli, P., Zovko, I.: Supplementary material for ‘the predictive power of zero intelligence in financial markets’ (2005), <http://www.santafe.edu/~jdf/papers/zerosuppl.pdf>
18. Gode, D., Sunder, S.: Allocative efficiency of markets with zero-intelligence traders. *Journal of political economy* 101, 119–137 (1993)
19. Guo, T.: An agent-based simulation of double-auction markets. Master’s thesis, University of Toronto (2005)
20. Keim, D., Madhavan, A.: The cost of institutional equity trades. *Financial Analysts Journal* 54(4), 50–52 (1998)
21. Kim, K.: *Electronic and Algorithmic Trading Technology*. Academic Press, U.S.A (2007)
22. Kissell, R.: *Algorithmic Trading Strategies*. PhD thesis, Fordham University (2006)
23. Kissell, R., Malanur, R.: Algorithmic decision-making framework. *Journal of Trading* 1(1), 12–21 (2006)
24. LeBaron, B.: Agent-based computational finance: suggested readings and early research. *J. Econom. Dynam. Control* 24, 679–702 (2000)
25. LeBaron, B.: Agent-based computational finance. In: Tesfatsion, L., Judd, K. (eds.) *Handbook of Computational Economics. Agent-based Computational Economics*, vol. 2, pp. 134–151. Elsevier, Amsterdam
26. LeBaron, B.: A builder’s guide to agent based financial markets. *Quantitative Finance* 1(2), 254–261 (2001)
27. Lim, M., Coggins, R.: Price impact of trades on the ASX. Presented at Australasian Finance and Banking Conference (2003)
28. Lim, M., Coggins, R.: The immediate price impact of trades on the Australian stock exchange. *Quantitative Finance* 5(4), 365–377 (2005)
29. Lim, M., Coggins, R.: Optimal trade execution: an evolutionary approach. In: *Proc. IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1045–1052 (2005)
30. Nevmyvaka, Y., Feng, Y., Kearns, M.: Reinforcement learning for optimized trade execution. In: *ICML 2006: Proceedings of the 23rd International Conference on Machine Learning*, pp. 673–680. ACM Press, New York (2006)
31. Raberto, M., Cincotti, S.: Modeling and simulation of a double auction artificial financial market. *Physica A: Statistical Mechanics and its applications* 355(1), 34–45 (2005)

32. Samanidou, E., Zschischang, E., Stauffer, D., Lux, T.: Agent-based models of financial markets. *Reports on Progress in Physics* 70(3), 409–450 (2007)
33. Swarm. Swarm package can be obtained from <http://www.swarm.org>
34. Tesfatsion, L.: Agent-based computational economics: A constructive approach to economic theory. In: Tesfatsion, L., Judd, K. (eds.) *Handbook of computational economics: agent-based computational economics*, pp. 269–277. Elsevier, North-Holland, Amsterdam (2006)
35. Wang, J., Zhang, C.: Dynamic focus strategies for electronic trade execution in limit order markets. In: *CEC-EEE 2006: Proceedings of the 8th IEEE International Conference on E-Commerce Technology and the 3rd IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, pp. 26–35. IEEE Press, Washington (2006)
36. Yang, J.: The efficiency of an artificial double auction stock market with neural learning agents. In: Chen, S. (ed.) *Evolutionary Computation in Economics and Finance*, pp. 85–106. Springer, Berlin (2002)

# Agent-Based Co-operative Co-evolutionary Algorithms for Multi-objective Portfolio Optimization

Rafał Dreżewski, Krystian Obrocki, and Leszek Siwik

Department of Computer Science

AGH University of Science and Technology, Kraków, Poland

drezew@agh.edu.pl, kobrocki@gmail.com, siwik@agh.edu.pl

**Summary.** Co-evolutionary techniques makes it possible to apply evolutionary algorithms in the cases when it is not possible to formulate explicit fitness function. In the case of social and economic simulations such techniques provide us tools for modeling interactions between social and economic agents—especially when agent-based models of co-evolution are used. In this chapter agent-based versions of multi-objective co-operative co-evolutionary algorithms are presented and applied to portfolio optimization problem. The agent-based algorithms are compared with classical versions of SPEA2 and NSGA2 multi-objective evolutionary algorithms with the use of multi-objective test problems and multi-objective portfolio optimization problem. Presented results show that agent-based algorithms obtain better results in the case of multi-objective test problems, while in the case of portfolio optimization problem results are mixed.

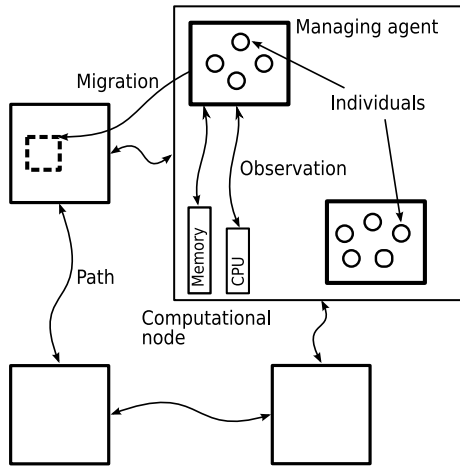
## 5.1 Introduction

Evolutionary algorithms are heuristic techniques which can be used for finding approximate solutions of global optimization problems. Evolutionary algorithms were also applied with great success to multi-modal and multi-objective problems (for example compare [11]), however in these cases some special mechanisms should be used in order to obtain good results. These are of course mechanisms specific for problems being solved but it seems that very important mechanisms in the case of multi-modal and multi-objective problems are the ones that maintain population diversity, because we are interested in finding not a single solution (as in the case of global optimization problems) but rather the whole sets of solutions.

Co-evolution is one of the mechanisms that can support maintaining of population diversity (see [14]). Another effect of applying co-evolutionary mechanisms is that we do not have to explicitly formulate the fitness function—we can just encode solutions in the genotypes and approximate fitness values for individuals on the basis of tournaments (*competitive co-evolutionary algorithms*) or co-operation (*co-operative co-evolutionary algorithms*).

Agent-based co-evolutionary algorithms are decentralized models of co-evolutionary computation. In fact two approaches are possible when we try to mix agent-based and evolutionary paradigms. In the first one agents are used to “manage” the evolutionary computations (see Figure 5.1). In such an approach each agent has the population of





**Fig. 5.1.** Agent-based layer used for managing evolutionary computations

individuals inside of it, and this sub-population is evolved with the use of a standard evolutionary algorithm. Agents themselves can migrate within the computational environment, from one computational node to another, trying to utilize in a best way free computational resources.

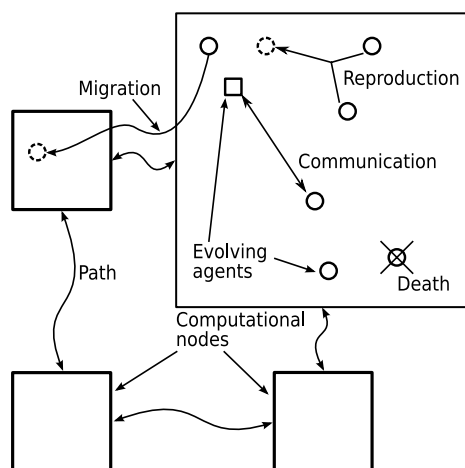
The example of the second approach is *co-evolutionary multi-agent system (Co-EMAS)* which results from the realization of co-evolutionary processes in multi-agent system (for example see [3, 4]). In such systems agents “live” within the environment (see fig. 5.2). All agents possess the ability to reproduce, they can compete for limited resources present within the environment, and die when they run out of resources.

In order to realize the selection process “better” (what means that they simply better solve the given problem) agents are given more resources from the environment (or from other agents) and “worse” agents are given less resources (or should give some of its resources to “better” agents). Such mechanisms result in decentralized evolutionary processes in which individuals (agents) make independently all their decisions concerning reproduction, migration, interactions with other agents, etc., taking into consideration conditions of the environment, other agents present within the neighborhood, and resources possessed.

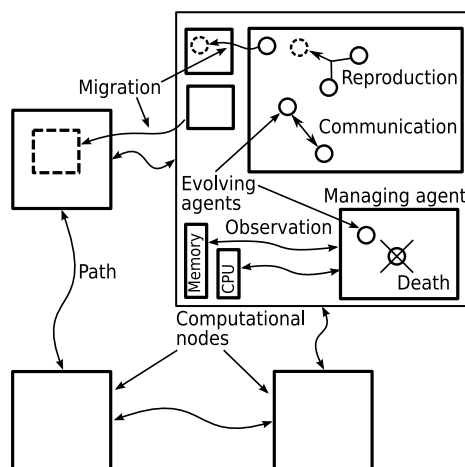
The approaches described above can be mixed. For example, one can imagine a system in which agents serve as management layer, and individuals, which “live” within such agents are also agents (see Figure 5.3). They can also migrate from one management agent to another and make independently all decisions (the system in which such approach was proposed is presented for example in [4]).

Agent-based co-evolutionary systems have some distinguishing features, among which the most interesting seem to be:

- the possibility of constructing hybrid systems, in which many different bio-inspired algorithms and techniques are used together within one coherent agent-based computational model,



**Fig. 5.2.** Co-evolutionary multi-agent system—population of evolving agents



**Fig. 5.3.** Mixed approach—agent-based layer is used for managing computations and evolving individuals are agents

- relaxation of computational constraints (because of the decentralization of evolutionary computations),
- the possibility of introducing new biologically and socially inspired operators or relations, which were hard or impossible to introduce in the case of “classical” evolutionary algorithms.

In the case of modeling and simulation of social and economic phenomena the model of co-evolutionary multi-agent system provides all necessary mechanisms like: agents, environment, agent-agent and agent-environment interactions needed for simulation of

complex social systems. The basic model with biological (evolutionary) layer can be easily extended—social and economical layers can be added on the top of biological one. Thus, we can construct artificial worlds and observe different emergent phenomena resulting from agents activities and interactions.

Multi-agent co-evolutionary algorithms based on CoEMAS model (utilizing different co-evolutionary interactions like: predator-prey, host-parasite, and sexual selection) were already applied to multi-objective problems (for example see [9], [7], [6]).

One of the first attempts of applying agent-based co-operative co-evolutionary approach to multi-objective optimization problems was presented in [8]. In the system presented in that paper the approach that uses agents as individuals living and evolving within the environment was used. There were several sub-populations (species) in the system. One criteria was assigned to each species. Agents competed for resources only within the species—there was no competition between agents that belonged to different species. Reproduction took place when the agent had enough resources to perform it. The agent searched for a reproduction partner from one of the opposite species. As the multi-objective test problems ZDT1, ZDT2, ZDT3, ZDT4 and ZDT6 functions [15] were used. Co-operative co-evolutionary multi-agent system was compared to NSGA2 and SPEA2 algorithms. Obtained results showed that proposed algorithm initially allowed for obtaining better solutions, but with time classical algorithms—especially NSGA2—were the better alternatives. However, in the case of ZDT4 problem this characteristic was reversed—co-operative co-evolutionary multi-agent system finally obtained better results.

Agent-based co-evolutionary algorithms have also been applied to financial problems. Agent-based co-evolutionary algorithm with predator-prey interactions solving multi-objective portfolio optimization problem was presented in [9]. Co-operative co-evolutionary algorithm using genetic programming approach for generating investment strategies was described in [10]. These two systems were based on the first presented above approach to constructing agent-based co-evolutionary algorithms—individuals were agents, which competed for limited resources, could reproduce, migrate, and which could eventually die when they ran out of resources.

The system presented in this chapter uses agents for managing evolutionary computations (first of the presented above approaches of mixing agent-based systems and evolutionary algorithms). Additionally, agent-based co-operative co-evolutionary approach is adapted for solving the multi-objective problem of portfolio optimization. The results of experiments with multi-objective test problems and portfolio optimization problem are used to compare proposed agent-based co-operative co-evolutionary algorithm, agent-based co-operative versions of well known SPEA2 and NSGA2 algorithms, and original versions of SPEA2 and NSGA2. The chapter is organized in the following way:

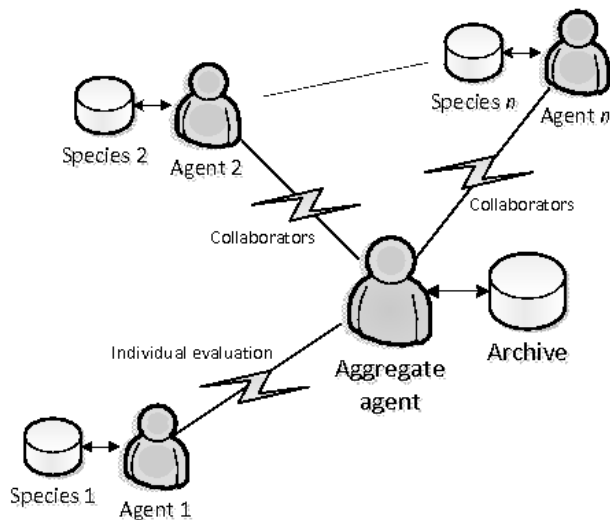
- In section 5.2 we will present the system and algorithms used in experiments: co-operative co-evolutionary multi-agent algorithm, agent-based co-operative co-evolutionary version of NSGA2 algorithm, and agent-based co-operative co-evolutionary version of SPEA2 algorithm.
- In section 5.3 the results of experiments with the proposed algorithms are presented. The problems used during experiments include commonly used multi-objective

test functions: ZDT [16] and DTLZ [2], and multi-objective portfolio optimization problem.

- Conclusions and future work plans are presented in section 5.4

## 5.2 Agent-Based Co-operative Co-evolutionary System

In the presented system co-operative co-evolutionary techniques were adapted to meet the demands of multi-objective problems and implemented with the use of mechanisms supported by the Java based framework jAgE [12]. This framework is particularly suitable for implementing agent-based evolutionary algorithms because it provides all necessary elements like environment composed of computational nodes, agents, basic mechanisms for agent-agent and agent-environment interactions.



**Fig. 5.4.** The architecture of agent-based co-operative co-evolutionary system

The co-operative co-evolutionary approach can be easily parallelized because the interaction between individuals from different sub-populations takes place only during forming complete solutions and evaluating their fitness. In co-operative co-evolutionary algorithm computational nodes do not have to communicate very often—communication is needed only during evaluation of the solutions—thus the parallelization of computations can be realized effectively in the decentralized system (like jAgE), not only on parallel machines.

Because the representatives of each species (sub-populations) had to be aggregated (in order to form the complete solution) and also because of the necessity of storing the complete non-dominated solutions, the central computational node (agent-aggregate) was introduced (see Figure 5.4). Its tasks include forming complete solutions (composed of the representatives of each species) and evaluation of the solutions. It also

maintains the set of non-dominated solutions found so far (the definition of domination relation and other issues connected with the Pareto approach to multi-objective optimization can be found for example in [1] or [9]). Each sub-population is responsible only for the selected part of solution, and evolved by one computational agent.

The system which we describe here has five implemented algorithms. Agent-based algorithms utilize agent layer for managing evolutionary computations. Three versions of agent-based co-evolutionary algorithms were implemented: co-operative co-evolutionary multi-agent algorithm (*CCEA-jAgE*), agent-based co-operative co-evolutionary version of NSGA2 algorithm (*CCNSGA2-jAgE*), and agent-based co-operative co-evolutionary version of SPEA2 algorithm (*CCSPEA2-jAgE*). Also two classical multi-objective evolutionary algorithms were implemented: NSGA2 and SPEA2 (details of these algorithms may be found in [1]).

## 5.2.1 The Algorithms

### Co-Operative Co-Evolutionary Multi-Agent Algorithm

In the **co-operative co-evolutionary multi-agent algorithm (CCEA-jAgE)**, which is based on the co-operative algorithm proposed in [13], there are computational agents which have individuals inside of them. Computational agents are located within the computational nodes of the jAgE platform—these nodes can be located on the same machine or on different machines connected with network. Agent-aggregate (which is a kind of “central point” of the system) is responsible for the creation of complete solutions and maintaining the set of non-dominated solutions found so far.

In the first step of this algorithm each of the computational agents performs the initialization of its sub-population (which is associated with the selected part of the problem—in our case this is one decision variable). Aggregate agent waits for receiving all of the sub-populations. When it receives all sub-populations, it forms complete solutions and computes the contribution of individuals coming from each species (sub-populations) to

---

#### Algorithm 5.1. The first step of the aggregate agent

---

```

for  $a \leftarrow a_1$  to  $a_n$  do
  /*  $a_i$  is the  $i$ -th computational agent */;
  receive the initial population  $P_a^0$  from agent  $a$ ;
  /*  $P_a^0$  is the sub-population of agent  $a$  in step 0 */;
end
 $C$  = aggregation of the solutions from  $P^0$ ;
/*  $C$  is the set of complete solutions (co-operations) consisted of;
the individuals coming from different sub-populations */;
calculate the contribution of each of the individuals in the co-operation;
for  $a \leftarrow a_1$  to  $a_n$  do
  send the sub-population  $P_a^0$  to agent  $a$ ;
end
 $A^0$  = choose the non-dominated solutions from  $C$ ;
/*  $A$  is the set of non-dominated solutions found so far */

```

---

**Algorithm 5.2.** Step of the computational agent

---

```

receive sub-population  $P^t$  from aggregate agent;
/*  $P^t$  is the sub-population in time  $t$  */;
compute the fitness of individuals from  $P^t$  on the basis of their contribution to the
whole solution quality;
 $P^{t+1} \leftarrow \emptyset$ ;
while  $P^{t+1}$  is not full do
    | select parents from  $P^t$ ;
    | generate offspring from parents and apply recombination;
    |  $P^{t+1} = P^{t+1} + \text{offspring}$ ;
end
mutate individuals from  $P^{t+1}$ ;
send  $P^{t+1}$  to aggregate agent;

```

---

the whole solution quality. Then the aggregate sends back all sub-populations and puts copies of all non-dominated solutions into the set of non-dominated solutions found so far (see Algorithm 5.1).

Each following step of computational agents (see Algorithm 5.2) begins with receiving of the sub-population from aggregate agent, then fitness of the individuals is computed. Next the selection of parents is performed, followed by the reproduction, recombination and mutation. At the end, the set of generated offspring is again sent to the aggregate agent.

Actions performed by the aggregate agent in the following steps start from checking whether the stop condition is fulfilled (see Algorithm 5.3). If yes, then the whole algorithm stops and the set of non-dominated solutions is the resulting Pareto frontier.

**Algorithm 5.3.** Step of the aggregate agent managing the computations

---

```

while stop condition is not fulfilled do
    | for  $a \leftarrow a_1$  to  $a_n$  do
    | | receive sub-population  $P_a^t$  from agent  $a$ ;
    | end
    | for  $a \leftarrow a_1$  to  $a_n$  do
    | |  $P_a^{t+1} = \text{select individuals for new generation from } P_a^{t-1} \cup P_a^t$ ;
    | end
    |  $C^{t+1} \leftarrow \text{complete solutions formed from } P^{t+1}$ ;
    | calculate the contribution of individuals coming from different species to the
    | whole solution quality;
    | for  $a \leftarrow a_1$  to  $a_n$  do
    | | send the sub-population  $P_a^{t+1}$  to the agent  $a$ ;
    | end
    | update the set of non-dominated solutions  $A^{t+1}$  with the use of  $C^{t+1}$ ;
end

```

---

---

**Algorithm 5.4.** Calculating the contribution of individuals coming from different species to the whole solution quality

---

```

for species  $P_s \leftarrow P_0$  to  $P_n$  do
  | choose representatives  $r_s$  from  $P_s$ ;
end
 $C \leftarrow \emptyset$ ;
for species  $P_s \leftarrow P_0$  to  $P_n$  do
  | for individual  $i_s \leftarrow i_0$  to  $i_N$  do
    |  $c_{pool} \leftarrow \emptyset$ ;
    | for  $j \leftarrow 1$  to  $|r_s|$  do
      |  $x \leftarrow$  aggregation of  $i_s$  with the representatives of the other species;
      | compute  $F(x)$ ;
      |  $c_{pool} \leftarrow c_{pool} + \{x\}$ ;
    | end
    |  $x \leftarrow$  solution chosen from  $c_{pool}$ ;
    |  $C \leftarrow C + \{x\}$ ;
    |  $F(x)$  is set as the contribution of individual  $i_s$  to the whole solution quality;
  | end
end
return  $C$ ;

```

---

When the stop condition is not fulfilled then aggregate agent receives sub-populations from computational agents, and for each sub-population generates the set containing next generation of individuals ( $P^{t+1}$ ) using individuals from previous generation of the given species and offspring sent by the given computational agent. Next the new set of complete solutions is generated on the basis of  $P^{t+1}$  and the contribution of individuals coming from different species to the whole solution quality is computed. Then the set of non-dominated solutions is updated (the new non-dominated solutions are inserted into the set and then all dominated solutions are removed from the set)—if the number of individuals in the set is greater than the maximal value then some individuals are removed on the basis of crowding algorithm (individuals from the most “crowded” areas are removed in the first place). Next, sub-populations are sent back to computational agents.

The process of creating complete solutions (aggregating individuals) and computing the contribution of the given individual to the quality of the whole solution is made with the use standard co-operative co-evolutionary schema. Firstly representatives  $r_s$  of all species are chosen, and then for subsequent individuals  $i_s$  from subsequent species  $s$  the pool  $c_{pool}$  of complete solutions is created. For every solution from the pool (which is composed of the given individual  $i_s$  and representatives of all other species) the values of all criteria are computed. One solution is chosen from the pool and inserted into the set  $C$  of currently generated solutions. The vector of values  $F(x)$  of the chosen solution is the measure of contribution of the given individual  $i_s$  to the quality of the solution (see Algorithm [5.4](#)).

## Agent-Based Co-Evolutionary Version of NSGA2 Algorithm with Co-Operative Mechanism (CCNSGA2-jAgE)

**CCNSGA2-jAgE—agent-based co-operative co-evolutionary version of NSGA2 algorithm**—is possible to obtain via proper configuration of the previously described algorithm (very similar solution was in fact applied in non-dominated sorting co-operative co-evolutionary genetic algorithm [11]).

As a result of integration of the previously described algorithm and NSGA2 [1] the agent-based co-operative version of NSGA2 was created. Thanks to the computed contribution of the given individual to the quality of the complete solution, the fitness computation in agent-based co-evolutionary NSGA2 is realized with the use of non-dominated sorting and crowding distance metric (see [1]). Additionally, the aggregate agent joins the populations of parents and offspring, and chooses (on the basis of elitist selection and within each sub-population separately) individuals which will form the next generation sub-population used for the creation of complete solutions. The applied schema implies that  $N$  best (according to non-dominated sorting and crowding distance metric) individuals survive. Other parts of algorithm are realized in the same way as in the case of previously described agent-based co-operative algorithm.

## Agent-Based Co-Evolutionary Version of SPEA2 Algorithm with Co-Operative Mechanism

In the case of **agent-based co-operative co-evolutionary version of SPEA2 algorithm (CCSPEA2-jAgE)** some modifications of the algorithms presented previously had to be done. It was caused mainly by the fact that SPEA2 uses additional external set of solutions during the process of evaluating individuals (compare [17]). In the described agent-based co-evolutionary version of SPEA2 algorithm each computational agent has its own, local, external set of solutions ( $IA$ ) used during the fitness estimation. This set is also sent to the aggregate agent, along with the sub-population which is evolved by the given computational agent.

First step of aggregate agent and computational agents is the same as in the case of CCEA-jAgE. Next steps of the algorithm of computational agents begin with receiving of the sub-population  $P^t$  and local external set of solutions  $IA^t$  from the aggregate agent (see Algorithm 5.5). On the basis of the contributions of the individuals to the quality of the complete solutions (computed by the aggregate agent), the fitness of individuals is computed. Next the archive  $IA^{t+1}$  is updated with the use of environmental selection mechanism adapted from SPEA2 algorithm [17]. Parents are selected from  $IA^{t+1}$  and children generated with the use of recombination operator are inserted into  $P^{t+1}$  (offspring population). Then mutation is applied to the individuals from set  $P^{t+1}$  and this set is sent to the aggregate agent together with the individuals from  $IA^{t+1}$ .

In the case of aggregate agent, the changes include receiving and sending additional sets of individuals  $IA^t$  (see Algorithm 5.6). Due to the fact that  $IA^t$  is the set of parents, now the step of selecting individuals to the next generation sub-population may be omitted. In order to create the set of complete solutions  $C^t$  and compute contributions of the individuals to the quality of the complete solutions, the aggregates are created from the individuals coming from populations  $P^t$  and  $IA^t$ . Finally all sub-populations



**Algorithm 5.5.** Step of the computational agent of CCSPEA2-jAgE algorithm

---

```

receive sub-population  $P^t$  and local external set of solutions  $IA^t$  from aggregate
agent;
compute the fitness of individuals from  $P^t$  and  $IA^t$  on the basis of their
contribution to the whole solution quality;
 $IA^{t+1}$  = environmental selection from  $P^t \cup IA^t$ ;
 $P^{t+1} \leftarrow \emptyset$ ;
while  $P^{t+1}$  is not full do
    select parents (using tournament selection) from  $IA^{t+1}$ ;
    generate offspring from parents and apply recombination;
     $P^{t+1} = P^{t+1} + \{offspring\}$ ;
end
mutate individuals from  $P^{t+1}$ ;
send  $P^{t+1}$  and  $IA^{t+1}$  to the aggregate agent;

```

---

**Algorithm 5.6.** Step of the aggregate agent managing the computations of CCSPEA2-jAgE algorithm

---

```

while stop condition is not fulfilled do
    for  $a \leftarrow a_1$  to  $a_n$  do
        receive sub-population  $P_a^t$  and additional set of individuals  $IA^t$  from agent
         $a$ ;
    end
     $C^t \leftarrow$  complete solutions formed from  $P^t \cup IA^t$ ;
    calculate the contribution of individuals coming from different species to the
    whole solution quality;
    for  $a \leftarrow a_1$  to  $a_n$  do
        send the sub-population  $P_a^t$  and additional set of individuals  $IA^t$  to the
        agent  $a$ ;
    end
    update the set of non-dominated solutions  $A^{t+1}$  with the use of  $C^t$ ;
end

```

---

are sent back to the proper computational agents and the set of non-dominated solutions is updated.

### 5.3 The Experiments

The algorithms presented in the previous section were preliminary assessed with the use of commonly used multi-objective ZDT [16] and DTLZ [2] test functions (detailed description of these test problems is presented in sections 5.3.1 and 5.3.2). Some of the results of these experiments were also presented in [5]. Generally speaking, the results obtained with the use of agent-based algorithms (especially CCEA-jAgE) were comparable, and in the case of some test problems better, than those obtained with the use of

SPEA2 and NSGA2. In this section we will present results of selected experiments with test functions and the problem of multi-objective portfolio optimization.

### 5.3.1 ZDT Test Functions

Test problems designed by Zitzler, Deb and Thiele [16] represent typical difficulties faced while performing real life multi-objective optimization tasks. Each of the six problems consists in minimization of function:

$$\mathcal{T}(x) = (f_1(x_1), f_2(x)) \quad (5.1)$$

where  $x = (x_1, \dots, x_m)$ ,  $f_1$  is a function of the first decision variable  $x_1$  and  $f_2$  is defined as:

$$f_2 = g(x_2, \dots, x_m) \cdot h(f_1(x_1), g(x_2, \dots, x_m)) \quad (5.2)$$

where  $g$  is a function of the remaining  $m - 1$  decision variables and the parameters of  $h$  are the function values of  $f_1$  and  $g$ . Each of functions  $f_1$ ,  $g$  and  $h$  is separately defined for every ZDT test problem. The number of decision variables  $m$  as well as their range of permissible values varies. ZDT problems are defined in the following way [16]:

- Test function  $\mathcal{T}_1$  has a convex Pareto frontier formed with  $g(x) = 1$  and is defined as follows:

$$\begin{cases} f_1(x_1) = x_1 \\ g(x_2, \dots, x_m) = 1 + 9 \cdot \sum_{i=2}^m \frac{x_i}{m-1} \\ h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} \end{cases} \quad (5.3)$$

where  $m = 30$  and  $x_i \in \langle 0; 1 \rangle$ .

- Test function  $\mathcal{T}_2$  has a non-convex Pareto frontier formed with  $g(x) = 1$ :

$$\begin{cases} f_1(x_1) = x_1 \\ g(x_2, \dots, x_m) = 1 + 9 \cdot \sum_{i=2}^m \frac{x_i}{m-1} \\ h(f_1, g) = 1 - \left(\frac{f_1}{g}\right)^2 \end{cases} \quad (5.4)$$

where  $m = 30$  and  $x_i \in \langle 0; 1 \rangle$ .

- Test function  $\mathcal{T}_3$  has a Pareto frontier composed of several non-continuous convex parts formed with  $g(x) = 1$ . The function is defined as follows:

$$\begin{cases} f_1(x_1) = x_1 \\ g(x_2, \dots, x_m) = 1 + 9 \cdot \sum_{i=2}^m \frac{x_i}{m-1} \\ h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} - \left(\frac{f_1}{g}\right) \cdot \sin(10\pi f_1) \end{cases} \quad (5.5)$$

where  $m = 30$  and  $x_i \in \langle 0; 1 \rangle$ .

- Test function  $\mathcal{T}_4$  has  $21^9$  local Pareto frontiers. It is well suited for testing algorithm's capability of dealing with multi-modality. The true Pareto frontier is formed with  $g(x) = 1$ :

$$\begin{cases} f_1(x_1) = x_1 \\ g(x_2, \dots, x_m) = 1 + 10 \cdot (m-1) + \sum_{i=2}^m (x_i^2 - 10 \cdot \cos(4\pi x_i)) \\ h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} \end{cases} \quad (5.6)$$

where  $m = 10$  and  $x_i \in (0; 1)$ .

- Test function  $\mathcal{T}_5$  represents a problem with multiple deceptive local Pareto frontiers. The best of them is formed with  $g(x) = 11$  while the global Pareto frontier is formed with  $g(x) = 10$ . The function requires binary representation of decision variables:

$$\begin{cases} f_1(x_1) = 1 + u(x_1) \\ g(x_2, \dots, x_m) = \sum_{i=2}^m v(u(x_i)) \\ h(f_1, g) = \frac{1}{f_1} \end{cases} \quad (5.7)$$

where  $m = 11$ ,  $x_1 \in \{0, 1\}^{30}$  and  $x_2, \dots, x_m \in \{0, 1\}^5$ . Function  $u(x_i)$  gives the number of ones in the bit vector  $x_i$  and  $v(u(x_i))$  is defined as follows:

$$v(u(x_i)) = \begin{cases} 2 + u(x_i) & , u(x_i) < 5 \\ 1 & , u(x_i) = 5 \end{cases} \quad (5.8)$$

- Test function  $\mathcal{T}_6$  introduces difficulties based on non-uniformity of the search space. The density of solutions is decreasing while closing to the true Pareto frontier. The frontier is formed with  $g(x) = 1$ :

$$\begin{cases} f_1(x_1) = 1 - \exp(-4x_1) \cdot \sin^6(6\pi x_1) \\ g(x_2, \dots, x_m) = 1 + \left(9 \cdot \sum_{i=2}^m \frac{x_i}{m-1}\right)^{0.25} \\ h(f_1, g) = \frac{1}{f_1} \end{cases} \quad (5.9)$$

where  $m = 10$  and  $x_i \in (0; 1)$ .

### 5.3.2 DTLZ Test Functions

The main limitation of ZDT test functions is the use of two criteria only. Such simplification facilitates graphical illustration of solutions and their verification against true Pareto frontier localization. Scalable test problems proposed by Deb, Thiele, Laumanns and Zitzler [2] represent  $M$ -criteria optimization tasks. Each of the DTLZ problems consists in minimization of functions  $f_1, \dots, f_m$ . Due to space limitations, we define here only DTLZ1 problem, which will be used during presentation of the results of experiments [2]:

- DTLZ1 test problem has a linear Pareto frontier located on a hyperplane, which satisfies the condition  $\sum_{m=1}^M f_m = 0.5$ , and  $11^k - 1$  local frontiers:

$$\begin{cases} f_1(x) = \frac{1}{2}x_1x_2 \cdots x_{M-1}(1 + g(x_M)) \\ f_2(x) = \frac{1}{2}x_1x_2 \cdots (1 - x_{M-1})(1 + g(x_M)) \\ \vdots \\ f_{M-1}(x) = \frac{1}{2}x_1(1 - x_2)(1 + g(x_M)) \\ f_M(x) = \frac{1}{2}(1 - x_1)(1 + g(x_M)) \end{cases} \quad (5.10)$$

where  $x_i \in \langle 0; 1 \rangle$  for  $i = 1, 2, \dots, n$  and  $n = M + k - 1$  with suggested value of  $k = |x_M| = 5$ .

### 5.3.3 Methodology of the Experiments

In all compared algorithms (CCEA, CCNSGA2, CCSPEA2, NSGA2 and SPEA2) the binary representation was used. One point crossover and bit inversion were used as genetic operators. As the selection mechanism tournament selection with elitism was used. The size of the population was set to 50. In order to minimize the differences between algorithms the values of crucial (and specific to each algorithm) parameters were obtained during preliminary experiments.

The results presented in this section include Pareto frontiers generated by the algorithms. Also, in order to better compare the generated results, hypervolume metric (HV) was used. Hypervolume metric [11] allows to estimate both the convergence to the true Pareto frontier as well as distribution of solutions over the whole approximation of the Pareto frontier. Hypervolume describes the area covered by solutions of obtained approximation of the Pareto frontier result set. For each found non-dominated solution, hypercube is evaluated with respect to the fixed reference point. In order to evaluate hypervolume ratio, value of hypervolume for obtained set is normalized with hypervolume value computed for true Pareto frontier.

HV is defined as follows:  $HV = v\left(\bigcup_{i=1}^N v_i\right)$ , where  $v_i$  is hypercube computed for  $i$ -th found non-dominated solution,  $PF^*$  represents obtained approximation of the Pareto frontier and  $PF$  is the true Pareto frontier.

Values presented in the figures are averages from 15 runs of each algorithm against each test problem. Due to space limitations only selected Pareto frontiers and values of hypervolume metrics are presented.

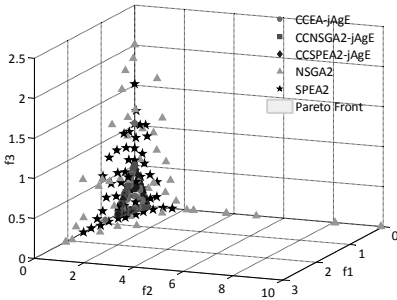
### 5.3.4 Experiments with Multi-objective Test Problems

Due to using three criteria in DTLZ problems it is possible to present the non-dominated solutions found by all compared algorithms. In the Figure 5.5 results from runs of all algorithms against DTLZ1 function are presented. Its Pareto frontier was properly located by all agent-based algorithms. It is worth to mention though, that the solutions obtained by CCNSGA2-jAgE and CCSPEA2-jAgE algorithms were located on the edges of the frontier rather than on its surface. Solutions found by NSGA2 and SPEA2 are all located at local Pareto frontiers (which are localized far away from the global frontier).

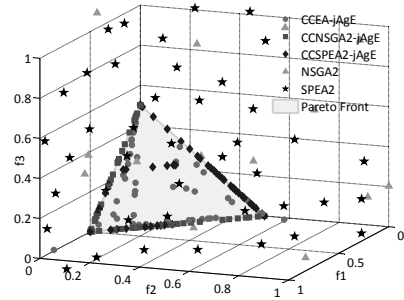
ZDT test problems are designed to use two criteria in order to facilitate presentation of the non-dominated solution sets in two dimensional space. In the Figures 5.6-5.8 we present selected Pareto frontiers obtained during experiments.

Multiple runs of the algorithms against test problems make it possible to present average values of hypervolume metric during experiments. In the Figures 5.9-5.11 values of hypervolume metric are presented for all six ZDT test problems.

In the case of ZDT1, ZDT2 and ZDT3 test problems the quality of non-dominated sets obtained with the use of CCEA-jAgE, CCNSGA2-jAgE, NSGA2 and SPEA2 algorithms is comparable. CCSPEA2-jAgE performs noticeably worse in this case. As it

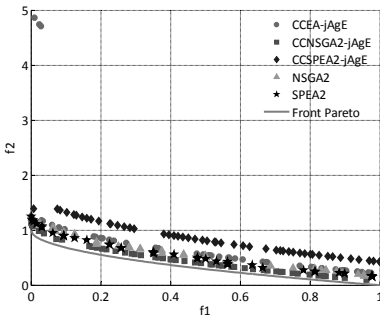


(a) The whole search domain

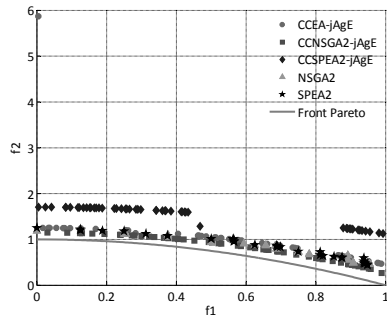


(b) Magnification of the Pareto frontier

Fig. 5.5. Pareto frontiers obtained for DTLZ1 problem



(a)



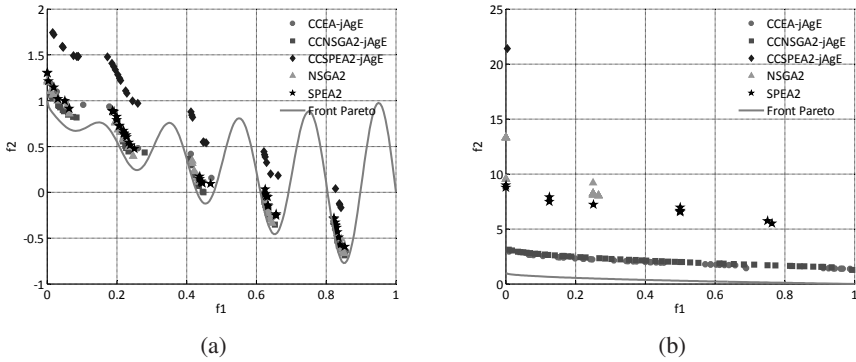
(b)

Fig. 5.6. Pareto frontiers obtained for ZDT1 (a) and ZDT2 (b) problems after 5000 fitness function evaluations for all compared algorithms

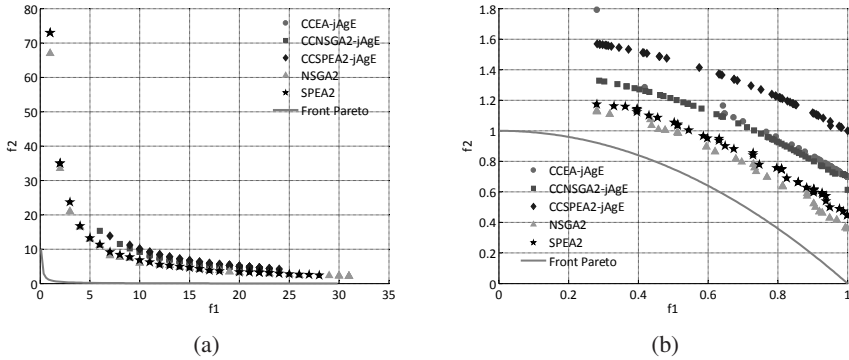
can be seen in Figure 5.7b and fig. 5.10b agent-based co-evolutionary algorithms generate significantly better results for ZDT4 test problem than NSGA2 and SPEA2. On the contrary, in the case of ZDT5, the latter two produce solutions wider spread and located closer to the true Pareto frontier (see Figure 5.8a and Figure 5.11a). The quality of the solutions generated for ZDT6 problems is comparable in the case of all algorithms, though NSGA2 and SPEA2 show slightly faster convergence to the true Pareto frontier (see Figure 5.8b and Figure 5.11b).

### 5.3.5 Experiments with Multi-objective Portfolio Optimization Problem

In experiments with multi-objective portfolio optimization problem complete solution is represented as a  $p$ -dimensional vector. Each decision variable represents the percentage participation of  $i$ -th ( $i \in 1 \dots p$ ) share in the whole portfolio. The problem is described with details in [9] (in that paper the agent-based predator-prey algorithm was used to solve this problem). Below we will present only the most important issues.



**Fig. 5.7.** Pareto frontiers obtained for ZDT3 (a) and ZDT4 (b) problems after 5000 fitness function evaluations for all compared algorithms



**Fig. 5.8.** Pareto frontiers obtained for ZDT5 (a) and ZDT6 (b) problems after 5000 fitness function evaluations for all compared algorithms

During presented experiments Warsaw Stock Exchange quotations from 2003-01-01 until 2005-12-31 were taken into consideration. Simultaneously, the portfolio consists of the three or seventeen stocks quoted on the Warsaw Stock Exchange. As the market index WIG20 has been taken into consideration.

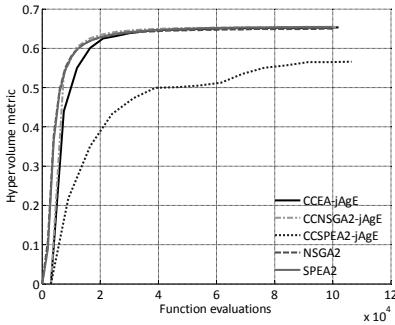
During experiments one-factor Sharpe model was used. This model was also used in [9] (in that work also comparison to other models and explanation why this particular model was used during experiments may be found). The algorithm (based on the one-factor Sharpe model) of computing the expected risk level and income expectation related to the portfolio of  $p$  assets is presented in Algorithm 5.7.

The meanings of the symbols used in Algorithm 5.7 are as follows:

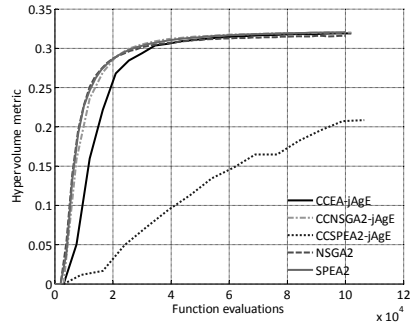
$p$  is the number of assets in the portfolio;

$n$  is the number of periods taken into consideration (the number of rates of return taken to the model);

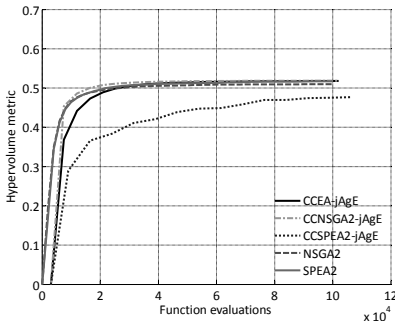
$\alpha_i, \beta_i$  are coefficients of the equations;



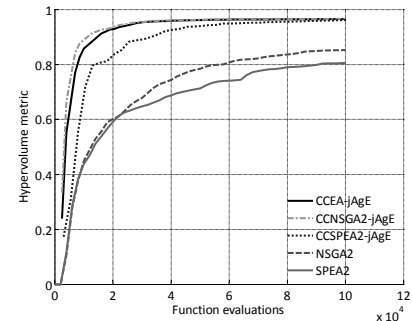
(a)



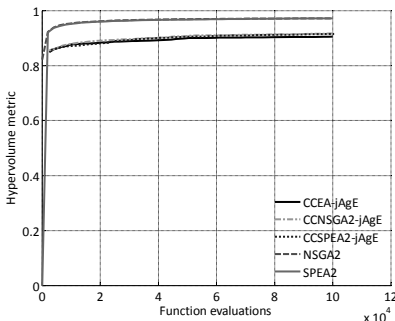
(b)

**Fig. 5.9.** Average values of hypervolume metric for ZDT1 (a) and ZDT2 (b) problems

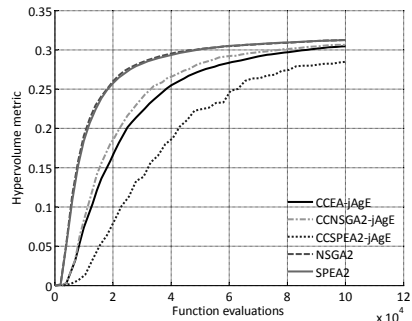
(a)



(b)

**Fig. 5.10.** Average values of hypervolume metric for ZDT3 (a) and ZDT4 (b) problems

(a)



(b)

**Fig. 5.11.** Average values of hypervolume metric for ZDT5 (a) and ZDT6 (b) problems

---

**Algorithm 5.7.** The algorithm (based on the one-factor Sharpe model) of computing the expected risk level and income expectation

---

Compute the arithmetic means on the basis of rate of returns;  
 Compute the value of  $\alpha$  coefficient  $\alpha_i = \overline{R_i} - \beta_i \overline{R_m}$ ;  
 Compute the value of  $\beta$  coefficient  $\beta_i = \frac{\sum_{t=1}^n (R_{it} - \overline{R_i})(R_{mt} - \overline{R_m})}{\sum_{t=1}^n (R_{mt} - \overline{R_m})^2}$ ;  
 Compute the expected rate of return of asset  $i$   $R_i = \alpha_i + \beta_i R_m + e_i$ ;  
 Compute the variance of random index  $s_{e_i}^2 = \frac{\sum_{t=1}^n (R_{it} - \alpha_i - \beta_i R_m)^2}{n-1}$ ;  
 Compute the variance of market index  $s_m^2 = \frac{\sum_{t=1}^n (R_{mt} - \overline{R_m})^2}{n-1}$ ;  
 Compute the risk level of the investing portfolio  $\beta_p = \sum_{i=1}^p (\omega_i \beta_i)$ ;  
 $s_{e_p}^2 = \sum_{i=1}^p (\omega_i^2 s_{e_i}^2)$ ;  
 $risk = \beta_p^2 s_m^2 + s_{e_p}^2$ ;  
 Compute the portfolio rate of return  $R_p = \sum_{i=1}^p (\omega_i R_i)$ ;

---

$\omega_i$  is percentage participation of  $i$ -th asset in the portfolio;

$e_i$  is random component of the equation;

$R_{it}$  is the rate of return in the period  $t$ ;

$R_{mt}$  is the rate of return of market index in period  $t$ ;

$R_m$  is the rate of return of market index;

$R_i$  is the rate of return of the  $i$ -th asset;

$R_p$  is the rate of return of the portfolio;

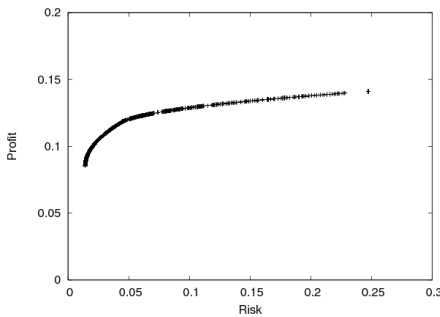
$s_i^2$  is the variance of the  $i$ -th asset;

$s_{e_i}^2$  is the variance of the random index of the  $i$ -th asset;

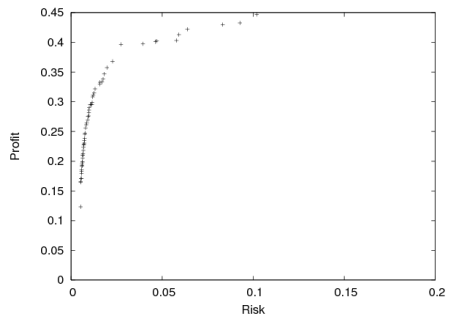
$s_{e_p}^2$  is the variance of the portfolio;

$\overline{R_i}$  is arithmetic mean of rate of return of the  $i$ -th asset;

$\overline{R_m}$  is arithmetic mean of rate of return of market index;



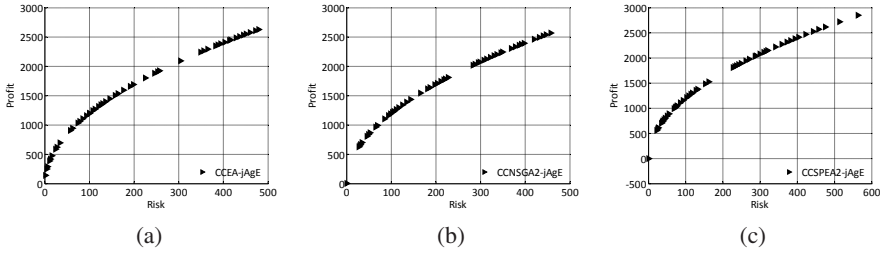
(a)



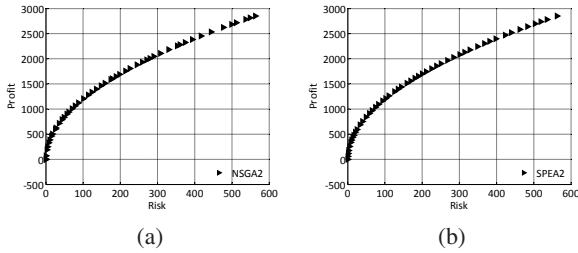
(b)

**Fig. 5.12.** The model Pareto frontier obtained using utter review method for 3 (a) and 17 (b) stocks set

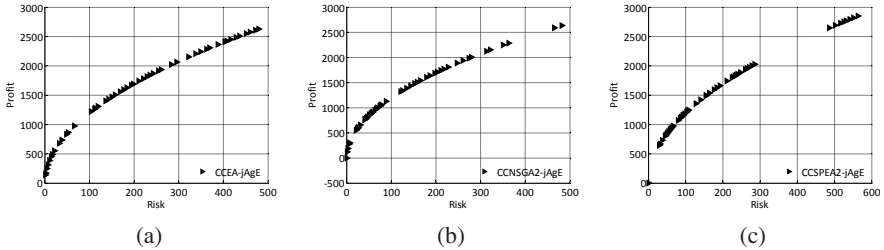




**Fig. 5.13.** Pareto frontiers obtained for 3 stocks problem after 2500 fitness function evaluations for CCEA (a), CCNSGA2 (b), and CCSPEA2 (c)



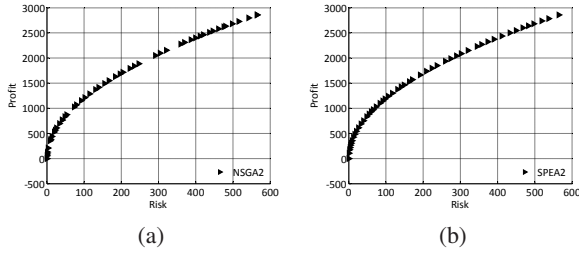
**Fig. 5.14.** Pareto frontiers obtained for 3 stocks problem after 2500 fitness function evaluations for NSGA2 (a) and SPEA2 (b)



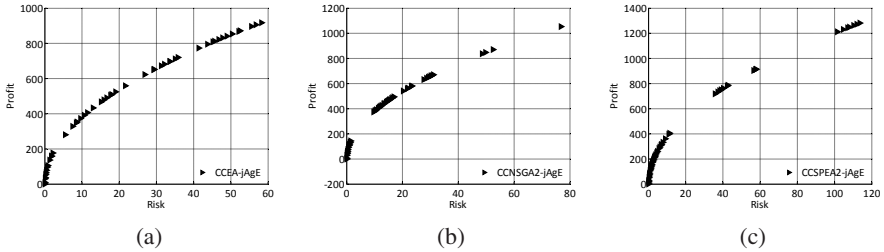
**Fig. 5.15.** Pareto frontiers obtained for 3 stocks problem after 5000 fitness function evaluations for CCEA (a), CCNSGA2 (b), and CCSPEA2 (c)

The goal of the optimization is to maximize the portfolio rate of return and minimize the portfolio risk level. The task consists in determining values of decision variables  $\omega_1 \dots \omega_p$  forming the vector  $\Omega = [\omega_1, \dots, \omega_p]^T$ , where  $0\% \leq \omega_i \leq 100\%$  and  $\sum_{i=1}^p \omega_i = 100\%$  and  $i = 1 \dots p$  and which is the subject of minimization with respect of two criteria  $F = [R_p(\Omega) * (-1), risk(\Omega)]^T$ .

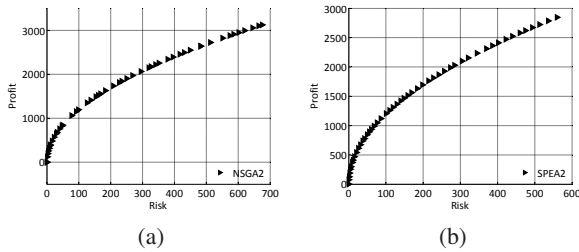
Model Pareto frontiers for two cases (portfolios consisting of three and seventeen stocks set), which are the subject of analysis in the following section, are presented in Figure [5.12](#)



**Fig. 5.16.** Pareto frontiers obtained for 3 stocks problem after 5000 fitness function evaluations for NSGA2 (a) and SPEA2 (b)



**Fig. 5.17.** Pareto frontiers obtained for 17 stocks problem after 25000 fitness function evaluations for CCEA (a), CCNSGA2 (b), and CCSPEA2 (c)

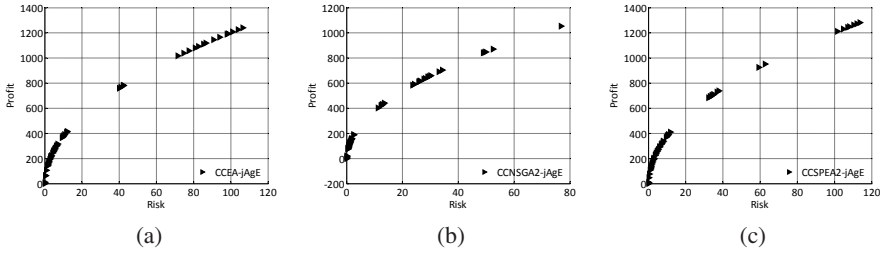


**Fig. 5.18.** Pareto frontiers obtained for 17 stocks problem after 25000 fitness function evaluations for NSGA2 (a) and SPEA2 (b)

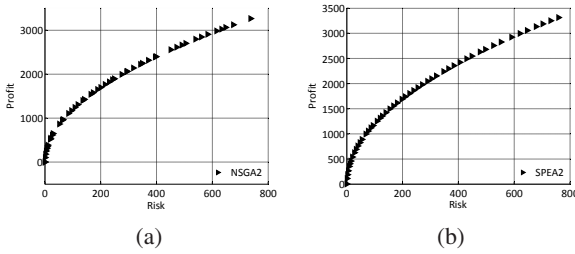
The Pareto frontiers obtained for 3 stocks problem after 2500 fitness function evaluations in typical experiment are presented in Figures 5.13 and 5.14. Pareto frontiers obtained after 5000 fitness function evaluations are shown in Figures 5.15 and 5.16.

The Figure 5.21a shows the average values of HV metric from 15 experiments for all compared algorithms. In this case (3 stocks) results are quite comparable for all implemented algorithms. Slightly worse results were obtained with the use of agent-based versions of SPEA2 and NSGA2 algorithms.

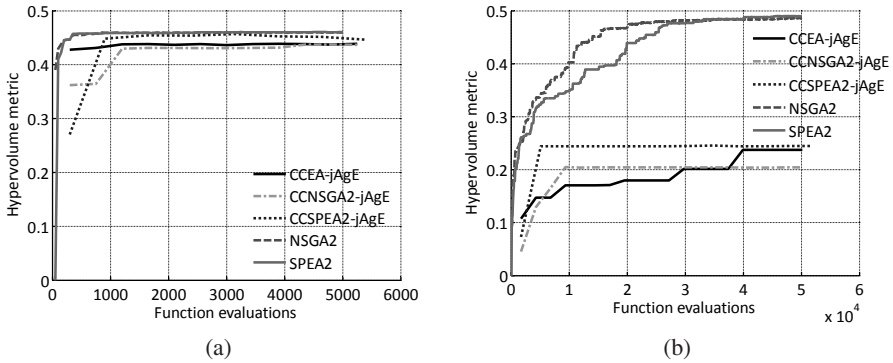
The Pareto frontiers obtained for 17 stocks problem after 25000 fitness function evaluations in typical experiment are presented in Figures 5.17 and 5.18. Pareto frontiers obtained after 50000 fitness function evaluations are shown in Figures 5.19 and 5.20.



**Fig. 5.19.** Pareto frontiers obtained for 17 stocks problem after 50000 fitness function evaluations for CCEA (a), CCNSGA2 (b), and CCSPEA2 (c)



**Fig. 5.20.** Pareto frontiers obtained for 17 stocks problem after 50000 fitness function evaluations for NSGA2 (a) and SPEA2 (b)



**Fig. 5.21.** Values of HV metrics for 3 (a) and 17 (b) stocks problems

In the Figure [5.21b](#) the average values of HV metric are presented (these are also average values from 15 experiments). In the case of the problem with 17 stocks the best results were obtained with the use of NSGA2 and SPEA2. When we look at the presented sample Pareto frontiers the CCEA-jAgE algorithm formed a quite comparable frontier—but the average value of HV metric was worse than in the case of NSGA2 and SPEA2. Agent-based versions of SPEA2 and NSGA2 decisively obtained worse results than other algorithms in this case.

## 5.4 Summary and Conclusions

In this chapter we have presented agent-based co-operative co-evolutionary algorithm for solving multi-objective problems. Also four other algorithms were implemented within the agent-based system: NSGA2, SPEA2 and agent-based co-operative co-evolutionary versions of these two state-of-the-art algorithms. The algorithms were verified with the use of standard multi-objective test problems—ZDT [16] and DTLZ [2] functions, and the multi-objective problem of constructing optimal portfolio.

In the case of ZDT and DTZL problems the winner was CCEA-jAgE algorithm—agent-based version of co-operative co-evolutionary algorithm. In the case of optimal portfolio problem the results were mixed. In the case of portfolio consisted of three stocks the results of all algorithms were rather comparable—only agent-based versions of SPEA2 and NSGA2 algorithms obtained slightly worse results. In the case of seven-teen stocks decisive winners were SPEA2 and NSGA2—especially when the values of HV metric were taken into consideration. Presented results lead to the conclusion that certainly more research is needed in the case of multi-objective agent-based techniques. But also it can be said that the results presented here (and in [5]) show that neither classical nor agent-based techniques can alone obtain good quality results for all kinds of multi-objective problems. We must carefully choose the right technique on the basis of the problem characteristics because there are no universal solutions. The algorithm that can obtain very good solutions for all types of multi-objective problems simply does not exist and we think that results presented here and in other our papers show this fact clearly.

When the future work is taken into consideration we can say that certainly presented agent-based algorithms will be further developed and tested on other multi-objective problems. Another direction of the research is (mentioned in section 5.1) the other way of merging multi-agent and evolutionary paradigms—the way in which agents are not used as the management layer but as the individuals that live, evolve and co-operate or compete with each other. Beside the financial problems which we have already used in our research, like investment strategies generation or multi-objective portfolio optimization, we are also planning to use agent-based co-evolutionary approach in modeling and simulation of economical and social phenomena.

## References

1. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester (2001)
2. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multi-objective optimization. Tech. rep., Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (2001)
3. Dreżewski, R.: A model of co-evolution in multi-agent system. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS (LNAI), vol. 2691, pp. 314–323. Springer, Heidelberg (2003)
4. Dreżewski, R.: Co-evolutionary multi-agent system with speciation and resource sharing mechanisms. *Computing and Informatics* 25(4), 305–331 (2006)

5. Dreżewski, R., Obrocki, K.: Co-operative co-evolutionary approach to multi-objective optimization. In: Corchado, E., Wu, X., Oja, E., Herrero, Á., Baroque, B. (eds.) HAIS 2009. LNCS (LNAI), vol. 5572, pp. 277–284. Springer, Heidelberg (2009)
6. Dreżewski, R., Siwik, L.: Co-evolutionary multi-agent system with sexual selection mechanism for multi-objective optimization. In: Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2006). IEEE press, Los Alamitos (2006a)
7. Dreżewski, R., Siwik, L.: Multi-objective optimization using co-evolutionary multi-agent system with host-parasite mechanism. In: Alexandrov, V.N., van Albada, G.D., Sloat, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3993, pp. 871–878. Springer, Heidelberg (2006b)
8. Dreżewski, R., Siwik, L.: Agent-based co-operative co-evolutionary algorithm for multi-objective optimization. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 388–397. Springer, Heidelberg (2008a)
9. Dreżewski, R., Siwik, L.: Co-evolutionary multi-agent system for portfolio optimization. In: Brabazon, A., O’Neill, M. (eds.) Natural Computation in Computational Finance, pp. 271–299. Springer, Heidelberg (2008b)
10. Dreżewski, R., Sepielak, J., Siwik, L.: Classical and agent-based evolutionary algorithms for investment strategies generation. In: Brabazon, A., O’Neill, M. (eds.) Natural Computation in Computational Finance, vol. 2. Springer, Heidelberg (2009)
11. Iorio, A., Li, X.: A cooperative coevolutionary multiobjective algorithm using non-dominated sorting. In: Deb, K., Poli, R., Banzhaf, W., Beyer, H.G., Burke, E.K., Darwen, P.J., Dasgupta, D., Floreano, D., Foster, J.A., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Thierens, D., Tyrrell, A.M. (eds.) GECCO 2004. LNCS, vol. 3102, pp. 537–548. Springer, Heidelberg (2004)
12. jAgE—Agent-Based Evolution Platform (2009), <http://age.iisg.agh.edu.pl>
13. Keerativuttitumrong, N., Chaiyaratana, N., Varavithya, V.: Multi-objective co-operative co-evolutionary genetic algorithm. In: Merelo, J.J., Adamidis, P., Beyer, H.G. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 288–297. Springer, Heidelberg (2002)
14. Paredis, J.: Coevolutionary algorithms. In: Bäck, T., Fogel, D., Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, (suppl.1). IOP Publishing/Oxford University Press (1998)
15. Zitzler, E.: Evolutionary algorithms for multiobjective optimization: methods and applications. PhD thesis, Swiss Federal Institute of Technology, Zurich (1999)
16. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* 8(2), 173–195 (2000)
17. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength pareto evolutionary algorithm. Tech. Rep. TIK-Report 103, Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (2001)

## Inferring Trader's Behavior from Prices

Louis Charbonneau<sup>1</sup> and Nawwaf Kharma<sup>2</sup>

<sup>1</sup> Computer Science Department, Concordia University, Montreal, Canada  
louischa@jmsb.concordia.ca

<sup>2</sup> Electrical and Computer Engineering Department, Concordia University, Montreal, Canada  
kharma@ece.concordia.ca

**Summary.** We propose a representation of the stock market as a group of rule-based trading agents, with the agents evolved using past prices. We encode each rule-based agent as a genome, and then describe how a steady-state genetic algorithm can evolve a group of these genomes (i.e. an inverted market) using past stock prices. This market is then used to generate forecasts of future stock prices, which are compared to actual future stock prices. We show how our method outperforms standard financial time-series forecasting models, such as ARIMA and Lognormal, on actual stock price data taken from real-world archives.

### 6.1 Introduction

The standard financial economics models of markets usually take the form of linear regressions, in which a key market variable is “explained” by independent variables. For instance, the Capital Asset Pricing Model (CAPM) posits that the rate of return on a stock,  $r_s$  depends on that of the general market,  $r_m$ , through the relationship  $r_s = \alpha + \beta(r_m - r_f) + \epsilon$ , where  $r_f$  is the risk-free rate [13]. Regression-based models have the advantage of being parsimonious, and their parameters can be inferred from market data. Their disadvantage lies in the remoteness of the specification from how markets work in reality, which generally results in a low statistical explanatory power.

In reality, financial market prices are created by the interaction between traders and market-makers. In organized exchanges, market-makers are the entities that take the counterparty to each order from the public, and have the power to set prices to the level they choose. For instance, if one wants to buy 100 shares of company XYZ as a member of the public, one will first look at the market-maker's quote - on a broker's website, for instance. If the quote is satisfactory, then a transaction will be recorded at the quoted price and the shares bought will come from the market-maker's inventory. The market-maker may then update its quote after the transaction for the next customer.

If we knew what rules traders from the public use to submit their orders and what rules market-makers follow to update their quoted prices, we would be in a better position to explain the dynamics of market prices than by using the more abstract market models: we would simply need to observe the initial conditions, and forward-simulate trading and market-making for as many periods as we please to get forecasts, assuming that the rules of the market participants remain reasonably stable during the simulation

period. We would obtain a high statistical explanatory power because of the similarity between our model's structure and the reality of the market.

So the question is, how much do we know about the behaviour rules of market participants? The behaviour of the market-maker is addressed in the literature of market micro-structure [12, 10]; the variables that market-makers consider in order to adjust their quotes are relatively well-defined and it is a simple matter to set up a stylized behaviour model. The difficulty lies on the side of the public traders: there are a huge variety of patterns of behaviour known as "trading rules" that public traders can potentially follow - assuming that traders follow a principled approach. Since there is a great variety of trading rules and no rule is clearly the best, traders cannot be expected to follow the same rules. For instance, the groups of traders on stock ABC may follow different rules than those trading stock DEF. It may be that traders on ABC are largely institutional and therefore conservative in their approach; traders on DEF may be hedge funds, and therefore more adventurous. Our main hypothesis is that the difference between the price time-series of ABC and DEF is due to the different rules that traders use on these stocks. Market-makers, on the other hand, follow the same pattern of behavior for both stocks.

This chapter explains how to use a *genetic algorithm* to infer the rules that traders use on a given stock by inferring these rules from past stock prices, if we assume suitable restrictions on the nature of these rules as well as on the set of trader's states. For instance, we assume memoryless traders, with no budget constraints - because we try to capture wide behaviour patterns in the market, not individual agent histories. A genetic algorithm is a parallel search technique in which a population of possible solutions to a problem evolves through many generations, the survival of each solution from a generation to the next depending on its quality, or "fitness". Eventually, after many generations, the best solutions would have been found. In this application, we do not know what rules traders use, so we create a population of randomly selected rule sets. These rule sets are used to simulate stock prices in the presence of an artificial market-maker, and the closeness of the resulting time-series of simulated prices compared to the real historical prices is evaluated for each rule set and constitutes its fitness. The final section of the chapter examines the forecasting power of the rule sets that were inferred by evolution, and conclusions are drawn.

## 6.2 An Artificial Market Model

### 6.2.1 Price Discretization and Trading Rules

We first define the *empirical cumulative distribution* of 1-day continuously compounded returns. An empirical cumulative distribution is obtained by sorting returns taken from a sample of historical observations, assuming that each sample return is equiprobable. Formally, if we have a sample of  $n$  return observations  $x_1, x_2, x_3, \dots, x_n$ , the empirical cumulative distribution  $\widehat{F}_n[x]$  is defined as the number of observations that are less than or equal to  $x$ , divided by the size of the sample  $n$ . If the sample is sorted such that  $x_1 \leq x_2 \leq x_3 \leq \dots \leq x_n$ , then we have:

$$\widehat{F}_n[x] = \frac{1}{n} \cdot \#\{i; 1 \leq i \leq n; x_i \leq x\}$$

where the symbol  $\#$  denotes the cardinality of the set. This empirical distribution is used to extract empirical return octiles, which are the returns for which  $F_n[\widehat{x}] = \frac{1}{8}$ ,  $F_n[x] = \frac{2}{8}$ , etc. We refer to these octiles as  $r(o_1)$ ,  $r(o_2)$ ,  $\dots$ ,  $r(o_8)$ . Due to the nature of continuously compounded returns, and the fact that in weakly stationary markets the median return is about zero, it follows that return octiles for  $n$  days can be approximated by a linear scaling of the 1-day return octiles  $n \cdot r(o_1)$ ,  $n \cdot r(o_2)$ ,  $\dots$ ,  $n \cdot r(o_8)$ .

Once the octiles are obtained, each return is coded in terms of which octile it belongs to. Denoting the present moment as time  $t$ , we write the  $n$ -days return from  $i - n$  days back in the past to  $i$  days back as  $r_{t-i-n,t-i}$ . We denote the coded return in term of octiles as  $\left\langle r_{t-i-n,t-i} \right\rangle_8^n \in \{1, 2, \dots, 8\}$ ,  $\forall n, \forall i$ .

A trading rule is defined as a IF-THEN construct based on a certain number of past stock return octiles relative to the current stock return octile. The choice of octiles as opposed to any other percentile is arbitrary; however, octiles offer an intuitively acceptable compromise between data granularity and rule specificity. Since continuous returns are centered around zero, choosing octiles gives us four different grades of down markets – that we could paraphrase as mildly down, moderately down but less than average, down and more than average, and severely down – and four different grades of up markets, which is probably sufficient for most practical purposes. When the logical expression in the IF part evaluates to TRUE, a *market position* is established for a specific amount. There are two kinds of market positions: a “long” (buy) or a “short” (sale). A *market* is defined as a population of trading rules. At any given point, only a certain subset of rules will fire under a given series of past prices, generating a market demand that varies through time.

We now turn to a general representation for trading rules, that subsumes fundamental as well as technical schemes.

We define the *rule lookahead*  $H$  as the maximum number of past periods a rule considers in order to make its decision. A variable-length generalized trading rule is a quadruple  $(P, v_{boundary}, v_{thresholds}, v_{conditions})$  containing the following elements:

- A symbol  $P$  taken from the set  $\{B, S\}$ , which determines whether a positive identification of the rule's condition results in a buy (B) or a sell (S) order.
- A variable-length vector  $v_{boundary}$  of integers taken between 1 and  $H - 1$  (inclusively), selected without replacement. These numbers define the boundary points for return calculations. This vector of integers needs not be sorted in the genome (since the application of genetic operators will scramble their order), but is sorted when interpreted. For instance, if  $H = 10$  and the numbers selected are 8, 2 and 5, this means that the rule will look at returns from  $t - 2$  to present (a 2-days return),  $t - 5$  to  $t - 2$  (a 3-days return),  $t - 8$  to  $t - 5$  (a 3-days return) and  $t - 10$  to  $t - 8$  (a 2-days return). If the vector has  $k$  elements, then  $k+1$  periods are implicitly defined.
- A vector  $v_{thresholds}$  of  $2(k+1)$  integers  $t_\alpha, t_\beta \in \{1, 2, \dots, 8\}$ , one pair for each interval defined implicitly by vector  $v_{boundary}$ . Again, it is not necessary to keep the pairs sorted.
- A vector  $v_{conditions}$  of  $k+1$  symbols taken with replacement from the set  $\{a, b, c, d\}$ . These define four possible relationships between the returns on the intervals defined in  $v_{boundary}$  and the corresponding threshold numbers  $t_\alpha$  and  $t_\beta$  from  $v_{thresholds}$ . For instance, if a given interval is from  $t - i - n$  to  $t - i$ :



1.  $\langle r_{t-i-n,t-i} \rangle_8^n \leq t_\alpha$ , (denoted by the symbol “a”)
2.  $t_\alpha \leq \langle r_{t-i-n,t-i} \rangle_8^n$ , (denoted by the symbol “b”)
3.  $t_\alpha \leq \langle r_{t-i-n,t-i} \rangle_8^n \leq t_\beta$ , (denoted by the symbol “c”)
4.  $(\langle r_{t-i-n,t-i} \rangle_8^n \leq t_\alpha) \vee (t_\beta \leq \langle r_{t-i-n,t-i} \rangle_8^n)$ , (denoted by the symbol “d”)

Note that relationships “a” or “b” require the definition of only one threshold number, and in this case, the rule simply ignores the second number  $t_\beta$  from the pair. This second number is kept in the chromosome and may become expressed in later generations if transmitted to the offspring. “Don’t care” conditions may occur in multiple ways in this representation: for instance, with relationship “a”, when  $t_\alpha = 8$ ; with relationship “b”, when  $t_\alpha = 1$ ; with relationship “c”, when  $t_\alpha = 1$  and  $t_\beta = 8$ ; and with relationship “d”, when  $t_\alpha = t_\beta$ .

We can obtain a more parsimonious representation for simple trading rules by setting a global  $v_{boundary}$  vector for the whole population of rules. This may be a good idea, since it allows us to impose as a constraint on the genetic algorithm coming from what is known about the behaviour of real traders, namely that more weight is usually given to more recent events. This restricts the search space to regions that we know are more likely to contain reasonable solutions.

This convention allows us to define rule specificity in the following obvious fashion: if the global  $v_{boundary}$  has  $k$  elements, the specificity  $S$  of a fixed-length generalized rule is defined as  $9(k+1)$  minus the sum of the  $(k+1)$  integers defined in the  $v_{conditions}$  and the  $v_{threshold}$  sets, where the integers to sum are:

$$\begin{aligned}
 & t_\alpha, \text{ if condition} = \text{“a”} \\
 & 8 - t_\alpha, \text{ if condition} = \text{“b”} \\
 & \max(t_\alpha, t_\beta) - \min(t_\alpha, t_\beta) + 1, \text{ if condition} = \text{“c”} \\
 & (\min(t_\alpha, t_\beta) - 1) + (8 - \max(t_\alpha, t_\beta)), \text{ if condition} = \text{“d”}
 \end{aligned}$$

This is simply the sum of the “off” octiles for each condition in the rule. Since we have a fixed-length representation, the maximum specificity of a rule is  $9(k+1)$  (a rule that can never be satisfied), and the minimum specificity is 0 (a rule that is always satisfied). We make the amount bet on each trade depend proportionally on specificity: for instance, as  $\$100 \frac{S}{9(k+1)}$ . That way, a rule that is hard to satisfy will bet more money when it fires, and a rule that gets satisfied more often will bet less every time it is satisfied. This setup ensures that populations do not get dominated by sets of looser rules, which provide little forecasting power. Setting the amount of money bet based on specificity also reflects the behaviour of real traders.

## 6.2.2 Calibrating an Empirical Market-Maker

The market-maker posts new prices based on the state of its inventory. This process can be very complex and may require sophisticated artificial intelligence; what we want here is to capture the broad aspects of this behaviour. So we assume that the only thing that matters to market-makers is to minimize their inventory. Large inventories (long

or short) create a speculative position in the stock, almost certainly contrary to the direction of where the smart money is. Indeed, one of the main observations of the literature on market-making is that asymmetric information is one of the greatest risk to the business. Indeed, traders having privileged information will trade early against the market-makers; therefore, any accumulation of inventory should prompt a change in price to neutralize the direction of the change. The disadvantage coming from asymmetric information should thus be combated by the “price adjustment function”, which is based on the following observations:

- if the value of the market-maker's inventory is high (an indication that stocks were recently bought), then it indicates that the current price is too high: too many trading agents want to sell. The function should therefore return a lower price, which will discourage agents from selling further.
- if the value of the market-maker's inventory is low – meaning in this case that it is a large short position (an indication that stocks were recently sold), then it indicates that the current price is too low: too many trading agents want to buy. The function should therefore return a higher price, which will discourage agents from buying further.

The only remaining problem is to determine what should be considered a low and a high inventory. To do that, we create an empirical distribution of inventory values that is calibrated to the price time-series and the market under consideration as follows:

- a historical price time-series of at least 100 observations is selected.
- we set the H first prices from the time-series as the current price stack.
- all the “IF” conditions of the current market are evaluated on the H prices in the stack, causing a subset of rules from the population to fire.
- firing rules create *orders* that are put in the book and immediately submitted to the market-maker. These orders constitute a market demand for the current moment in time.
- the market-maker fills the orders at the current price and computes his new stock inventory.
- the value of the stock inventory of the day is stored. Initially, the inventory is zero.
- the time counter is increased; the market-maker does not push a new price on the stack, but the next historical price is pushed onto the stack.

The behavioural rule of the market-maker is specified by matching the two empirical distributions (price returns and inventory values) octile by octile. As inventory octiles go from 1 to 8, corresponding price octiles go from 8 to 1. For any probability  $p \in (0, 1)$ , the percentile to percentile matching condition can be written as:

$$1 - \widehat{G}_n[y] = \widehat{F}_n[x],$$

and a behavioural response function that maps inventory levels to price changes, expressed as a continuously compounded rate  $x$ , is given by:

$$x = \widehat{F}_n^{-1}[\widehat{G}_n[y]]$$

At this stage, we substitute the empirical distribution with a parameterized form, such as a logistic distribution:

$$F(\alpha, \beta) = \frac{1}{1 + e^{(-\frac{x-\alpha}{\beta})}}$$

There are many ways to do so; we can optimize the parameters  $\alpha$  and  $\beta$  by using steepest descent, while minimizing the Kolmogorov-Smirnov statistic, given by:

$$D = \max_i D_i = \max_i \left[ \left| F(\alpha, \beta) - \widehat{F}_n[x] \right| \right]$$

An interesting fact about our calibration procedure is that genomes depend on the nature of the market-maker to affect prices via forward simulation. Not unlike actual biological genes, our trading rules can be traced to a position in the genome *in abstracto*, but they do not exist independently when the genome is activated within a particular environment. Depending on the environment (in this case the past price history), this or that different gene may or may not be expressed (or responded to by the price-adjustment mechanism) as much as another. Gene dominance is a phenomenon emerging from the interaction between genome and current state of the environment.

### 6.3 The Genetic Algorithm

We want to find what is the best population of trading rules that will generate market prices that are closest to a given market price history. Once a population of rules is evolved and generates prices that are close to a given historical interval on a training sample, we consider this population as a reconstruction of the underlying dynamic of the market. Hence, we forward simulate the market, generating a forecast, which may be evaluated against actual market performance. We divide a historical sample of data into two intervals:

1. the pre-processing interval, used to construct the empirical distributions  $\widehat{F}_n[x]$  and  $\widehat{G}_n[y]$ .  $\widehat{F}_n[x]$  is pre-computed and stored, whereas  $\widehat{G}_n[y]$  is computed at each genome evaluation. In our implementation, we use a pre-processing sample of 100 observations, which is reasonably large.
2. the training interval, composed of the data that comes next to the pre-processing sample, is used by the inner GA to evaluate the fitness of a given genome. The length of this interval can be selected at will. The trade-off is that the longer the interval, the more lengthy the resulting genome, and the more processing time. In the experiments described in the next section, a training sample length of 50 weekly observations has been used, for forecasts of up to one year. Judging from the amount of time it takes to evolve a market on a standard machine, it would be quite impractical to attempt to train the GA on much longer intervals.

When an acceptably fit individual has been found, the individual can be used to simulate a market run beyond the training interval. The newly generated prices can be used as a forecast. If the model is used in a business setting to generate forecasts, the training interval ends at the present day and a forecast can be made for any desired number of

future periods. The forecast interval is composed of the prices immediately following the training sample. This forecast interval contains historical data that the model has never seen during training via evolution. We will evaluate the forecast performance on these prices.

Forecast accuracy is expected to get worse, as the forecast is extended further into the future. It makes sense to evaluate forecasts over a length that is commensurate with the training interval: it would indeed be surprising if a model trained using a small historical sample succeeded in generating reliable forecasts over a much longer period into the future.

### 6.3.1 Genome Structure

Since our rules are composed of tuples, they can be easily expressed as a variable-length or a fixed length representation. In practice, variable-length rules are implemented as tuples having a specific maximum length, which keeps memory usage under reasonable bounds. Rules are collected into a variable-length set that constitutes a genome, the unity of selection. The data structure used for a genome is a two-dimensional array of alleles, composed of symbols taken from the set  $\{1, 2, \dots, 8\}$ . We do not need the symbols  $\{B, S\}$  (“B” is for a buy condition and “S” for a sell); we interpret the second allele as meaning a “B” if the allele is between 1 and 4, and a “S” otherwise.

To implement a variable number of rules per genome within a data structure of fixed dimensions, we interpret the very first allele of every line as an “on/off” expression. Genomes therefore carry junk genetic baggage that play no role in the current generation, but may be reinterpreted and play an useful role in future generations if the expression allele flips due to a mutation or a crossover.

We have already mentioned the important parameter  $H$  that we call “rule lookback”, which controls how far in the past the rules of a genome can look to take their decision. This should intuitively set to match the number of lags that are significant in the data correlogram and autocorrelogram.

### 6.3.2 Genetic Operators and Parent Selection

We evolve populations of genomes (which are themselves sets of rules) within an overlapping steady-state genetic algorithm. Each genome is a two-dimensional array of dimensions  $N \times M$ .  $N$  represents the maximum number of rules in a genome and  $M$  is

**Table 6.1.** Genetic algorithm parameters for a typical run

Genetic parameter	value
elitism proportion	40%
population size	200 (constant)
maximum number of rules (N)	70
crossover type	one-point
mutation type	uniform, $p = 10\%$
parent selection	3-way tournament
termination criterion	number of generations (500000)

the maximum rule representation length, which is simply defined as  $4H + 5$  to allow for sufficient space to represent even the longest rules. Parents are selected by tournaments with a window size of 3, and crossover is of the single point variety. Mutation is uniform.

To choose acceptable values for the simulation parameters, we use a simple static parameter strategy combined with a grid search over the space of the two most problematic model-specific parameters.

### 6.3.3 Fitness Evaluation and Objective Function

The objective function is bipartite; it is calculated for each genome as a weighted sum of two criteria:

1. a statistical closeness criteria that compares the price path generated by a genome to the prices in the training sample, namely the sum of squared errors between the price forecast given by market forward simulation and the actual price training sample.
2. a parsimony criterion, the effective genome length, defined as the number of “on” rules in the current genome. Recall that this varies from one individual to the other, as a specific allele on every line of the code matrix defines whether the corresponding rule is active or not.

Formally, the objective function of a genome is:

$$Obj(genome) = w(SSE) + (1 - w)(len(genome))$$

where the parameter  $w \in [0, 1]$  is the relative importance of the error minimization part of the objective function. From this definition, it is plain that the higher the objective function, the lower the fitness of an individual.

The objective function of an individual genome is computed with the following steps:

1. the empirical market-maker calibration process is performed for the market defined by the genome, using the pre-processing interval.
2. the market defined by the genome is forward-simulated to generate as many prices as there are in the training interval. The initial price conditions for this forward simulation are the  $H$  last prices from the pre-processing interval.
3. These forward simulated prices are compared to the actual training data to compute the sum of squared errors over the training interval. Denote the actual prices as  $a_1, a_2, a_3, \dots, a_n$ , and the simulated prices as  $s_1, s_2, s_3, \dots, s_n$ , we have:

$$SSE = \sum_i^n (s_i - a_i)^2$$

4. The length of the genome is calculated. At this point, we have all the required elements to compute the objective function of the individual.

## 6.4 Model Evaluation

### 6.4.1 Experimental Data Selection

It would be a waste of time to apply the model to time-series that exhibit a clear trend. At the cost of much computation, the end result would be a forecast that would be very similar to that obtained by fitting a linear trend line to the same data. More importantly, testing a nonlinear model on linearly trending data would tend to over-estimate the true performance of the nonlinear model. It is indeed an easy problem to infer a linear tendency in data.

So the stocks we take to evaluate the model on are the components of the S&P500 that are stationary. Formally, a stochastic process  $y_t$  (indexed by time  $t$ ) is weakly stationary [8] if:

1.  $E[y_t]$  is independent of  $t$ .
2.  $Var[y_t]$  is independent of  $t$ , and
3.  $Cov[y_t, y_s]$  is a finite function of  $|t - s|$ , but not of only  $t$  or  $s$ .

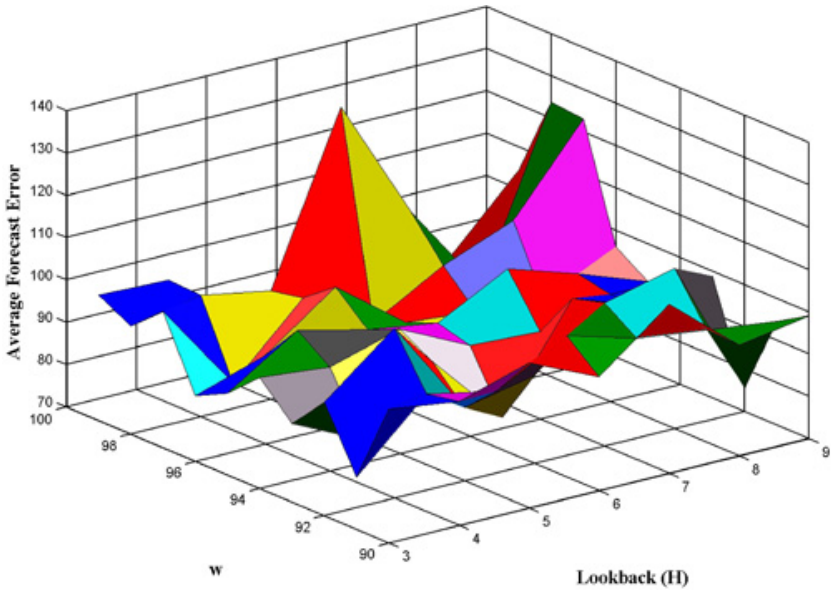
To test for stationarity, we have used the Dickey–Fuller test [5]. Of all the components of the S&P500 index over the 2004–2006 period, 276 were found to be stationary. 200 stocks from that sample were used in our experiments [3].

### 6.4.2 Parameter Optimization

Any evolutionary computation model has a wealth of parameters. As shown in Table 6.1, we have chosen most general GA parameters *a priori* (with some degree of experimentation in the course of developing the software). This is justified by the literature, as it has been shown by trials on test suites that GAs are quite robust with respect to their parameter settings [7].

However, it is apparent from our discussion that the value of parameter  $w$  in the objective function (the relative importance of the SSE, which is model-specific) has a potentially important effect on the nature of the solution that is found at the end of the evolution process: no *a priori* choice is obvious in this case. Another problematic

<sup>3</sup> The ticker symbols of these stocks are ASH, AIZ, ADSK, ADP, AN, AZO, AVY, AVP, BHI, BLL, BK, BCR, BRL, BAX, BBT, BBBY, BBY, BIIB, BJS, BDK, HRB, BMY, BRCM, BNI, CHRW, CA, CAM, COF, CAH, CCL, CAT, CBG, CNP, CTX, CHK, CVX, CIEN, CI, CTAS, CC, CSCO, CIT, C, CTXS, CCU, CLX, COH, CCE, CMA, CBH, CSC, CPWR, CAG, COP, CNX, ED, STZ, CEG, CBE, GLW, COST, CFC, CVH, DHI, DRI, DE, DVN, DDS, D, RRD, DOW, DTE, DD, ETFC, EMN, EK, ETN, EBAY, EIX, EP, ERTS, EDS, EMC, ESV, EOG, EFX, EL, ESRX, FDO, FNM, FRE, FII, FDX, FIS, FITB, FHN, FRX, FO, GPS, GE, GM, GPC, GENZ, GR, GT, GOOG, GWW, HAL, HOG, HAR, HAS, HNZ, HPC, HES, HD, HON, HSP, HBAN, IACI, ITW, RX, IR, TEG, INTC, IBM, IFF, IP, IPG, ITT, JBL, JEC, JNS, JDSU, JNJ, JCI, JNY, KBH, KMB, KG, KLAC, LLL, LH, LM, LEG, LEH, LEN, LUK, LXX, LLY, LIZ, LOW, LSI, MMC, MAS, MAT, MBI, MKC, MHP, MCK, MWV, MHS, MDT, MDP, MTG, MCHP, MU, MSFT, MIL, MOLX, TAP, MNST, MUR, MYL, NBR, NCC, NOV, NSM, NTAP, NEM, NKE, NI, NE, NBL, NSC, NOVL, NVLS, NYX, PLL, PH, PAYX, BTU, PKI, PFE, PNW, PBI, PCL, PPG, PGR, PRU and PHM.



**Fig. 6.1.** Average SSE of forecast errors for 20 stationary stocks

model-specific parameter is the rule lookback  $H$ , which determines how far back in the past rules look to make their decisions, and represents the maximum memory size of traders. Ideally, all GA and model parameters should be selected as to maximize forecast performance of the forward simulated price on the forecast interval, but in practice this is too computationally expensive to do. We performed instead a grid search over a range of plausible values for these two problematic parameters: rule lookback and relative importance of SSE. Figure 6.1 shows a “fitness landscape” for these two parameters. Each point on the surface represents the average SSE of forecast errors for 20 stationary stocks<sup>4</sup> over the forecast interval of 50 weekly observations from 2006, produced by allocating the value of the lookback parameter  $H$  to the range  $\{3, 4, \dots, 9\}$  and of the SSE weight parameter  $w$  to the range  $\{0.91, 0.92, \dots, 0.99\}$ .

All the other parameters were set to the values shown in Table 6.1. The SSE of forecasts is minimized by selecting  $H = 6$  (each rule looks back 6 periods in the past) and  $w = 93\%$  (weight SSE criterion in the objective function as 93% of the total). The graph has the shape of a bowl, but is quite noisy, a fact that is not surprising considering the stochastic nature of financial prices. As a matter of fact, the point we have selected is not the lowest on figure 6.1:  $H = 9$  and  $w = 92\%$  is. However, since this better point is in a corner, and it beats the selected point by only a small margin, it was rejected. Moreover, model evaluations for  $H = 9$  are twice as expensive in terms of time (19 hours) as for  $H = 6$  (9 hours). An important detail is that the 20 stocks selected for this

<sup>4</sup> These stocks are MMM, ABT, ADBE, AMD, AET, ACS, AFL, A, AA, ALTR, AMZN, AIG, AMGN, APC, ADI, BUD, APA, AAPL, AMAT and ADM.

graph were chosen at random from the stationary sample, and were not re-used in any future experiment, in order to eliminate any data snooping bias.

Optimizing parameter values using the fitness landscape is not the only way to proceed; another approach to this problem is to set up a nested evolution strategy, such that the top-level GA evolves the parameters of the lower-level GA. De Jong [24] cogently remarks that this leaves open the question of which parameters are to be selected for the top-level GA, opening the door to infinite regression. Nested evolution therefore does not solve the problem of parameter selection, but simply transfers to a higher level, where parameter choice is even less intuitive. Apart from the difficulty and computational cost of a nested scheme, we feel that due to the large amount of noise present in financial data, the parameter's "sweet spot" is probably going to be quite large.

In what follows, we will apply the fixed-length generalized trading rule version of our model for different experiments using the parameter values we just found.

## 6.5 Forecast Quality Evaluation

Our GA model, apart from its theoretical interest, ultimately results in a forecast. From a practical point of view, one may want to know whether it improves upon other standard methods. There has been a very wide variety of forecasting methods that have been tried, using varied angles of attack. For instance, studies have proposed using attractor reconstruction from chaos theory, or Fourier analysis, wavelet analysis, Hilbert transforms, artificial neural networks, genetic programming, kernels, evolutionary strategies and many other approaches, with various degrees of success. There is even a mythical folklore of forecasting; for instance, several popular books report that chaos theory has been applied to financial forecasting with great success, but the precise way to do so is of course not divulged [11].

We chose not to compare our method to any such complex models, since they are often ill-defined (because of the strong incentive to publish only the unsuccessful models and keep the successful ones under wraps for profitable use). Moreover, in many cases their implementation is non trivial, because of the fair amount of tuning required. Skeptics could always claim that we compare our method to a straw man. So we have chosen to compare our method to two very simple and accepted methods of financial forecasting: a linear trend based on the lognormal random walk model, and ARIMA forecasting.

### 6.5.1 Comparison to Two Benchmarks

According to the lognormal model of stock prices, the expected value of the stock at time  $T$  is  $S_0 \cdot e^{\mu T}$ , where  $\mu$  is the drift parameter, which can be estimated by linear regression. This forecast is of course only a baseline that should be easy to beat: the drift estimator is known to have an unreasonably high variance, and is therefore close to be useless for forecasting purposes.

The standard algorithm for the ARIMA model was proposed by Box and Jenkins in 1970 [2]. It is a very tough benchmark to beat, as it is the optimal linear forecast. Denoting  $Y^*$  as the stationary series, the general ARIMA is:



$$Y_t^* = \phi_1 Y_{t-1}^* + \phi_2 Y_{t-2}^* + \dots + \phi_p Y_{t-p}^* + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}$$

where the  $\theta_i$  and  $\phi_i$  are the parameters to be estimated (along with the correct lags  $p$  and  $q$ ), and  $\varepsilon_i$  is Gaussian white noise. Any time-series is therefore explained solely by its previous values.

Figures 6.2 and 6.3 display examples of weekly forecasts over one year for the Ashland, Inc. stock (ASH), and for the McCormick & Co. stock (MKC). The blue line represents the 2007 historical weekly data that none of the models have seen in their estimation procedure. Each model used only the 2006 data as a training period. Our model (“GA”) was trained with 500 000 generations.

We can see that in the case of ASH, the forecasts from ARIMA and the GA are comparable. However, the GA exhibits a trending behaviour for MKC. Over the 200 stocks in the sample, ARIMA is never far from the general data trend to the point that it seems to “guess” the future quite well, even for a forecast period of one full year. However, the ARIMA forecast is always a bit lagging with respect to actual data. The GA model is sometimes good, but most of the time it is far off the mark, as for MKC. It tends to exhibit explosive or cyclical behaviour often. Table 6.2 confirms this impression: for all the stocks, the RMSE (root mean square errors) and the MAD (mean absolute deviations) of the forecasts are higher for the GA model.

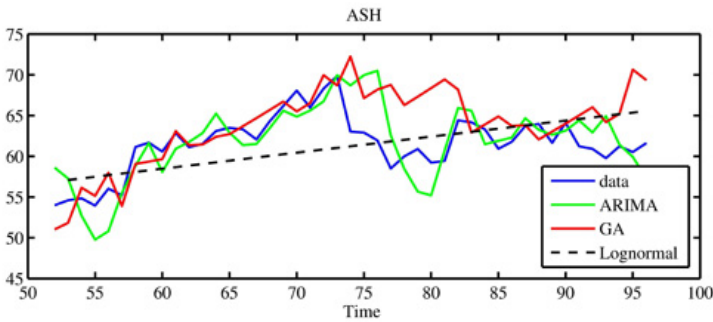


Fig. 6.2. Forecast comparison for Ashland, Inc. (2007)

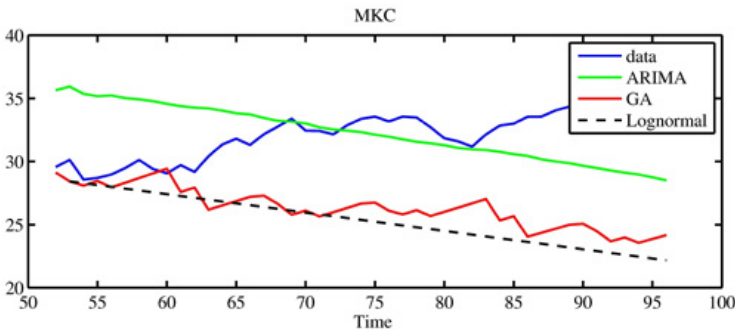


Fig. 6.3. Forecast comparison for McCormick & Co. (2007)

**Table 6.2.** Average RMSE and MAD of forecasts for all 200 stocks, year 2007

Model	Average RMSE over 200 stocks	Average MAD over 200 stocks
lognormal	50.54	6.40
ARIMA	58.61	3.71
GA	64.08	7.58

**Table 6.3.** Average RMSE and MAD of forecasts for all stocks and noncyclical subset

Model	Average RMSE over 200 stocks	Average RMSE over noncyclical subset
lognormal	50.54	45.18
ARIMA	58.61	57.41
GA	64.08	39.22
Model	Average MAD over 200 stocks	Average MAD over noncyclical subset
lognormal	6.40	5.62
ARIMA	3.71	3.63
GA	7.58	4.58

The fact that the GA model often shows cyclical behaviour opens the door to an interesting question: what would be the performance of the GA model if we filtered out the cyclical forecasts? This can be done by a user of the system in a automatic objective manner by applying the Box-Pierce-Ljung “Q-statistic” [11]. Looking at the first 13 lags, we retain only the GA forecasts that exhibit no cyclical behaviour: any forecast with a Q-statistic rejected for any lag at the 1% level is excluded. This has the disadvantage of reducing drastically the number of usable forecasts; we find only 15 noncyclical forecasts in the original set of 200.

We can now revisit the previous table by comparing models again on that noncyclical subset. Table 6.3 displays the results.

When the cyclical forecasts are removed, the GA model *performs better* than ARIMA according to the RMSE criterion, but worse than ARIMA by the MAD criterion. The lognormal model is the worst performer according to both criteria over that restricted subset. In all fairness, even if the models were ranked identically by both closeness criteria, this experiment would not be conclusive statistically, since the number of stocks involved in the restricted subset is *too small*. We can however take note of an interesting fact: the GA fitness function minimizes the MSE, and not the MAD. It is therefore encouraging to see that the GA model beats ARIMA with respect to that criterion, which after all is the one that we explicitly targeted.

We make one last point concerning RMSE and MAD. It could be argued that financial practitioners would be *a priori* more interested in models that are heavily penalized when they make large errors, which is what the RMSE criterion does. The reason is that large forecast deviations are likely to cause catastrophic financial losses, and any model that provides a closer fit for these large deviations is more useful than a model that fits well on average, but does not capture exceptional movements. This argument would tend to confirm that the minimization criterion should indeed be the RMSE, and that the GA would indeed be more useful than ARIMA.

**Table 6.4.** Number of noncyclical forecasts by number of generations

Number of generations	Noncyclical forecasts over 200 stocks
50 000	15
100 000	26
200 000	20
300 000	14
400 000	19
500 000	15

**Table 6.5.** Average RMSE and MAD of forecasts by number of generations

Number of generations	Average RMSE			Average MAD		
	GA	lognormal	ARIMA	GA	lognormal	ARIMA
50 000	45.03	58.64	62.79	4.99	7.09	4.57
100 000	74.67	81.35	74.07	8.91	10.26	5.24
200 000	51.57	48.95	60.36	5.91	6.05	4.02
300 000	51.20	37.75	52.35	6.26	4.71	3.38
400 000	42.48	46.45	58.50	4.83	5.63	3.77
500 000	39.21	45.18	57.41	4.58	5.62	3.63

What causes cyclical forecasts to appear in the GA model? A possible explanation lies in the fact that we selected individuals through a fitness function that encouraged parsimonious representations. Obtaining cycles is a natural consequence of that selection pressure since cycles are about the most economical way to represent any given data, as in Fourier decompositions. But cycles do not do well out-of-sample, so they defeat the broader purpose of our model, which is forecasting. This phenomenon is typical of genetic algorithms. The specification of the fitness function can have consequences unintended by the modeller. Evolution produces individuals exactly in accordance with the kind of selection pressure applied.

We can check whether parsimony pressure is responsible for cyclical behaviour by examining how many noncyclical forecasts remain when we evolve our model for a smaller number of generations, on the same stocks. Table 6.4 shows the number of remaining noncyclical forecasts for various number of generations. Table 11.4 reveals no clear pattern. It seems that parsimony may not be responsible and that cyclicity may be a feature of the model.

It may be interesting to see the average RMSE and MAD of the forecasts produced when we evolve the algorithm for a lesser number of generations compare to those obtained when the model is fully evolved (Table 6.5). Notice that in this table, the lognormal and ARIMA models do not give always the same RMSE, since they are evaluated not on all samples, but only those determined by the Q-statistic for the GA forecast.

Again, the results of Table 6.5 send a mixed message. GA usually outperforms both models by the RMSE criterion except for the 300 000 generations noncyclical sample. ARIMA is always better by the MAD criterion. We conclude that it is probably

better to let evolution follow its course, even if that means that only a small percentage of forecasts are going to be noncyclical; at any rate, there is no clear advantage of evolving less.

In practice, a end-user of the GA model with enough patience can always obtain a noncyclical forecast: evolution can indeed be re-initiated from another random seed until a noncyclical forecast is obtained. If we model this process as a geometric random variable with the probability of success being 15/200, we would need on average 13 trials before getting a noncyclical forecast, so about 120 hours of continuous processing time on a 2.2 GHz machine.

### 6.5.2 An Encompassing-in-Forecast Test

Ordering forecasts on the basis of RMSE or MAD is standard in the literature, and it is why we have provided these statistics in the previous section. However, an important shortcoming of this procedure lies in the fact that we cannot tell what constitutes a significant difference between RMSE or MAD scores. Suppose, for instance, that the RMSE of model A is 2% higher than the RMSE of model B. Does that constitute sufficient reason to prefer model A over model B? We do not know, since the statistical distribution of RMSE and MAD scores is unknown. A remedy to this situation would be to get bootstrap estimates of these scores and test significance on that basis. Since this is slightly controversial, another avenue is to use encompassing-in-forecast tests.

Encompassing-in-forecast tests [3, 9] provide a way of deciding whether forecasts produced by one model provide significantly more information than forecasts from another model. To give an example, if the forecast errors of model A are explained by the model B forecasts, then this is evidence that model A does not add anything new to our knowledge. But in order to be certain that model B is better than model A, one has to examine the converse, namely whether the forecast errors of model B are *not* explained by the model A forecasts. If that last proposition is not true, the test is inconclusive as we might have a case of forecast multicollinearity. The various combinations that can occur are:

1. forecast errors of model A are explained by model B forecasts AND forecast errors of model B are unexplained by model A forecasts  $\implies$  then model B encompasses-in-forecast model A.
2. forecast errors of model B are explained by model A forecasts AND forecast errors of model A are unexplained by model B forecasts  $\implies$  then model A encompasses-in-forecast model B.
3. forecast errors of model A are explained by model B forecasts AND forecast errors of model B are explained by model A forecasts  $\implies$  then the test is inconclusive.
4. forecast errors of model A are unexplained by model B forecasts AND forecast errors of model B are unexplained by model A forecasts  $\implies$  then the test is inconclusive.

To test whether the forecast errors of a model are explained or not by another, we perform the following ordinary least-squares regression:

$$\widehat{Y}_{A_t} - Y_t = \beta_0 + \beta_1 \widehat{Y}_{B_t} + \varepsilon_t$$

**Table 6.6.** p-values of encompassing-in-forecast tests for the three models

Regressor ↓	Regressand		
	GA	ARIMA	Lognormal
GA	–	0.0356	0.0607
ARIMA	0.2761	–	0.1714
Lognormal	0.1047	0.0141	–

where  $\widehat{Y}_{A_t}$  refers to the forecast of model A at time  $t$  on the forecast interval and  $Y_t$  to the realized value. If the coefficient  $\beta_1$  is significantly different from zero, we conclude that forecast errors of model A are explained by model B forecasts. Table 6.6 presents the p-values of the t-statistics of the regression coefficient  $\beta_1$  for all possible combinations of regressor and regressand<sup>5</sup> permitted by our three models.

At the 5% significance level, ARIMA is explained by GA, but GA is not explained by ARIMA. We conclude that GA encompasses-in-forecast ARIMA, and this is good news for our model.

## 6.6 Is the Market Reconstruction Unique?

Since the method we propose creates a demand function that is consistent with the prices in the sample, we could wonder whether this reconstruction is unique. A worrying fact is the so-called “curse of dimensionality”: specifying (or reconstructing) a demand function requires orders of magnitudes more information than what is available in prices, so the process should logically fail.

To illustrate, we attempt to reconstruct a known market. We take an arbitrary market genome with 50 fixed-length generalized trading rules that we call the “target” market, that we use to forward-simulate 50 weekly stock observations from arbitrary initial conditions. We then apply our method on these prices in the hope of reconstructing the target market.

The main difficulty of this experiment is that deciding whether one set of rules is close to another; we need to define a measure of similarity between market genomes, but an easy solution such as computing the Levenshtein distance between alleles in corresponding positions is not suitable, for the following reasons:

- rules are in no specific order in individuals, so we can neither compare genes (i.e. rules) nor alleles based on their position within the individual.
- even though rules are of the fixed-length type, the individuals themselves are of variable length, with a maximum set to 70. We know as experiment designers that our target individual has length 50, but this is not information that is made available to the GA model. The reconstructed individual can therefore end up having any length. So we need a similarity measure that will still work if presented with individuals of differing lengths.

---

<sup>5</sup> In a regression, the regressand is the variable we attempt to explain (to the left of the equality sign), and the regressor(s) is (are) the explanatory variables (to the right of the equality sign).

- recalling that “don’t care” conditions are possible, a rule present in one chromosome may be expressed as a *set* of rules in another. It is also possible that a rule or a rule set cancel another rule in a given individual, if the respective rule activation conditions are identical and if one rule is a “buy” while the other is a “sell”.

In light of these problems, we suggest comparing markets not structurally, but functionally. Recall that a market “demand” means the sum of individual rule demands when the market is presented with a particular market run, which is a stack of  $H$  discretized historical prices. Using octiles, we denote this stack by  $\{1, 2, \dots, 8\}^H$ . Formally, a demand function is a mapping:

$$D : \{1, 2, \dots, 8\}^H \mapsto \mathbb{R}$$

The set of  $H$ -tuples  $\{1, 2, \dots, 8\}^H$  constitutes a  $H$ -dimensional lattice over the integers 1 to 8. Recall that the market-maker is presented with the demand for each period, and applies its own response function, which is another mapping:

$$f : \mathbb{R} \mapsto \mathbb{R}$$

and that this mapping produces a new price which is discretized based on an empirical price distribution:

$$g : \mathbb{R} \mapsto \{1, 2, \dots, 8\}$$

and pushed onto the current price stack, which is another function:

$$h : \{1, 2, \dots, 8\}^H \mapsto \{1, 2, \dots, 8\}^H.$$

With given initial conditions, the repeated application of these four functions creates an orbit of a dynamical system. A market orbit will therefore repeat after a certain time, since the lattice is finite in size. This explains why prices often exhibit cyclical behaviour. As a matter of fact, they *always* exhibit cyclical behaviour, and all we did previously was to exclude cases where the period is *too short*. The largest possible period is obviously  $Q^H$ , where  $Q$  is the number of quantiles used for discretization, but in practice the average period is much shorter. Forecast cyclicity is therefore an unavoidable property of the model, and the frequency of cyclical behaviour should be inversely proportional to the value of the lookback parameter  $H$ .

Recalling that the optimized lookback parameter was found to be  $H = 6$ , we now describe a graphical representation of a demand function on the  $\{1, 2, \dots, 8\}^6$  lattice to  $\mathbb{R}$ . Since there is no convenient way to represent such a high-dimensional mapping, we project each point of the lattice to a two-dimensional square array indexed by a row and a column number:

```

i = 0;
For (a = 1 ... 8)
  For (b = 1 ... 8)
    For (c = 1 ... 8)
      For (d = 1 ... 8)
        For (e = 1 ... 8)
          For (f = 1 ... 8) {

```

```

array[floor(i/8^3), i mod (8^3)] =
demand(a,b,c,d,e,f)
i <- i + 1
}

```

This projection can be displayed in color as a “market tapestry”, using a colormap that maps net “sell” states to blue and net “buy” values to red, and intermediate values to a continuum of colors. All demand values have been scaled to between zero and one. Obviously, a projection from a high-dimensional space to a square destroys the proximity relation: proximity of two pixels in the image does not mean proximity in the lattice. This unfortunately cannot be avoided, since we are compressing six dimensions into two. However, we get pictures that can be compared visually; the human brain is a good judge of visual similarity. Independently of the visual observer, reconstructed markets can be objectively compared by computing the Pearson correlation coefficient between the pixel intensities of the two images.

Figure 6.4 displays the target market; figure 6.5 shows the market obtained with our method, after evolving one run for 500 000 generations. The colormap appears to the right of each figure. It is clearly visible that there is little similarity between the two demand functions for that run, although both produce about the same prices using the same initial conditions for a simulation of length 50. The Pearson correlation between these two markets is  $-0.07$ , which shows that this resulting market cannot be said to be a faithful reconstruction of the target market.

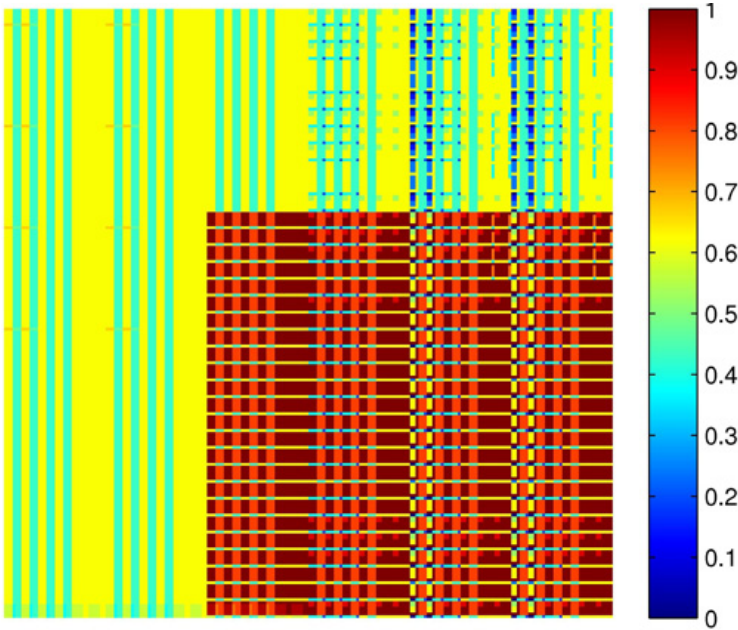


Fig. 6.4. Market tapestry of target demand function

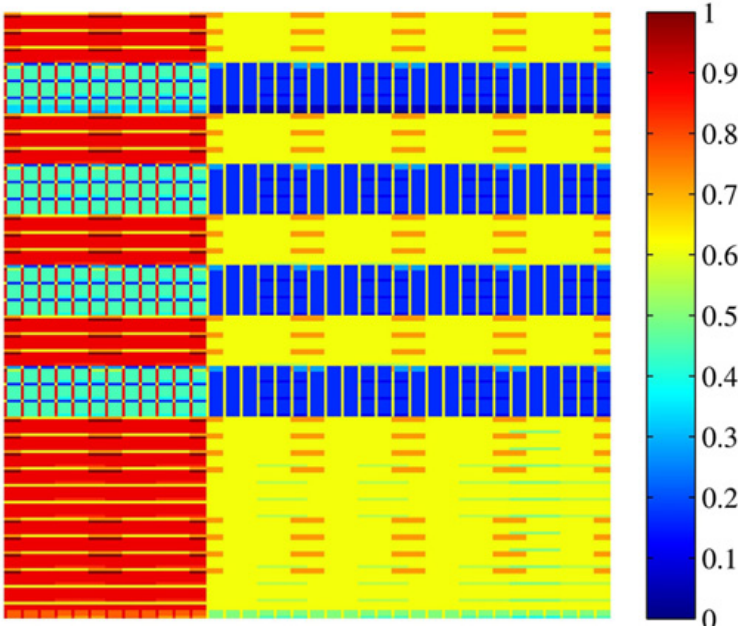


Fig. 6.5. Resulting market tapestry after 500 000 generations

Table 6.7. Average correlation by number of generations

Number of generations	Restricted demand $\bar{\rho}$ with target (100 runs)	$\widehat{\sigma}_{\rho}$
50 000	10.74%	7.38%
100 000	9.83%	6.64%
200 000	12.34%	6.52%
300 000	19.42%	5.39%
400 000	22.50%	6.27%
500 000	21.45%	5.14%

Recall that the model is applied to a training sample with length 50, namely one year of weekly observations. On the other hand, the graphical representation of the demand function shows demands for all possible inputs, a set of  $8^6 = 262144$  different price stacks. There is huge discrepancy in size between the full range of behaviours that can be inferred from the genome and the restricted part of that range acted upon by evolution, which represents only a fraction of  $\frac{19}{100000}$  of it. We may still examine whether evolution is effective on the part of the genome that is restricted to the historical set of H-tuples  $\{1, 2, \dots, 8\}^H$  that were encountered in the training sample. Since there were 50 of them, we remove the first to get arrays of  $7^2$  pixels. Table 6.7 displays the average correlations for 100 runs, if we restrict demand for only the actual market conditions that were encountered historically.



We can conclude from Table 6.7 that although the goal of evolution is not to reproduce demand functions, but prices, reproducing prices will create a certain similarity between the demand functions inferred. Because of the curse of dimensionality, this similarity will not raise beyond a certain point.

## 6.7 Conclusions and Future Work

We have presented a model of an artificial stock market that is simple enough as to allow calibration to a time-series of market prices, by using evolution. To our knowledge, this has been attempted in the literature only once, and using a very different artificial market model [6]. The main drawback of this method is a rather prohibitive computational cost; instances of the model had to be run in parallel on a cluster machine.

While the model cannot be said to faithfully reconstruct the underlying market structure because of the curse of dimensionality, we have presented empirical evidence that forward-simulation provides a better forecast, over one year, compared to the lognormal model and ARIMA, if one uses the financially-relevant RMSE criterion. However, when cyclical forecasts are *not* removed from consideration, our model forecasting performance is clearly dominated by both ARIMA and the lognormal model. Cyclical behaviour is a natural consequence of the price discretization that is used to simplify rule representation within genomes.

Useful noncyclical forecasts are obtained in 1 case out of 10 on average, which limits the practical use of the model. However, this proportion can possibly be raised, by increasing the lookback parameter  $H$  to a higher value, which however increases the computational burden.

## References

1. Bass, T.: *The Predictors: How a Band of Maverick Physicists Used Chaos Theory to Trade Their Way to a Fortune on Wall Street*, Holt, USA (2000)
2. Box, G.E.P., Jenkins, G.M.: *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco (1970)
3. Chong, Y.Y., Hendry, D.F.: *Econometric Evaluation of Linear Macroeconomic Models*. *Review of Economic Studies* 53, 671–690 (1986)
4. De Jong, K.: *Parameter Setting in EA's: A 30 Years Perspective*. In: Lobo, F., et al. (eds.) *Parameter Setting in Evolutionary Algorithms*. *Studies in Computational Intelligence*, vol. (54). Springer, Heidelberg (2007)
5. Dickey, D., Said, E.: *Testing for Unit Roots in Autoregressive Moving Average Models of Unknown Order*. *Biometrika* 71, 599–607 (1984)
6. Ecemis, I., Bonabeau, E., Ashburn, T.: *Interactive Estimation of Agent-Based Financial Market Models: Modularity and Learning*. In: *Proceedings of GECCO 2005*, Washington, DC, USA, June 25-29. ACM Press, New York (2005)
7. Goldberg, D.: *The Design of Innovation: Lessons From and For Competent Genetic Algorithms*. Kluwer, Boston (2002)
8. Greene, W.H.: *Econometric Analysis*. Prentice-Hall, New Jersey (2003)

9. Harrald, P.G., Kamstra, M.: Evolving Artificial Neural Networks to Combine Financial Forecasts. *IEEE Transactions on Evolutionary Computation* 1(1), 40–52 (1997)
10. Hasbrouck, J.: *Empirical Market Microstructure*. Oxford Press, Oxford (2007)
11. Ljung, G.M., Box, G.E.P.: On a Measure of Lack of Fit in Time-Series Models. *Biometrika* 65, 297–303 (1978)
12. O'Hara, M.: *Market Microstructure Theory*. Blackwell, Oxford (1995)
13. Sharpe, W.F.: Capital Asset Prices: A Theory of Market Equilibrium Under Conditions of Risk. *Journal of Finance* 19(3), 425–442 (1964)

**Dynamic Strategies and Algorithmic Trading**

# Index Mutual Fund Replication

Jin Zhang<sup>1</sup> and Dietmar Maringer<sup>2</sup>

<sup>1</sup> Centre for Computational Finance and Economic Agents, University of Essex,  
United Kingdom  
jzhangf@essex.ac.uk

<sup>2</sup> Business and Economics Faculty, University of Basel, Switzerland  
dietmar.maringer@unibas.ch

**Summary.** This chapter discusses the application of an index tracking technique to mutual fund replication problems. By using a tracking error (TE) minimization method and two tactical rebalancing strategies (i.e. the calendar based strategy and the tolerance triggered strategy), a multi-period fund tracking model is developed that replicates S&P 500 mutual fund returns. The impact of excess returns and loss aversion on overall tracking performance is also discussed in two extended cases of the original TE optimization. An evolutionary method, Differential Evolution, is used for optimizing the asset weights. According to the experiment results, it is found that the proposed model replicates the first two moments of the fund returns by using only five equities. The TE optimization strategy under loss aversion with tolerance triggered rebalancing dominates other combinations studied with regard to tracking ability and cost efficiency.

## 7.1 Introduction

In the last decade, individual holdings of corporate stocks have decreased while holdings through fund management institutions have correspondingly increased. According to the Investment Company Institute's official survey, the combined assets of U.S. mutual funds reached a peak of 12 trillion dollars in May 2008; although there was a great redemption pressure on the fund industry due to the recent credit crunch, the net asset value of the funds was in excess of 9 trillion dollars at the end of 2008. As the survey shows, approximately half of the fund holdings were claimed and managed by equity funds. The latter typically choose from three management styles, namely active management, passive management, or a blend of the two. The literature shows that most of actively managed equity funds underperform their passive benchmark portfolios after adjustments are made for fund management fees and expenses. For example, actively managed funds usually do not outperform index mutual funds which are passively managed to mimic certain indices in the long-term (see [10], [6], [5] and [9]). According to the official survey, the number of U.S. index funds increased almost three-fold from 134 to 373, in which the number of S&P 500 index funds rose from 72 to 124 at a growth rate of over 70% in the last decade. And more recently, the institutional investment in index funds increased dramatically after the bankruptcy of major investment banks on Wall Street in September 2008.

While investing in funds brings several advantages over direct investments in equities, expenses such as management fees or distribution fees have continued to increase

during the last two decades. The question of whether charging higher fund fees benefits investors has been discussed at great length (see [3]). Furthermore, researchers have drawn attention to a confusing phenomenon in the fund market: while fund fees and expenses vary quite a lot, the return patterns of the funds typically show relatively small amounts of dispersion. Therefore, it is not necessary to use expensive funds if the performance of funds are similar. As [17] pointed out, funds with lower expenses tend to have higher reward-to-risk ratios; and investors should avoid using expensive funds since the high fund fees reduce the overall payout.

This paper extends index tracking techniques to perform index fund return replication. From the literature, several index tracking methods have been discussed by researchers: the classic tracking error (TE) minimization method of [16]; the principal components factor model and the cointegration based index tracking method in [1, 2]. The classic index TE minimization model has been widely studied by researchers. [14] used quadratic programming to construct equity index funds by minimizing tracking errors between index returns and asset returns which were generated from the autoregressive conditional heteroscedastic (ARCH) process. [15] applied an optimized sampling method in order to select equities to minimize tracking errors. As an alternative to traditional numerical methods, heuristic methods provide ways of approaching difficult combinatorial optimization problems, especially of solving financial optimization problems. An interesting feature of heuristic methods is that they combine stochastic search with supervised search; this can provide investors with new and efficient methods to obtain good solutions for complex and constrained optimization problems. Traditional deterministic optimization approaches tend to be poorly served for these problems whose solution space becomes rough and discontinuous after imposing different types of constraints on optimization problems. There exists some literature on the use of heuristic methods for index tracking problems. [8], e.g., adopt the Threshold Accepting algorithm, a trajectory search method, while [4] and [12] use evolutionary approaches. A classification of index tracking problems as well as a survey of applications of heuristics methods to this type of problems can be found in [19].

There are three main reasons to study the fund replication problem by using index tracking methods. First, although index mutual funds are usually considered to be cheap, they tend to outperform most of the actively managed funds in the long-term according to the literature. Secondly, the application of index tracking techniques in index mutual fund replications has not been widely discussed in the literature yet; this chapter therefore discusses such applications in order to address this issue. Thirdly, if the index fund returns can be replicated, the dispersion in respect of the fund fees and expenses should not be great. Several important issues which are not usually included in the index tracking discussion are addressed in this study. For example, the proposed multi-period tracking model involves the tactical rebalancing issue, the transaction cost limitation and the cash reservation issue. Furthermore, a cardinality constraint is imposed that limits the number of different equities included in the portfolio to lower managing and survey costs.

The remainder of this chapter is organized as follows: Section 7.2 introduces the fund tracking model and a population based heuristic method for tackling the

multi-period optimization problem; Section 7.3 provides the results and discussions from the in-sample and out-of-sample experiments; and Section 7.4 concludes.

## 7.2 Tracking Error Minimization and Multi-period Readjustment

### 7.2.1 The Optimization Problem for Tracker Construction

The optimal solution for the fund tracking problem is a portfolio that exhibits properties as similar to those of the tracked fund as possible. Primarily, this means that both the original fund and the tracker ought to have similar return patterns. From a management point of view, these similarities might also affect the frequency of portfolio revisions. If an index fund adopts a passive management, its composition should rarely change; in consequence, this should also apply to the tracker. However, since the tracker funds are not perfect replications of the original funds, and are potentially subject to certain restrictions, the suitability of the this selection might change over time and the tracker requires revision. Hence, the initial construction of a tracker fund is just one part of the optimization problem while the second part is the optimal readjustment of asset weights and, where necessary, change of included equities. One can therefore distinguish an (initial) construction stage and an (ongoing) adjustment stage.

At the construction stage, an investor observes the daily prices of  $N$  equities, as well as the daily net asset values (NAV) of target funds over historical time  $[T_{\varpi}, T_0]$ . Let  $T_{\varpi}$  and  $T_0$  denote the first day and the end of the construction stage respectively. At time  $T_0$ , an optimal set of  $k$  equities ( $k < N$ ) and holding quantities  $n_{i,T_0}$  need to be known in order to construct a tracker which best replicates its target over the period  $[T_{\varpi}, T_0]$ . To measure the similarity of tracker returns and fund returns, a difference measure between the returns is adopted, i.e. the tracking error  $TE = \sqrt{\frac{1}{T} \sum_t (r_{P,t} - r_{I,t})^2}$ . The market value of the tracker at time  $t$  is the sum of the market value of holdings,  $P_t = \sum_{i=1}^N n_{i,t} \cdot S_{i,t}$ . Since the short selling of equities is not considered in this study,  $n_{i,t}$  must be positive integers. Let  $S_{i,t}$  denote the market price of equity  $i$  at time  $t$ , and the tracker daily return at time  $t$  is defined as  $r_{P,t} = \ln(P_t/P_{t-1})$ . The NAV reflects the dollar value of one share of a fund, which is used to compute the fund return at time  $t$ :  $r_{I,t} = \ln(NAV_t/NAV_{t-1})$ .

The tracker portfolio is constructed with three constraints. Suppose that an initial budget  $B_{T_0}$  is available at  $T_0$ , and an amount of  $P_{T_0} = \sum_{i=1}^N n_{i,T_0} \cdot S_{i,T_0} \leq B_{T_0} \cdot (1 - C)$  is used to purchase a set of non-negative and integer quantities  $n_{i,T_0}$  of equities from the market, where  $C$  is the initial cash reserve rate. If an equity has a price  $S_{i,T_0}$  at time  $T_0$ , the weight invested can be written as  $x_{i,T_0} = (n_{i,T_0} \cdot S_{i,T_0})/P_{T_0}$ . For the purpose of portfolio diversification, if an equity  $i$  is included, its corresponding weight should satisfy two weight constraints,  $x_g^l \leq x_{i,T_0} \leq x_g^u$ , otherwise its holding quantity  $n_{i,T_0}$  should be zero. After taking out the value  $P_{T_0}$  from the initial budget  $B_{T_0}$ , one has the initial cash reserve:  $Cash_{T_0} = B_{T_0} - P_{T_0}$ , which is used to cover transaction costs. Secondly, the tracker can only use a subset  $k$  out of the market equities  $N$  to track funds. Therefore, a cardinality constraint  $\#C_g = \sum_{i=1}^N I_{C_g}(i) \leq k$  is imposed to control the number of equities purchased.  $C_g$  is the equity set;  $k$  the number of equities purchased; and  $I_{C_g}(i)$  is an indicator function. It should be noted that introducing lower and upper limits also incurs implicit cardinality constraints: if the weight of each equity must be kept below a weight

$x_g^u$ , at least  $k^{\min} = \lfloor 1/x_g^u \rfloor$  equities must be bought; on the other hand, the maximum number of equities which should be purchased to satisfy the minimum weight constraint is  $k^{\max} = \lceil 1/x_g^l \rceil$ . Finally, the tracker has an upper limit on the costs of each transaction: the costs cannot grow beyond the amount which is a small proportion  $\gamma$  of the tracker value  $P_t$ , i.e.  $TC_t \leq \gamma \cdot P_t$ . The transaction costs  $TC$  are set as linear functions of the amount for buying the equities. Thus, the costs can be modelled as  $TC_{T_0} = \sum_{i \in C_g} \rho \cdot n_{i,T_0} \cdot S_{i,T_0}$ , where  $\rho$  is defined as a transaction cost coefficient. Although the current work uses only linear transaction cost functions, the model can handle also transaction cost functions when the solution spaces are neither continuous nor convex.

For ease of reading, the TE optimization problem is summarized using the following notation.

$$\min_{\mathbf{n}} TE = \sqrt{\frac{\sum_t (r_{P,t} - r_{I,t})^2}{T_0 - T_w}}$$

s.t.

$$\begin{aligned} n_{i,T_0} &\in \mathbb{N}_0^+ \\ t &\in [T_w, T_0] \\ k^{\min} &< \#C_g = \sum_{i=1}^N I_{C_g}(i) \leq k < k^{\max} \\ x_g^l &\leq \frac{n_{i,T_0} \cdot S_{i,T_0}}{P_{T_0}} \leq x_g^u \quad \text{for } i \in C_g \\ TC_{T_0} &= \sum_{i \in C_g} \rho \cdot n_{i,T_0} \cdot S_{i,T_0} \leq \gamma \cdot P_{T_0} \end{aligned}$$

where

- $n_{i,T_0}$  number of shares of the  $i$ -th equity invested at time  $T_0$
- $\gamma$  the transaction cost limiting ratio
- $P_t$  market value of the tracker at time  $t$
- $r_{P,t}$  tracker return at time  $t$
- $r_{I,t}$  index fund return at time  $t$
- $C_g$  tracker equity set
- $x_g^l$  minimum weight of each equity
- $x_g^u$  maximum weight of each equity
- $TC_t$  transaction cost at time  $t$
- $\rho$  transaction cost coefficient
- $Cash_t$  cash reserve at time  $t$
- $C$  cash reserve rate
- $B_t$  sum of the tracker market value and cash reserve at time  $t$
- $S_{i,t}$  per-share market value of the  $i$ -th equity at time  $t$
- $N$  number of available equities in the equity market

While the market moves over time, the tracker holdings should be revised if the tracker return drifts away from its target return. The following subsection introduces the rebalancing problem and two rebalancing strategies.

### 7.2.2 Tracker Rebalancing Stage

At rebalancing time  $T_j$ , an optimal adjustment set  $\delta(n_{i,T_j})$  of  $k$  equities should be known. After rebalancing, the replicator should best track its target over the period  $[T_{j-1}, T_j]$ . The decision variables are a set of adjusted quantities  $\delta(n_{i,t})$ , which can be either positive or negative integers. The holding of equity  $i$  after the rebalancing at time  $t = T_j$  can be written as:  $n_{i,T_j} = n_{i,T_{j-1}} + \delta(n_{i,T_j})$ . As the rebalancing involves both selling and buying, the transaction cost is modelled as twice the cost at the construction stage:  $TC_{T_j} = \sum_{i \in C_g} 2 \cdot \rho \cdot |n_{i,T_j} - n_{i,T_{j-1}}| \cdot S_{i,T_j}$ . If the cash reserve is not enough to cover transaction costs, the model will recover the cash reserve by liquidating assets, which depends on the reserve rate  $C$ , the tracker value  $P_t$ , and the optimal holdings of the next period. At the rebalancing stage, all the constraints from the construction stage must be satisfied. Thus the optimization problem at this stage is summarized as follows.

$$\min_{\delta(\mathbf{n})} TE = \sqrt{\frac{\sum_t (r_{P,t} - r_{I,t})^2}{T_j - T_{j-1}}}$$

s.t.

$$\begin{aligned} \delta(n_{i,T_j}) &\in \mathbb{Z} \\ n_{i,T_j} &\in \mathbb{N}_0^+ \\ t &\in [T_{j-1}, T_j] \\ k^{\min} &< \#C_g = \sum_{i=1}^N I_{C_g}(i) \leq k < k^{\max} \\ x_g^\ell &\leq \frac{(n_{i,T_{j-1}} + \delta(n_{i,T_j})) \cdot S_{i,T_j}}{P_{T_j}} \leq x_g^u \quad \text{for } i \in C_g \\ TC_{T_j} &= \sum_{i \in C_g} 2 \cdot \rho \cdot |n_{i,T_j} - n_{i,T_{j-1}}| \cdot S_{i,T_j} \leq \gamma \cdot P_{T_j} \end{aligned}$$

### 7.2.3 Rebalancing Strategies

Two portfolio rebalancing strategies are usually adopted by market practitioners (see [7]). One is portfolio readjustment at regular calendar intervals (e.g. quarterly), which is referred to as calendar based rebalancing. The other is a tolerance triggered strategy which initiates readjustments when certain thresholds or limits are exceeded.

#### Calendar Based Rebalancing

This strategy schedules rebalancing at a regular calendar interval  $T_\psi$ .



1. The model splits the future time horizon  $[T_0, T_\omega]$  into  $M$  subintervals  $[T_0, T_1]$ ,  $[T_1, T_2]$ ,  $\dots$ ,  $[T_{M-1}, T_\omega]$  according to a fixed calendar interval  $T_\psi$ . The interval number  $M$  is decided by the length of the rebalancing stage and the time interval:  $M = \lfloor (T_\omega - T_0)/T_\psi \rfloor$ .
2. At the rebalancing time  $T_j$ , the model
  - a) decides an optimal set of quantities  $\delta(n_{i,T_j})$  based on the market information over the time period  $[T_{j-1}, T_j]$ ,
  - b) adjusts portfolio holdings  $n_{i,T_j} = n_{i,T_{j-1}} + \delta(n_{i,T_j})$ ,
  - c) updates cash reserves  $Cash_{T_j} = Cash_{T_{j-1}} - TC_{T_j}$ , and
  - d) waits till the next planned rebalancing point  $T_{j+1} = T_j + T_\psi$ .
3. The model repeats the second step until the end of the rebalancing stage  $T_\omega$ .

### Tolerance Triggered Rebalancing


The second strategy checks whether the properties of the original and the tracker funds are similar enough or have drifted too far apart. If a certain tolerance level has been exceeded and the differences are too big, the system triggers rebalancing activities. Also, due to price changes, the portfolio weights of individual equities might have changed so much that they now exceed the upper or lower limits. In this case, a readjustment of the portfolio is necessary to ensure that the solution is valid again.

Practically speaking, for the rebalancing stage  $[T_0, T_\omega]$ ,  $M = \lfloor (T_\omega - T_0)/\varphi \rfloor$  check-points are introduced that are equidistant with an interval length of  $\varphi$  days. At each of these check-points  $T_j$  with  $j = 1..M$ , the system evaluates whether the asset weights at time  $T_j$  are still within  $[x_g^\ell, x_g^u]$  or whether the tracking error over a recent period of time  $[T_j - W_L, T_j]$  exceeds the prespecified tolerance level of  $\xi_1$ . The rebalancing procedure is described as follows.

1. At each check-point  $T_j$ , the tracker has the starting point of the  $j$ -th window,  $T_{s,j} = T_j - W_L$  with  $j = 1, 2, \dots, M$ .
2. a) If any one of the following conditions is violated:
 
$$\sqrt{\frac{1}{W_L} \cdot \sum_{t=T_{s,j}}^{T_j} |r_{P,t} - r_{I,t}|^2} < \xi_1, \frac{n_{i,T_j} \cdot S_{i,T_j}}{P_{T_j}} > x_g^\ell, \text{ and } \frac{n_{i,T_j} \cdot S_{i,T_j}}{P_{T_j}} < x_g^u,$$
 the model
  - (i) finds an optimal set of  $\delta(n_{i,T_j})$  based on the market information in the time period  $[T_{s,j}, T_j]$ ,
  - (ii) adjusts portfolio holdings:  $n_{i,T_j} = n_{i,T_{j-1}} + \delta(n_{i,T_j})$ ,
  - (iii) updates cash reserves:  $Cash_{T_j} = Cash_{T_{j-1}} - TC_{T_j}$ ;
- b) otherwise the model keeps the holdings unchanged:  $n_{i,T_j} = n_{i,T_{j-1}}$ ;
- c) the model waits till the next check-point  $T_{j+1} = T_j + \varphi$ .
3. The model repeats the second step up till the end of rebalancing stage  $T_\omega$ .

## 7.2.4 Extensions of Traditional Tracking Error Optimization

### Extension to Include Excess Return

 considered the following average positive deviations from market benchmarks, or the average excess return (ER):

$$ER = \frac{1}{T_N} \sum_t (r_{P,t} - r_{I,t}), \text{ for } r_{P,t} \geq r_{I,t}$$

as a part of the index tracking objective, where  $T_N$  represents the number of returns observed over the period. The model considers index fund return as the benchmark, and extends the traditional TE optimization objective to

$$\min(\lambda \cdot TE - (1 - \lambda) \cdot ER).$$

$\lambda$  is a value between 0 and 1, representing the trade-off between TE and ER.

### Extension to Include Loss Aversion

The classic TE minimization objective does not distinguish between positive and negative deviations of the tracker relative to its target, due to ignorance of the sign of return deviations. Loss averse investors tend to strongly prefer avoiding losses to acquiring gains, therefore the behaviour can be modelled by introducing an aversion coefficient  $\vartheta$  to the TE measure with  $\vartheta > 1$  (see [12]).

$$\tilde{\Delta}_r = \begin{cases} r_{P,t} - r_{I,t} & r_{P,t} \geq r_{I,t} \\ (r_{P,t} - r_{I,t}) \cdot \vartheta & r_{P,t} < r_{I,t} \end{cases}.$$

Thus the original objective function at the construction and rebalancing stage is modified to:

$$\min \widetilde{TE} = \sqrt{\frac{1}{T_N} \sum_t (\tilde{\Delta}_r)^2}.$$

### 7.2.5 The Optimization Method

Heuristic methods provide ways of tackling combinatorial optimization problems, and they are most suitable for solving constrained financial optimization problems. Differential Evolution (DE) is a population based heuristic method which was originally proposed by [18] for continuous optimization problems. It generates new solutions by combining three solutions, and cross-over with a fourth solution. With the standard DE method, only the population size  $P$ , the scaling factor  $F$  for the linear combination and the cross-over probability  $\pi_1$  need to be considered. A crucial property of this version of DE is that all individuals of the population eventually converge to the same optimum. For genuinely different new solutions to emerge, however, the parent individuals must represent different solutions. The more challenging the search space and the (relatively) smaller the population, the higher the chance that the population converges prematurely to what is only local optima. The lack of diversity within the population requires additional perturbation by introducing noise. This is typically done by adding normally distributed random numbers to  $F$  value and the difference of two solution vectors respectively. Vectors  $\mathbf{z}_1$  and  $\mathbf{z}_2$  represent the extra noise in the algorithm; they contain random numbers being zero with probability  $\pi_2$  and  $\pi_3$  respectively, or following normally independent distribution  $N(0, \sigma_1^2)$  and  $N(0, \sigma_2^2)$ . The linear combination and cross-over procedure are described as follows:

$$\tilde{v}_c[i] := \begin{cases} v_{p_1}[i] + (F + z_1[i]) \cdot (v_{p_2}[i] - v_{p_3}[i] + z_2[i]) & \text{with probability } \pi_1 \\ v_{p_4}[i] & \text{otherwise,} \end{cases}$$

where  $\pi_1$  is the cross-over probability and  $p_1, \dots, p_4$  are four distinct solutions. After creating a set of new candidate solutions, a tournament mechanism replaces some of the existing solutions with new ones. The process is repeated until a halting criterion is met. The DE algorithm is described by using the pseudo code as follows.

---

**Algorithm 7.1.** Differential Evolution.

---

```

randomly initialize population of vectors  $v_p, p= 1 \dots P$ ;
while the halting criterion is not met do
  for all current solutions  $v_c, c=1 \dots P$  do
    randomly pick  $p_1 \neq p_2 \neq p_3 \neq p_4$ ;
    for all elements  $i$  do
       $\tilde{v}_c[i] \leftarrow v_{p_1}[i] + (F + z_1[i])(v_{p_2}[i] - v_{p_3}[i] + z_2[i])$  with probability  $\pi_1$ , or
       $\tilde{v}_c[i] \leftarrow v_{p_4}[i]$  otherwise;
    end
    interpret  $\tilde{v}_c$  into equity weights and compute the fitness value;
  end
  for the current solution  $v_c, c = 1 \dots P$  do
    if Fitness( $\tilde{v}_c$ ) > Fitness( $v_c$ ) then  $v_c \leftarrow \tilde{v}_c$ ;
  end
end

```

---

Since the solutions from DE may be either positive or negative, the no-short-selling constraint would be violated if one directly took  $v_p$  as weight solutions. In other words, the solutions from DE may not be valid for the current problem. According to the literature, one may use penalty functions to impair the fitness of solutions which violate constraints. Despite the straightforwardness of using penalty functions, computational efficiency would be reduced if one intended to use penalty functions to satisfy all constraints. A mapping function is used to translate  $v_p$  into valid equity weights, in order to satisfy the integer, cardinality and weight constraints. The mapping function first checks the number of positive elements  $\kappa$  in  $v_p$ . If  $\kappa \leq 0$ , the function prohibits the  $v_p$  entering the fitness evaluation procedure. Otherwise, the mapping function selects the  $k$  largest positive elements in  $v_p$  giving  $\kappa > k$ . If the positive number satisfies the condition  $\kappa < k^{\min}$ , the function picks  $k^{\min}$  largest elements from  $v_p$ . If the positive number of elements satisfies the condition  $k^{\min} < \kappa < k$ , those equities with positive values are included in the tracker.

The included equities are first assigned the minimum weight  $x_g^l$ , and then they are increased in proportion to the values in  $v_p$  until the sum of them add up to unity. If an equity weight exceeds the maximum weight, its weight is decreased to  $x_g^u$ , and the excess part is superadded proportionally to other selected equities according to their weights. The optimal holding of the  $i$ -th equity is computed by rounding up  $x_{i,t}P_t/S_{i,t}$  to the closest integer. The simple rounding approximation may bring neutrality bias to

the final solutions of such problems with binary variables, e.g. the problem of modelling yes/no decisions; a unit difference in stock holdings due to the rounding approximation could be usually ignored when the holding amount of a stock is large. In addition to the mapping function, a penalty function is used to guarantee the solutions satisfying the transaction cost constraint. The penalty function impairs the fitness value of the unsatisfied solutions by imposing a punishment:  $-\max(CT_t - 2 \cdot \gamma \cdot P_t, 0)$ .

The mapping function is largely based on the one applied in [13] for a similar optimization problem. It must be noted that some elements in this procedure can potentially generate some bias in the search process. Preliminary experiments found that this bias has typically no negative impact on the actual quality of the final result and that it was the most efficient of the tested alternatives.

### 7.2.6 Experiment Settings and Data

The following experiment settings were used: Initial budget  $B_t = 10,000,000$  dollars; Cash Reserve Rate  $C = 10\%$ ; Transaction Cost Limiting Rate  $\gamma = 1\%$ ; Transaction Cost Coefficient  $\rho = 0.1\%$ ; Cardinality Size  $k = 5$ . To the portfolios with fewer dimensions, the transaction costs will be low; and the costs can easily be modelled if the cost functions are nonlinear. Furthermore, a well diversified portfolio can be achieved by using limited assets (see [11]). Therefore, the cardinality size in this study was set at the relatively low value of five. The calendar based rebalancing strategy considered a rebalancing time interval  $T_\psi$  being 60 trading days, representing quarterly readjustment. In the tolerance triggered rebalancing, the window size  $W_L$  was set at 60. As a result, the trigger values were computed by using the information in the last 60 trading days. A step size  $\varphi = 10$  was adopted, i.e. the minimum rebalancing frequency was two weeks to prevent the rebalance from occurring too frequently. The TE value was used as a trigger in the tolerance triggered rebalancing. The TE tolerance  $\xi_1$  was set at 0.004, which is proposed to set the value as twice the in-sample tracking errors. The equity weight trigger  $x_{i,t}$  had minimum and maximum values being 1% and 50%, respectively. Setting such a high upper limit helps to study the actual diversification ability of the tracking model, as the DE algorithm has greater degrees of freedom in choosing asset weights. For real-world applications, the upper limit should be reduced further to avoid the risk due to sudden changes of major index shares (in this study it was found that the highest optimized weights always varied around 40%). In the TE and ER optimization experiments, the weighting difference  $\lambda$  was set at 50%. In the TE with loss aversion experiments, the aversion coefficient  $\vartheta$  was set at 2, thereby doubling the impairment of negative deviations.

The technical parameters of DE algorithm are listed as follows. Population size and iteration number were set at 1,000 and 4,000; the factor  $F$  was set at a value 0.5; and the cross-over probability  $\pi_1$  was at 60%. The parameters were used to generate the artificial noise:  $\pi_2 = 50\%$ ,  $\pi_3 = 10\%$ ,  $\sigma_1 = 0.1$  and  $\sigma_2 = 0.1$ . The above settings was found to be highly suited for solving this index mutual fund tracking problem; in preliminary experiments, the relative differences between the TE and its corresponding lowest TE recorded after independent restarts were found to be zero or negligible.

A total of 445 equities were used to track index mutual funds. The following five S&P 500 index funds were considered as targets in this study: ETRADE S&P 500 Index;

Vanguard 500 Index; USAA S&P 500 Index; UBS S&P 500 Index A; and TIAA-CREF S&P 500 Index Retire. The five index funds were traced by Tracker 1 to 5, respectively. The data comprise of daily prices and the net asset values of the equities and funds in the period January 2004 to December 2007, downloaded from Datastream. The equities have price sequences with 1,043 observations. To decide whether a large or a small data sample is suitable for this fund replication, different in-sample data sizes are considered at the construction stage and at the rebalancing stage. The first 250 observations (i.e. the information in 2004) were used to construct trackers, which would be held from the beginning of 2005. At the rebalancing stage, the latest 60 observations at each rebalancing point were employed to decide on the optimal adjusted quantities.

## 7.3 Experiment Results

### 7.3.1 In-Sample Tracking Performance

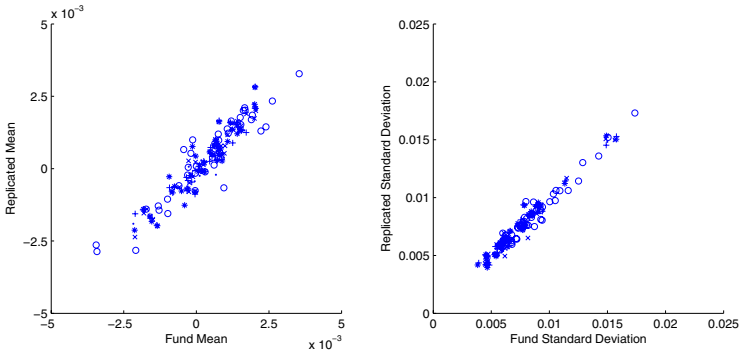
In order to determine how well the model replicates the fund returns over the in-sample periods for calendar-based rebalancing, scatter plot of the means and standard deviations of the actual and replicated daily returns at monthly intervals are shown in Figure 7.1. The first to fifth replicated case are identified by using different markers: Plus sign, Circle, Asterisk, Point and Cross. It can be seen that there are linear relationships existing between the fund return moments and replicated return moments. In order to show clearly the impact of the three objective functions on the in-sample tracker performance, Figure 7.2 provides the TE and the excess return to risk ratio (or the excess Sharpe ratio ( $r_p/\sigma_p - r_f/\sigma_f$ )), which were computed at monthly intervals. The return to risk ratio was approximately equal to the Sharpe Ratio when one considers the daily safe rate being tiny. In the figure, different line styles (i.e. solid, solid-circle, dashed, dotted, and dash-dot) illustrate the performance of the five trackers respectively. In Figure 7.2, the left and right panels show the in-sample TE and the excess Sharpe ratio respectively. As the figure shows, the excess Sharpe ratios are improved after considering the ER and loss aversion; and the TE from the two extended objective functions are higher than that from the classic TE optimization as expected. For the tolerance triggered rebalancing, the in-sample tracker performance from the three objective functions should have been similar to those from the calendar based rebalancing, since the in-sample results are independent of the rebalancing strategies.

The next subsection provides the results from the out-of-sample experiments and analysis in order to judge whether the model replicates the fund returns over the out-of-sample periods. The impact of the objective functions and the two rebalancing strategies on tracker performance are further explored and discussed.

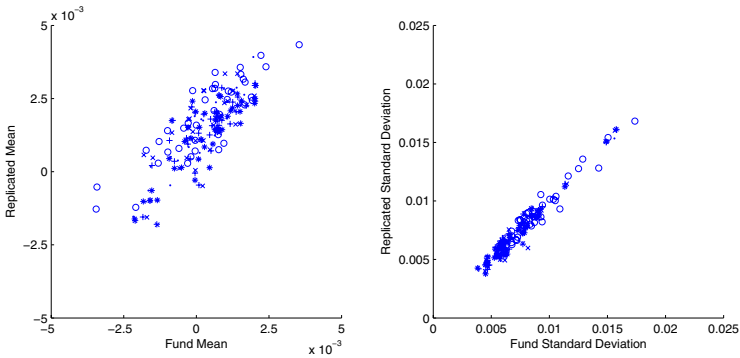
### 7.3.2 The Out-of-Sample Analysis of Replicators

Figure 7.3 and Figure 7.4 provide scatter plots of the return means and return standard deviations. The scatters were computed on the basis of the replicated returns and fund returns at monthly intervals.

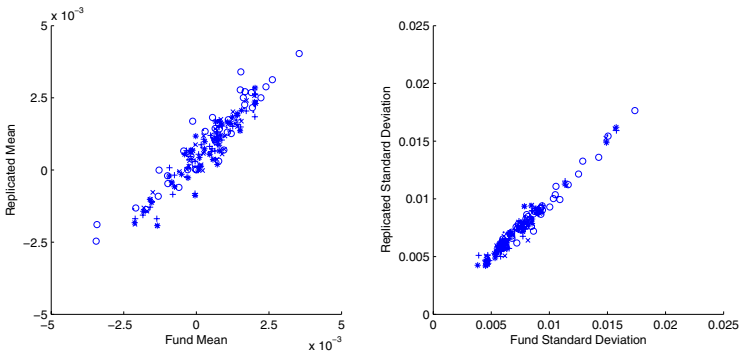
In order to quantify the relationship between the fund return and tracker return moments, linear regression analysis is employed to study the observations shown in the



(a) Tracking Error Optimization

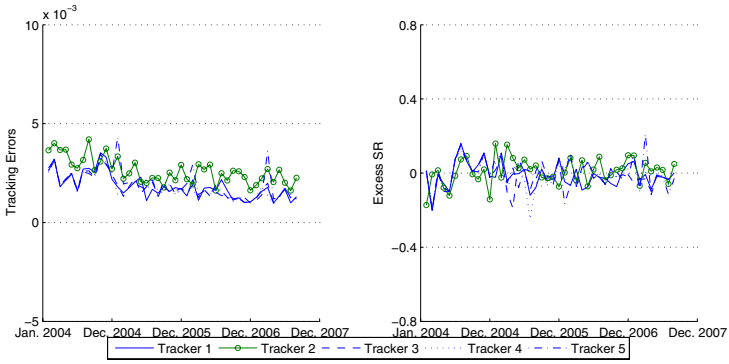


(b) Tracking Error with Excess Return Optimization

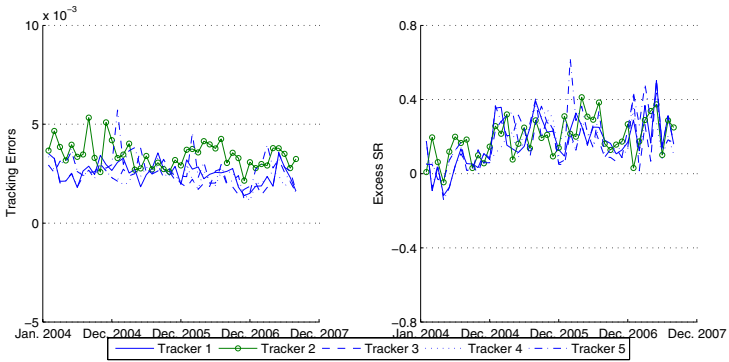


(c) Tracking Error with Loss Aversion Optimization

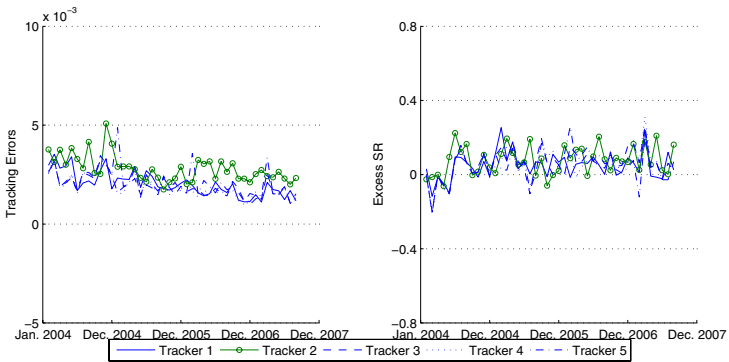
**Fig. 7.1.** In-Sample Monthly Means and Standard Deviations of Actual and Replicated Returns from Calendar Based Rebalancing.



(a) Tracking Error Optimization



(b) Tracking Error and Excess Return Optimization



(c) Tracking Error and Loss Aversion Optimization

**Fig. 7.2.** In-Sample Monthly Tracking Errors and Excess Sharpe Ratios from Calendar Based Rebalancing.

**Table 7.2.** Regression Analysis of Actual and Replicated Return Means.

	TE Opt.		Ext. 1		Ext. 2	
	C.	T.	C.	T.	C.	T.
SSE ( $10^{-3}$ )	0.0983	0.0818	0.1296	0.1427	0.1110	0.0919
$R^2$	0.6494	0.6924	0.5576	0.6682	0.6117	0.7094
$\bar{\alpha}$ ( $10^{-3}$ )	-0.0199	-0.1942	-0.1054	-0.0692	-0.0969	-0.1239
S.D.( $\alpha$ ) ( $10^{-3}$ )	0.0661	0.0603	0.0758	0.0796	0.0702	0.0639
$p(\alpha \neq 0)$	0.7436	0.0016	0.1665	0.3860	0.1696	0.0542
$\bar{\beta}$	0.8950	0.8999	0.8474	1.1238	0.8771	0.9931
S.D.( $\beta$ )	0.0532	0.0485	0.0610	0.0640	0.0565	0.0514
$p(\beta \neq 1)$	0.0497	0.0413	0.0124	0.0536	0.0308	0.8966

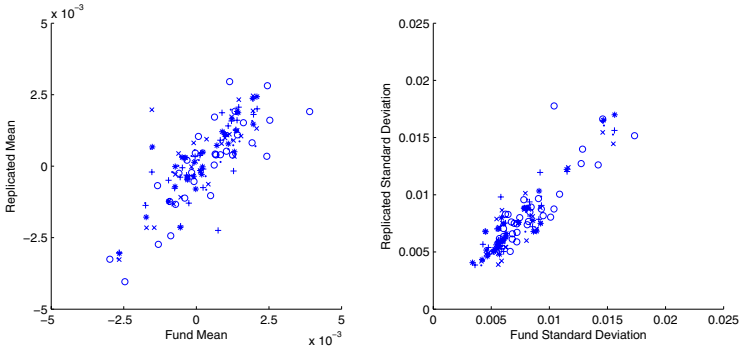
**Table 7.3.** Regression Analysis of Actual and Replicated Return Standard Deviations.

	TE Opt.		Ext. 1		Ext. 2	
	C.	T.	C.	T.	C.	T.
SSE ( $10^{-3}$ )	0.2516	0.1623	0.3170	0.1736	0.2793	0.2316
$R^2$	0.8206	0.8650	0.7597	0.8664	0.7908	0.8294
$\bar{\alpha}$ ( $10^{-3}$ )	0.5485	0.7363	1.6700	1.1030	1.0100	0.6546
S.D.( $\alpha$ ) ( $10^{-3}$ )	0.2920	0.2346	0.3278	0.2426	0.3077	0.2802
$p(\alpha \neq 0)$	0.0623	0.0020	< .0001	< .0001	0.0013	0.0208
$\bar{\beta}$	0.9654	0.9176	0.9008	0.9548	0.9245	0.9548
S.D.( $\beta$ )	0.0365	0.0293	0.0410	0.0303	0.0384	0.0350
$p(\beta \neq 1)$	0.3472	0.0051	0.0163	0.1362	0.0495	0.1973

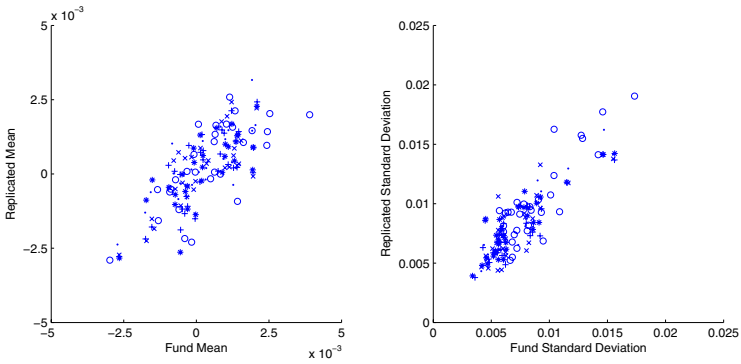
scatter plots. Tables [7.2](#) and [7.3](#) summarize the regression analysis results. As the two tables show, the  $R^2$  values from the standard deviation regression are all higher than the values from the mean regression, indicating that the model replicates the return standard deviations better than the return means. The intercept  $\alpha$  and slope  $\beta$  values from the analysis can be used as indicators to evaluate tracking performance: if a tracker perfectly replicates a fund, the regression intercept and slope should be equal to 0 and 1 respectively. According to the table, most of the intercept  $\alpha$  values from the mean regression are close to 0, and not statistically different from 0. Although the  $\alpha$  values in the standard deviation regression case are small, they are statistically different from 0 at a 5% confidence level, indicating the risk of trackers are slightly higher than their targets. The impact of the extra risk on tracker performance will be discussed together with the excess Sharpe ratio.

When using calendar based rebalancing, most of the  $\beta$  values are lower than 1, and statistically different from 1. However, while using the tolerance triggered rebalancing, there is evidence that the  $\beta$  values are not statistically different from 1. Therefore, the tracker using the tolerance triggered strategy should have had a higher tracking ability than the one using the calendar based strategy. Moreover, the  $\beta$  values from the two extended objective functions are higher than the one from the classic TE optimization

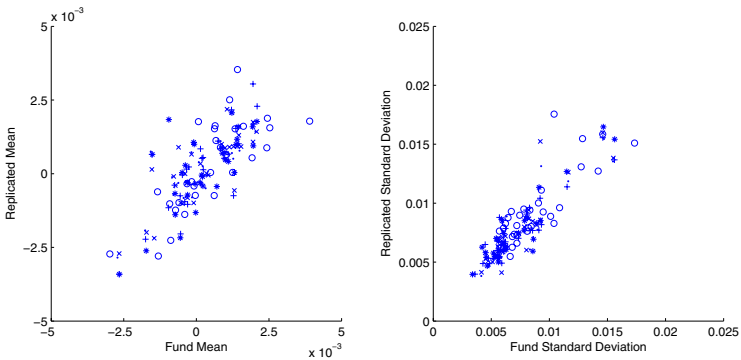




(a) Tracking Error Optimization

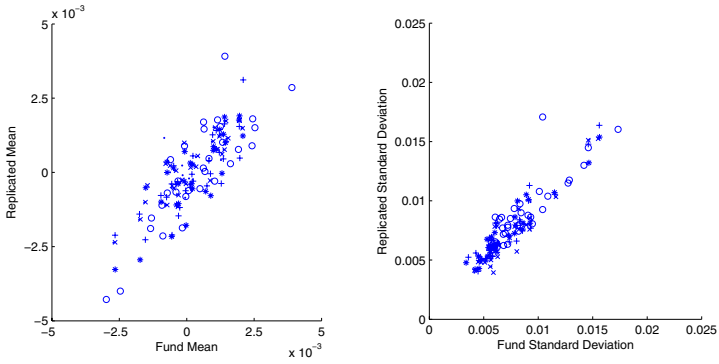


(b) Tracking Error with Excess Return Optimization

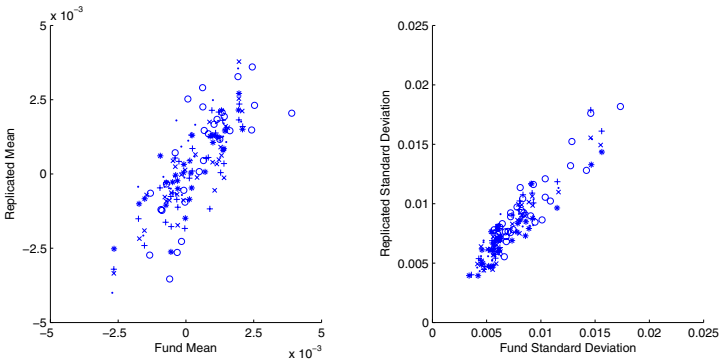


(c) Tracking Error with Loss Aversion Optimization

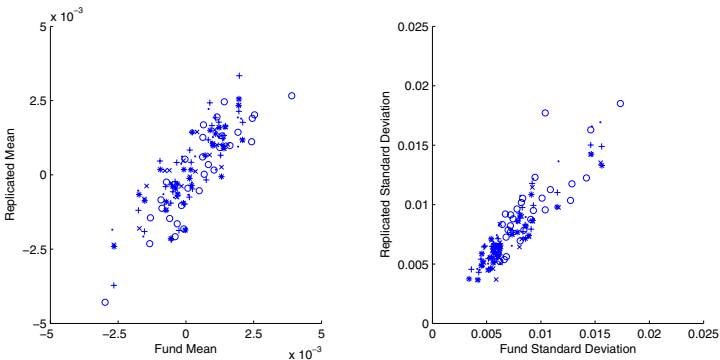
**Fig. 7.3.** Out-of-Sample Means and Standard Deviations of Actual and Replicated Returns from Calendar Based Rebalancing.



(a) Tracking Error Optimization

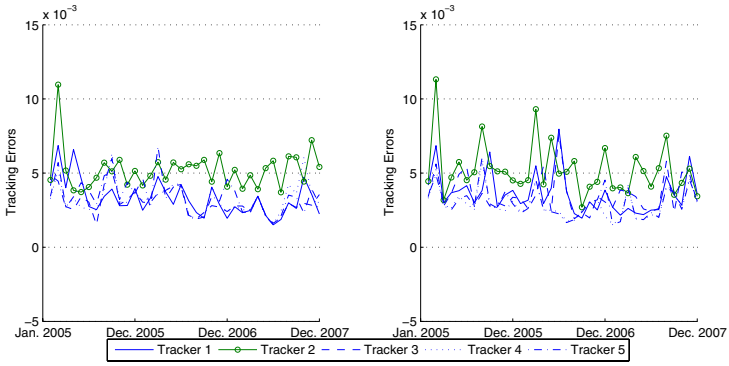


(b) Tracking Error with Excess Return Optimization

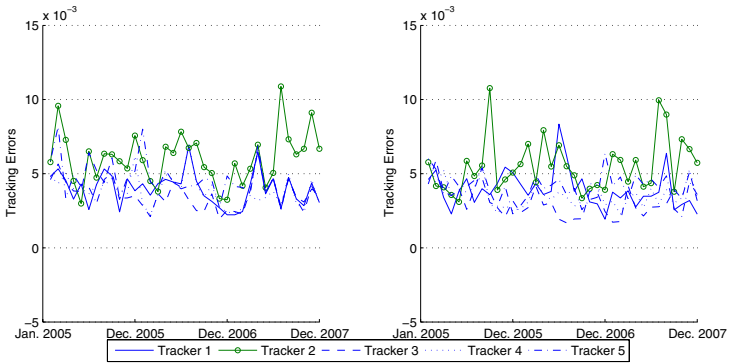


(c) Tracking Error with Loss Aversion Optimization

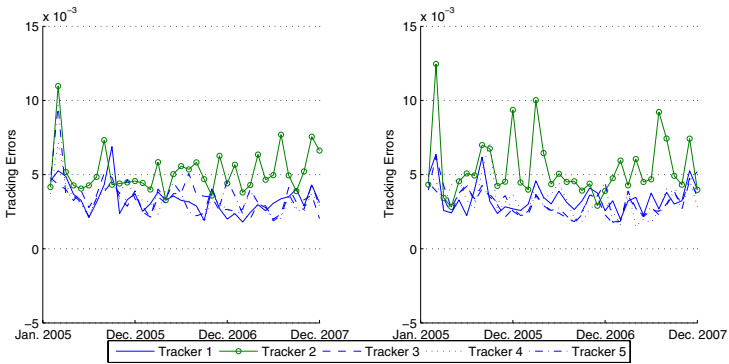
**Fig. 7.4.** Out-of-Sample Means and Standard Deviations of Actual and Replicated Returns from Tolerance Triggered Rebalancing.



(a) Tracking Error Optimization

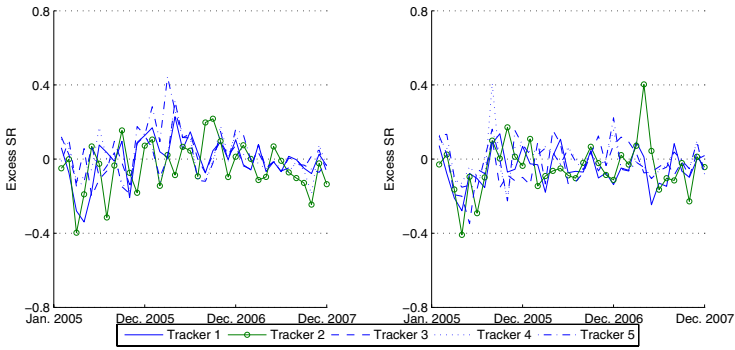


(b) Tracking Error with Excess Return Optimization

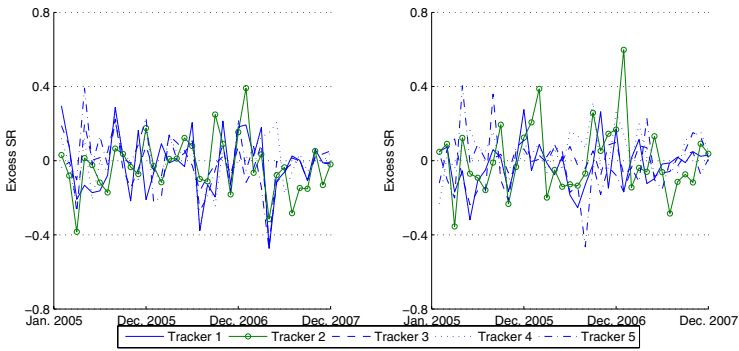


(c) Tracking Error with Loss Aversion Optimization

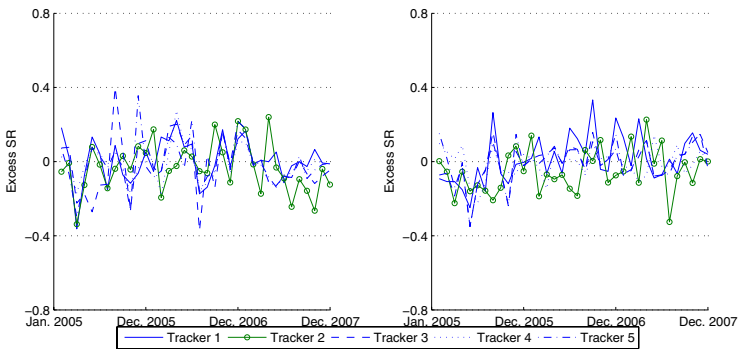
**Fig. 7.5.** Out-of-Sample tracking errors from using the three objective functions and two rebalancing strategies (left – calendar based rebalancing, right – tolerance triggered rebalancing).



(a) Tracking Error Optimization



(b) Tracking Error and Excess Return Optimization



(c) Tracking Error and Loss Aversion Optimization

**Fig. 7.6.** Out-of-Sample excess Sharpe ratios from using the three objective functions and two rebalancing strategies (left – calendar based rebalancing, right – tolerance triggered rebalancing).

**Table 7.4.** Rebalancing Times at Rebalancing Stage.

	Replicator 1	Replicator 2	Replicator 3	Replicator 4	Replicator 5
TE Opt. 2005	3	7	3	2	4
2006	3	8	2	0	0
2007	2	7	3	0	0
sum	8	22	8	2	4
Ext. 1 2005	3	9	4	5	7
2006	5	13	0	0	2
2007	2	13	1	1	6
sum	10	35	5	6	15
Ext. 2 2005	3	8	3	4	4
2006	0	7	0	1	0
2007	1	9	1	2	1
sum	4	24	4	7	5

when the trackers employ the tolerance triggered strategy. Due to the strict constraints imposed, it is reasonable that the replication criteria, i.e. the  $\alpha = 0$  and  $\beta = 1$  criteria are not perfectly satisfied. However, the current tracking performance could be improved further by relaxing some constraints, e.g. increasing the cardinality size.

It may be interesting to further compare the tracker performance of the model using the rebalancing strategies. The sub-figures in Figure 7.5 report the TE computed at monthly intervals; and the left and right panels of Figure 7.5 show the TE as a result of using the calendar based rebalancing and tolerance triggered rebalancing respectively. There is no significant difference found between the TEs which are optimized by using the two different rebalancing strategies. In other words, setting the TE tolerance  $\xi_1$  as twice the in-sample TE achieved the same result as that from the calendar based strategy case.

Figure 7.6 shows the impact of different objective functions and rebalancing strategies on the excess Sharpe ratio over the out-of-sample periods. The upper panel of Figure 7.6 reveals that there are consistent negative deviations of the ratios in year 2005. While considering the ER maximization as a part of the objective function, the negative excess Sharpe ratios in the year are reduced. However, it should be noted that both the positive and negative deviation of the ratio are larger than those in the upper panel. In the lower panel, the effect of loss aversion can be seen. The negative deviations of the excess Sharpe ratios are reduced, while the positive deviations can still be maintained at the same magnitude as those in the middle panel. The statistical results suggest that the excess Sharpe ratios are not significantly different from 0 at a 5% confidence level. The insignificance of the excess Sharpe ratios indicates that the extra risk taken by the trackers has been compensated by return premiums.

When comparing the left and right panels of Figure 7.6, the tolerance triggered rebalancing tends to produce more consistent positive deviations of the excess Sharpe ratio, than that using the calendar based rebalancing over the years 2006 and 2007. These positive deviations also explain the reason of the higher  $\beta$  values shown in Table 7.2.

**Table 7.5.** Means and Standard Deviations of Actual and Replicated Returns.

Replications	Fund Return		Replicated Return														
	Mean( $10^{-3}$ )	S.D.( $10^{-2}$ )	TE Optimization					Ext. 1					Ext. 2				
			C.	T.	C.	T.	C.	T.	C.	T.	C.	T.	C.	T.	C.	T.	
2005	1	0.18	0.64	-0.44	-0.34	0.75	0.67	-0.06	-0.28	0.66	0.79	-0.15	-0.36	0.73	0.72		
	2	0.38	0.76	-0.23	-0.16	0.89	0.88	-0.05	0.21	0.93	0.93	-0.18	-0.44	0.92	0.91		
	3	0.18	0.64	-0.06	-0.46	0.69	0.69	0.10	0.17	0.75	0.76	-0.13	-0.24	0.72	0.70		
	4	0.19	0.64	-0.18	-0.04	0.66	0.67	-0.03	-0.02	0.77	0.79	-0.03	0.12	0.77	0.74		
	5	0.15	0.66	-0.18	0.05	0.73	0.65	0.21	-0.05	0.75	0.73	-0.27	0.09	0.79	0.74		
2006	1	0.42	0.63	0.95	0.21	0.63	0.68	0.35	0.34	0.75	0.77	0.63	0.68	0.61	0.66		
	2	0.20	0.90	0.45	-0.08	0.91	0.91	0.44	-0.01	0.98	1.00	0.43	0.14	0.98	0.96		
	3	0.41	0.62	0.89	0.23	0.60	0.65	0.44	0.15	0.70	0.70	0.70	0.59	0.65	0.60		
	4	0.42	0.63	0.84	0.56	0.60	0.59	0.33	-0.01	0.76	0.77	0.55	0.32	0.63	0.64		
	5	0.43	0.62	1.11	0.56	0.58	0.59	0.45	0.29	0.79	0.73	0.75	0.58	0.64	0.59		
2007	1	0.17	0.98	0.11	-0.18	1.03	0.97	-0.01	0.22	1.00	0.98	0.20	0.51	0.99	1.00		
	2	0.71	1.10	0.03	0.39	1.07	1.11	-0.03	0.55	1.29	1.18	0.07	0.40	1.08	1.14		
	3	0.18	0.98	0.13	0.05	1.04	0.95	-0.11	0.11	1.01	1.01	0.07	0.43	1.00	0.92		
	4	0.17	0.98	0.20	0.07	1.03	0.96	0.33	0.34	1.01	1.04	0.13	0.26	0.95	1.07		
	5	0.17	0.98	0.11	0.08	1.01	0.96	-0.16	0.11	1.00	1.03	0.06	0.36	1.00	0.92		

Cost efficiency is an important criterion for rebalancing strategy evaluation. In the experiments, the rebalancing stage consisted of three years. Thus there would be 12 rebalances if one takes 60 trading days as rebalancing interval in the calendar based

strategy. It will be interesting to know how many rebalancing times occur during the rebalancing period while using the tolerance triggered rebalancing; Table 7.4 records the number of rebalances in this case. From the table, it is found that excepting Replicator 2, most of the rebalancing times of others in the rebalancing stage are less than 12, suggesting the tolerance triggered strategy is more cost-efficient than the calendar based strategy.

To explore the reason for the high rebalancing times of Replicator 2, the actual fund returns are examined. Table 7.5 shows the means and standard deviations which were computed based on the actual and replicated returns. It should be noted that Fund 2 exceeds the other four funds by yielding almost two and four times more than that of others in 2005 and 2007, respectively, but it gains only half of the rewards of others in 2006. The remarkable performance of Fund 2 implies that the fund may adopt a different management strategy from that of the passive management style, thereby resulting in the high tracking errors (the case of solid line with circle marker in Figure 7.5) and the high rebalancing times.

In Table 7.5 one may note that most of the negative returns of the replicators occur in 2005; correspondingly, it is straightforward to observe that the excess Sharpe ratios are negative during 2005 from Figure 7.6. This evidence indicates that the tracker portfolios constructed at the beginning are not robust. One possible explanation for this is that the authors made use of one year observations, i.e. the first 250 daily data to construct the trackers. Using long historical data, e.g. one year or half year observations to analyze index compositions may be appropriate, but it may lead to unreliable outcomes for fund tracking. The main difference between indices and index funds is that the funds are under professional management. To maintain effective diversification of portfolios, fund managers normally do not keep their holdings unchanged for such long periods, e.g. one year. Therefore, a long data sample will shield the true fund compositions if there is rebalance taken place just before the tracker construction; thus, using short term data, e.g. quarterly data may be more appropriate. Apart from directly using short term data samples, one can modify the TE definition to include a weighting factor, for more recent time periods getting a higher weighting than other time periods (see 4).

## 7.4 Conclusion

In this paper, the authors develop a fund tracking model in order to track index mutual funds with constraints on the cardinality size, assets' weights, transaction costs, and integer constraints. This paper proposes a novel way of decomposing the index mutual fund replication into the traditional index tracking problem and a multi-period optimization problem. By employing a heuristic method to solve the optimization problem, an empirical study is performed to track five S&P 500 mutual funds dynamically. The regression results show that the model replicates the first two moments of index fund returns by using limited equities; moreover, the optimized tracker portfolios do not exhibit significant difference between the original and replicated Sharpe ratios. By setting the tracking error tolerance at twice the in-sample tracking error in the tolerance triggered rebalancing, the model produced the same tracking error magnitude as that using the calendar based rebalancing. Also, it has been shown that tolerance triggered

rebalancing outperformed the calendar based rebalancing in terms of both tracking ability and cost-efficiency. Financial practitioners may employ the model to build up their own portfolios based on any interesting index mutual funds for certain purposes, such as the funds from different geographical areas for global investments. In future research, the short selling constraint may be relaxed; some financial products, such as bonds, futures and options may be included into tracker portfolios for more advanced applications, e.g. enhanced index funds (EIF) replications. Differential Evolution is reliable and flexible enough for these further extensions.

## Acknowledgement

The authors gratefully acknowledge financial support from the EU Commission through MRTN-CT-2006-034270 COMISEF.

## References

1. Alexander, C., Dimitriu, A.: Sources of out-performance in equity markets. *The Journal of Portfolio Management* 30, 170–185 (2004)
2. Alexander, C., Dimitriu, A.: Indexing and statistical arbitrage. *The Journal of Portfolio Management* 31, 50–63 (2005)
3. Anderson, S.C., Ahmed, P.: Mutual Fund Fees and Expenses. In: *Mutual Funds: Fifty Years of Research Findings*, ch. 3, pp. 57–69. Springer, Heidelberg (2005)
4. Beasley, J.E., Meade, N., Chang, T.J.: An evolutionary heuristic for the index tracking problem. *European Journal of Operational Research* 148, 621–643 (2003)
5. Bogle, J.C.: *Common Sense on Mutual Funds: New Imperatives for the Intelligent Investor*. Wiley, John & Sons (1999)
6. Carhart, M.M.: On persistence of mutual fund performance. *Journal of Finance* 52, 57–82 (1997)
7. Eakins, G.S., Stansell, S.: An examination of alternative portfolio rebalancing strategies applied to sector funds. *Journal of Asset Management* 8, 1–8 (2007)
8. Gilli, M., K ellezi, E.: The threshold accepting heuristic for index tracking. In: Pardalos, P., Tsitsiringos, V. (eds.) *Financial Engineering, E-Commerce, and Supply Chain*. Kluwer Applied Optimization Series, ch. 1, pp. 1–18 (2002)
9. Haslem, J.A., Baker, H.K., Smith, D.M.: Performance and characteristics of actively managed retail equity mutual funds with diverse expense ratios. *Financial Services Review* 17, 49–68 (2008)
10. Malkiel, B.G.: Returns from investing in equity mutual funds. *Journal of Finance* 50, 549–572 (1995)
11. Maringer, D.: Small is beautiful: Diversification with a limited number of assets, centre for Computational Finance and Economic Agents Working Paper Series (2006)
12. Maringer, D.: Constrained index tracking under loss aversion using differential evolution. In: Brabazon, A., O’Neill, M. (eds.) *Natural Computing in Computational Finance*, ch. 2, pp. 7–24. Springer, Berlin (2008)
13. Maringer, D., Oyewumi, O.: Index tracking with constrained portfolios. *Intelligent Systems in Accounting and Finance Management* 15(1), 51–71 (2007)
14. Meade, N., Salkin, G.R.: Developing and maintaining an equity index fund. *Journal of the Operational Research Society* 41, 599–607 (1990)



15. Montfort, K.V., Visser, E., Draat, L.F.V.: Index tracking by means of optimized sampling. *The Journal of Portfolio Management* 34, 143–151 (2008)
16. Roll, R.: A mean–variance analysis of tracking error. *Journal of Portfolio Management* 18, 13–22 (1992)
17. Sharpe, W.F.: Mutual fund performance. *Journal of Business* 39, 119–138 (1966)
18. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* 11(4), 341–359 (1997)
19. di Tollo, G., Maringer, D.: Metaheuristics for the index tracking problem. In: Geiger, M.J., Habenicht, W., Sevoux, M., Sörensen, K. (eds.) *Metaheuristics in the Services Industry. Lecture Notes in Economics and Mathematical Systems*, vol. 624, ch. 8, pp. 127–154. Springer, Heidelberg (2009)

# Frequent Knowledge Patterns in Evolutionary Decision Support Systems for Financial Time Series Analysis

Piotr Lipinski

Institute of Computer Science,  
University of Wrocław, Wrocław, Poland  
lipinski@ii.uni.wroc.pl

**Summary.** This chapter discusses extracting and reusing frequent knowledge patterns in building trading experts in an evolutionary decision support system for financial time series analysis. It focuses on trading experts built by an evolutionary algorithm as binary sequences representing subsets of a specific set of trading rules, where frequent knowledge patterns correspond to common building blocks of trading rules occurring in previous trading experts. Reusing frequent knowledge patterns leads to a significant reduction of the search space, due to fixing a part of chromosome and running the evolution process to set only the remaining genes, without significant decreases of results.

This chapter presents a number of experiments carried out on financial time series from the Paris Stock Exchange, discusses some examples of the frequent knowledge patterns as well as analyses the results obtained in terms of their financial relevance and compares them with some popular benchmarks.

## 8.1 Introduction

Evolutionary algorithms are often applied in economic and financial modelling, [3], [6], [10], [12], where their capabilities of robust prediction and possibilities of dynamic modelling are particularly important to elaborate an efficient economic or financial model. For instance, genetic programming was applied to building decision trees for supporting financial decision making, [20], various evolutionary algorithms were constructed for portfolio optimization, [9], [14], grammatical evolution was applied to discovering trading rules for stock market speculations, [7].

One popular application concerns decision support systems for economic and financial time series analysis, especially time series containing stock price quotations, foreign exchange rates as well as gold or oil price quotations. Although there are a number of classic approaches, such as those based on statistical or stochastic modelling, evolutionary algorithms often outperform them in short-term or real-time analysis, where the mathematical trading model becomes too complex, uncertainty and information noise become very high and a solution must be found as soon as possible, [2], [4]. However, one of the bottlenecks in real-time evolutionary systems is the lengthy computing time, which creates a serious barrier to further development of such approaches.

This chapter refers to a class of decision support systems for time series analysis that bases its knowledge on evolutionarily generated artificial experts built on a set

of defined rules. These experts are generated on the basis of available past data and applied to new data until they are efficient. Since the analysed data exhibit time-varying properties, the efficiency of experts falls, at which point they should be regenerated or updated. The process of expert generation is time-consuming, prompting many ideas for its improvement. Updating an expert takes less time than regenerating one, although efficiency of an updated expert is often slightly lower than that of a newly-generated one. In this chapter, we discuss methods of expert updating based on discovering and reusing frequent knowledge patterns from a set of experts generated over a specific time period.

This chapter focuses on the optimization of expertise elaboration, which is discussed in the context of real-time stock trading systems and discovering frequent knowledge patterns in trading experts. The significance of delays in such systems, along with the issue of time necessary for discovering a profitable composition of indicator-based trading rules, is discussed. The proposed optimization was applied to a real-time financial data analysis system based on the genetic algorithm presented in [10].

The goal of the presented approach is to improve the throughput of the expert discovery process by discovering frequent knowledge patterns and shortening the time of expert computing by using this knowledge. The idea is to reuse previous trading experience of generated experts instead of computing experts *ex nihilo*. The trading experience can be found in patterns that are a kind of abstraction from the most recent sequence of chromosomes of the best experts. These patterns can be easily used to generate an initial population of experts that are located close to the place where the final solution might be found. Two effects of using patterns as building blocks are expected: first, reduction of the search space by decreasing the number of trading rules used for evaluation; second, acceleration of the convergence process due to the heuristics used to create an initial population of experts. The building blocks discovery process is based on sequential pattern mining algorithms applied to a list of previously generated experts [1].

This chapter is structured in the following manner: Section 2 defines the trading rules and the financial knowledge base containing them. Section 3 discusses the decision making process and Section 4 presents an overview of the evolutionary decision support system for such a decision making process. Section 5 describes frequent knowledge patterns and their applications in decision support systems. In Section 6, experiments on real-data from the Paris Stock Exchange are presented. Finally, Section 7 concludes the chapter and points out some directions for further research.

## 8.2 Financial Knowledge Base

A popular technique of financial time series analysis is the technical analysis, [5], [17], [11], which considers past stock price quotations as a principal factor affecting future price directions. It introduces many indicators, which characterise financial time series, and uses them to forecast future trends and particular events like falls and rises in stock prices. Such indicators may be formalised by a concept of a *stock market trading rule*.

**Definition 1.** A *stock market trading rule* is a function

$$f : \mathcal{K} \mapsto y \in \mathbf{R} \quad (8.1)$$

which maps a *factual financial knowledge*  $\mathcal{K}$  to a real number  $y$ , interpreted later as a trading signal: values lower than a certain threshold  $\alpha_1$  correspond to a sell signal (-1.0), values greater than a certain threshold  $\alpha_2$  correspond to a buy signal (+1.0), and remaining values correspond to no signal (0.0). In the case of the technical analysis, the factual financial knowledge represents past stock price quotations, but in other cases, it may also include fundamental information, tax rates, exchange rates, etc.

In the experiments, *the financial knowledge base* was composed of 250 popular trading rules, defined on the basis of [5] and [17], where the factual financial knowledge was composed of financial time series containing one-minute stock price quotations (i.e. open, high, low, close prices, transaction volumes and index values). Although, in all the experiments,  $\alpha_1 = -1$  and  $\alpha_2 = 1$ , different thresholds may be also tested in order to tune the number of trading signals obtained (the closer the values of  $\alpha_1$  and  $\alpha_2$ , the larger the number of trading signals obtained).

### 8.3 Decision Making Process

In most cases, different trading rules produce different, often opposing, trading signals, so the decision support system must spend some effort to transform them into one final trading decision [18]. Such a problem may be solved by selecting an efficient set of trading rules, referred to as a *stock market trading expert*, and defining the final trading decision by the trading signal proposed by the majority of the trading rules from the chosen set.

**Definition 2.** A *stock market trading expert* over a set of trading rules  $\mathcal{R} = \{f_1, f_2, \dots, f_d\}$  is a subset of the entire set of trading rules

$$E \subset \mathcal{R}. \quad (8.2)$$

A result  $E(\mathcal{K})$  of a trading expert  $E$ , for a given factual financial knowledge  $\mathcal{K}$ , is an arithmetic average of the results of the trading rules from the subset  $E$ ,

$$E(\mathcal{K}) = \frac{1}{|E|} \sum_{f \in E} f(\mathcal{K}), \quad (8.3)$$

interpreted later as a trading signal in the same way as in the case of a single trading rule.

In practice, the decision support system usually focuses on trading strategies for a specific time period rather than on single trading decisions for particular moments. For successive time instants  $t_0, t_1, t_2, \dots, t_{T-1}$  of the specific time period, the trading expert  $E$  produces trading decisions  $d_0^{(E)}, d_1^{(E)}, d_2^{(E)}, \dots, d_{T-1}^{(E)}$ . In order to apply such trading decisions on the stock market, it is also necessary to define the amount of stocks to be bought or sold. Such amounts may be obtained by simulating the behavior of a hypothetical investor, who is given an initial endowment  $(c_0, s_0)$  with  $c_0$  the initial amount of cash and  $s_0$  the initial quantity of stocks (in the experiments,  $c_0 = 10000$  and  $s_0 = 100$ ) and follows the trading decisions of the trading expert.

At time  $t_0$ , the investor takes the decision  $d_0^{(E)}$ . If the decision is to sell, i.e.  $d_0^{(E)} = -1$ , he sells  $q\%$  of stocks (in experiments,  $q = 50\%$ ), i.e. the amount of stock  $\Delta s$  in the investor's order is equal to

$$\Delta s = s_0 \cdot q. \quad (8.4)$$

If the decision is to buy, i.e.  $d_0^{(E)} = 1$ , he invests  $q\%$  of money in stocks (in experiments,  $q = 50\%$ ), i.e. the amount of stock  $\Delta s$  in the investor's order is equal to

$$\Delta s = \frac{c_0 \cdot q}{(1 + \tau) \cdot \text{Open}(t_1)}, \quad (8.5)$$

where  $\text{Open}(t)$  denotes the open price at time  $t$  and  $\tau$  denotes transaction costs (in experiments,  $\tau = 0.2\%$ ). Next, the transaction is executed at time  $t_1$  and the investor's capital changes accordingly. At time  $t_1$ , the investor's capital consists of the amount of cash

$$c_1 = c_0 - d_0^{(E)} \cdot \Delta s \cdot \text{Open}(t_1) - \tau \cdot \Delta s \cdot \text{Open}(t_1), \quad (8.6)$$

and the quantity of stocks

$$s_1 = s_0 + d_0^{(E)} \cdot \Delta s. \quad (8.7)$$

Next, the investor takes the decision  $d_1^{(E)}$ , which is executed at time  $t_2$  and the investor's capital changes again, and so on. Finally,  $c_0, c_1, \dots, c_T$  denote the successive amounts of cash and  $s_0, s_1, \dots, s_T$  the corresponding quantities of stocks at successive time instants  $t_0, t_1, t_2, \dots, t_{T-1}$  of the specific time period.

## 8.4 Evolutionary Decision Support System

### 8.4.1 Objectives and Optimization Problem

In practice, the objectives of the decision support system are to build a trading expert efficient over a future time period with unknown stock prices, where exact sequences of amounts of cash and quantities of stocks cannot be evaluated, so the decision support system must estimate the future performance of the trading expert on the basis of its behavior over a past time period. It may use a number of so-called performance measures, [16], [13], which consider not only the future return rates, but also the risk factors related to achieving them. Experiments focus on the Sharpe ratio, [19], but different performance measures, such as the Sortino ratio, the Sterling ratio or the Treynor ratio, may also be tested in order to adjust the financial model and the risk definition.

Formally, the performance measure  $\varrho(E)$  of the trading expert  $E$  (the Sharpe ratio) is equal to

$$\varrho(E) = \frac{\mathbf{E}[R] - r_0}{\mathbf{Std}[R]}, \quad (8.8)$$

where  $\mathbf{E}[R]$  denotes the expected return rate of the trading expert  $E$ , the  $\mathbf{Std}[R]$  denotes the standard deviation of the return rate and  $r_0$  denote the reference return rate of risk-free asset.

For the specific time period studied in the previous section, let  $C_i$  denote the capital at time  $t_i$ , for  $i = 0, 1, 2, \dots, T - 1$ ,

$$C_i = c_i + s_i \cdot \text{Open}(t_i) \quad (8.9)$$

and  $R_i$  denote the return rate for the time period  $[t_{i-1}, t_i]$ , for  $i = 1, 2, \dots, T-1$ ,

$$R_i = \frac{C_i - C_{i-1}}{C_{i-1}}, \quad (8.10)$$

therefore, the expected return rate  $\mathbf{E}[R]$  and the standard deviation of the return rate  $\mathbf{Std}[R]$  may be estimated on the basis of  $R_1, R_2, \dots, R_{T-1}$ .

Discovering efficient trading experts is an optimization problem with the objective function  $\varrho(E)$  over the search space of all trading experts  $E$ . Formally, let  $\mathcal{R} = \{f_1, f_2, \dots, f_d\}$  be the set of all available trading rules and  $\mathcal{E}(\mathcal{R})$  be the set of all available trading experts built on these trading rules. The objective is to find a trading expert  $E \in \mathcal{E}(\mathcal{R})$  such as

$$\varrho(\tilde{E}) \leq \varrho(E) \quad (8.11)$$

for all  $\tilde{E} \in \mathcal{E}(\mathcal{R})$ , for a given time period and for given the other parameters of the financial simulation described in the previous section.

#### 8.4.2 Binary Representation of Trading Experts

Each trading expert  $E$  may be represented in a natural way by a binary sequence  $\mathbf{e} = (e_1, e_2, \dots, e_d)$ , in such a way that, for  $i = 1, 2, \dots, d$ ,  $e_i$  corresponds to the  $i$ -th trading rule  $f_i \in \mathcal{R}$ ;  $e_i = 0$  denotes the absence and  $e_i = 1$  denotes the presence of the trading rule  $f_i$  in the set  $E$ .

Such a representation defines a one-to-one map between trading experts and binary sequences of length  $d$ , so the search space in the optimization problem defined in the previous section is simply  $\{0, 1\}^d$  and the objective function  $\varrho(\mathbf{e})$  is the performance measure of the trading expert  $E$  corresponding to the binary vector  $\mathbf{e}$ .

#### 8.4.3 Building Trading Experts Using Evolutionary Algorithm

Building an efficient trading expert was transformed in the previous sections to the optimization problem with the objective function defined by the performance measure  $\varrho$ , being the Sharpe ratio, and the search space  $\{0, 1\}^d$ . Naturally, the objective function is defined in the context of a given set  $\mathcal{R}$  of trading rules, a given stock, a given training period and given the other parameters of the financial simulation.

Algorithm 8.1 shows the framework of the evolutionary algorithm, based on the Simple Genetic Algorithm [8], [15], designed to solve the optimisation problem.

First, the algorithm creates an initial population  $\mathcal{P} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_N\} \subset \{0, 1\}^d$  in such a way that each gene  $e_{ij}$  of each chromosome  $\mathbf{e}_i$  is generated randomly using the uniform distribution, i.e. the probability of  $e_{ij} = 0$  and the probability of  $e_{ij} = 1$  are both equal to 0.5. After creation, the population is evaluated.

Afterwards, the population evolves under the influence of four evolutionary operators, namely *parent selection*, *crossover*, *mutation* and *replacement*, until a termination condition is satisfied ( $M$  denotes the number of parent individuals chosen for reproduction,  $\theta_C$  and  $\theta_M$  denote the probabilities of crossing over and mutation, respectively).

---

**Algorithm 8.1.** The Simple Genetic Algorithm designed to optimise the objective function  $\varrho$  with a population  $\mathcal{P}$  composed of  $N$  individuals, where  $M$ ,  $\theta_C$ ,  $\theta_M$  are some algorithm parameters

---

```

 $\mathcal{P}$  = Random-Population( $N$ );
Population-Evaluation( $\mathcal{P}$ ,  $\varrho$ );
while not Termination-Condition( $\mathcal{P}$ ) do
     $\mathcal{P}^{(P)}$  = Parent-Selection( $\mathcal{P}$ ,  $M$ );
     $\mathcal{P}^{(C)}$  = Crossover( $\mathcal{P}^{(P)}$ ,  $\theta_C$ );
    Mutation( $\mathcal{P}^{(C)}$ ,  $\theta_M$ );
    Replacement( $\mathcal{P}$ ,  $\mathcal{P}^{(C)}$ );
    Population-Evaluation( $\mathcal{P}$ ,  $\varrho$ );
end
Best-Expert( $\mathcal{P}$ );

```

---

The evolution terminates when it completes a specific number of iterations or when there is no increase in objective function values over a specific number of recent iterations. Due to size constraints, complete specifications of evolutionary operators are omitted and may be found in [8] or [15].

## 8.5 Frequent Knowledge Patterns

Trading experts, built on the basis of financial information related to a specific time period, become outdated and lose profitability when the stock market starts to differ significantly from its initial state. Normally, the stock market changes incessantly, so trading experts must be built anew or updated very often.

In the simplest approach, the decision support system builds a new trading expert each time when the current trading expert loses profitability, without reusing any information from previous trading experts. Algorithm 8.2 shows the framework of such an approach. When the decision support system starts, it initialises the financial knowledge base and builds the initial trading expert using the Simple Genetic Algorithm (see Algorithm 8.1). Next, in the loop, usually once per minute, it publishes the current trading expert and updates the financial knowledge base. If the efficiency of the current trading expert evaluated on the current knowledge base falls below a specific threshold, a new trading expert is built using the same algorithm (but with a new objective function related to the new knowledge base).

However, studies on trading experts built in successive iterations in the decision support system suggest that a new trading expert often has a similar structure to a number of recent trading experts, i.e. it often contains trading rules included in previous trading experts and does not usually contain trading rules excluded in previous trading experts. Such information may be formalised by a concept of a *frequent knowledge pattern*.

**Definition 3.** A *knowledge pattern* of trading experts over a set of trading rules  $\mathcal{R} = \{f_1, f_2, \dots, f_d\}$  is a sequence

$$\mathbf{k} = (k_1, k_2, \dots, k_d) \in \{0, 1, \#\}^d,$$

---

**Algorithm 8.2.** The decision support system, which builds new trading experts without reusing any information from previous ones

---

```

Financial-Knowledge-Base-Initializing( $\mathcal{K}$ );
 $\mathbf{e} = \text{SGA}(\varrho, N, M, \theta_C, \theta_M)$ ;
while System-Is-Running() do
  Trading-Expert-Publishing( $\mathbf{e}$ );
  Financial-Knowledge-Base-Updating( $\mathcal{K}$ );
  if Trading-Expert-Efficiency( $\mathbf{e}$ )  $< \zeta$  then
    |  $\mathbf{e} = \text{SGA}(\varrho, N, M, \theta_C, \theta_M)$ ;
  end
  Wait();
end

```

---

such that each element  $k_i$ , for  $i = 1, 2, \dots, d$ , corresponds to the  $i$ -th trading rule  $f_i$  in such a way that  $k_i = 0$  denotes the permanent absence and  $k_i = 1$  denotes the permanent presence of the trading rule  $f_i$  in the trading experts representing the knowledge pattern  $\mathbf{k}$  (if  $k_i = \#$ , trading experts representing the knowledge pattern  $\mathbf{k}$  may include the trading rule  $f_i$  or not). An *order* of the knowledge pattern  $\mathbf{k}$  is a number of its elements equal to 0 or to 1, i.e.

$$\text{ord}(\mathbf{k}) = \#\{i = 1, 2, \dots, d : k_i = 0 \vee k_i = 1\}.$$

**Definition 4.** A trading expert  $\mathbf{e} = (e_1, e_2, \dots, e_d)$  represents a knowledge pattern  $\mathbf{k} = (k_1, k_2, \dots, k_d)$ , which is denoted as  $\mathbf{k} \vdash \mathbf{e}$ , if it excludes all the trading rules  $f_i$  for which  $k_i = 0$  and includes all the trading rules  $f_i$  for which  $k_i = 1$ , i.e. if  $e_i = k_i$ , for each  $i = 1, 2, \dots, d$ , such that  $k_i = 0$  or  $k_i = 1$ .

**Definition 5.** For a set  $\mathcal{E}$  of trading experts, a *support* of a knowledge pattern  $\mathbf{k}$  is a subset of the set  $\mathcal{E}$  containing the trading experts representing the knowledge pattern, i.e.

$$\text{supp}(\mathbf{k}; \mathcal{E}) = \{\mathbf{e} \in \mathcal{E} : \mathbf{k} \vdash \mathbf{e}\}.$$

**Definition 6.** For a set  $\mathcal{E}$  of trading experts, a *frequent knowledge pattern* is the knowledge pattern of the highest order which is represented by all the trading experts, i.e.

$$\arg \max\{\text{ord}(\mathbf{k}) : \mathbf{k} \vdash \mathbf{e} \text{ for each } \mathbf{e} \in \mathcal{E}\}.$$

More efficient approaches may reuse frequent knowledge patterns, extracted earlier from previous trading experts, by *a priori* excluding or including some trading rules in all the trading experts generated and running the evolutionary process only to set the remaining part of trading rules. Algorithm 8.3 shows the framework of such an approach. When the decision support system starts, it initialises the financial knowledge base and builds the initial trading expert using the original algorithm. Next, in the loop, usually once per minute, it publishes the current trading expert and updates the financial knowledge base. If the efficiency of the current trading expert evaluated on the current knowledge base falls below a specific threshold, the decision support system extracts a frequent knowledge pattern from a number of previous trading experts and builds a



---

**Algorithm 8.3.** The decision support system which builds new trading experts using frequent knowledge patterns extracted earlier from previous trading experts

---

```

Financial-Knowledge-Base-Initializing( $\mathcal{K}$ );
 $\mathbf{e} = \text{SGA}(\varrho, N, M, \theta_C, \theta_M)$ ;
while System-Is-Running() do
    Trading-Expert-Publishing( $\mathbf{e}$ );
    Financial-Knowledge-Base-Updating( $\mathcal{K}$ );
    if Trading-Expert-Efficiency( $\mathbf{e}$ ) <  $\zeta$  then
         $\mathbf{k} = \text{Frequent-Knowledge-Pattern-Extracting}()$ ;
         $\mathbf{e} = \text{SGA-With-Knowledge-Patterns}(\varrho, N, M, \theta_C, \theta_M, \mathbf{k})$ ;
        if Trading-Expert-Efficiency( $\mathbf{e}$ ) <  $\zeta$  then
             $\mathbf{e} = \text{SGA}(\varrho, N, M, \theta_C, \theta_M)$ ;
        end
    end
    Wait();
end

```

---

new trading expert using a modified algorithm with the frequent knowledge pattern extracted (and with a new objective function related to the new knowledge base). If the efficiency of the new trading expert is still below the specific threshold, a new trading expert is built using the original algorithm.

## 8.6 Experiments

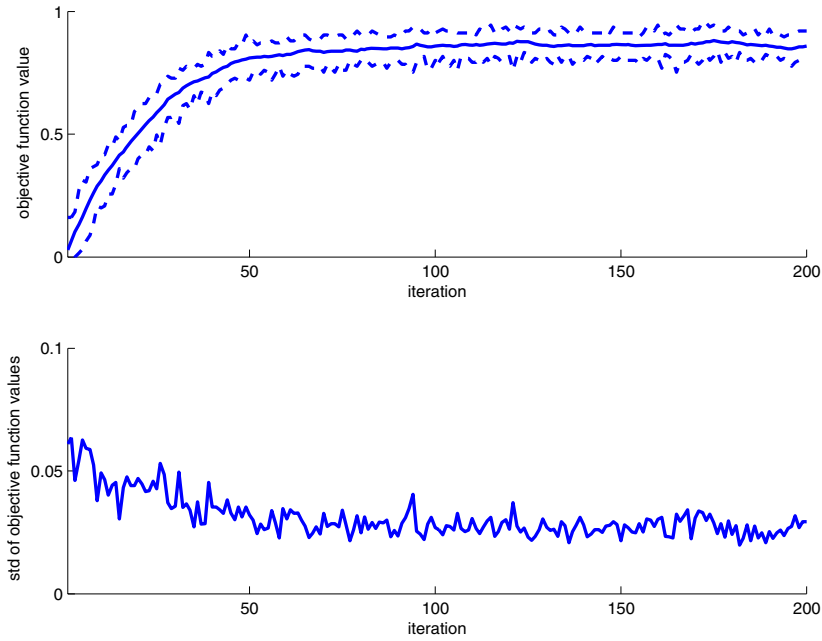
In order to illustrate the discovery and usage of frequent knowledge patterns, a number of experiments were carried out on 4 financial time series containing one-minute stock price quotations from the Paris Stock Exchange (namely, AXA, Peugeot, Renault and STMicroelectronics). Each experiment concerned one of these time series and a chosen time period.

In all the experiments, the same set of 250 trading rules was used and the Simple Genetic Algorithm was run with the same configuration (the Sharpe ratio objective function  $\varrho$ , the population size  $N$  equal to 50, the number of iterations  $n$  equal to 200 and  $M = 45$ ,  $\theta_C = 0.95$ ,  $\theta_M = 0.05$ ).

Although each experiment was considered separately, similar results were obtained. Due to size constraints, the further discussion focuses on only one of the cases.

### 8.6.1 Evolution of Gene Values

First, an initial trading expert was built using the original algorithm with a population of 50 individuals, each consisted of 250 genes, evolved in 200 iterations. Figure 8.1 presents values of the objective function in populations in successive iterations of the Simple Genetic Algorithm. It is easy to see that the optimisation algorithm led to a significant improvement of values of the objective function and consequently to a construction of an efficient trading expert. In practice, the number of iterations could be



**Fig. 8.1.** Values of the objective function (lowest, average and highest), as well as the standard deviation, in populations in successive iterations of the Simple Genetic Algorithm

decreased to about 75 – 100, but for the sake of comparison with other experiments it remains unchanged.

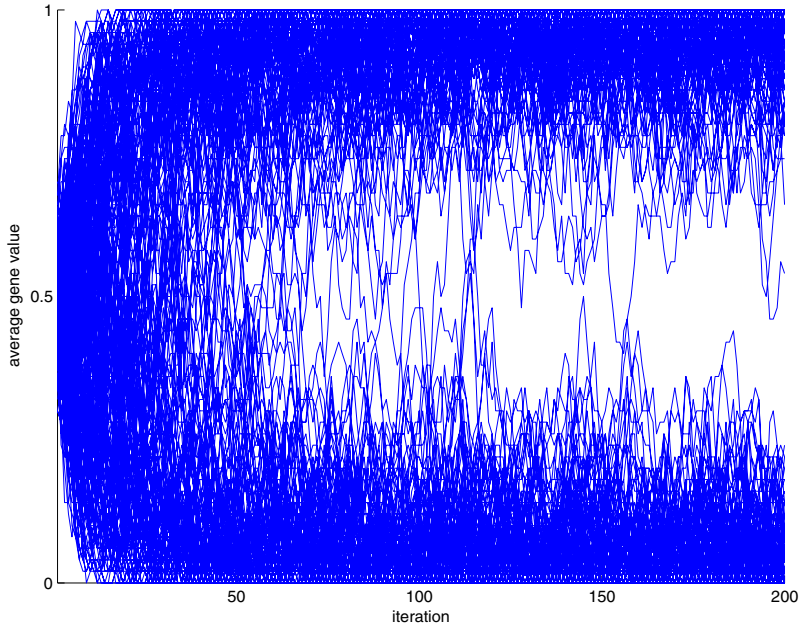
Further studies on the evolution of particular genes and its stabilisation in successive iterations of the Simple Genetic Algorithm, which is crucial to analyse frequent knowledge patterns, requires definitions of two additional factors: a frequency of usage  $\alpha$  and a stabilisation factor  $\beta$ .

Let  $\mathcal{P}^{(0)} = \{\mathbf{e}^{(0,1)}, \mathbf{e}^{(0,2)}, \dots, \mathbf{e}^{(0,N)}\}$  denote the initial population created before the first iteration starts, and  $\mathcal{P}^{(k)} = \{\mathbf{e}^{(k,1)}, \mathbf{e}^{(k,2)}, \dots, \mathbf{e}^{(k,N)}\}$  denote the current population after the  $k$ -th iteration of the Simple Genetic Algorithm, for  $k = 1, 2, \dots, n$ , where  $n$  is the number of iterations and  $N$  is the population size. For  $i = 1, 2, \dots, d$  and  $k = 0, 1, \dots, n$ , let

$$\alpha_i^{(k)} = 1/N \cdot \sum_{j=1}^N e_i^{(k,j)} \quad \text{and} \quad \beta_i^{(k)} = |0.5 - \alpha_i^{(k)}|$$

be the frequency of usage and the stabilisation factor of the  $i$ -th gene in the  $k$ -th iteration, respectively.

Figure 8.2 presents frequencies of usage  $\alpha_i^{(k)}$  of particular genes,  $i = 1, 2, \dots, d$ , in successive iterations,  $k = 0, 1, \dots, n$ . Each line corresponds to one gene. Values close to 0 correspond to rare usage of the gene, values close to 1 correspond to frequent usage of the gene in the individuals of the population, values close to 0.5 denote that the gene is set to 0 in approximately a half of the population and is set to 1 in the remaining



**Fig. 8.2.** Frequencies of usage  $\alpha_i^{(k)}$  of particular genes,  $i = 1, 2, \dots, d$ , in successive iterations,  $k = 0, 1, \dots, n$  ( $d = 250, n = 200$ )

individuals. Clearly, after a number of iterations, gene values stabilise and most of genes tend either to be used in most of individuals in the population or not used in any.

More detailed experiments showed that sets of well-stabilised genes in the last iteration of the Simple Genetic Algorithm are similar for successive iterations of the decision support system. For instance, Table 8.1 presents the comparison of 5 sets built from 200 best-stabilised genes (of all 250 genes in total) in the last iteration of the Simple Genetic Algorithm for 5 successive iterations of the decision support system. Although, it is easy to see that they are similar, which suggests that frequent knowledge patterns built on the basis of these trading experts would be stable and also large enough to reasonably

**Table 8.1.** Comparison of 5 sets of 200 best-stabilised genes (of all 250 genes in total) in trading experts built in 5 successive iterations of the decision support system (each element  $m_{ij}$  of the matrix corresponds to the number of trading rules from the  $i$ -th set occurring also in the  $j$ -th set)

	1	2	3	4	5
1	200	191	183	172	159
2	191	200	193	182	174
3	183	193	200	191	184
4	172	182	191	200	189
5	159	174	184	189	200

reduce the search space, further studies are necessary to estimate the proper number of previous trading experts used for extraction of the frequent knowledge pattern.

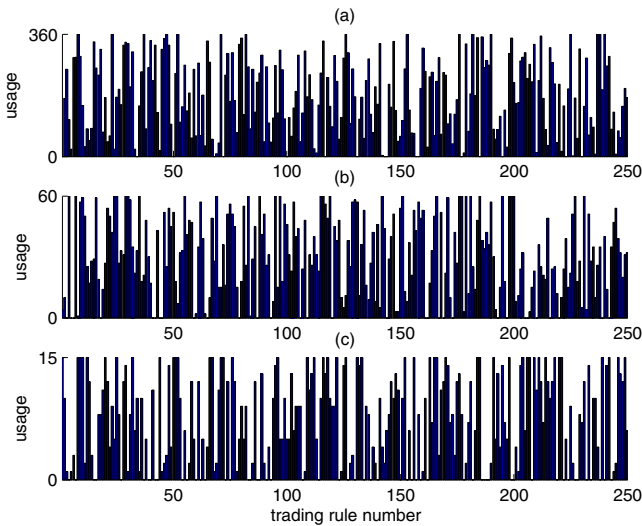
### 8.6.2 Extraction of Frequent Knowledge Patterns

In order to study the relation between the number of previous trading experts used in the extraction and the order of the frequent knowledge pattern obtained, long sequences of trading experts built in successive iterations of the decision support system were considered. Each sequence considered a time period of 360 minutes, where a new trading expert was built in each minute, even if the previous one was efficient enough. All the new trading experts were built independently without reusing any information from previous ones.

Figure 8.3 (a) presents usage of 250 trading rules in 360 trading experts built for successive minutes of a time period of 360 minutes. Each bar corresponds to one trading rules and depicts the number of trading experts. It is easy to see that there is a number of frequently chosen trading rules as well as a number of infrequently chosen trading rules.

Unfortunately, only 10% to 15% of trading rules were either constantly used, or not used at all, in trading experts built during time periods of 360 minutes, which limits the order of the 360-minute frequent knowledge patterns to 25–40 and which does not lead to a large reduction of the search space and the computing time. In order to obtain frequent knowledge patterns of higher order, shorter time periods were considered.

Figures 8.3 (b) and (c) present usage of 250 trading rules in trading experts built for successive minutes of a time period of 60 and 15 minutes, respectively. 60-minute



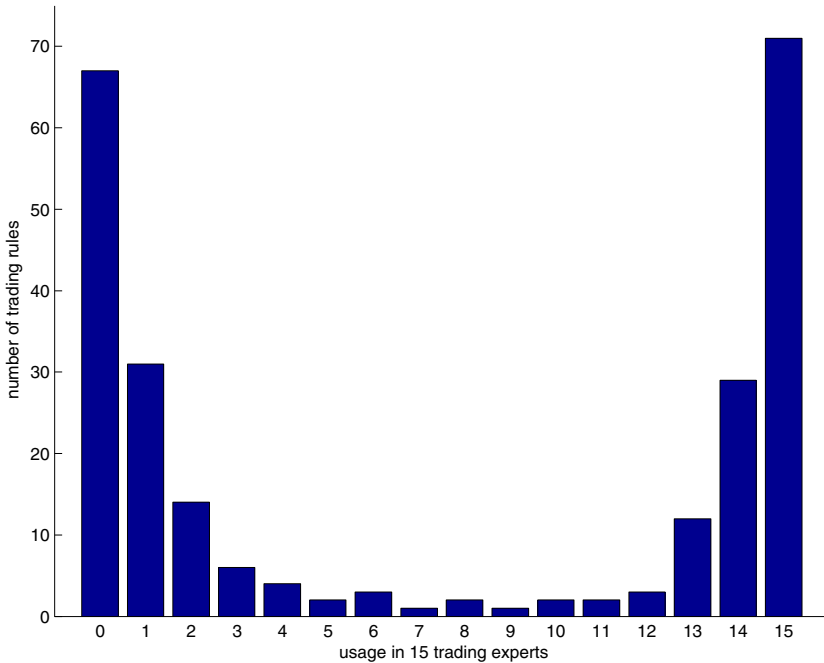
**Fig. 8.3.** Usage of 250 trading rules in trading experts built for successive minutes of a time period of 360, 60 and 15 minutes (a, b, and c, respectively)

**Table 8.2.** Usage of 250 trading rules in trading experts built for successive minutes of a time period of 360, 60, 15, 10 and 5 minutes (orders and coverages of 360-, 60-, 15-, 10- and 5-minute frequent knowledge patterns)

	order	coverage
360-minute frequent knowledge patterns	32	13%
60-minute frequent knowledge patterns	60	24%
15-minute frequent knowledge patterns	98	39%
10-minute frequent knowledge patterns	145	58%
5-minute frequent knowledge patterns	207	83%

frequent knowledge patterns included approximately 24% of trading rules, while 15-minute frequent knowledge patterns included approximately 39% of trading rules. In additional experiments, 10-minute and 5-minute frequent knowledge patterns covered 58% and 83% of trading rules, respectively. Table 8.2 presents a brief summary of results.

Clearly, shorter time periods lead to frequent knowledge patterns of higher order and consequently to better optimisation of the original algorithm, assuming that the trading experts built with frequent knowledge patterns would be efficient enough. In



**Fig. 8.4.** Histogram of usage of 250 trading rules in trading experts built in successive minutes of a time period of 15 minutes

**Table 8.3.** Excess of the return rate of the trading strategies defined by trading experts built in both decision support systems over the return rate of the B&H strategy

Market Condition	Return over B&H	
	with patterns	without patterns
extremely positive B&H, i.e. $0.01 \leq B\&H$	0.0323% $\pm$ 0.0051	0.0412% $\pm$ 0.0013
positive B&H, i.e. $0.00 \leq B\&H < 0.01$	0.0225% $\pm$ 0.0033	0.0268% $\pm$ 0.0024
negative B&H, i.e. $-0.01 \leq B\&H < 0.00$	0.0265% $\pm$ 0.0075	0.0283% $\pm$ 0.0082
extremely negative B&H, i.e. $B\&H < -0.01$	0.0142% $\pm$ 0.0029	0.0157% $\pm$ 0.0015

practice, the reasonable trade off between the optimisation of the original algorithm and the efficiency of solutions was achieved with 15-minute frequent knowledge patterns.

### 8.6.3 15-Minutes Frequent Knowledge Patterns

Further studies concerned 15-minute frequent knowledge patterns. Figure 8.4 presents a histogram of usage of 250 trading rules in trading experts built in successive minutes of a time period of 15 minutes. The first bar corresponds to 71 trading rules which are not included in any trading expert. The last bar corresponds to 67 trading rules which are included in each trading expert. In total, these 138 trading rules form the 15-minute frequent knowledge pattern.

Introducing the 15-minute frequent knowledge patterns to the decision support system, described in Algorithm 8.3, led to a significant reduction of the search space. New trading experts were built by evolving only the part of the chromosome not covered by the knowledge pattern (usually, about 150 genes). Naturally, such an approach could result in premature convergence of the evolutionary algorithm and finding local solutions instead of global ones, so when the efficiency of the new trading experts decreased excessively, new ones were built using the original algorithm evolving the entire chromosome.

In order to study the financial aspect of the approach proposed, the trading strategies defined by trading experts built using frequent knowledge patterns were compared with the original trading strategies defined by trading experts built without reusing any information from previous trading experts in terms of profitability. 40 experiments were carried out. Each experiment concerned a different stock and a different time period of 30 minutes. In each experiment, for each minute, two trading experts were built – one using knowledge patterns and one using the original algorithm.

Table 8.3 presents the excess of the return rate of the trading strategies over the return rate of the simple Buy-and-Hold (B&H) strategy, which consists in investing all the capital in stocks at the start of the time period and keeping it until the end of the time period. Results usually depend on the stock market conditions, so experiments were divided into 4 groups according to the state of the stock market, defined by ranges of B&H values over the time period.

Although results depend on the stock market conditions, in most cases, the modified algorithm seems not to be worse than the original algorithm and both trading strategies usually outperforms the B&H strategy.

## 8.7 Conclusions

This chapter discussed extracting and reusing frequent knowledge patterns in building trading experts in evolutionary decision support systems for financial time series analysis. It focused on trading experts built by an evolutionary algorithm from a specific set of trading rules, where frequent knowledge patterns corresponded to common building blocks of trading rules occurring in most previous trading experts. Reusing frequent knowledge patterns led to a significant reduction in the search space (in experiments, from  $2^{250}$  to about  $2^{150}$ ) without significant decreases in objective function values.

In general, results of experiments were encouraging. Many tests showed that the knowledge pattern captured a subset of an already successful group of trading rules applied to a recurring trading problem arisen within a certain context. Decreasing the time of trading expert building significantly improved the overall throughput of the decision support system.

At the moment, the approach uses 15-minute frequent knowledge patterns extracted from 15 previous trading experts coming from the last 15 minutes. Although it is tempting to shorten the time horizon to 10 or even 5 minutes, which would lead to larger knowledge patterns, this may provoke premature convergence of the evolutionary algorithm, so this may also require additional studies on mutation operators capable of diversifying the population and avoiding focusing on a small part of the search space around local maxima.

On the other hand, one can analyse knowledge patterns occurring over longer time horizon in order to promote or eliminate trading rules that are permanently chosen or rejected. In this case, it would appear to be necessary to develop a strategy which would allow a trading rule to go back to the previous status after a certain time. Otherwise, a rule might be eliminated forever, even if it would turn out to be efficient in the future.

It is also worth noticing that relaxing the definition of frequent knowledge patterns and accepting in knowledge patterns also the trading rules that occur or not occur in majority of trading experts, but not in all of them, may lead to further increases in orders of frequent knowledge patterns, even over shorter time horizons.

Reduction of the computing time enables further development of the decision support system. Currently, the data aggregation period is equal to one minute, which implies that the system generates trading decisions every minute. Since the decision making process has been accelerated, it is tempting to lessen the aggregation period so that the system will react faster, for instance, every 30 or 15 seconds.

## References

1. Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: Proc. of the International Conference on Data Engineering, ICDE, Taipei, Taiwan (1995)
2. Allen, F., Karjalainen, R.: Using Genetic Algorithms to Find Technical Trading Rules. *Journal of Financial Economics* 51, 245–279 (1999)
3. Bauer, R.: *Genetic Algorithms and Investment Strategies*. Wiley, Chichester (1994)
4. Brabazon, A., O'Neill, M.: *Biologically Inspired Algorithms for Financial Modelling*. Springer, Heidelberg (2006)

5. Colby, W., Meyers, T.: *The Encyclopedia of Technical Market Indicators*. Down Jones-Irwin (1990)
6. Dempster, M., Jones, C.: A Real-Time Adaptive Trading System using Genetic Programming. *Quantitative Finance* 1, 397–413 (2001)
7. Dempsey, I., O'Neill, M., Brabazon, A.: Adaptive Trading with Grammatical Evolution. In: *Proc. of the 2006 Congress on Evolutionary Computation (CEC 2006)*, pp. 2587–2592. IEEE, Los Alamitos (2006)
8. Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading (1989)
9. Korczak, J., Lipinski, P., Roger, P.: Evolution Strategy in Portfolio Optimization. In: Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M. (eds.) *EA 2001*. LNCS, vol. 2310, pp. 156–167. Springer, Heidelberg (2002)
10. Korczak, J., Lipinski, P.: Evolutionary Building of Stock Trading Experts in a Real-Time System. In: *Proc. of the 2004 Congress on Evolutionary Computation (CEC 2004)*, pp. 940–947. IEEE, Los Alamitos (2004)
11. Lipinski, P.: Discovering Stock Market Trading Rules using Multi-Layer Perceptrons. In: Sandoval, F., Prieto, A.G., Cabestany, J., Graña, M. (eds.) *IWANN 2007*. LNCS, vol. 4507, pp. 1114–1121. Springer, Heidelberg (2007)
12. Lipinski, P.: ECGA vs. BOA in Discovering Stock Market Trading Experts. In: *Proc. of Genetic and Evolutionary Computation Conference, GECCO 2007*, pp. 531–538. ACM, New York (2007)
13. Lipinski, P., Korczak, J.: Performance Measures in an Evolutionary Stock Trading Expert System. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) *ICCS 2004*. LNCS, vol. 3039, pp. 835–842. Springer, Heidelberg (2004)
14. Loraschi, A., Tettamanzi, A.: An Evolutionary Algorithm for Portfolio Selection within a Downside Risk Framework. In: Dunis, C.L. (ed.) *Forecasting Financial Markets*, pp. 275–286. Wiley, Chichester (1996)
15. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, New York (1994)
16. Moody, J., Wu, L., Liao, Y., Saffell, M.: Performance Function and Reinforcement Learning for Trading Systems and Portfolios. *Journal of Forecasting* 17(56), 441–470 (1998)
17. Murphy, J.: *Technical Analysis of the Financial Markets*, NUIF (1998)
18. Neely, C., Weller, P., Dittmar, R.: Technical Analysis in the Foreign Exchange market Profitable? A Genetic Programming Approach. *Journal of Financial Quantitative Analysis* 32, 405–426 (1997)
19. Sharpe, W.: Capital Asset Prices: A Theory of Market Equilibrium under Conditions of Risk. *Journal of Finance* 19, 425–442 (1964)
20. Tsang, E., Li, J., Markose, S., Er, H., Salhi, A., Iori, G.: EDDIE in Financial Decision Making. *Journal of Management and Economics* 4(4) (November 2000)



---

# Modeling Turning Points in Financial Markets with Soft Computing Techniques

Antonia Azzini, Célia da Costa Pereira, and Andrea G.B. Tettamanzi

Università degli Studi di Milano, Dipartimento di Tecnologie dell'Informazione  
via Bramante 65, I-26013 Crema, Italy  
antonia.azzini, celia.pereira, andrea.tettamanzi@unimi.it

**Summary.** Two independent evolutionary modeling methods, based on fuzzy logic and neural networks respectively, are applied to predicting trend reversals in financial time series of the financial instruments S&P 500, crude oil and gold, and their performances are compared. Both methods are found to give essentially the same results, indicating that trend reversals are partially predictable.

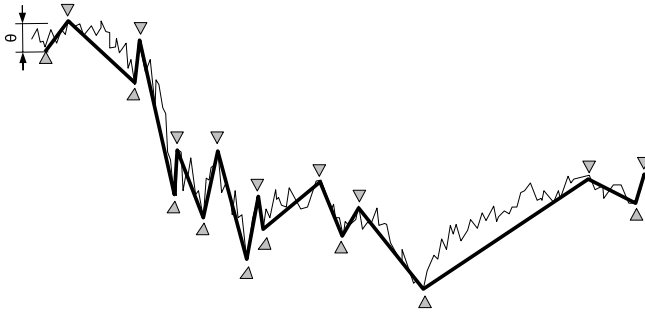
## 9.1 Introduction

Even the most casual observer of a financial time series will notice that prices of financial instruments move up and down [11]; furthermore, this behaviour happens and can be observed at all scales [13]. However, price movements are not regular and look unpredictable. In general, market action consists of alternating up-trends and down-trends, separated by turning points, which correspond to maxima and minima. A trader able to buy at minima and sell at maxima, i.e., trade exactly at the turning points, would gain the maximum profit possible. For this reason, the main objective of financial market forecasting techniques is to call turning points consistently and correctly. Two approaches for calling turning points in a financial time series are possible:

- reveal turning points when they occur, or just after they have occurred — this is what most technical analysis indicators are all about, starting from simple moving averages to the most sophisticated indicators;
- predict the price at which the next turning point will most likely occur — this is the approach we follow in this chapter.

To do that, we summarize the past history of the series up to the last confirmed turning point by applying a noise-eliminating filter. The output of the filter is given as input to a predictive model, whose output provides an estimate of the price at which the next turning point is going to happen.

Biologically inspired methods have become extremely popular as tools for modeling financial markets [6, 7]; these include evolutionary algorithms (EAs), neural networks (NNs), and fuzzy logic. In this chapter we employ two types of models, namely fuzzy rule bases, i.e., sets of fuzzy IF-THEN rules, and feedforward neural networks. Both types of models are designed and optimized by means of evolutionary algorithms. The



**Fig. 9.1.** A schematic illustration of the construction of the zig-zag indicator. Significant turning points (tops and bottoms) are indicated by triangles. The original price chart is the light line; the thick trendlines are the swings that make up the zig-zag.

performance of models of the two types are compared with the aim of assessing which one is more suited to the task.

The chapter is organized as follows: Section 9.2 states the problem, Sections 9.3 and 9.4 provide a description of the two modeling methods, and Section 9.5 reports the experiments and discusses their results. Section 9.6 concludes.

## 9.2 Problem Description

The *zig-zag* indicator [1, 14] is essentially used to help individuating changes by highlighting the most significant reversals and by filtering out changes less than a specified amount. Such an indicator is a series of trendlines that connect significant tops and bottoms on a price chart, as illustrated by Figure 9.1. The minimum price reversal parameter  $\theta$  specifies the percentage that the price must trace back from a top or bottom in order to form a new “zig” or “zag” line.

The time series of prices of the financial instrument under study is pre-processed by applying the *zig-zag* filter with a threshold  $\theta$  — the reversal amount set by the user. The threshold is used as follows. If a reversal in price trend fails to follow through to  $\theta$ , zig-zag will revise itself. The apparent upswing or downswing will be filtered out and the previous trend will appear to continue without interruption. Once the price reversal has reached or exceeded the threshold, zigzag’s last leg is no longer revisable.

Such a filter is used primarily to help us see changes by both punctuating the most significant reversals and to eliminate noise by filtering out small and unimportant price movements. This is done by decomposing the input time series into a series of alternating up- and down-swings. The length of each swing which is given by the price difference of its ends is divided by the length of the previous swing. The output resulting from the noise-eliminating filter is composed by the series of the resulting ratios.

The problem of predicting the price at which the next turning point will occur can thus be transformed into the problem of predicting the next ratio of the series of ratios of swing lengths, since the price of a turning point  $x_{t+1}$  can be calculated by knowing the price of the two previous turning points  $x_{t-1}$  and  $x_t$  and the ratio  $r_t = \frac{|x_{t+1} - x_t|}{|x_t - x_{t-1}|}$ . The

working hypothesis is that such prediction can be done at any time by considering the  $n$  most recent ratios. In other words, we are searching the space of all autoregressive models of  $\{r_t\}_t$  of order  $n$ .

A baseline for any model is provided by the simplest model possible, namely a model that always predicts  $E[r_t]$ , without taking the last  $n$  ratios into account.  $E[r_t]$  can be estimated by taking the longest available time series for the financial instrument considered and computing the average ratio of a swing length to the length of the previous swing. Such a model has a mean absolute error  $E[|r_t - E[r_t]|]$  and a mean square error  $E[(r_t - E[r_t])^2] = \text{Var}[r_t]$ , while the relative mean absolute error corresponds to  $E[|r_t - E[r_t]|]/E[r_t]$ . This, by the way, is the best performance one would expect from a predictive model if the time series under observation were completely random or, which is roughly equivalent [13], if the market were perfectly efficient.

### 9.3 Fuzzy Rule Base Optimization

Data mining is a process aimed at discovering meaningful correlations, patterns, and trends between large amounts of data collected in a dataset. This process is able to tell us important features that we didn't know (or we could not see) or what is going to happen next. This technique is called modeling, that is the act of building a model of a situation where the answer is known and then applying that model to another situation where the answer is unknown. In our case, a model is determined by observing past behaviour of a financial instrument and extracting the relevant variables and correlations between the data and the dependent variable. More precisely, model is described through a set of fuzzy rules, made by one or more antecedent clauses ("IF . . .") and a consequent clause ("THEN . . ."). Clauses are represented by a pair of indices referring respectively to a variable and to one of its fuzzy sub-domains, i.e., a membership function.

Using fuzzy rules makes it possible to get homogenous predictions for different clusters without imposing a traditional partition based on crisp thresholds, that often do not fit the data, particularly in financial applications. Fuzzy decision rules are useful in approximating non-linear functions because they have good interpolative power and are intuitive and easily intelligible at the same time. Their characteristics allow the model to give an effective representation of the reality and simultaneously avoid the "black-box" effect of, e.g., neural networks.

The intelligibility of the model is useful for a trader, because understanding the rules helps the user to judge if a model can be trusted.

#### 9.3.1 Fuzzy Rule-Based Systems

A prominent role in the application of fuzzy logic to real-world problems is played by fuzzy rule-based systems. Fuzzy rule-based systems are systems of fuzzy rules that embody expert knowledge about a problem, and can be used to solve it by performing fuzzy inferences.

The ingredients of a fuzzy rule-based systems are *linguistic variables*, *fuzzy rules*, and *defuzzification* methods.

### Linguistic Variables

A linguistic variable [23] is defined on a numerical interval and has linguistic values, whose semantics is defined by their membership function. For example, a linguistic variable *temperature* might be defined over the interval  $[-20^{\circ}\text{C}, 50^{\circ}\text{C}]$ ; it could have linguistic values like *cold*, *warm*, and *hot*, whose meanings would be defined by appropriate membership functions.

### Fuzzy Rules

A fuzzy rule is a syntactic structure of the form

$$\text{IF antecedent THEN consequent,} \tag{9.1}$$

where each *antecedent* and *consequent* are formulas in fuzzy logic.

Fuzzy rules provide an alternative, compact, and powerful way of expressing functional dependencies between various elements of a system in a modular and, most importantly, intuitive fashion.

### Inference in Fuzzy Rule-Based Systems

The semantics of a fuzzy rule-based system is governed by the calculus of fuzzy rules [24]. In summary, all rules in a fuzzy rule base take part simultaneously in the inference process, each to an extent proportionate to the truth value associated with its antecedent. The result of an inference is represented by a fuzzy set for each of the dependent variables. The degree of membership for a value of a dependent variable in the associated fuzzy set gives a measure of its compatibility with the observed values of the independent variables.

Given a system with  $n$  independent variables  $x_1, \dots, x_n$  and  $m$  dependent variables  $y_1, \dots, y_m$ , let  $R$  be a base of  $r$  fuzzy rules

$$\begin{aligned} &\text{IF } P_1(x_1, \dots, x_n) \text{ THEN } Q_1(y_1, \dots, y_m), \\ &\quad \vdots \qquad \qquad \qquad \vdots \\ &\text{IF } P_r(x_1, \dots, x_n) \text{ THEN } Q_r(y_1, \dots, y_m), \end{aligned} \tag{9.2}$$

where  $P_1, \dots, P_r$  and  $Q_1, \dots, Q_r$  represent fuzzy predicates respectively on independent and dependent variables, and let  $\tau_P$  denote the truth value of predicate  $P$ . Then the membership function describing the fuzzy set of values taken up by dependent variables  $y_1, \dots, y_m$  of system  $R$  is given by

$$\begin{aligned} &\tau_R(y_1, \dots, y_m; x_1, \dots, x_n) \\ &= \sup_{1 \leq i \leq r} \min\{\tau_{Q_i}(y_1, \dots, y_m), \tau_{P_i}(x_1, \dots, x_n)\}. \end{aligned} \tag{9.3}$$

### The Mamdani Model

The type of fuzzy rule-based system just described, making use of the min and max as the triangular norm and co-norm, is called the Mamdani model. A Mamdani system [12] has rules of the form

$$\text{IF } x_1 \text{ is } A_1 \text{ AND } \dots \text{ AND } x_n \text{ is } A_n \text{ THEN } y \text{ is } B, \quad (9.4)$$

where the  $A_i$ s and  $B$  are linguistic values (i.e., fuzzy sets) and each clause of the form “ $x$  is  $A$ ” has the meaning that the value of variable  $x$  is in fuzzy set  $A$ .

### Defuzzification Methods

There may be situations in which the output of a fuzzy inference needs to be a crisp number  $y^*$  instead of a fuzzy set  $R$ . Defuzzification is the conversion of a fuzzy quantity into a precise quantity.

At least seven methods in the literature are popular for defuzzifying fuzzy outputs [10], which are appropriate for different application contexts. The *centroid method* is the most prominent and physically appealing of all the defuzzification methods. It results in a crisp value

$$y^* = \frac{\int y\mu_R(y)dy}{\int \mu_R(y)dy}, \quad (9.5)$$

where the integration can be replaced by summation in discrete cases.

The next section introduces evolutionary algorithms, a biologically inspired technique which we use to learn and optimize fuzzy rule bases. We describe below a data-mining approach based on the use of EAs, which recognize patterns within a dataset, by learning models represented by sets of fuzzy rules.

### 9.3.2 The Evolutionary Algorithm

The described approach incorporates an EA for the design and optimization of fuzzy rule-based systems originally developed to learn fuzzy controllers [17, 16], then adapted for data mining, which has already been used for financial modeling by two of the authors [9].

A model is a rule base, whose rules comprise up to four antecedent and one consequent clause each. Input and output variables are partitioned into up to 16 distinct linguistic values each, described by as many membership functions. Membership functions for input variables are trapezoidal, while membership functions for the output variable are triangular. Models are encoded in three main blocks:

1. a set of trapezoidal membership functions for each input variable; a trapezoid is represented by four fixed-point numbers;
2. a set of symmetric triangular membership functions, represented as an area-center pair, for the output variable;
3. a set of rules, where a rule is represented as a list of up to four antecedent clauses (the IF part) and one consequent clause (the THEN part); a clause is represented by a pair of indices, referring, respectively, to a variable and to one of its membership functions.

An island-based distributed EA is used to evolve models. Island-based approaches are useful in harnessing distributed computing power and maintaining solution diversity and thereby reducing the chance of premature convergence of the population.

The sequential algorithm executed on every island is a standard generational replacement, elitist EA. Crossover and mutation are never applied to the best individual in the population.

The recombination operator is designed to preserve the syntactic legality of models. A new model is obtained by combining the pieces of two parent models. Each rule of the offspring model can be inherited from one of the parent models with probability 1/2. When inherited, a rule takes with it to the offspring model all the referred domains with their membership functions. Other domains can be inherited from the parents, even if they are not used in the rule set of the child model, to increase the size of the offspring so that their size is roughly the average of its parents' sizes.

Like recombination, mutation produces only legal models, by applying small changes to the various syntactic parts of a fuzzy rulebase.

Migration is responsible for the diffusion of genetic material between populations residing on different islands. At each generation, with a small probability (the migration rate), a copy of the best individual of an island is sent to all connected islands and as many of the worst individuals as the number of connected islands are replaced with an equal number of immigrants. A detailed description of the algorithm and of its genetic operators can be found in [17].

## 9.4 Neuro Genetic Optimization

The second evolutionary approach [3, 4, 5] considered in this work, evolves a population of neural networks. It is based on a particular type of evolving systems, namely neuro-genetic systems, which have become a very important topic of study in neural network design. They make up so-called Evolutionary Artificial Neural Networks (EANNs) [18, 19, 20], i.e., biologically-inspired computational models that use evolutionary algorithms in conjunction with neural networks in a synergetic way.

Several approaches presented in the literature have been developed to apply evolutionary algorithms to neural network design [2]. Some consider the setting of the weights in a fixed topology network. Others optimize network topologies, or evolve the learning rules, the input feature selection, or the transfer function used by the network. Several systems also allow an interesting conjunction of the evolution of network architecture and weights, carried out simultaneously.

Important aspects of such a simultaneous evolution underline that an evolutionary algorithm allows all aspects of a neural network design to be taken into account at once, without requiring any expert knowledge of the problem. Furthermore, the conjunction of weights and architecture evolution overcomes the possible drawbacks of each single technique and combines their advantages. The main advantage of weight evolution, for example, is to simulate the learning process of a neural network, avoiding the drawbacks of the traditional gradient descent techniques, such as the backpropagation algorithm (BP). Generally, the ANN architecture design follows some performance optimality criteria, regarding, for example, the setting of two network parameters, like the learning rate and the momentum. The first affects the speed at which the ANN arrives at the minimum solution, and it is analogous to the step-size parameter of a gradient-descent algorithm when the BP is used.

The second is used to prevent the system from converging to local minima. A high value of this parameter can help to increase the speed of convergence of the system. However, if too high, it could create a risk of overshooting the minimum, causing the system to become unstable. Similarly, if too low, it could slow down the training of the system.

The performance level of such network's parameters forms a surface in the design space. The advantage of such a representation is that determining the optimal architecture design is equivalent to finding the highest point on this surface.

The simultaneous evolution of architecture and weights limits the negative effects of a noisy-fitness evaluation in a NN structure optimization, by defining a one-to-one mapping between genotypes and phenotypes of the individuals.

### 9.4.1 The Neuro-genetic Approach

The neuro-genetic approach considered in this financial application restricts the attention to a specific subset of feedforward neural networks, namely Multi-Layer Perceptrons (MLPs), which make use of advantages like the ability of EAs and NNs to learn and generalize, reduced dataset dimensions and computational time requirements, ease of implementation and the use of simple structures.

The approach can be considered as a hybrid algorithm. The basic idea is to exploit the ability of the EA to find a solution close enough to the global optimum, together with the ability of a gradient descent technique, the Backpropagation algorithm (BP), to finely tune a solution and reach the nearest local minimum. BP is not able to find a global minimum if the error function is multimodal and/or non-differentiable, but it becomes useful when the minimum of the error function currently found is close to a solution but not close enough to solve the problem. The adaptive nature of NN learning by examples is a very important feature of these methods, and the training process modifies the weights of the ANN, in order to improve a pre-defined performance criterion, that corresponds to an objective function over time. In several methods to train neural networks, BP has emerged as a suitable solution for finding a set of good connection weights and biases.

In the overall evolutionary process the population is randomly created, initialized, and the genetic operators are then applied to each network until termination conditions are satisfied. The idea proposed in this work is close to the solution presented in EPNet [20]: a new evolutionary system for evolving feedforward ANNs, that puts emphasis on evolving ANN's behaviours. This neuro-genetic approach evolves ANN's architecture and connection weights simultaneously, just like EPNet, in order to reduce noise in fitness evaluation. Close behavioural link between parents and offspring is maintained by applying the genetic operators and a partial training, in order to reduce behavioural disruption. The genetic operators include selection and two kinds of mutations, that are applied, respectively, to the weights and to the network topologies. Mutation is applied before the training process; in particular, weight mutation is carried out before topology mutation, in order to perturb the connection weights of the neurons in a neural network. After each weight mutation, a weight check is carried out, in order to delete neurons whose contribution is negligible with respect to the overall network output. This allows

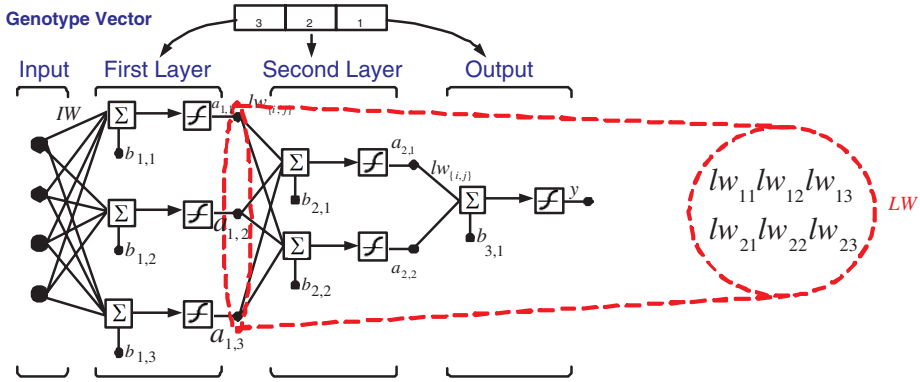


Fig. 9.2. Example of an individual.

us to obtain, if possible, a reduction of the computational cost of the entire network before any structure mutation.

In the literature it is well known that recombination of neural networks of arbitrary structure is a very hard issue, due to the detrimental effect of the permutation problem. No satisfactory solutions have been proposed so far. As a matter of fact, the most successful approaches to neural network evolution do not use recombination at all [20]. Therefore, in this approach the crossover operator is not applied, due to the disruptive effects it could have on the neural models, after the cut and recombination processes on the network structures of the selected parents.

### 9.4.2 Individual Encoding

As described in detail in [3], individuals are not constrained to a pre-established topology, and the population is initialized with different hidden layer sizes and different numbers of neurons for each individual, according to two exponential distributions, in order to maintain diversity among all the individuals in the new population. Such dimensions are not bounded in advance, even though the fitness function may penalize large networks. The number of neurons in each hidden layer is constrained to be greater than or equal to the number of network outputs, in order to avoid *hourglass structures*, whose performance tends to be poor. Indeed, a layer with fewer neurons than the outputs destroys information which later cannot be recovered. An example of an individual encoded through a neural network is shown in Figure 9.2. The genotype vector represents the number of elements defining the hidden layers and the output.

Each individual is encoded in a structure in which basic information are maintained as illustrated in Table 9.1. A normal distribution is applied to determine the weights, the biases, and variance matrices. The latter are initialized to one for all individuals and are applied in conjunction with evolutionary strategies in order to perturb network weights and biases.

The values of all the parameters are affected by the genetic operators during evolution, in order to perform incremental (adding hidden neurons or hidden layers) and decremental (pruning hidden neurons or hidden layers) learning.



**Table 9.1.** Individual Representation.

Element	Description
$l$	Length of the topology string, corresponding to the number of layers.
topology	String of integer values that represent the number of neurons in each layer.
$\mathbf{W}^{(0)}$	Weights matrix of the input layer neurons of the network.
$\mathbf{Var}^{(0)}$	Variance matrix of the input layer neurons of the network.
$\mathbf{W}^{(i)}$	Weights matrix for the $i$ th layer, $i = 1, \dots, l$ .
$\mathbf{Var}^{(i)}$	Variance matrix for the $i$ th layer, $i = 1, \dots, l$ .
$b_{ij}$	Bias of the $j$ th neuron in the $i$ th layer.
$\text{Var}(b_{ij})$	Variance of the bias of the $j$ th neuron in the $i$ th layer.

### 9.4.3 The Evolutionary Process

The general framework of the evolutionary process can be described by the following pseudo-code. Individuals in a population compete and communicate with other individuals through genetic operators applied with independent probabilities, until termination conditions are satisfied.

1. Initialize the population by generating new random individuals.
2. Create for each genotype the corresponding MLP, and calculate its cost and its fitness values.
3. Save the best individual as the best-so-far individual.
4. While not termination condition do
  - a) Apply the genetic operators to each network.
  - b) Decode each new genotype into the corresponding network.
  - c) Compute the fitness value for each network.
  - d) Save statistics.

In each new generation a new population has to be created, and the first half corresponds to the best parents that have been selected with the truncation operator, while the second part of the new population is defined by creating offspring from the previously selected parents. Then, each individual of the new population is mutated, trained and the new corresponding fitness function is calculated, in order to determine the new best one. Genetic operators are applied to each network as follows:

1. Select from the population (of size  $n$ )  $\lfloor n/2 \rfloor$  individuals by truncation and create a new population of size  $n$  with copies of the selected individuals.
2. For all individuals in the population:
  - a) Mutate the weights and the topology of the offspring.
  - b) Train the resulting network using the training set.
  - c) Calculate  $f$  on the test set (see Section 9.4.3).
  - d) Save the individual with lowest  $f$  as the best-so-far individual if the  $f$  of the previously saved best-so-far individual is higher (worse).
3. Save statistics.

For each generation of the population all information of the best individual is saved.

### Selection

The selection method implemented in this work is taken from the breeder genetic algorithm [8], and differs from natural probabilistic selection in that evolution considers

only the individuals that best adapt to the environment. Elitism is also used, allowing the best individual to survive unchanged in the next generation and solutions to monotonically get better over time.

The selection strategy implemented is truncation. It is not a novel solution, indeed, several evolutionary approaches described this operator in order to prevent the population from remaining too static and perhaps not evolving at all. Moreover, this kind of selection is a very simple technique and produces satisfactory solutions through conjunction with other strategies (like elitism).

## Mutation

The aim of this operator is to introduce new genetic material and to maintain diversity in the population. Generally, the purpose of mutation is to simulate the effect of transcription errors that can occur with a very low probability, the mutation rate, when a chromosome is duplicated. The evolutionary process applies two kinds of neural network perturbations:

- *Weights mutation*, that perturbs the weights of the neurons before performing any structural mutation and applying BP. This kind of mutation defines a Gaussian distribution for the Variance matrix values  $\mathbf{Var}^{(i)}$  of each network weight  $\mathbf{W}^{(i)}$ , defined in Table 9.1. This solution is similar to the approach implemented by Schwefel [15], who defined *evolution strategies*, algorithms in which the strategy parameters are proposed for self-adapting the mutation concurrently with the evolutionary search. The main idea behind these strategies, moreover considered in this approach, is to allow a control parameter, like mutation variance, to self-adapt rather than changing their values by some deterministic algorithms. Evolution strategies perform very well in numerical domains, since they are dedicated to (real) function optimization problems.

This kind of mutation offers a simplified method for self-adapting the variance matrix  $\mathbf{Var}_j^{(i)}$ , whose values are defined as log-normal perturbations of their parent parameter values.

- *Topology mutation* is defined with four types of mutation by considering neurons and layer addition and elimination. It is implemented after weight mutation because a perturbation of weight values changes the behaviour of the network with respect to the activation functions; in this case, all neurons whose contribution becomes negligible with respect to the overall behaviour, will be deleted from the structure. The addition and the elimination of a layer and the insertion of a neuron are applied with independent probabilities, corresponding respectively to three algorithm parameters  $p_{\text{layer}}^+$ ,  $p_{\text{layer}}^-$  and  $p_{\text{neuron}}^+$ . These parameters are set at the beginning and maintained unchanged during the entire evolutionary process. Anyway, such topology mutation operators are aimed at minimizing their impact on the behaviour of the network; in other words, they are designed to be as little disruptive, and as much neutral, as possible, preserving the behavioural link between the parent and the offspring better than by adding random nodes or layers.

## Fitness Function

An important aspect that has to be considered in the overall evolutionary process is that the depth of the network structure could in principle increase without limits under the influence of some of the topology mutation operators, defining a so-called bloating effect. In order to avoid this problem some penalization parameters are introduced in the fitness function in order to control the structure growth, reducing the corresponding computational cost.

Then, the fitness of an individual depends both on its accuracy (i.e., its mse, the mean square error) and on such cost. Although it is customary in EAs to assume that better individuals have higher fitness, the convention that a lower fitness means a better NN is adopted in this work. This maps directly the objective function into a cost minimization problem, that is defined as:

$$f = \lambda kc + (1 - \lambda) * mse, \quad (9.6)$$

where  $\lambda$  corresponds to the desired tradeoff between network cost and accuracy, and has been set to 0.2 after some preliminary experiments.  $k$  is a scaling constant set to  $10^{-6}$ , and  $c$  models the computational cost of a neural network, defined by

$$c = \alpha N_{hn} + \beta N_{syn}, \quad (9.7)$$

where  $N_{hn}$  is the number of hidden neurons,  $N_{syn}$  is the number of synapses, and  $\alpha = 2$  and  $\beta = 4$  represent, respectively, the costs of each hidden neuron and of each synapse. This term has been introduced to keep the demand of computational resources at a reasonable level by penalizing large networks. In this work we have assumed these values in order to put more emphasis the number of neural network synapses. The fitness is calculated according to Equation 9.6 over the test set.

## 9.5 Experiments and Results

To empirically assess the extent to which turning points in financial time series are predictable, we have applied the two evolutionary modeling methods described above in Sections 9.3 and 9.4 to predicting turning points for three very diverse financial instruments, namely the S&P 500 stock index, a crude oil future, and gold.

First of all, we would like to stress the fact that, although they both rely on evolutionary algorithms for optimization, the two modeling methods are as different from each other as two modeling methods can be, in that they use diametrically opposed languages to express their models: on one side, we have linguistic, symbolic models, expressed in the language of IF-THEN rules; on the other side, we have connectionist, subsymbolic models, expressed as multilayer perceptrons.

Furthermore, the three financial instruments under study have been selected to be heterogeneous and representative of different types of markets:

- the S&P 500 is a value-weighted index of the prices of 500 large-cap common stocks actively traded in the United States, to date still the strongest economy of the world; the S&P 500 is one of the most widely followed indices of large-cap American stocks and is considered a bellwether for the American economy;

**Table 9.2.**  $E[r_t]$ ,  $Var[r_t]$  and relative error values obtained from different settings of  $\theta$  for each financial instrument.

Instrument	$\theta$	$E[r_t]$	$Var[r_t]$	Relative Error
S&P 500	0.01	1.2918	1.1062	0.5586
	0.02	1.2418	0.6895	0.4962
crude oil	0.005	1.3811	1.6176	0.6494
	0.01	1.3199	1.2811	0.5864
gold	0.005	1.2916	1.0146	0.5680
	0.01	1.2815	1.0781	0.5622

- crude oil is perhaps the most representative and influential industrial commodity, whose demand goes hand in hand with global industrial production; it may be regarded as a convenient proxy for the complex of all raw matters that constitute the input to the world's industry;
- gold is a commodity too, but a very special one, with unique features: since the dawn of civilization, gold has been used as a store of value and a medium of exchange; until very recently, it served as a standard for monetary systems; to date, most of above-surface gold is hoarded as reserve by central banks and supernational institutions; its price follows patterns that are more typical of foreign exchange markets than anything else.

For the S&P 500 index, we have considered the time series of daily prices from September 26, 1985 to October 31, 2008, while for the crude oil future and gold we have used time series of five-minute prices ranging from May 5, 2008 at 3:00 am to April 9, 2009 at 5:10 pm for crude oil and from June 5, 2008 at 3:00 am to April 9, 2009 at 5:10 pm for gold.

With  $n = 12$ , applying the *zig-zag* filter with two different values of threshold  $\theta$  produces datasets of, respectively, 2,769 and 1,095 records for the S&P 500 index, 4,708 and 1,892 for crude oil, and 1,960 and 620 for gold.

The values of  $E[r_t]$ ,  $Var[r_t]$ , and the relative error of the baseline model, which always predicts a swing of length  $E[r_t]$ , are reported in Table 9.2 for the different combinations of instrument and *zig-zag* threshold  $\theta$  that have been employed.

The reason why different values of  $\theta$  were selected for S&P 500 and for the two commodities is the time series have different temporal resolutions (daily vs. five-minute), which result in different scales of period-over-period variations. The values selected for  $\theta$  somehow compensate for such difference in scale.

Following the commonly accepted practice of machine learning, the problem data are partitioned into training, test and validation sets, used, respectively for training, to stop learning avoiding overfitting, and to test the generalization capabilities of each model.

We have set aside, for the validation of the S&P 500, the 200 most recent records for  $\theta = 0.01$  and the 100 most recent records for  $\theta = 0.02$ . Of the remaining records, a random 10% is used as the test set by the fuzzy-evolutionary method and 18% and 10% respectively by the neuro-genetic method.

**Table 9.3.** A comparison of the results obtained by the two methods on the two datasets generated for  $\theta = 0.01$  and  $\theta = 0.02$  when applied to the validation set (out of sample) for the S&P 500 index.

Financial Instrument	Dataset → Method	$\theta = 0.01$			$\theta = 0.02$		
		mean	stdev	best	mean	stdev	best
S&P 500	Fuzzy-Evolutionary	0.4720	0.0055	0.4657	0.4110	0.0049	0.4057
	Neuro-Evolutionary	0.4813	0.0096	0.4740	0.40307	0.0250	0.3958

**Table 9.4.** A comparison of the results obtained by the two methods on the two datasets generated for  $\theta = 0.005$  and  $\theta = 0.01$  when applied to the validation set (out of sample) for crude oil and gold.

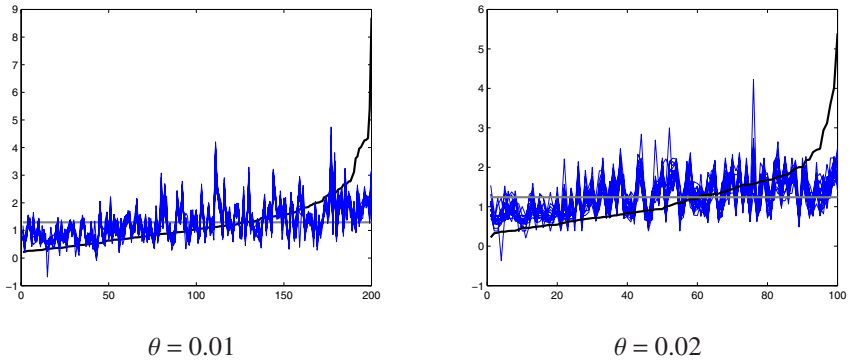
Financial Instrument	Dataset → Method	$\theta = 0.005$			$\theta = 0.01$		
		mean	stdev	best	mean	stdev	best
crude oil	Fuzzy-Evolutionary	0.5370	0.1501	0.4807	0.4768	0.0072	0.4682
	Neuro-Evolutionary	0.5109	0.0059	0.4984	0.5150	0.0119	0.4806
gold	Fuzzy-Evolutionary	0.4679	0.0071	0.4574	0.4631	0.0116	0.4399
	Neuro-Evolutionary	0.5096	0.0081	0.4945	0.5145	0.0400	0.4278

Similarly, we have set aside, for the validation of crude oil the 709 most recent records for  $\theta = 0.005$  and 193 for  $\theta = 0.01$ , while, for the validation of gold, we have considered, respectively, the 161 and the 100 most recent records. Then, the fuzzy-evolutionary method used the 10% of the remaining dataset as the test set, while 10% and 25% are used by the neuro-genetic method as the test set, respectively, for  $\theta = 0.005$  and  $\theta = 0.01$ .

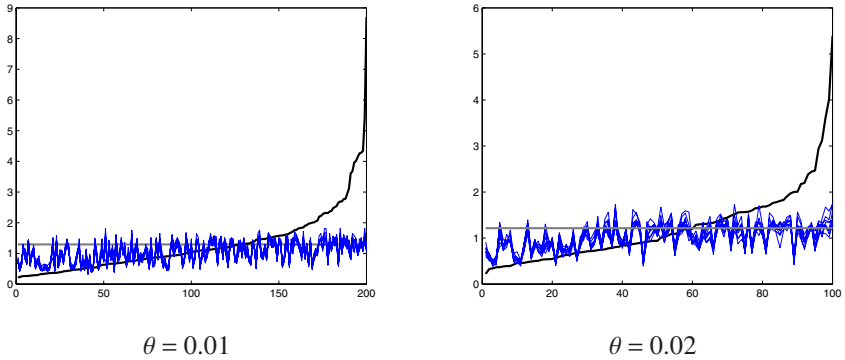
We performed 20 runs for either dataset with the neuro-genetic method and 10 runs for either dataset with the fuzzy-evolutionary method, which is more demanding in terms of computational resources. The results of those runs are shown in Tables 9.3 and 9.4.

Tables 9.3 and 9.4 summarize, respectively, the performance of the two modeling methods on the validation set for S&P 500 (for  $\theta = 0.01$  and  $\theta = 0.02$ ) and on crude oil and gold (for  $\theta = 0.005$  and  $\theta = 0.01$ ). Although the fuzzy-evolutionary method appears to slightly outperform the neural-evolutionary method, it does not do so consistently, nor by a wide margin. Instead, both methods perform largely better than the baseline model for every instrument and threshold  $\theta$ , suggesting that some regularities exist that may be exploited by the evolved models.

The fact that two independent methods, both based on global optimization methods like evolutionary algorithms, yet using two radically different “languages” to represent models, have obtained very similar results, leads us to doubt that there is room for significant further improvement. In other words, there is reason to presume that their results are very close to the optimum. Of course, this might also mean that the zig-zag indicator can only capture a limited amount of information—and no matter how one processes it, one can only do so well.



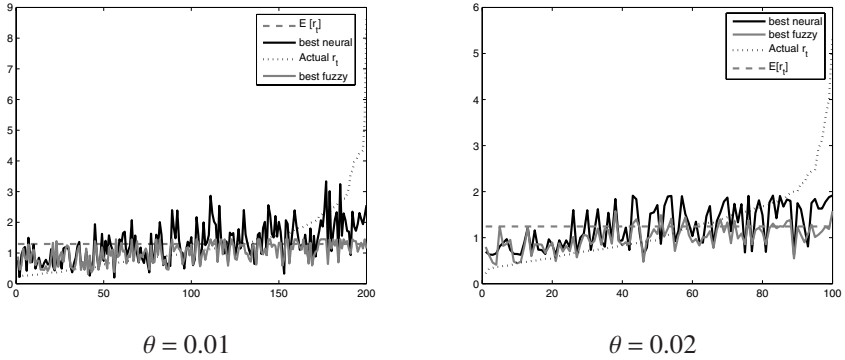
**Fig. 9.3.** Performance on the validation set (out-of-sample data) of the neural networks evolved by independent runs of the neuro-genetic method for the two datasets considered for S&P 500. The records of the validation set have been sorted by increasing  $r_t$ . The thick black line is the actual  $r_t$ , the thick light-grey line is  $E[r_t]$ , while the other lines represent the predictions of each neural network.



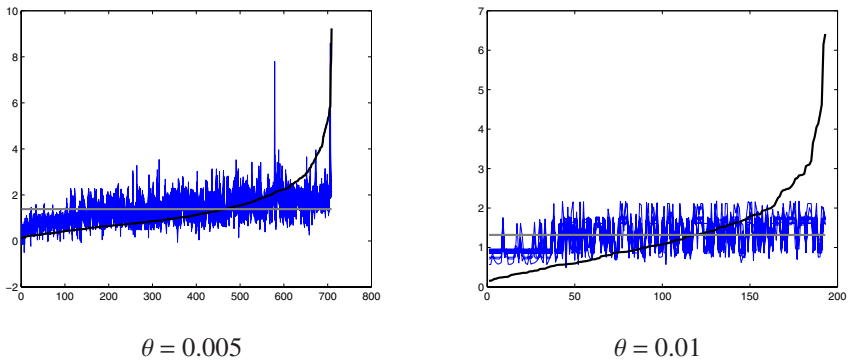
**Fig. 9.4.** Performance on the validation set (out-of-sample data) of the fuzzy rule bases evolved by independent runs of the fuzzy-evolutionary method for the two datasets considered for S&P 500. The records of the validation set have been sorted by increasing  $r_t$ . The thick black line is the actual  $r_t$ , the thick light-grey line is  $E[r_t]$ , while the other lines represent the predictions of each fuzzy model.

Figures 9.3 and 9.4 plot, respectively, the performance of the best neural networks and fuzzy rule-based models produced by each run for the financial instrument S&P 500, while Figure 9.5 compares the two best solutions obtained from the fuzzy and neuro-genetic approaches over the validation set of S&P 500 for, respectively,  $\theta = 0.01$  and  $\theta = 0.02$ .

In the same way, Figures 9.6, 9.7, 9.9, and 9.10 plot, respectively, the performance of the best neural networks and fuzzy rule-based models produced by each run for the crude oil and gold commodities, while Figures 9.8 and 9.11 compare the two best



**Fig. 9.5.** Comparison of the best results on the validation set of the fuzzy- and the neuro-evolutionary approaches for S&P 500. The dashed black line is the actual  $r_t$ , the dashed grey line is  $E[r_t]$ , while the grey line represents the best fuzzy model and the black line the best neural model.

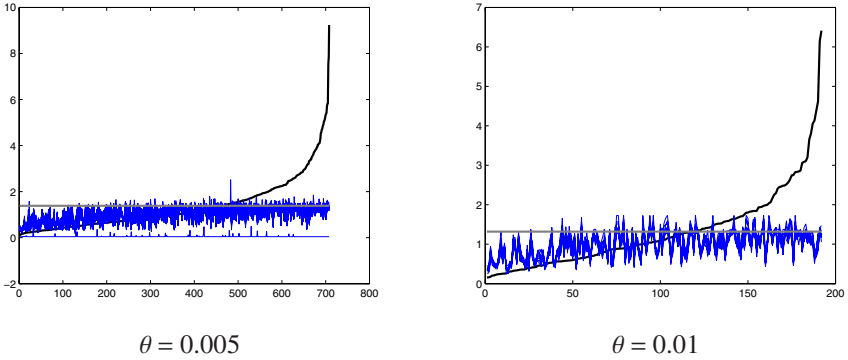


**Fig. 9.6.** Performance on the validation set (out-of-sample data) of the neural networks evolved by independent runs of the neuro-genetic method for the two datasets considered for crude oil. The records of the validation set have been sorted by increasing  $r_t$ . The thick black line is the actual  $r_t$ , the thick light-grey line is  $E[r_t]$ , while the other lines represent the predictions of each neural network.

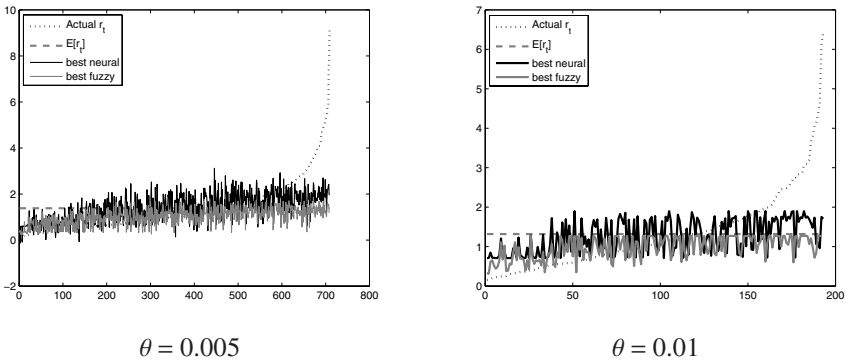
solutions obtained by the fuzzy- and neuro-evolutionary approaches over the validation sets of these two commodities for, respectively,  $\theta = 0.005$  and  $\theta = 0.01$ .

Although at first sight the predictions provided by the models found by both methods may appear disappointing, a closer examination of the graphs in these figures reveals something interesting.

What is striking in all these figures is that the lines representing the predictions by the various models appear to follow the same or very similar patterns. The graphs have been obtained by sorting the records of the validation set by increasing  $r_t$ , and then plotting the actual  $r_t$  (which, by construction, comes out as a monotonically increasing



**Fig. 9.7.** Performance on the validation set (out-of-sample data) of the fuzzy rule bases evolved by independent runs of the fuzzy-evolutionary method for the two datasets considered for crude oil. The records of the validation set have been sorted by increasing  $r_t$ . The thick black line is the actual  $r_t$ , the thick light-grey line is  $E[r_t]$ , while the other lines represent the predictions of each fuzzy model.



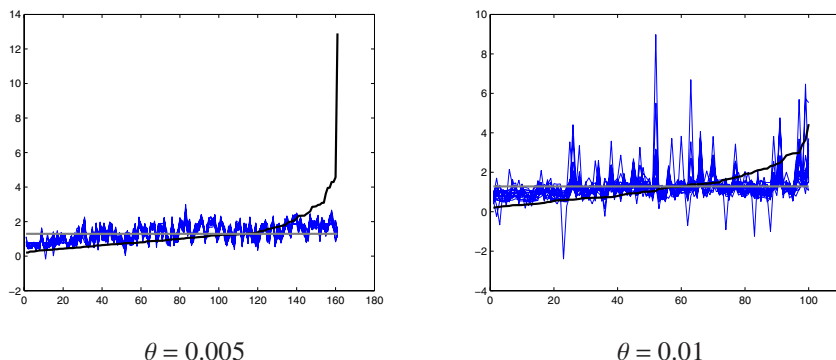
**Fig. 9.8.** Comparison of the best results on the validation set of the fuzzy- and the neuro-evolutionary approaches for crude oil. The dashed black line is the actual  $r_t$ , the dashed grey line is  $E[r_t]$ , while the grey line represents the best fuzzy model and the black line the best neural model.

line), the prediction of the baseline model  $E[r_t]$  (which, being constant, comes out as a flat line), and the predictions of the various models.

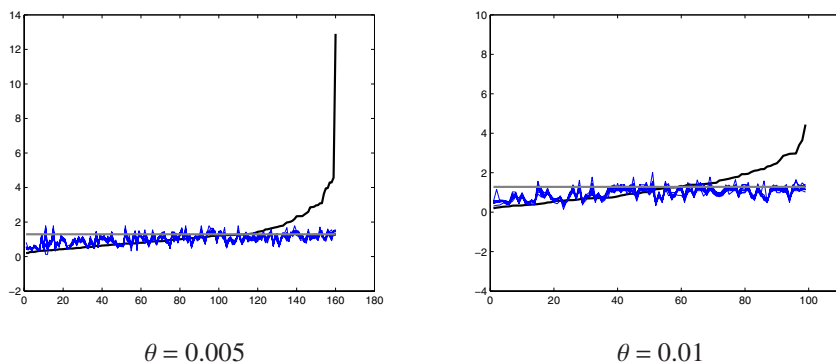
A clear tendency can be observed for models evolved by independent runs of both methods to provide similar predictions for the same records.

The fact that independently evolved models commit similar errors for the same records cannot be a coincidence, but it must be understood as evidence that the models are striking systematic trade-offs among errors committed when predicting  $r_t$  in contexts that look similar as far as the recent previous history of the time series is concerned, in a struggle to achieve the smallest possible overall error.



 $\theta = 0.005$  $\theta = 0.01$ 

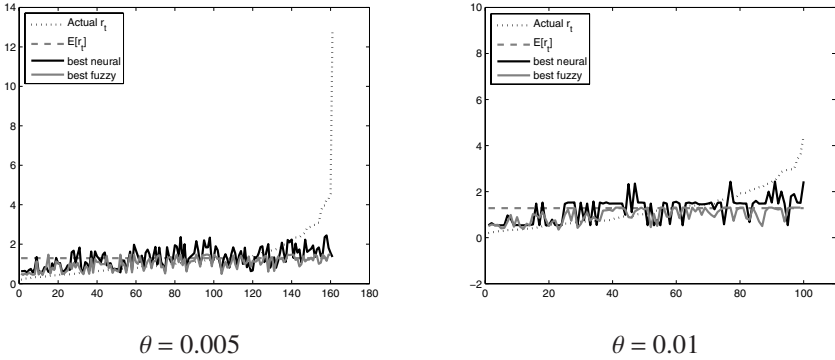
**Fig. 9.9.** Performance on the validation set (out-of-sample data) of the neural networks evolved by independent runs of the neuro-genetic method for the two datasets considered for gold. The records of the validation set have been sorted by increasing  $r_t$ . The thick black line is the actual  $r_t$ , the thick light-grey line is  $E[r_t]$ , while the other lines represent the predictions of each neural network.

 $\theta = 0.005$  $\theta = 0.01$ 

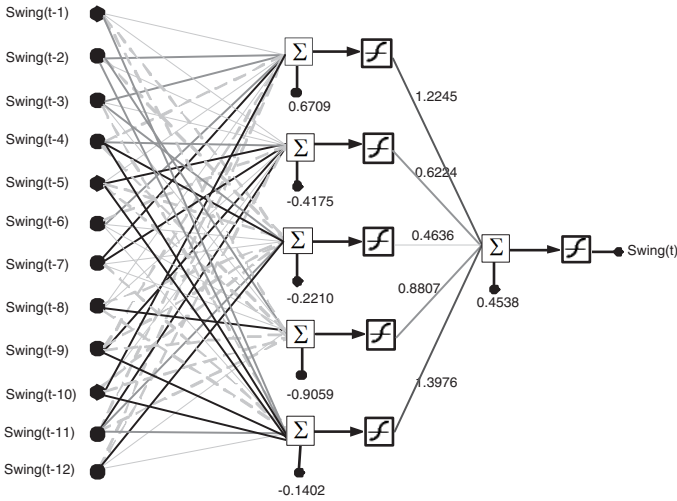
**Fig. 9.10.** Performance on the validation set (out-of-sample data) of the fuzzy rule bases evolved by independent runs of the fuzzy-evolutionary method for the two datasets considered for gold. The records of the validation set have been sorted by increasing  $r_t$ . The thick black line is the actual  $r_t$ , the thick light-grey line is  $E[r_t]$ , while the other lines represent the predictions of each fuzzy model.

Figures [9.5](#), [9.8](#), and [9.11](#) are even more impressive because they show that the best fuzzy and neural models, which are not only independently evolved, but also based on different representations, behave in the same manner, although neural networks seem more “extremist” in their predictions, whereas fuzzy rule bases look more “moderate”. However, in despite of this apparent quantitative difference, their qualitative behaviour is almost identical.

To give the curious reader an idea of the kinds of models that turn out to be good at predicting turning points, the best topologies obtained from the neuro-genetic

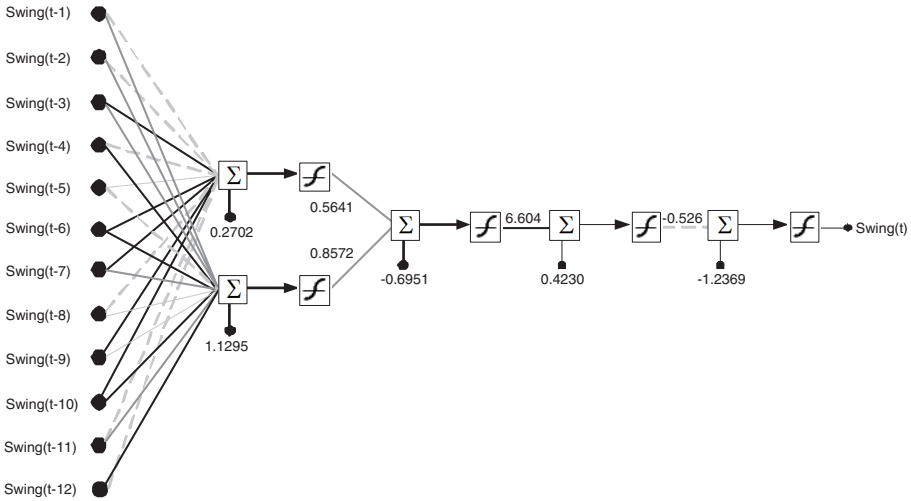


**Fig. 9.11.** Comparison of the best results on the validation set of the fuzzy- and the neuro-evolutionary approaches for gold. The dashed black line is the actual  $r_t$ , the dashed grey line is  $E[r_t]$ , while the grey line represents the best fuzzy model and the black line the best neural model.



**Fig. 9.12.** Topology of the best neural network for  $\theta = 0.01$ . The thick black line refers to connection weights  $w \geq 1$ . The thick dark grey line corresponds to values defined as  $0.5 \leq w < 1$ , while the light grey line to those defined as  $0 \leq w < 0.5$ . Finally the dashed light grey line corresponds to negative correlations  $w < 0$ .

approach for each of the two datasets for the financial instrument S&P 500 are shown in Figures 9.12 and 9.13, corresponding, respectively, to  $\theta = 0.01$  and  $0.02$ . An interesting aspect that can be highlighted is that, while the first model corresponds to a more usual network with normal connections, in the second network (see Figure 9.13) the last hidden connection is set, together with the bias of the last node, to a negative value. Such connection could represent a reversal (through a NOT operator) of the information



**Fig. 9.13.** Topology of the best neural network for  $\theta = 0.02$ . As reported in Figure 9.12, the thick black line refers to connection weights  $w \geq 1$ . The thick dark grey line corresponds to  $0.5 \leq w < 1$ , and the light grey line to  $0 \leq w < 0.5$ . Finally the dashed light grey line corresponds to negative correlations  $w < 0$ .

```

IF TRUE THEN swing(t) is 1.44
IF swing(t - 1) is medium-large THEN swing(t) is 0.12
IF swing(t - 3) is very-large AND swing(t - 11) is small-to-medium THEN swing(t) is 0.12
IF swing(t - 1) is medium AND swing(t - 2) is medium-to-large THEN swing(t) is 0.12
IF swing(t - 1) is medium-to-huge AND swing(t - 2) is medium THEN swing(t) is 0.12
IF swing(t - 1) is medium-to-huge THEN swing(t) is 0.12
IF swing(t - 1) is medium-to-huge THEN swing(t) is 0.12
IF swing(t - 1) is medium AND swing(t - 2) is medium-to-large THEN swing(t) is 0.12
    
```

**Fig. 9.14.** The best fuzzy rule base for  $\theta = 0.01$ . The linguistic values have been manually given meaningful names to improve readability.

flow; however, it seems to be an effect of the backpropagation algorithm used to train the networks.

The best fuzzy rule bases obtained by the fuzzy-evolutionary approach for either S&P 500 dataset are shown in Figures 9.14 and 9.15. It can be observed that both rule bases feature a sort of *default* rule, which always fires, and sets the prediction of  $\text{swing}(t)$  to 1.44 for  $\theta = 0.01$  and 1.57 for  $\theta = 0.02$ , while all the remaining rules appear to be there to recognize and handle significant patterns where different predictions can be made. Interestingly, the rule base in Figure 9.14 uses but the three most recent swings to predict the next one; the most recent swings have a prevailing role in the rule base in Figure 9.15 as well, although “older” swings are looked at in a few rules.

```

IF swing( $t-7$ ) is large AND swing( $t-11$ ) is medium-large AND swing( $t-5$ ) is large
AND swing( $t-2$ ) is medium THEN swing( $t$ ) is 8.65
IF swing( $t-7$ ) is large AND swing( $t-5$ ) is medium-large AND swing( $t-1$ ) is large
THEN swing( $t$ ) is 15
IF TRUE THEN swing( $t$ ) is 1.57
IF swing( $t-1$ ) is large AND swing( $t-8$ ) is medium-small THEN swing( $t$ ) is 0.13
IF swing( $t-1$ ) is small-to-large AND swing( $t-2$ ) is medium THEN swing( $t$ ) is 0.13
IF swing( $t-2$ ) is medium THEN swing( $t$ ) is 0.13
IF swing( $t-6$ ) is around 11 AND swing( $t-12$ ) is between 6 and 10
AND swing( $t-8$ ) is medium-small THEN swing( $t$ ) is 0.13
IF swing( $t-1$ ) is small-to-large THEN swing( $t$ ) is 0.13
IF swing( $t-1$ ) is large AND swing( $t-9$ ) is small THEN swing( $t$ ) is 0.13
IF swing( $t-1$ ) is large THEN swing( $t$ ) is 0.13

```

**Fig. 9.15.** The best fuzzy rule base for  $\theta = 0.02$ . The linguistic values have been manually given meaningful names to improve readability.

## 9.6 Conclusion and Future Work

Two independent evolutionary modeling methods have been applied to predicting trend reversals in financial time series. The results obtained indicate that the lengths of price swings of financial instruments follow a largely, but not exclusively, random pattern. However, the results of the experiments that have been performed provide empirical evidence that the non-random part of their behaviour can be modeled and predicted.

Whether this predictable part of a financial time series behaviour can be effectively exploited for gaining excess returns is an open question, but we believe our results might constitute yet another small piece of evidence against the efficient market hypothesis, if one could trade them profitability having adjusted for risk.

Detailed results have been shown regarding the S&P500 index, crude oil, and gold, which, we argued, form a representative selection of financial instruments. However, the time series of all the other instruments we have tried to apply the same approach to, show the same behaviour.

## References

1. Achelist, S.: Technical analysis from A to Z. Probus Publisher, Chicago (1995)
2. Azzini, A.: A New Genetic Approach for Neural Network Design. Ph.D. Thesis (March 2007), <http://www.dti.unimi.it/azzini/wwwmat/azzinipublication.htm>
3. Azzini, A., Tettamanzi, A.: A neural evolutionary approach to financial modeling. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006, vol. 2, pp. 1605–1612. Morgan Kaufmann, San Francisco (2006)
4. Azzini, A., Tettamanzi, A.: Evolving Neural Networks for Static Single-Position Automated Trading. Journal of Artificial Evolution and Applications (2008), doi:10.1155/2008/184286
5. Azzini, A., Tettamanzi, A.: Evolutionary Single-Position Automated Trading. In: Proceedings of European Workshop on Evolutionary Computation in Finance and Economics, EVOFIN 2008, pp. 1605–1612 (2008)
6. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Springer, Berlin (2006)

7. Brabazon, A., O'Neill, M.: *Natural Computing in Computational Finance*. Springer, Berlin (2008)
8. Muhlenbein, H., Schlierkamp-Voosen, D.: The science of breeding and its application to the breeder genetic algorithm (bga). *Evolutionary Computation* 1(4), 335–360 (1993)
9. da Costa Pereira, C., Tettamanzi, A.: Fuzzy-evolutionary modeling for single-position day trading. In: Brabazon, A., O'Neill, M. (eds.) *Natural Computing in Computational Finance*, pp. 131–159. Springer, Berlin (2008)
10. Hellendoorn, H., Thomas, C.: Defuzzification in Fuzzy Controllers. *Intelligent and Fuzzy Systems* 1, 109–123 (1993)
11. Herbst, A.: *Analyzing and Forecasting Futures Prices*. Wiley, New York (1992)
12. Mamdani, E.: Advances in Linguistic Synthesis of Fuzzy Controllers. *International Journal of Man Machine Studies* 8, 669–678 (1976)
13. Peters, E.: *Chaos and Order in the Capital Markets*, 2nd edn. Wiley, New York (1996)
14. Raftopoulos, S.: The Zigzag Trend Indicator. *Technical Analysis of Stocks and Commodities* 21(11), 26–32 (2003)
15. Schwefel, H.: *Numerical Optimization for Computer Models*. John Wiley, Chichester (1981)
16. Tettamanzi, A.: An evolutionary algorithm for fuzzy controller synthesis and optimization. In: *IEEE International Conference on Systems, Man and Cybernetics*, vol. 5/5, pp. 4021–4026. IEEE Systems, Man, and Cybernetics Society (1995)
17. Tettamanzi, A., Poluzzi, R., Rizzotto, G.: An evolutionary algorithm for fuzzy controller synthesis and optimization based on SGS-Thomson's W.A.R.P. fuzzy processor. In: Zadeh, L., Sanchez, E., Shibata, T. (eds.) *Genetic algorithms and fuzzy logic systems: Soft computing perspectives*. World Scientific, Singapore (1996)
18. Yao, X.: Evolving artificial neural networks. *Proceedings of the IEEE* 87(9), 1423–1447 (1999)
19. Yao, X.: *Evolutionary Optimization*. Kluwer Academic Publishers, Norwell (2002)
20. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks* 8(3), 694–713 (1997)
21. Yao, X., Liu, Y.: A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks* 8(3), 694–713 (1997)
22. Zadeh, L.: Fuzzy Sets. *Information and Control* 8, 338–353 (1965)
23. Zadeh, L.: The concept of a Linguistic Variable and its application to Approximate Reasoning, I–II. *Information Science* 8, 199–249, 301–357 (1975)
24. Zadeh, L.: The Calculus of Fuzzy If-Then Rules. *AI Expert* 7(3), 22–27 (1992)

# Evolutionary Money Management

Philip Saks<sup>1</sup> and Dietmar Maringer<sup>2</sup>

<sup>1</sup> Centre for Computational Finance and Economic Agents, University of Essex  
psaks@essex.ac.uk

<sup>2</sup> Economics and Business Faculty, University of Basel  
dietmar.maringer@unibas.ch

**Summary.** This paper evolves trading strategies using genetic programming on high-frequency tick data of the USD/EUR exchange rate covering the calendar year 2006. This paper proposes a novel quad tree structure for trading system design. The architecture consists of four trees each solving a separate task, but mutually dependent for overall performance. Specifically, the functions of the trees are related to initiating (“entry”) and terminating (“exit”) long and short positions. Thus, evaluation is contingent on the current market position. Using this architecture the paper investigates the effects of money management. Money management refers to certain measures that traders use to control risk and take profits, but the findings in this paper suggest that it has detrimental effects on performance.

## 10.1 Introduction

The foreign exchange (FX) market is the largest financial market in the world. In theory, exchange rates should be intimately linked to macro economic variables, such as interest rates, inflation, money supply and real income. However, in their seminal paper, Meese and Rogoff [15] found that these fundamentals are useless in forecasting exchange rate changes even at medium frequencies. This is known as the *determination puzzle of foreign exchange*. At shorter time horizons many traders tend to use *technical analysis* in decision making [1]. Technical analysis attempts to forecast future price changes based on historical observations. This broad definition covers a wide range of methods from visual pattern recognition to moving averages and more elaborate schemes. Traditionally, it has encountered much skepticism from academia since it clearly contradicts the *Efficient Market Hypothesis* (EMH) – one of the cornerstones of modern finance [7]. During the past few decades, however, there has been an increasing interest in technical analysis among financial economists and extensive literature has emerged on the subject. There is a general consensus that technical analysis on a daily frequency has been profitable in the past [3, 13, 12].

In contrast, this paper uses high-frequency intraday tick data where trading occurs under market frictions through the quoted bid-ask spread. Instead of using a predetermined strategy as a moving average rule or a chart pattern, the computer evolves its own trading strategies using *genetic programming* (GP). GP has previously been applied to trading rule induction for foreign exchange markets, but the results are mixed. In the

early nineties GP was found to produce significant profits when trading in the presence of realistic transaction costs [9, 2]. But performance has deteriorated since then [6, 16].

As mentioned above, this paper considers high-frequency intraday tick data on the USD/EUR exchange rate covering the full year of 2006. Besides being a much needed update on GP in this domain it adds to the existing literature in a number of ways. The standard approach of GP in trading rule induction is to use a single tree structure that makes buy or sell recommendations. This paper proposes a novel multiple tree structure consisting of four (quad) trees for ternary decision problems. Hence, strategies can take short, neutral and long positions. Which tree is evaluated is contingent on the current market position. Each of the four trees returns Boolean values and their functions can be characterised as long entry, long exit, short entry and short exit. The entry trees initiate either long or short positions, while the exit trees terminate those positions and revert to a neutral state. Using this division of entry and exit strategies, the benefits of money management are examined. Money management refers to certain measures that traders use to control risk and take profits, implying that closing of positions can be initiated by events other than “standard” buy/sell signals. To reflect this in an automated trading system, an extension to standard approaches needs to be made. Traditionally, one common rule for both positions is evaluated, and depending on the outcome, the signal is to enter (stay in) these positions or exit (stay out of) them, respectively. In money management, different rule sets, contingent on the current position, are used. Hence, a negative entry signal is not necessarily seen as an exit signal, but entirely different rules are evaluated to find exit signals. Furthermore, these exit signals can be based on other indicators or information. For example, stop losses are often placed to trigger an exit signal in order to limit downside risk. Since money management is a practitioner’s way of controlling risk, the effects of evolving strategies under different utility functions are investigated.

The rest of this Chapter is structured as follows. Section 10.2 provides a brief introduction to genetic programming. Section 10.3 presents the data. The fitness function, model and parameter settings are described in Section 10.4. This is followed by empirical results in Section 10.5. Finally, Section 10.6 concludes and gives pointers to possible future research.

## 10.2 Genetic Programming

Genetic programming (GP) was pioneered by Koza [11] and is often seen as a derivative of genetic algorithms (GA). The GA was popularised by Holland [8] in his ambitious quest to understand the principles of adaptive systems in a broad sense. An obvious inspiration came from biology, where the success of natural adaptive systems rests on competition and innovation in order to survive in changing and uncertain environments.

The GA is a population based search method, where the individuals are fixed-length binary strings, known as *genotypes* or *chromosomes* [21]. Generally, this representation requires an encoding which is problem specific. For example if the GA is used for real-valued parameter optimization, then it is necessary to discretise the search space, where the resolution depends on the number of bits chosen to represent a given variable.

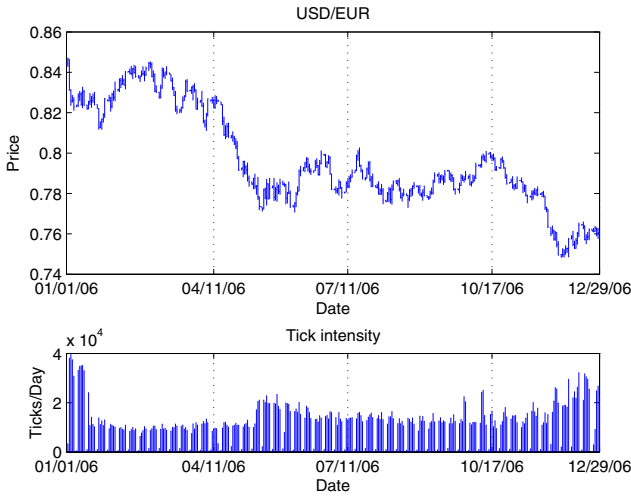
There exist many variations of GA, but their basic workings are illustrated in the following. To begin with an initial population of  $M$  individuals is generated randomly in generation zero. Hereafter, the fitness of each individual is calculated according to the pre-specified objective function. Then a new population is created by selecting between the operators reproduction, crossover and mutation according to the probabilities  $p_r$ ,  $p_c$  and  $p_m$ , respectively. The reproduction and mutation operators select individuals from the parent population, such that better solutions are favored. A popular mechanism for doing this is *tournament selection*, in which a fixed number of individuals are chosen uniformly from the parent population, and the fittest individual wins the tournament and is selected. By controlling the tournament size it is possible to regulate the selection pressure. The reproduction operator simply copies the selected string to the new population. For the crossover operation, two individuals are selected from the parent population. Hereafter a position or index is uniformly selected within the bit-string and genetic material from the two parents is simply swapped around this point. The two resulting offspring are then inserted into the new population. The mutation operator can be applied to individuals from the population or resulting from reproduction. It simply selects a random element within an individual and negates the value, i.e., zero becomes one and vice versa. An advantage of the mutation operator is that it can introduce diversity into a population, but usually mutation is only invoked with a small probability. When the population size of the new population is equal to  $M$ , the algorithm has completed one generation and the process repeats itself until a termination criterion has been satisfied, e.g., until a maximum number of generations is reached.

Genetic programming is basically a GA operating on hierarchical computer programs instead of binary strings. Any problem that is concerned with finding an optimal mapping from a set of inputs to a set of outputs, can be reformulated as a search for an optimal computer program. GP provides the means to search the space of possible programs. It is therefore a much more direct approach to problem solving than GAs, that are heavily dependent on problem encoding. In practice the individuals in GP are computer programs represented as tree structures. The programs are constructed from *functions* and *terminals*, where by definition the former take arguments and the latter do not. The sets of available functions and terminals for a given problem is known as the *function set* and *terminal set*. For more details on GP and their application to automated trading, see [11] and [19].

### 10.3 Data

The data is provided by OANDA FXticks, and comprises of 24-hour tick data on the USD/EUR exchange rate covering the calendar year 2006. This constitutes a total of 3 894 525 bid-ask observations, and on average there are more than 12 000 per trading day. Figure 10.1 shows the daily prices, together with the number of ticks per day. Over the entire period the Dollar depreciates from 0.8439 to 0.7577, corresponding to 10.77%. The daily tick intensity appears fairly constant most of the year, but at the beginning and the end of the year the activity is considerably higher. Moreover, during the





**Fig. 10.1.** Daily USD/EUR exchange rate (top), and number of ticks per day (bottom).

**Table 10.1.** Summary statistics of log-returns of 10-tick sampled USD/EUR middle prices. AC denotes the auto-correlation and KS is the  $p$ -value of a Kolmogorov-Smirnov test.

Mean ( $\cdot 10^{-6}$ )	Std ( $\cdot 10^{-4}$ )	Skewness	Kurtosis	AC (lag-1)	AC (lag-2)	KS ( $p$ )
-0.277	1.323	-0.215	18.823	-0.048	0.002	0.000

month of May, there is also increased activity which coincides with a crisis in the equity markets. The USD/EUR series exhibits strong intraday effects, e.g., the trading activity is higher during European business hours. Moreover, there are two distinct peaks around 08:00 and 14:00 GMT with high activity, separated by lower activity at noon. By using an equidistant intraday sampling in calendar time this seasonality is neglected leading to disproportionately many samples during the night compared to daytime. Hence, sampling is done in trading time. Specifically, the exchange rate is sampled every 10 ticks, which yields 389 452 samples. On average this is close to 1 minute sampling in calendar time. The data comprises bid-ask quotes, but in the following the statistical properties of the logarithmic middle prices are analyzed.

Due to large spikes in the spreads, the top decile is winsorised. For the entire sample, the median spread is 1.1870 bp, with an interquartile range of 0.0707 bp. Table 10.1 contains summary statistics for the log-returns of the sampled USD/EUR series. The series has negative skewness and significant excess kurtosis, thus strongly rejecting the null hypothesis of Gaussianity in a Kolmogorov-Smirnov test. At the 10-tick sampling frequency there is a significantly negative first order auto-correlation. This phenomenon has previously been reported in the literature using a one-minute sampling in calendar time [5].

## 10.4 Framework

### 10.4.1 Objectives and Fitness Function

The choice of objective function is essential in evolutionary computation. In relation to trading system design an obvious candidate would simply be to maximise the return of a strategy. This measure, however, is problematic because it ignores the risk dimension associated with financial decision making and trading. In this respect, the Sharpe ratio is better in that it evaluates the expected return relative to the standard deviation of returns. While this is a popular measure it has certain undesirable properties. For the Sharpe ratio to be an accurate measure, either the returns must be Gaussian, or people should only have preferences for the first and second moment of returns. The findings in Section 10.3 refute the former, while the latter has unrealistic implications. The standard deviation is a symmetric measure, which implies that investors are equally concerned with the positive as they are with the negative variation in returns.

In the context of financial decision making, *economic utility theory* provides a better description of the objectives that drive investors. In classical utility theory, the sole objective of agents is to maximise expected utility of wealth, where more wealth is always preferred to less [4]. However, cognitive psychology has revealed that, in evaluating different outcomes, reference dependence plays a crucial role [10]. In the context of financial investments the reference point is determined by how myopic an agent is, i.e., how frequently wealth is evaluated [20]. This implies that both the long-term level and short-term changes in wealth are important factors in determining overall happiness of investors. Hence, happiness is a path-dependent function of the evolution of wealth. To capture this path dependency, the period over which the trading rule is optimised is divided into  $I$  sub-intervals and the utility is evaluated for each interval. Let the return within an interval  $i$  be defined as

$$r_i = (w_i - w_{i-k})/w_{i-k} \quad (10.1)$$

where  $k$  is the length of the interval. A modified terminal interval wealth is then introduced,

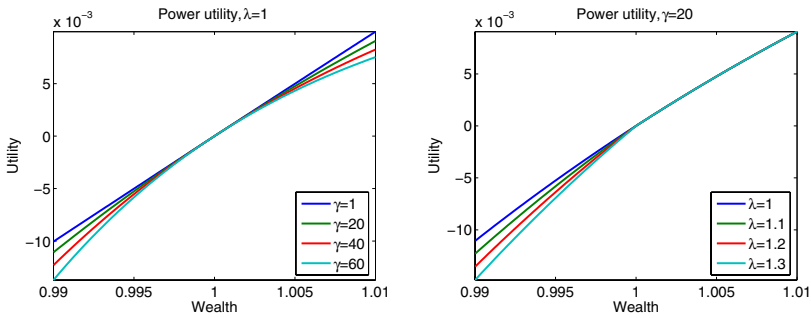
$$\hat{v}_i = \begin{cases} v_0 \cdot (1 + r_i) & \text{if } r_i \geq 0 \\ v_0 \cdot (1 + r_i)^\lambda & \text{if } r_i < 0 \text{ with } \lambda \geq 1 \end{cases} \quad (10.2)$$

where  $\lambda > 1$  implies loss aversion, i.e., a greater sensitivity to decreases in wealth, while  $\lambda = 1$  models no additional disutility from losses beyond risk aversion. The initial endowment at the beginning of the interval is  $v_0$ , but in this paper a unit investor is considered such that  $v_0 = 1$ . Assuming a power utility function, the value of an interval is

$$U(\hat{v}_i) = \begin{cases} \frac{\hat{v}_i^{1-\gamma}}{1-\gamma} - \frac{1}{1-\gamma} & \text{if } \gamma > 1 \\ \ln(\hat{v}_i) & \text{if } \gamma = 1 \end{cases}. \quad (10.3)$$

For a risk averse trader,  $\gamma > 1$ , whereas  $\gamma = 1$  implies risk neutrality. From this the objective function simply follows as the average interval utility,

$$F = \frac{1}{I} \sum_{i=1}^I U(\hat{v}_i). \quad (10.4)$$



**Fig. 10.2.** Power utilities for different values of  $\gamma$  (left) and  $\lambda$  (right).

The power utility function is shown for different values of  $\gamma$  and  $\lambda$  in Figure 10.2.  $\gamma$  controls the concavity of the utility function, and  $\lambda$  regulates the loss aversion by decreasing utility for losses while leaving utility for gains unchanged.

#### 10.4.2 Model

The purpose of this paper is to evolve trading models using genetic programming. The traditional approach in the context of FX forecasting is to evolve binary decision rules where outputs correspond to either long or short positions in a given currency. Using this representation a trading model is forced to take a directional view and cannot remain neutral. To overcome this problem in a single tree framework, it is possible to construct programs that return a trinary Boolean variable instead of the normal binary Boolean variable [2].

In the context of binary trading models it has previously been found that using a dual tree structure instead of the traditional single tree model has significant impact on performance, especially if market frictions are taken into account [18]. Capitalizing on these findings this paper proposes a unique quad tree structure. The four trees consist of a long entry (T1), long exit (T2), short entry (T3) and short exit (T4). Unlike a stock, an exchange rate does not have a distinct up and down. The inverse relationship of exchange rates dictates that up for one currency is down for the other and vice versa. In this paper the positions relate to dollar. Thus, when a long (short) position is initiated we expect an appreciation (depreciation) of the dollar relative to the euro.

The workings of the four trees is illustrated in Table 10.2. Each tree returns a Boolean variable, but which tree is evaluated depends on the current market position. For example if the current position is neutral, then either a long or a short position can be initiated. This happens if either T1 or T3 is true. If both T1 and T3 are true, the signal is ambiguous and a neutral position is maintained. If the current position is long and T2 is true, a transition is made to a neutral position. If T3 is true, a short position is initiated. In the short state, T1 initiates a long position and T4 results in a neutral position. In order to resolve conflicting decisions, the strongest views are given precedence such that directional views trump neutrality.

One objective of this paper is to examine the effects of money management on trading strategies. This is done by comparing strategies with special grammars for the exit

**Table 10.2.** Transition table of the quad tree structure consisting of long entry (T1), long exit (T2), short entry (T3) and short exit (T4), going from the current position to neutral (N), short (S), long (L) position. 0=FALSE, 1=TRUE and -=not evaluated

	Current position											
	Neutral				Long				Short			
T1	0	0	1	1	-	-	-	-	0	0	1	1
T2	-	-	-	-	0	0	1	1	-	-	-	-
T3	0	1	0	1	0	1	0	1	-	-	-	-
T4	-	-	-	-	-	-	-	-	0	1	0	1
new position	N	S	L	N	L	S	N	S	S	N	L	L

strategies (T2 and T4), to strategies where the grammar is the same across all the trees. In addition to type constraints the trees have semantic restrictions, which improves the search efficiency significantly, since computational resources are not wasted on non-sensical solutions [2]. The function set for the entry strategies (T1 and T3) consists of numeric comparators, Boolean operators and addition. Furthermore, three special functions have been introduced. BTWN takes three arguments and evaluates if the first is between the second and third. HASINC (HASDEC) returns true if the second argument has increased (decreased) over the lag period given by the first argument. The terminals include the variables price and moving averages thereof (price) and the time of day (time). While sampling is done in trading time, the calendar time of each observation is recorded with a minutes precision (hhmm format). Special constants are available for conditioning on time (timeConst), and the difference between price indicators (pConst). The entry strategy grammar is documented in Table 10.3.

In practice, traders employ various exit strategies for money management, such as stop losses and profit targets. A stop loss automatically exits the strategy when the current profit is below a certain level. Likewise a profit target closes out a position when a given profit is obtained. A more elaborate scheme is a trailing stop, which ensures that the drawdown does not exceed a given value. Simple stop losses and profit targets might be augmented with time exits such that the duration of a trade is constrained. To capture these ideas the exit strategy grammar contains information about the current profit, drawdown and duration of a trade. The exit strategy grammar is an extension to the entry strategy grammar and their difference is documented in Table 10.4.

### 10.4.3 Parameter Settings

In the following computational experiments a population of 500 individuals is initialised using the *ramped half-and-half* method. It evolves for a maximum of 50 generations, but is stopped after 20 generations if no new elitist (*best-so-far*) individual has been found. A normal tournament selection is used with a size of 5, and the crossover and mutation probabilities are 0.9 and 0.05, respectively. Moreover, the probability of selecting a function node during reproduction is 0.5, and each of the trees in the programs are constrained to a maximum complexity of 25 nodes. This constraint might seem overly restrictive, but given the highly specialised grammar described in Section 10.4.2 it is

**Table 10.3.** Entry strategy grammar. BTWN checks if the first argument is *between* the second and third. HASINC (HASDEC) returns true if the price has increased (decreased) over the last period.

Function	Arguments	Return Type
+	(price, pConst)	priceNew
<=, >=	(price, price)	bool
<=, >=	(price, priceNew)	bool
<=, >=	(time, timeConst)	bool
BTWN	(price, price, price)	bool
BTWN	(price, priceNew, priceNew)	bool
BTWN	(time, timeConst, timeConst)	bool
HASDEC, HASINC	(lag, price)	bool
AND, OR, XOR	(bool, bool)	bool
NOT	(bool)	bool

**Table 10.4.** Additional exit strategy grammar. The complete grammar is composed of the entry strategy grammar and the functions in this table.

Function	Arguments	Return Type
<=, >=	(duration, durationConst)	bool
<=, >=	(profit, pConst)	bool
>=	(drawdown, drawdownConst)	bool
BTWN	(duration, durationConst, durationConst)	bool
BTWN	(profit, pConst, pConst)	bool

possible to construct very complicated rules within that limit. Hence the complexity constraint aims to minimise the risk of overfitting.

The entire data set is split into four equal-size blocks of 97,364 samples each. Henceforth the blocks are denoted; I, II, III and IV. Since the blocks have been constructed in trading time, their durations in calendar time differ. Block I covers the period from 01-Jan-2006 to 11-Apr-2006, Block II continues to 11-Jul-2006, Block III ends 17-Oct-2006, and Block IV is the remainder until 29-Dec-2006. The four blocks are shown in Figure 10.1. The returns in each of the blocks are -2.37%, -4.75%, 1.40% and -5.06%, respectively. In the following experiments, rolling window estimation is made on blocks I-III and successive out-of-sample tests are made on blocks II-IV.

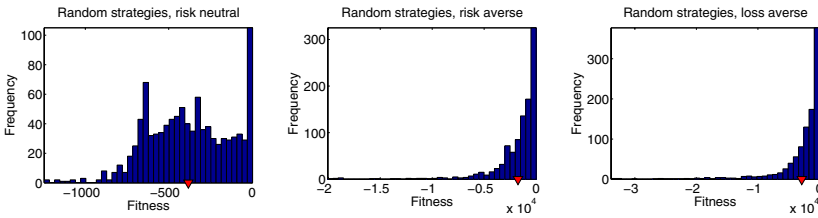
As fitness functions, three different utility functions are considered: risk neutral ( $\gamma = 1$ ,  $\lambda = 1$ ), risk averse ( $\gamma = 35$ ,  $\lambda = 1$ ) and loss averse ( $\gamma = 35$ ,  $\lambda = 1.15$ ). Each block is divided into a number of sub-intervals, each consisting of 1000 samples. This implies that, on average, wealth is evaluated between one and two times per day, which does not seem unreasonable for a high-frequency trader. As mentioned in Section 10.4.1, the fitness of an individual trading strategy is the average utility obtained within each sub-interval. Due to the high-frequency domain considered in this paper the overnight interest rates are neglected when calculating the returns of the strategies.

### 10.5 Empirical Results

In this section the results from rolling window estimation and testing on blocks I to IV are presented. For statistical inferences ten independent runs are considered for both the entry-entry and entry-exit grammar strategies. As mentioned previously the strategies are evolved in the presence of market frictions.

Appendix A contains detailed results of interval statistics, for both the entry-entry and entry-exit grammar strategies under various utility functions. Naturally, the introduction of market frictions has an adverse effect on performance. Under risk neutrality the in-sample median fitness in Block I drops to 7.78 and 8.08 for the entry-entry and entry-exit grammar strategies, respectively. What is more interesting is that the out-of-sample median fitnesses in Block II are positive for both grammars (4.51 and 4.37). Unfortunately, this pattern does not repeat itself during the later periods. This holds for all utility functions. However, it should be noted that negative utility need not imply unprofitability – the out-of-sample results in Block III for the entry-exit grammar strategies under loss aversion illustrate this point (Table 10.12). When risk aversion and loss aversion are introduced it has an adverse effect on in-sample performance for both grammars as expected.

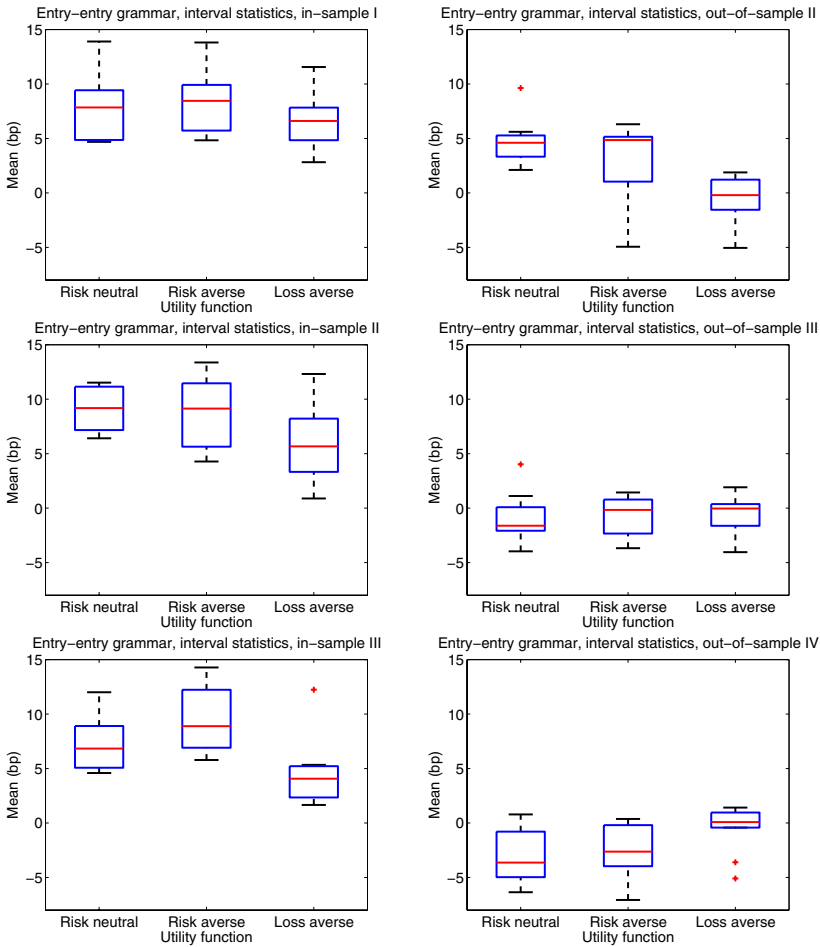
To test the null hypothesis that the evolved strategies have uncovered significant regularities, their fitness values are compared to that of 1000 randomly initialised strategies. Figure 10.3 show the empirical fitness distributions for the random strategies under the different utility functions. Given a frictionless environment and risk neutral utility function, then trading would be a fair game where the expected utility is zero. However, under market frictions the expected utility is negative even for risk neutral agents. Risk



**Fig. 10.3.** Histograms of the performance of 1000 random strategies for different utility functions; risk neutral (left), risk averse (center) and loss averse (right). The triangle marks the average utility.

**Table 10.5.** Rank sum test *p*-values for the null hypothesis of equal median fitnesses of the entry-entry and entry-exit grammar strategies.

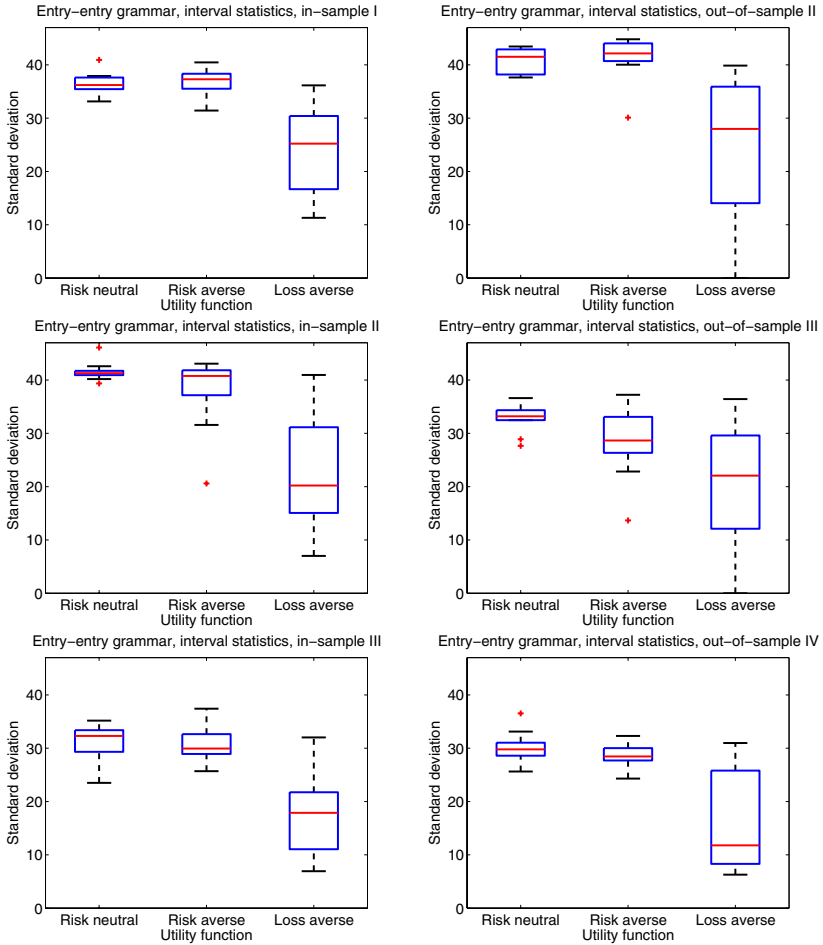
Utility function	In-sample			Out-of-sample		
	I	II	III	II	III	IV
Risk neutral	0.3847	0.6232	0.2730	0.5708	0.9097	0.0312
Risk averse	0.6776	0.7913	0.3447	0.0757	0.4727	0.9097
Loss averse	0.7337	0.4274	0.2413	0.5452	0.9097	0.2413



**Fig. 10.4.** Boxplots of the average of interval returns across the 10 runs for each block and different utility functions.

aversion and loss aversion only exacerbates this effect. Most of the evolved strategies significantly outperform the random strategies both in-sample and out-of-sample, despite being less successful in monetary terms. This suggests that real speculation in the considered currency markets requires either a strong risk-seeking behavior or a significant edge.

As mentioned previously, a main objective of this chapter is to examine the effects of money management for different types of speculators. To test formally whether money management, i.e., an extended exit grammar, makes a difference, the Wilcoxon rank sum test is employed for each block and for each utility function. Table 10.5 lists the  $p$ -values for the null hypothesis of identical median fitnesses for the two grammars. Only Block IV out-of-sample under risk neutrality is significant at the usual level. In the context of the other results, this can clearly be treated as a spurious rejection and does

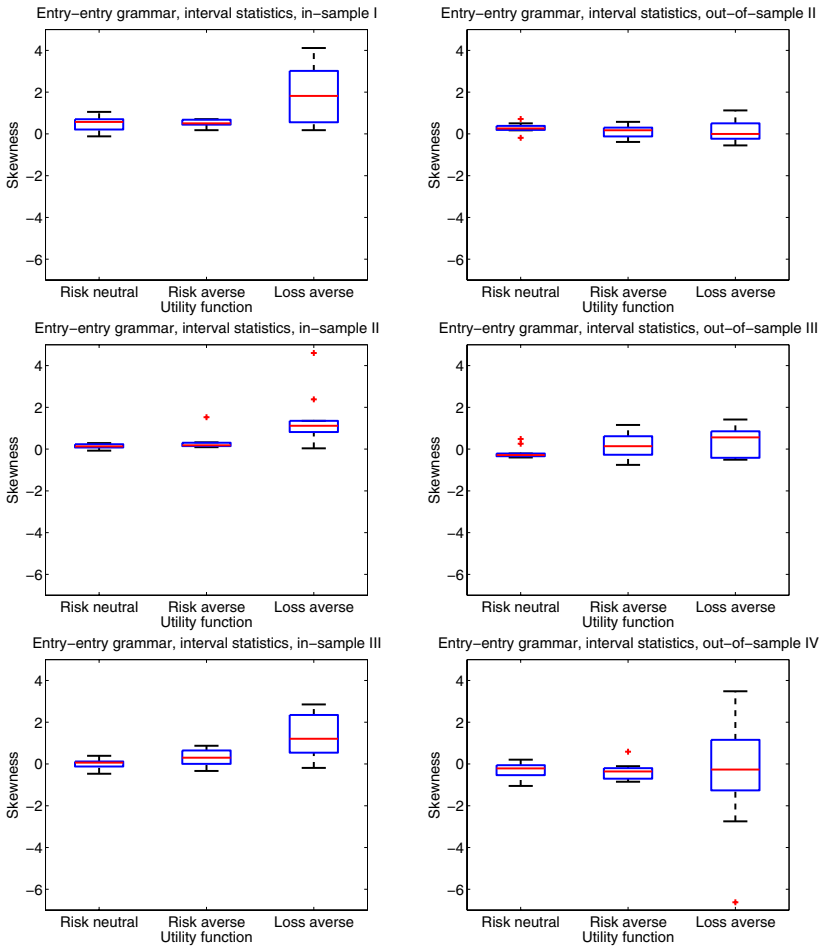


**Fig. 10.5.** Boxplots of the standard deviation of interval returns across the 10 runs for each block and different utility functions.

not lead to an overall rejection of the null hypothesis. It must therefore be concluded that money management has a detrimental effect on utility, since the evolved strategies do not make use of it. Osler [17] offers a possible explanation for this result: in the foreign exchange markets, stop and limit orders tend to be clustered around round numbers giving rise to distinct support and resistance levels, where trend reversals are more likely to occur. This has not been taken into account in this paper. Having concluded that money management does not add significant value, a more detailed analysis of the entry-entry grammar results is provided in the following.

Figures 10.4 to 10.7 show boxplots of the moments of the interval return distributions under risk neutrality, risk aversion and loss aversion. To determine if medians differ across utility functions, the Kruskal-Wallis test is employed. Table 10.6 reports the  $p$ -values for the null hypothesis of equal medians. For the mean interval returns the



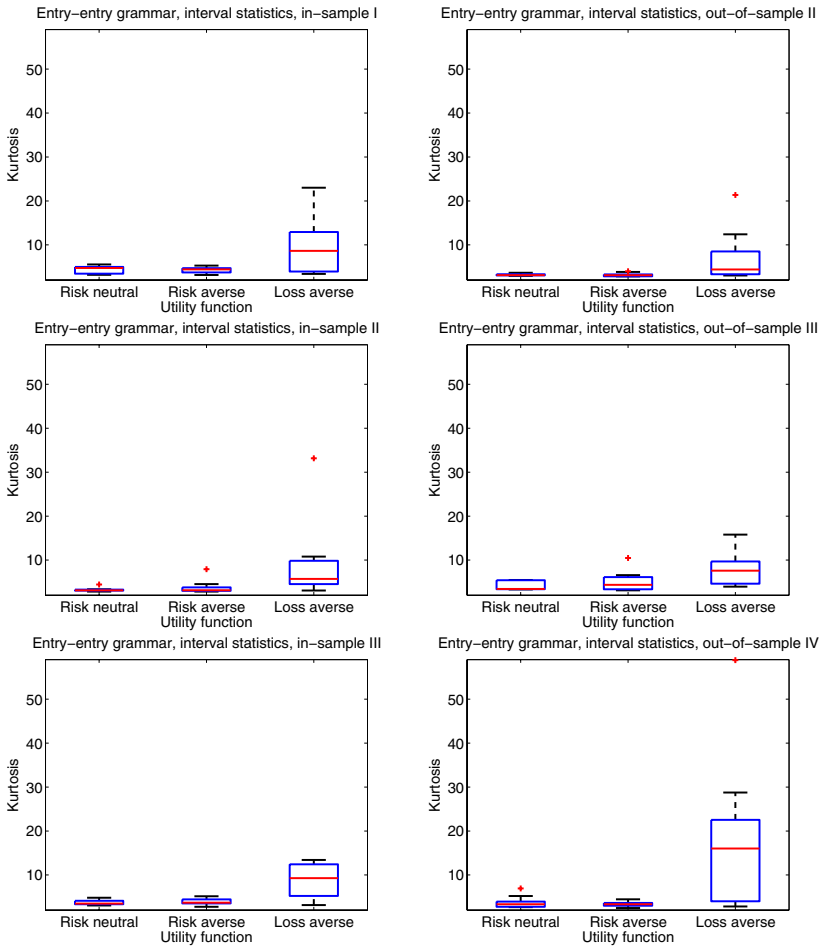


**Fig. 10.6.** Boxplots of the skewness of interval returns across the 10 runs for each block and different utility functions.

different utility functions have the same median values in-sample on Block I and II, but for Block III the median under loss aversion is significantly lower. Out-of-sample on Block II, loss aversion produces lower means, while on Block IV the opposite holds.

Strategies evolved under loss aversion have significantly smaller standard deviations of interval returns both in-sample and out-of-sample. In accordance with [14], the skewness of the interval returns is generally also higher in-sample under loss aversion, but it does not seem to generalise out-of-sample. Finally, the kurtoses are significantly higher across all blocks under loss aversion.

This can be explained from the proportion of time that the strategies have neutral exposure. Under risk neutrality and risk aversion, the strategies generally remain neutral less than 10% of the time, but when loss aversion is introduced it increases significantly to around 70%. Having a neutral position results in zero return. Thus, by increasing the



**Fig. 10.7.** Boxplots of the kurtosis of interval returns across the 10 runs for each block and different utility functions.

time with neutral exposure the effect is that more zero interval returns are introduced. Naturally, this decreases the standard deviation whilst increasing the kurtosis, *ceteris paribus*.

As the random strategies in Figure 10.3 indicate, the required risk premium to enter a market position grows significantly under loss aversion, and as a result the number of opportunities in strategy space decreases. The interval returns under loss aversion have higher skewness in-sample, but do not generalise out-of-sample. This suggests that loss aversion can lead to a higher degree of overfitting. However, loss aversion is not necessarily a bad thing. As mentioned previously the mean interval returns are smaller out-of-sample for Block II and larger for Block IV under loss aversion. The main difference between these two blocks is that during the former there is significant generalization from the in-sample results, while in the latter there is none. Consequently,

**Table 10.6.** Kruskal-Wallis test  $p$ -values for the null hypothesis of equal median moments of the interval return distributions across utility functions.

Mean						
Grammar	In-sample			Out-of-sample		
	I	II	III	II	III	IV
Entry-entry	0.2058	0.1756	0.0025	0.0013	0.6755	0.0649
Entry-exit	0.1194	0.8973	0.0038	0.0097	0.0969	0.0087

Standard deviation						
Grammar	In-sample			Out-of-sample		
	I	II	III	II	III	IV
Entry-entry	0.0003	0.0004	0.0004	0.0004	0.0099	0.0013
Entry-exit	0.0001	0.0110	0.0306	0.0026	0.0363	0.1756

Skewness						
Grammar	In-sample			Out-of-sample		
	I	II	III	II	III	IV
Entry-entry	0.0466	0.0019	0.0039	0.2359	0.5693	0.6941
Entry-exit	0.0373	0.4569	0.3352	0.0985	0.2838	0.2360

Kurtosis						
Grammar	In-sample			Out-of-sample		
	I	II	III	II	III	IV
Entry-entry	0.0451	0.0022	0.0013	0.0087	0.0203	0.0032
Entry-exit	0.0164	0.0118	0.0965	0.0226	0.0049	0.2441

if persistent patterns exists in the data then loss aversion is problematic because it limits the space for viable strategies. However, if persistent patterns have not been uncovered, then it is beneficial because it encourages conservative trading under market frictions and therefore minimises losses.

## 10.6 Conclusions

This chapter evolves trading strategies using genetic programming (GP) on high-frequency tick FX data, an area that has been widely neglected in the literature so far. Furthermore, this chapter proposes a novel quad tree structure for trading system design to allow for a money management system where exit rules can be based on additional indicators and triggers other than the rules for entering positions.

In practice traders often use so-called money management that builds on a different information set when deciding on whether to exit a trade. For example, a stop loss is a measure to control downside risk and exits a position when a loss has exceeded a given threshold. This chapter investigates the potential use of money management by comparing strategies composed of two different grammars. The first is an entry-entry

grammar, where the information set is the same for both entry and exit trees. The second is an entry-exit grammar that has a larger information set including variables such as current profit, drawdown and duration of a trade. Evolving money management as an endogenous feature has not previously been attempted in the literature.

The quad tree architecture consists of four trees each solving a separate task, but mutually dependent for overall performance. Specifically, the functions of the trees are: long entry, long exit, short entry and short exit. Thus, evaluation is contingent on the current market position. For example if the current position is neutral it is possible to go either long or short, but if the current position is long, then the long exit and short entry are evaluated. Making this distinction provides a more accurate description of the decision problem facing real traders.

The trading strategies are evolved using a fitness measure based on the power utility function, where three different kinds of behavior are investigated: risk neutral, risk averse and loss averse. The empirical investigation uses the USD/EUR exchange rate covering the calendar year 2006, sampled at 10 tick intervals. Under market frictions and loss aversion, the strategies spend considerably more time in a neutral position since there are fewer satisfying opportunities. The downside is that generalization can suffer as a result. However, it cannot be concluded that loss aversion always has an adverse effect on performance; it depends on the quality of the patterns discovered in-sample. If the strategies have overfitted the data, or the patterns cease to exist out-of-sample, then loss aversion is beneficial because it promotes cautious trading that limits transaction costs.

When comparing the entry-entry and entry-exit grammar strategies, the null hypothesis of identical median performance is not rejected, neither in-sample nor out-of-sample. Hence, the results are not significantly different. This suggests that money management has a detrimental effect on utility, and raises the question as to why it is extensively used by practitioners. A possible explanation is that stop orders and limit orders, in the foreign exchange market, tend to be clustered around round numbers, thus giving rise to distinct support and resistance levels where trend reversals are more likely to occur [17]. This has not been taken into account in this study, but could be an interesting avenue for future research.

## References

1. Allen, H., Taylor, M.P.: Charts, noise and fundamentals in the London foreign exchange market. *The Economic Journal* 100(400), 49–59 (1990)
2. Bhattacharyya, S., Pictet, O.V., Zumbach, G.: Knowledge-intensive genetic discovery in foreign exchange markets. *IEEE Transactions on Evolutionary Computation* 6(2), 169–181 (2002)
3. Chang, K., Osler, C.L.: Methodical madness: Technical analysis and the irrationality of exchange-rate forecasts. *The Economic Journal* 109, 636–661 (1999)
4. Copeland, T.E., Weston, J.F., Shastri, K.: *Financial Theory and Corporate Policy*. Pearson Addison Wesley (2005)
5. Dacorogna, M.M., Gencay, R., Müller, U.A., Olsen, R.B., Pictet, O.V.: *An Introduction to High-Frequency Finance*. Academic Press, London (2001)
6. Dempster, M.A.H., Jones, C.M.: A real-time adaptive trading system using genetic programming. *Quantitative Finance* 1, 397–413 (2001)

7. Fama, E.F.: Efficient capital markets: A review of theory and empirical work. *Journal of Finance* 25(2), 383–417 (1970)
8. Holland, J.H.: *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (1975)
9. Jonsson, H., Madjidi, P., Nordahl, M.G.: Evolution of trading rules for the FX market or how to make money out of GP. Tech.rep., Institute of Theoretical Physics, Chalmers University of Technology (1997)
10. Kahneman, D., Tversky, A.: Prospect theory: An analysis of decision under risk. *Econometrica* 47(2), 263–291 (1979)
11. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge (1992)
12. LeBaron, B.: Technical trading profitability in foreign exchange markets in the 1990's. Tech.rep., Brandeis University (2002)
13. Maillat, B., Michel, T.: Further insights on the puzzle of technical analysis profitability. *The European Journal of Finance* 6, 196–224 (2000)
14. Maringer, D.: Risk preferences and loss aversion in portfolio optimization. In: Kontoghiorhes, E.J., Rustem, B., Winker, P. (eds.) *Computational Methods in Financial Engineering*, pp. 27–45. Springer, Heidelberg (2008)
15. Meese, R., Rogoff, K.: Empirical exchange rate models of the seventies, do they fit out-of-sample? *Journal of International Economics* 14, 3–24 (1983)
16. Neely, C.J., Weller, P.A.: Intraday technical trading in the foreign exchange market. Tech.rep., Federal Reserve Bank of St. Louis (1999)
17. Osler, C.L.: Currency orders and exchange rate dynamics: An explanation for the predictive success of technical analysis. *The Journal of Finance* 58(5), 1791–1819 (2003)
18. Saks, P., Maringer, D.: Genetic programming in statistical arbitrage. In: Giacobini, M. (ed.) *EvoWorkshops 2008*. LNCS, vol. 4974, pp. 73–82. Springer, Heidelberg (2008a)
19. Saks, P., Maringer, D.: Single versus multiple tree genetic programming for dynamic decision making. Tech.rep., Centre for Computational Finance and Economic Agents, University of Essex (2008b)
20. Thaler, R., Tversky, A., Kahneman, D., Schwartz, A.: The effect of myopia and loss aversion on risk taking: An experimental test. *The Quarterly Journal of Economics* 112(2), 647–661 (1997)
21. Whitley, D.: A genetic algorithm tutorial. *Statistics and Computing* 4, 65–85 (1994)

## A Interval Statistics

### A.1 Entry-Entry Grammar, Market Frictions

**Table 10.7.** Interval statistics of entry-entry grammar strategies under market frictions and risk neutrality.

Risk neutral												
Run	In-sample I						Out-of-sample II					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	7.16	0.000	7.22	36.04	0.70	4.98	5.09	0.002	5.16	37.64	0.37	3.42
2.	4.79	0.000	4.86	37.91	0.55	4.71	4.75	0.009	4.84	43.44	0.17	2.95
3.	7.66	0.000	7.72	35.61	1.06	5.11	4.28	0.009	4.36	40.93	0.71	3.08
4.	4.60	0.000	4.69	40.92	-0.12	3.19	2.02	0.013	2.11	41.86	0.25	3.06
5.	9.74	0.000	9.80	33.54	0.21	3.46	3.25	0.011	3.33	38.19	-0.19	3.14
6.	4.74	0.000	4.81	37.55	0.63	4.78	5.52	0.000	5.61	42.03	0.27	3.05
7.	9.36	0.000	9.43	36.40	0.70	4.83	3.45	0.011	3.55	43.11	0.24	3.09
8.	13.84	0.000	13.90	33.13	0.17	3.34	3.00	0.011	3.07	37.87	0.51	3.66
9.	8.01	0.000	8.08	35.44	0.57	5.54	9.54	0.000	9.63	41.17	0.19	3.32
10.	7.90	0.000	7.97	37.61	0.58	4.70	5.18	0.002	5.27	42.90	0.39	3.27
Med	7.78	0.000	7.85	36.22	0.58	4.74	4.51	0.009	4.60	41.51	0.26	3.12

Run	In-sample II						Out-of-sample III					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	11.43	0.000	11.52	41.11	0.08	3.06	1.05	0.011	1.11	36.61	0.49	5.41
2.	7.08	0.000	7.17	41.47	0.12	3.25	0.03	0.016	0.08	32.73	-0.28	3.39
3.	10.72	0.000	10.81	41.09	0.12	3.10	-0.23	0.051	-0.18	32.89	-0.26	3.32
4.	9.87	0.000	9.95	39.36	-0.01	4.45	-1.98	0.065	-1.93	32.47	-0.29	3.40
5.	6.62	0.000	6.71	42.59	0.18	3.11	-1.36	0.053	-1.31	34.23	-0.30	3.38
6.	6.32	0.000	6.41	41.72	0.24	2.83	-3.12	0.068	-3.06	35.27	-0.35	3.38
7.	8.34	0.000	8.42	40.18	0.24	3.27	-1.98	0.065	-1.94	28.89	-0.21	5.14
8.	11.27	0.000	11.36	40.92	-0.07	3.36	3.96	0.000	4.02	33.51	-0.34	3.48
9.	7.34	0.000	7.45	46.10	0.29	3.02	-4.02	0.069	-3.97	34.34	0.26	5.47
10.	11.06	0.000	11.15	41.48	0.14	3.11	-2.11	0.066	-2.08	27.65	-0.40	5.43
Med	9.10	0.000	9.19	41.29	0.13	3.11	-1.67	0.059	-1.62	33.20	-0.29	3.44

Run	In-sample III						Out-of-sample IV					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	4.89	0.000	4.95	33.05	0.08	3.06	-5.02	0.054	-4.97	31.04	-0.06	2.76
2.	7.92	0.000	7.96	29.33	0.39	4.40	-4.45	0.054	-4.41	28.60	-0.23	3.95
3.	5.04	0.000	5.07	23.50	-0.47	4.12	-4.01	0.052	-3.97	26.79	-0.53	3.32
4.	10.81	0.000	10.86	32.10	0.12	3.38	-6.40	0.069	-6.35	29.50	-0.20	2.74
5.	4.54	0.000	4.59	30.53	-0.12	3.54	0.74	0.018	0.79	30.24	-0.22	3.60
6.	8.85	0.000	8.91	32.51	0.38	3.44	-3.34	0.051	-3.30	28.85	0.21	3.36
7.	11.93	0.000	12.00	35.17	0.05	3.33	-1.36	0.051	-1.29	36.53	-1.05	6.93
8.	7.96	0.000	8.01	33.38	0.08	3.89	-5.44	0.067	-5.39	30.07	0.05	2.68
9.	5.64	0.000	5.70	34.71	0.05	3.17	-0.85	0.049	-0.80	33.12	-0.09	2.80
10.	5.46	0.000	5.50	29.05	-0.38	4.81	0.59	0.018	0.62	25.64	-0.75	5.21
Med	6.78	0.000	6.83	32.30	0.06	3.49	-3.67	0.051	-3.63	29.79	-0.21	3.34

**Table 10.8.** Interval statistics of entry-entry grammar strategies under market frictions and risk aversion.

Risk averse												
Run	In-sample I						Out-of-sample II					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	11.19	0.000	13.81	37.62	0.49	4.32	2.54	0.000	5.29	40.02	0.23	3.28
2.	6.17	0.000	8.56	36.96	0.49	5.28	1.82	0.008	5.16	44.04	0.17	2.83
3.	6.06	0.000	8.25	35.52	0.68	5.10	0.37	0.011	3.60	42.76	-0.38	2.85
4.	3.71	0.000	5.41	31.42	0.58	3.53	-3.05	0.046	-1.50	30.09	0.58	3.85
5.	2.38	0.000	4.83	38.03	0.69	4.44	1.89	0.008	5.01	42.68	0.30	3.04
6.	2.89	0.000	5.72	40.46	0.18	4.66	1.52	0.008	4.85	44.03	0.18	2.94
7.	6.90	0.000	8.78	32.39	0.24	3.15	-2.48	0.044	1.03	44.80	-0.12	4.00
8.	5.79	0.000	8.34	38.32	0.44	4.41	1.98	0.000	4.86	40.71	0.01	2.86
9.	8.22	0.000	10.50	35.87	0.71	4.70	3.34	0.000	6.31	41.61	0.33	3.16
10.	7.08	0.000	9.92	40.32	0.53	3.74	-7.97	0.051	-4.94	40.71	-0.30	3.17
Med	6.11	0.000	8.45	37.29	0.51	4.42	1.67	0.008	4.85	42.14	0.17	3.10

Run	In-sample II						Out-of-sample III					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	7.96	0.000	11.01	41.38	0.14	3.13	-3.62	0.047	-2.34	26.58	-0.76	5.14
2.	8.57	0.000	11.46	40.15	0.19	2.84	-4.06	0.055	-2.79	26.35	-0.75	5.23
3.	8.63	0.000	11.83	42.29	0.16	3.01	-0.90	0.044	1.43	37.24	0.72	6.58
4.	7.98	0.000	11.04	41.39	0.14	3.13	-2.27	0.045	-0.35	33.10	-0.26	3.27
5.	2.54	0.000	4.28	31.57	0.09	4.54	-1.21	0.045	-0.31	22.83	0.23	3.59
6.	4.86	0.000	5.61	20.60	1.53	7.94	0.47	0.001	0.79	13.66	1.16	10.48
7.	3.95	0.000	6.57	38.92	0.30	3.63	-0.20	0.028	1.32	29.69	0.11	3.12
8.	4.87	0.000	7.27	37.15	0.32	3.80	-0.84	0.044	0.47	27.62	0.16	3.35
9.	2.45	0.000	5.64	43.07	0.18	3.03	-1.91	0.045	-0.03	32.78	-0.27	3.36
10.	10.18	0.000	13.37	41.84	0.19	2.85	-6.02	0.065	-3.68	36.69	0.62	6.13
Med	6.41	0.000	9.14	40.77	0.18	3.13	-1.56	0.045	-0.17	28.65	0.14	4.36

Run	In-sample III						Out-of-sample IV					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	5.32	0.000	6.91	30.19	0.65	3.68	-5.38	0.058	-3.96	28.39	0.59	3.41
2.	10.60	0.000	12.23	28.92	0.33	2.76	-4.56	0.056	-3.16	27.71	-0.70	3.49
3.	3.34	0.000	5.78	37.42	0.00	3.52	-3.93	0.055	-2.05	32.30	-0.62	3.08
4.	6.19	0.000	8.09	32.64	0.22	3.44	-8.62	0.080	-7.07	28.52	-0.20	3.00
5.	5.17	0.000	6.76	29.71	-0.34	5.15	-4.91	0.056	-3.56	27.51	-0.10	2.48
6.	8.43	0.000	9.69	25.70	0.39	4.67	-1.21	0.048	0.37	30.04	-0.37	3.65
7.	11.82	0.000	14.28	36.32	0.87	4.44	-3.50	0.054	-2.10	28.04	-0.34	3.25
8.	10.86	0.000	12.87	32.07	-0.25	3.66	-1.81	0.052	-0.19	30.02	-0.81	3.76
9.	5.71	0.000	7.22	29.08	0.27	3.73	-7.15	0.076	-5.54	29.50	-0.23	3.00
10.	8.59	0.000	9.94	26.98	0.87	4.26	-1.17	0.048	-0.12	24.30	-0.85	4.47
Med	7.31	0.000	8.89	29.95	0.30	3.70	-4.24	0.056	-2.63	28.46	-0.36	3.33

**Table 10.9.** Interval statistics of entry-entry grammar strategies under market frictions and loss aversion.

Loss averse												
Run	In-sample I						Out-of-sample II					
	F	p	Mean	Std	Skew	Kurt	F	p	Mean	Std	Skew	Kurt
1.	5.17	0.000	7.74	27.89	0.18	3.39	-4.33	0.055	0.15	34.84	0.13	3.19
2.	4.38	0.000	5.04	16.67	2.55	10.10	-0.78	0.047	-0.41	8.83	1.13	21.36
3.	4.02	0.000	6.63	30.40	0.73	5.54	-10.30	0.060	-5.05	35.90	-0.55	4.11
4.	7.17	0.000	8.39	23.20	2.15	7.71	-3.37	0.052	-0.42	28.23	-0.28	5.37
5.	3.87	0.000	4.12	11.31	3.15	12.93	0.53	0.001	1.22	14.04	1.07	12.40
6.	2.46	0.000	2.82	13.04	4.11	23.01	0.00	0.031	0.00	0.00	NaN	NaN
7.	4.65	0.000	6.59	27.21	1.50	9.58	-0.97	0.049	1.70	27.73	0.32	4.41
8.	4.09	0.000	7.83	36.16	0.38	3.94	-7.66	0.058	-1.77	39.86	-0.03	3.04
9.	4.06	0.000	4.84	19.02	3.02	16.06	-3.47	0.054	-1.55	21.73	-0.21	7.20
10.	8.90	0.000	11.56	30.96	0.56	3.82	-3.37	0.052	1.88	39.61	-0.00	3.37
Med	4.24	0.000	6.61	25.21	1.82	8.65	-3.37	0.052	-0.20	27.98	-0.00	4.41

Run	In-sample II						Out-of-sample III					
	F	p	Mean	Std	Skew	Kurt	F	p	Mean	Std	Skew	Kurt
1.	3.78	0.000	4.90	19.10	1.12	7.32	0.40	0.000	0.89	10.93	0.82	15.79
2.	0.75	0.001	0.88	7.00	4.60	33.16	0.00	0.033	0.00	0.00	NaN	NaN
3.	1.58	0.000	2.11	12.95	1.35	9.83	-3.61	0.054	-1.86	20.66	-0.37	7.59
4.	3.70	0.000	6.44	28.84	0.72	5.82	-3.13	0.053	0.37	29.83	-0.51	3.96
5.	2.59	0.000	3.32	15.07	0.92	5.63	-0.60	0.034	0.04	12.11	0.56	11.51
6.	6.94	0.000	8.21	21.33	1.26	5.47	0.49	0.000	1.91	21.32	1.42	8.85
7.	3.67	0.000	4.28	15.38	2.38	10.80	-2.85	0.051	-0.70	24.05	-0.39	9.08
8.	9.53	0.000	12.31	32.31	1.12	4.52	-3.84	0.054	-1.62	22.81	0.95	4.59
9.	6.14	0.000	11.21	40.95	0.04	3.08	-9.06	0.077	-4.04	36.42	0.70	6.85
10.	4.42	0.000	7.48	31.15	0.82	4.53	-3.55	0.053	-0.07	29.61	-0.51	4.63
Med	3.74	0.000	5.67	20.22	1.12	5.72	-2.99	0.052	-0.04	22.07	0.56	7.59

Run	In-sample III						Out-of-sample IV					
	F	p	Mean	Std	Skew	Kurt	F	p	Mean	Std	Skew	Kurt
1.	4.59	0.000	5.21	14.44	1.20	5.64	-1.19	0.044	1.42	26.13	-0.04	3.70
2.	2.10	0.000	2.49	11.05	2.35	11.73	-0.11	0.040	0.22	8.30	3.48	28.76
3.	1.25	0.000	1.66	10.50	-0.19	12.43	-0.73	0.044	-0.43	7.81	-6.62	58.87
4.	2.46	0.000	3.13	16.69	2.67	13.42	-8.36	0.049	-5.10	25.80	-0.35	4.02
5.	3.92	0.000	5.34	20.78	0.69	5.23	-0.66	0.044	-0.05	11.85	-2.75	22.54
6.	3.12	0.000	5.00	25.28	0.54	5.09	-0.14	0.041	0.32	11.10	1.16	19.12
7.	1.12	0.000	2.34	19.06	1.21	8.98	0.44	0.015	0.96	11.69	-1.26	20.11
8.	9.04	0.000	12.24	32.03	0.19	3.15	-7.81	0.049	-3.60	30.98	-0.18	2.84
9.	3.73	0.000	5.08	21.74	2.05	9.57	-1.26	0.044	-0.30	14.32	-0.44	7.39
10.	1.88	0.000	2.01	6.93	2.85	12.77	0.85	0.015	1.01	6.29	2.19	12.93
Med	2.79	0.000	4.07	17.87	1.21	9.28	-0.69	0.044	0.09	11.77	-0.27	16.03



## A.2 Entry-Exit Grammar, Market Frictions

**Table 10.10.** Interval statistics of entry-exit grammar strategies under market frictions and risk neutrality.

Risk neutral												
Run	In-sample I						Out-of-sample II					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	7.96	0.000	8.03	38.06	0.41	4.93	4.05	0.016	4.15	44.17	0.18	2.91
2.	7.10	0.000	7.17	36.01	0.99	4.62	3.12	0.022	3.17	31.79	1.01	5.51
3.	9.75	0.000	9.82	36.83	0.63	4.57	1.97	0.025	2.06	41.78	0.30	3.17
4.	7.49	0.000	7.56	37.73	0.21	4.01	4.06	0.016	4.15	41.69	-0.08	2.88
5.	8.78	0.000	8.87	40.85	0.21	5.85	6.13	0.000	6.22	41.61	0.24	2.93
6.	5.85	0.000	5.93	38.61	0.36	4.74	4.68	0.013	4.78	43.84	0.14	2.97
7.	8.17	0.000	8.24	38.08	0.47	5.13	4.72	0.012	4.83	47.29	-0.10	3.46
8.	11.10	0.000	11.16	33.35	0.21	2.89	-0.20	0.064	-0.15	32.16	0.38	3.07
9.	9.51	0.000	9.57	35.29	0.77	4.94	4.91	0.011	5.00	41.63	0.22	3.25
10.	7.99	0.000	8.06	36.37	0.77	5.23	5.11	0.003	5.20	42.76	0.20	3.08
Med	8.08	0.000	8.15	37.28	0.44	4.84	4.37	0.014	4.46	41.73	0.21	3.08

Run	In-sample II						Out-of-sample III					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	6.19	0.000	6.26	37.96	0.48	3.32	-4.05	0.076	-4.00	31.39	0.18	2.59
2.	8.94	0.000	9.04	42.87	-0.29	3.16	-1.94	0.058	-1.88	34.30	-0.34	3.16
3.	7.52	0.000	7.58	33.83	0.16	3.45	-1.60	0.046	-1.57	25.94	-0.04	3.61
4.	8.39	0.000	8.45	36.50	0.38	3.79	-2.07	0.059	-2.03	28.12	0.20	3.01
5.	7.31	0.000	7.38	36.46	0.74	3.99	-1.19	0.044	-1.14	30.96	0.44	2.91
6.	7.35	0.000	7.44	41.58	-0.05	2.98	-1.27	0.044	-1.21	34.62	-0.32	3.13
7.	10.80	0.000	10.87	35.39	0.39	3.98	1.47	0.007	1.50	26.30	-0.22	4.64
8.	8.93	0.000	9.00	36.21	0.09	3.74	0.39	0.014	0.43	29.23	-0.24	2.89
9.	6.29	0.000	6.39	42.49	0.09	3.13	-1.51	0.045	-1.46	32.76	-0.17	2.85
10.	12.01	0.000	12.11	41.73	0.10	3.06	-3.66	0.075	-3.59	36.79	0.64	6.24
Med	7.95	0.000	8.02	37.23	0.13	3.38	-1.56	0.045	-1.51	31.18	-0.11	3.07

Run	In-sample III						Out-of-sample IV					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	6.25	0.000	6.30	31.21	0.47	3.57	-9.26	0.109	-9.20	34.22	-2.15	15.26
2.	6.25	0.000	6.31	32.15	-0.12	2.72	-4.96	0.079	-4.91	29.79	-0.24	2.83
3.	4.64	0.000	4.69	30.39	0.28	3.53	-5.26	0.085	-5.21	31.07	0.21	3.47
4.	6.84	0.000	6.89	31.49	0.31	3.61	-6.54	0.095	-6.51	24.37	-0.14	3.68
5.	8.83	0.000	8.87	27.66	-0.59	3.54	-5.57	0.090	-5.53	28.63	-0.22	2.66
6.	4.74	0.000	4.79	31.61	0.21	3.23	-5.48	0.090	-5.43	29.59	0.34	3.29
7.	4.75	0.000	4.78	23.93	0.53	5.24	-4.93	0.078	-4.88	31.14	0.18	3.43
8.	4.67	0.000	4.71	28.11	0.88	4.38	-2.16	0.070	-2.14	21.90	0.65	3.20
9.	5.60	0.000	5.65	31.46	0.39	3.67	-4.97	0.079	-4.93	25.63	0.13	2.78
10.	5.66	0.000	5.69	22.99	0.98	7.14	-5.22	0.083	-5.17	29.84	0.22	3.76
Med	5.63	0.000	5.67	30.80	0.35	3.59	-5.24	0.084	-5.19	29.69	0.16	3.36

**Table 10.11.** Interval statistics of entry-exit grammar strategies under market frictions and risk aversion.

Risk averse												
Run	In-sample I						Out-of-sample II					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	5.52	0.000	7.96	37.10	-0.06	3.63	-8.87	0.091	-5.56	42.50	-0.29	3.35
2.	10.49	0.000	13.22	38.69	0.46	3.78	0.86	0.027	3.80	40.80	-0.33	3.48
3.	3.99	0.000	6.36	36.98	0.34	4.57	1.35	0.023	4.86	45.04	0.03	3.56
4.	7.57	0.000	9.75	35.21	0.90	4.95	-0.10	0.053	2.02	35.23	0.36	5.16
5.	3.45	0.000	6.02	38.82	0.56	4.33	-10.87	0.105	-7.42	43.35	0.04	2.91
6.	3.66	0.000	5.93	36.65	0.83	4.50	-2.51	0.061	-0.26	36.31	0.48	4.84
7.	3.72	0.000	5.97	36.36	0.72	4.69	2.37	0.003	4.78	37.62	0.50	3.50
8.	5.73	0.000	7.34	30.65	1.31	5.93	-4.36	0.063	-2.05	36.70	0.63	5.21
9.	5.68	0.000	7.96	36.50	0.84	3.65	-4.78	0.064	-2.70	34.37	0.09	2.91
10.	7.10	0.000	9.32	35.43	0.40	3.69	-0.57	0.056	2.14	39.50	-0.06	3.20
Med	5.60	0.000	7.65	36.58	0.64	4.41	-1.54	0.058	0.88	38.56	0.06	3.49

Run	In-sample II						Out-of-sample III					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	8.01	0.000	10.76	39.08	0.13	3.95	-0.91	0.056	0.63	29.80	0.14	3.26
2.	5.08	0.000	7.02	32.74	-0.52	3.43	-4.67	0.102	-2.87	31.62	-0.45	4.06
3.	5.19	0.000	7.59	37.29	0.60	3.39	-5.90	0.109	-4.14	31.41	0.15	3.10
4.	6.22	0.000	8.35	34.45	-0.05	4.09	-3.64	0.076	-1.54	34.38	-0.35	3.15
5.	4.44	0.000	5.32	22.35	1.04	8.71	1.69	0.000	1.92	11.53	2.29	13.51
6.	8.29	0.000	9.52	25.56	0.67	5.75	-0.97	0.057	-0.30	19.80	0.65	7.49
7.	3.58	0.000	6.82	43.13	0.04	2.85	-4.00	0.089	-1.94	33.95	-0.38	3.23
8.	7.17	0.000	9.00	31.84	0.40	4.09	-1.38	0.061	0.10	28.87	-0.78	4.80
9.	7.02	0.000	8.25	25.72	-0.03	3.87	-3.24	0.071	-1.86	27.72	-0.65	5.70
10.	10.94	0.000	12.35	27.02	1.84	7.50	-2.14	0.067	-1.15	23.66	-0.36	7.86
Med	6.62	0.000	8.30	32.29	0.27	4.02	-2.69	0.069	-1.34	29.33	-0.36	4.43

Run	In-sample III						Out-of-sample IV					
	F	<i>p</i>	Mean	Std	Skew	Kurt	F	<i>p</i>	Mean	Std	Skew	Kurt
1.	8.30	0.000	9.50	25.43	1.10	6.52	-3.01	0.067	-2.22	21.16	-0.04	6.64
2.	9.00	0.000	10.77	30.41	-0.44	3.69	0.57	0.032	2.04	29.09	0.05	4.14
3.	4.55	0.000	6.84	35.98	-0.15	2.75	-4.00	0.070	-2.07	32.70	-0.60	2.96
4.	8.48	0.000	10.31	31.46	0.17	4.34	-3.34	0.068	-1.46	32.49	-0.46	3.84
5.	5.87	0.000	8.70	39.87	-0.18	2.70	2.03	0.026	3.53	29.07	-0.60	3.69
6.	4.82	0.000	6.81	34.23	0.97	4.75	-6.40	0.076	-4.73	30.56	0.36	3.40
7.	5.48	0.000	7.00	29.46	0.66	3.81	-5.78	0.075	-4.20	29.80	0.30	2.98
8.	5.84	0.000	6.45	17.97	0.23	8.76	-6.41	0.076	-5.13	26.29	-0.47	3.78
9.	5.06	0.000	5.90	21.78	1.05	7.63	-5.15	0.073	-3.81	27.15	-0.41	3.58
10.	8.16	0.000	9.53	26.77	-0.32	2.76	-9.29	0.104	-7.28	32.50	-0.42	3.05
Med	5.85	0.000	7.85	29.94	0.20	4.07	-4.57	0.072	-3.01	29.45	-0.41	3.64

**Table 10.12.** Interval statistics of entry-exit grammar strategies under market frictions and loss aversion.

Loss averse												
Run	In-sample I						Out-of-sample II					
	F	p	Mean	Std	Skew	Kurt	F	p	Mean	Std	Skew	Kurt
1.	5.01	0.000	6.87	28.29	2.21	12.55	-3.99	0.076	1.72	41.13	0.03	3.34
2.	7.56	0.000	8.57	20.16	2.66	14.91	0.29	0.002	1.99	21.41	0.48	7.40
3.	1.77	0.000	2.04	11.09	4.22	30.31	0.00	0.038	0.00	0.00	NaN	NaN
4.	6.81	0.000	8.56	25.44	0.97	4.54	-10.66	0.090	-4.13	41.46	-0.06	3.05
5.	3.13	0.000	3.50	12.51	3.05	17.91	-3.91	0.076	-1.52	24.67	-1.29	9.98
6.	7.42	0.000	10.22	30.43	0.38	3.23	-6.25	0.080	-1.60	35.01	-0.45	4.26
7.	4.13	0.000	6.32	25.82	-0.57	5.74	0.36	0.002	3.73	32.43	0.09	5.42
8.	4.71	0.000	5.74	19.71	1.53	6.46	1.35	0.000	4.74	32.16	0.02	3.64
9.	3.65	0.000	4.16	14.32	2.49	9.98	-2.17	0.063	-0.69	20.17	-0.67	16.62
10.	5.84	0.000	7.00	19.80	0.70	4.75	-0.61	0.044	1.74	24.69	0.13	3.27
Med	4.86	0.000	6.60	19.98	1.87	8.22	-1.39	0.053	0.86	28.43	0.02	4.26

Run	In-sample II						Out-of-sample III					
	F	p	Mean	Std	Skew	Kurt	F	p	Mean	Std	Skew	Kurt
1.	6.16	0.000	9.51	33.59	0.23	5.63	-0.13	0.032	1.42	20.18	-1.05	10.12
2.	3.14	0.000	7.42	37.07	0.06	3.59	-3.50	0.057	-0.05	29.79	-0.45	4.16
3.	6.98	0.000	8.83	25.98	1.60	10.93	-6.78	0.091	-2.14	34.08	-0.33	3.22
4.	5.74	0.000	7.68	25.96	0.29	5.30	-4.70	0.064	-1.38	28.42	-0.58	4.40
5.	3.25	0.000	4.05	17.54	1.80	8.19	-0.20	0.032	0.21	9.47	0.16	15.98
6.	3.81	0.000	7.63	34.93	-0.08	4.07	-1.85	0.035	1.06	27.77	-0.31	4.23
7.	6.47	0.000	8.07	25.52	1.52	5.77	-0.84	0.033	1.38	25.28	0.08	5.07
8.	1.48	0.000	2.12	13.98	1.04	7.58	-0.79	0.033	-0.02	13.33	0.55	9.74
9.	6.78	0.000	11.88	41.46	0.14	3.06	-1.11	0.033	-0.38	15.32	-0.63	39.94
10.	6.35	0.000	8.96	29.15	0.25	4.99	-0.21	0.001	1.84	21.65	0.28	5.60
Med	5.95	0.000	7.87	27.57	0.27	5.46	-0.98	0.033	0.09	23.46	-0.32	5.34

Run	In-sample III						Out-of-sample IV					
	F	p	Mean	Std	Skew	Kurt	F	p	Mean	Std	Skew	Kurt
1.	3.70	0.000	4.74	17.97	0.38	5.23	-0.41	0.059	2.37	27.12	-0.38	4.17
2.	2.68	0.000	2.92	9.59	0.94	8.72	1.85	0.000	2.38	13.49	1.14	15.72
3.	5.97	0.000	6.93	16.14	0.48	3.92	-4.70	0.086	-2.42	20.94	-0.45	3.76
4.	3.33	0.000	5.63	25.78	0.64	5.28	-9.16	0.102	-5.47	27.05	-0.15	2.87
5.	3.20	0.000	3.80	14.14	1.15	9.06	0.87	0.016	2.33	19.30	-0.05	4.60
6.	2.89	0.000	4.25	22.20	1.47	8.61	-2.96	0.080	-0.86	21.89	-0.04	4.34
7.	5.03	0.000	8.18	31.75	0.26	3.32	-5.66	0.088	-1.48	32.58	0.18	3.23
8.	3.68	0.000	6.97	31.75	0.25	4.06	-9.03	0.101	-4.84	30.50	0.23	3.32
9.	2.73	0.000	4.44	20.93	0.15	4.36	-9.52	0.113	-5.18	31.07	0.21	3.47
10.	4.86	0.000	5.80	17.60	0.82	5.00	-1.61	0.069	0.75	23.91	-0.23	3.64
Med	3.50	0.000	5.19	19.45	0.56	5.12	-3.83	0.083	-1.17	25.48	-0.04	3.70

# Interday and Intraday Stock Trading Using Probabilistic Adaptive Mapping Developmental Genetic Programming and Linear Genetic Programming

Garnett Wilson and Wolfgang Banzhaf

Memorial University of Newfoundland, St. John's, NL, Canada  
gwilson@cs.mun.ca, banzhaf@cs.mun.ca

**Summary.** A developmental co-evolutionary genetic programming approach (PAM DGP) is compared to a standard linear genetic programming (LGP) implementation for trading of stocks in the technology sector. Both interday and intraday data for these stocks were analyzed, where both implementations were found to be impressively robust to market fluctuations while reacting efficiently to opportunities for profit. PAM DGP proved slightly more reactive to market changes compared to LGP for intraday data, where the converse held true for interday data. Both implementations had very impressive accuracy in choosing both profitable buy trades and sells that prevented losses for both interday and intraday stock data. These successful trades occurred in the context of moderately active trading for interday prices and lower levels of trading for intraday prices.

## 11.1 Introduction

Technical analysis of the stock market involves attempts to examine the past effects of market movements in order to anticipate what traders will do next to affect the market. Such analysis involves the use of technical indicators to examine price trends and trading volume in order to identify the likely future trading activity and change in price of an asset [1]. In recent years, a number of Evolutionary Computation-inspired algorithms, including genetic programming (GP), have been applied to the analysis of financial markets with a reassuring degree of success. This paper explores the use of a developmental GP system, Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP), that uses co-operative co-evolution of genotype solutions and genotype-phenotype mappings, as well as Linear Genetic Programming (LGP), for both interday and intraday stock trading. While the encoding of functions is static for LGP, PAM DGP allows emphasis of particular functions over others.

The following section describes previous related approaches to stock analysis. Section 11.3 describes the LGP and PAM DGP implementations and their application to trading of individual stocks, as well as the function set for this domain and the related interpretation of an individual's genotype. Results are provided in Section 11.4 for the interday analysis of three technology sector stocks, with intraday analysis following in Section 11.5. Conclusions and future work follow in Section 11.6.

## 11.2 Related Approaches to Stock Prediction

Genetic programming approaches have met with considerable success when applied to stock analysis. Yan et al. have shown standard GP to outperform other machine learning techniques such as support vector machines for application to portfolio optimization in highly volatile markets, where this success is attributed to adaptation to optimize profits rather than simply predict returns [7]. Furthermore, the authors found that GP was superior in its balance of Return On Investment (ROI) and robustness to volatility. LGP has been applied to market analysis previously by Grosnan et al. [4], where Nasdaq and Nifty indices were examined. Multi-expression programming (MEP), LGP, and an MEP/LGP ensemble were found to surpass the predictive performance of the particular neural network or neuro-fuzzy implementations chosen by the authors for next day prediction of stock prices. The PAM DGP algorithm that is used in this study relies on a co-evolutionary mechanism. A co-evolutionary process has also been applied to the creation of trading rules by Drezewski and Sepielak [3] where one species represented market entry strategies and one species represented exit strategies. In addition, a multi-agent version of the co-evolutionary algorithm and evolutionary algorithm were tried. For the particular data set used by the authors, the multi-agent co-evolutionary approach generated the most profit. To the authors' knowledge, developmental GP had not been applied to stock market analysis until the original study by the authors on which this chapter is partially based [6].

In terms of the application of the GP algorithm to interday trading rule generation, a technique somewhat similar to the grammatical evolution (GE) approach of Brabazon and O'Neill [1] was adopted: After a period of initial training, the best evolved rules in the population were used to trade live for a window of  $n$  days. The window is then shifted ahead and the current population is retrained on the data within the window on which it was previously trading live in order to trade live on the following  $n$  days, and so on. The authors compare two versions of the GE system, one that maintains its population across window-based training periods and one that re-initializes the population with each window shift / training period. The authors found that maintaining the populations, rather than re-initializing them with each window, provided better trading rules that yielded greater profits. As detailed in the following section, our technique uses a shifting window of length 5 (increments of one day for interday analysis, minute interval ticks for intraday), but shifts only in increments of 1 day/tick. Following the findings and recommendations of [1], populations are not re-evolved with the shifting of each window.

## 11.3 Stock Analysis Using Developmental and Linear GP

Genetic programming is one of a family of algorithms in machine learning classified as evolutionary methods. In such methods, a population of candidate solutions (called "individuals") are ranked in their ability to perform the objective of solving an optimization problem based on some measure of error or success, called a "fitness" function. The individuals in the population are ranked using the fitness function, and the fittest individuals are biased for selection as parents and used to create a new population of

solutions. The genetic material composing the parents is then manipulated, after being copied, to create children who typically replace some individuals originally in the population. Population size thus does not typically change. The manipulation of genetic material is accomplished through the use of the two operators of crossover and mutation. Crossover allows swapping of genetic material between two individuals, while mutation causes the creation of new genetic material by altering the genotype that is already present.

### 11.3.1 Description of PAM DGP and LGP Algorithms

This work uses a well-established variant of genetic programming called “linear genetic programming” or “LGP” [2]. In this variant, the genetic material of the individuals (called “genotype”) has the form of a linear list of instructions. Program execution is that of a simple register machine, and instructions are made up of opcodes and operands. The opcodes correspond to functions in a functional set, with arguments to the functions being considered a terminal set. As the program executes, it alters the contents of internal registers and perhaps a separate solution register. The structure of a linear GP individual is depicted below in Figure 11.1. When the bit strings are interpreted, they correspond to members of the functional sets to produce a solution that makes semantic sense in terms of the original problem, also called the “phenotype.” For instance, the binary sequence “011” in the individual’s genotype could be interpreted as the functional set member “addition” in the phenotype. All the instructions in the phenotype are then evaluated to determine its fitness.

A modern trend in genetic programming, called “developmental genetic programming (DGP)” has been to develop models that more closely mirror the higher-level developmental processes of nature. One type of developmental GP is accomplished by using a genotype-phenotype mapping as an intermediary between an individual’s genotype and phenotype. Specifically, a genotype-phenotype mapping is the encoding of a phenotypic symbol by one or more codons, where a codon is a non-zero contiguous bit sequence from a binary genotype. The biological analogue of the evolution of a genotype-phenotype mapping in a developmental system, as is described in this work,

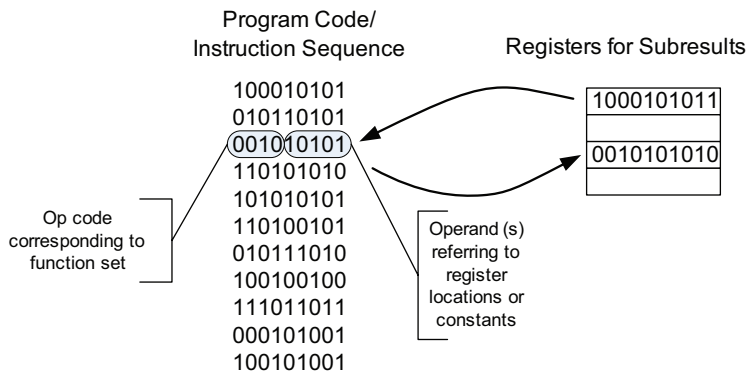


Fig. 11.1. Linear Genetic Programming (LGP) genotype

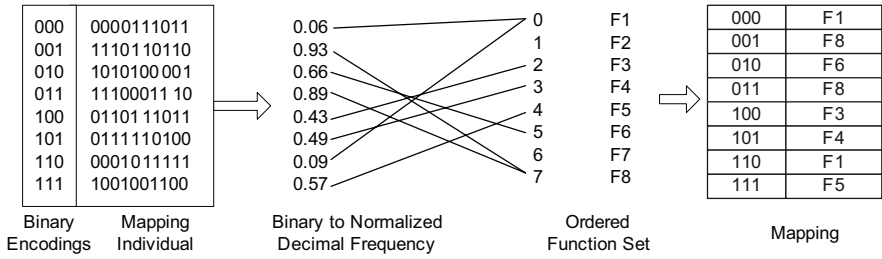


Fig. 11.2. PAM DGP mapping process

is evolution of individuals and a shared “genetic code” that maps codons to amino acids in actual biological organisms.

The developmental GP variant used in this work for stock analysis is Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP), introduced in [5]. Genotypes in PAM DGP are binary strings, with interpretation of sections of the binary string being instruction-dependent (see next Section 11.4). Mappings in this work are redundant such that individuals are composed of  $b \geq s$  10-bit binary strings, where  $b$  is the minimum number of binary sequences required to represent a function set of  $s$  symbols. Each 10 bit mapping section is interpreted as its decimal equivalent, normalized to the range  $[0, 1]$ , and mapped to an ordered function set index by multiplying by  $s - 1$  and truncating to an integer value to yield an index value from 0 to  $s - 1$  (allowing redundant encoding of symbols). For example, in the mapping expressed in Figure 11.2 the first 10-bit binary sequence of the mapping individual “0000111011” corresponds to an encoding for the genotype binary sequence “000.” By dividing the decimal equivalent of the mapping individual’s 10-bit sequence (59) by the maximum value (“0000000000”, 1023 in decimal), a normalized value of 0.06 results. Truncating 0.06 to the integer value 0 means that the genotype binary sequence (op code) corresponds to the member of the ordered function set at index position 0, which is function (F1). Thus, genotype binary sequence “000” corresponds to the function set member F1.

Using this mapping mechanism with co-evolutionary selection, PAM DGP will emphasize the most useful members of the function set, ignore members of the function set which are not pertinent, and simultaneously evolve an appropriate genotype solution. PAM DGP is compared to the standard LGP implementation as described by Brameier and Banzhaf in [2]. LGP individuals are also bit strings, and there is naturally only a genotype population. The interpretation of instructions in PAM DGP can be considered the same for LGP, only LGP uses a static mapping and constant function set. Thus, PAM DGP extends LGP such that members of a function set are adaptively emphasized.

In PAM DGP there is a population of genotypes that cooperatively coevolves with a separate population of mappings. A probability table is updated throughout algorithm execution with entries corresponding to each pair of individual genotype and mapping from both populations. The table entries represent frequencies that dictate the probability that roulette selection in a steady state tournament will choose the genotype-phenotype pairing of individuals determined by the indices of the table. The genotype

and mapping individual that are members of the current best genotype-mapping pairing are immune to mutation and crossover to maintain the current best solution discovered. Each tournament round involves the selection of four unique genotype-mapping pairings. Following fitness evaluation and ranking, the probability table columns associated with the winning combinations have the winning combination in that column updated using Equation 11.1 and the remaining combinations in that column updated using Equation 11.2

$$P(g,m)_{new} = P(g,m)_{old} + \alpha(1 - P(g,m)_{old}) \tag{11.1}$$

$$P(g,m)_{new} = P(g,m)_{old} - \alpha(P(g,m)_{old}) \tag{11.2}$$

where  $g$  is the genotype individual / index,  $m$  is the mapping individual / index,  $\alpha$  is the learning rate (corresponding to how much emphasis is placed on current values versus previous search), and  $P(g, m)$  is the probability in table element  $[g, m]$ . To prevent premature convergence, the algorithm uses a noise threshold. If an element in the table exceeds the noise threshold following a tournament round, a standard Gaussian probability in the interval  $[0, 1]$  is placed in that element and all values in its column are re-normalized so the column elements sum to unity. The PAM DGP algorithm and selection mechanism are summarized in Figure 11.3. Additional details of PAM DGP and its improvements over other, related systems are described in [5].

Each steady state tournament consists of 1000 rounds (4 individuals per round). PAM DGP uses a genotype population of size 10 (as does LGP) and mapping population of size 10. Each genotype consists of 320 bits and 4 subresult registers, and each mapping consists of 160 bits (10 bits for each of 16 required encodings for a function set of size 16). XOR mutation on a (uniform) randomly chosen instruction was used on genotypes, with low threshold point mutation used on mappings to provide a more stable context against which the genotype could evolve. The genotype population used a mutation rate of 0.5 and a crossover rate of 0.9. The mapping population uses a lower crossover and mutation rate, both set at 0.1. PAM DGP used a conservative learning rate of 0.1 and noise threshold of 0.95 to prevent premature convergence.

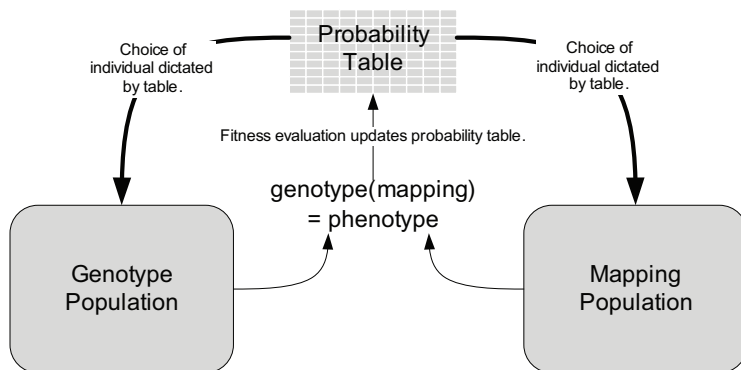


Fig. 11.3. Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP)



### 11.3.2 Stock Analysis with PAM DGP and LGP Algorithms

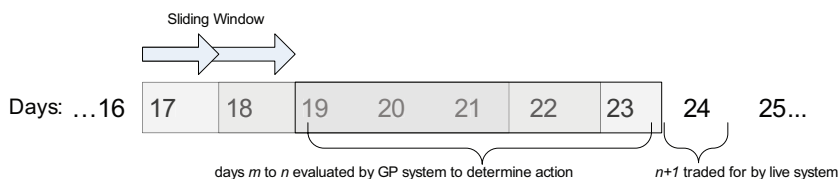
The PAM DGP and LGP implementations are applied to three stocks in the technology sector: Google Inc., ticker symbol “NASDAQ:GOOG,” Apple Inc., ticker symbol “NASDAQ:AAP,” and Microsoft Corporation, ticker symbol “NASDAQ:MSFT.” The initial exchange portion of the ticker symbols will be removed for brevity in the remainder of the paper. High, low, open, and close data was provided as input for the 200 day period in 2007 for interday data analysis, which we use here to explain the operation of the algorithm. (Particular differences in implementation for intraday data are discussed in Section 11.5) The first 16 days of the 200 day intraday data set were reserved as a basis on which to draw technical indicator data. After the first 16 days, the GP fitness was evaluated on data corresponding to a moving window of 5 days. Individuals represent sets of trading rules, based on functions in the function set (to be described). For each window of 5 days corresponding to trading days  $m$  to  $n$ , each of  $m$  to  $n - 1$  days were used for calculation of a trading decision given the individual’s rule set, with  $m + 1$  to  $n$  being used to evaluate the recommendation based on the immediately preceding day. Days used for the calculation of a trading decision were normalized using two-phase preprocessing similar to [11]: All daily values were transformed by division by a lagged moving average, and then normalized using linear scaling into the range  $[0, 1]$  using

$$v_{scaled} = \frac{v_t - l_n}{h_n - l_n} \quad (11.3)$$

where  $v_{scaled}$  is the normalized trading value,  $v_t$  is the transformed trading value at time step  $t$ ,  $h_n$  is highest transformed value in the last  $n$  time steps,  $l_n$  is the lowest transformed value in the last  $n$  time steps, and  $n$  is length of the time lag chosen for the initial transformation.

In addition to an instruction set, each individual consists of a set of four registers, a flag for storing the current value of logical operations, and a separate output (trade) register for storing a final value corresponding to a trade recommendation. Following the execution of the trading rules of a GP individual, if the value of the trade register is 0, no action is recommended. Otherwise, the final value in the trade register corresponds to a value in the range  $[0, 1]$ . This value was multiplied by a maximum dollar amount to be bought or sold per trade (\$10,000 was used here based on an initial account balance of \$100,000 with which to trade) to give some portion of \$10,000 to be traded. For each trade conducted, there is a \$10 commission penalty. The trading system is permitted to run a small deficit  $\geq$  \$10 to either handle a sell recommendation when maximally invested (where the deficit would be immediately recouped) or, similarly, to allow a buy in order to be maximally invested. Fitness of an individual is the value of the cash and shares held.

The best individual consisting of the best trading rule set is used by a “live” trading algorithm. That is, the live trader provides known information to the GP for days  $m$  to  $n$ . The GP algorithm returns a recommendation on which the live trading system bases its decision to trade on the following day,  $n + 1$ . In particular, the net number of shares bought and sold by the best evolved individual (trading rules) given the recommendation of the trade register over all the fitness cases (4 cases given a 5 day window) is the buy or sell recommendation to the “live” trading system. The best GP individual can thus



**Fig. 11.4.** Relationship between “live” trading system and GP tournament execution

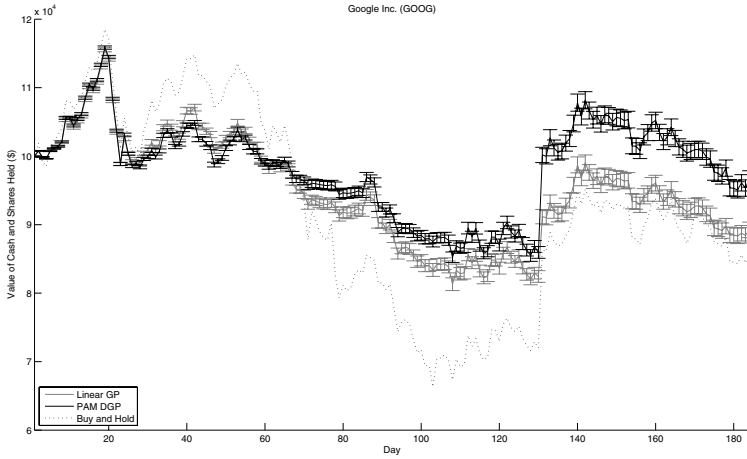
recommend up to \$40,000 worth of share selling or buying per actual trading day to the live system. With the next window shift, the current cash and shares of stock held by the “live” trading system are the new initial amounts for the GP individuals in the next tournament on the new window content. The transactions of the live trading system are what are actually based on unknown data, and determine the success of the algorithms. The process is summarized in Figure 11.4.

While PAM DGP uses co-evolution to refine function set composition, the appropriate initial function set members must be provided as a basis upon which the algorithm can select its optimum function set. In the case of standard GP, this initial function set remains constant throughout execution. The function set includes standard mathematical operators (+, -, \*) and instructions to trade based on logical operators (<, >, =) applied to the four internal registers. In addition, there are four established financial analysis metrics of moving average, momentum, channel breakout, and current day high, low, open, or close price. The financial technical indicator moving average is the mean of the previous  $n$  share prices. The momentum indicator provides the rate of change indicator, and is the ratio of a particular time-lagged price to the current price. Momentum is used to measure the strength of the trend of a stock price, and is often used to predict price peaks [11]. Channel breakout establishes a trading range for a stock, and reflects its volatility. The most popular solution places Bollinger bands around a  $n$ -day moving average of the price at  $\pm 2$  standard deviations of the price movement over the last  $n$  days. A trader is typically alerted when the stock price passes the upper or lower bound of the Bollinger bands.

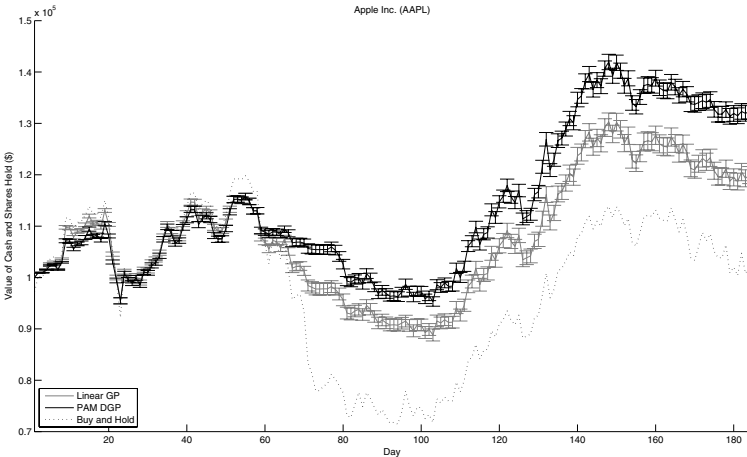
## 11.4 Interday Trading Results

The worth of the assets held by the live trading system for each of 184 days of trading is initially analyzed (200 fitness cases were used overall, with the initial 16 being reserved so initial technical financial indicators had values). Fifty such trials over 184 days of trading were conducted for each of the four stocks using an Apple iMac Intel Core 2 Duo 2.8 GHz CPU and 4GB RAM using OS X Leopard v10.5.4. Starting trading with \$100,000, the mean worth (with standard error) of the live trading system for PAM DGP, LGP, and naïve buy-and-hold strategies is provided in Figures 11.5 to 11.7.

Given Figures 11.5 to 11.7, the prevalent observation is that PAM DGP and LGP are both impressively robust to share price fluctuations (as indicated by the buy-and-hold trend line). The evolved solutions seem to take advantage of the upward trends, although the solutions reflect a conservative strategy overall, adept at anticipating and buffering against sharp share price declines and volatility in general. In the instance

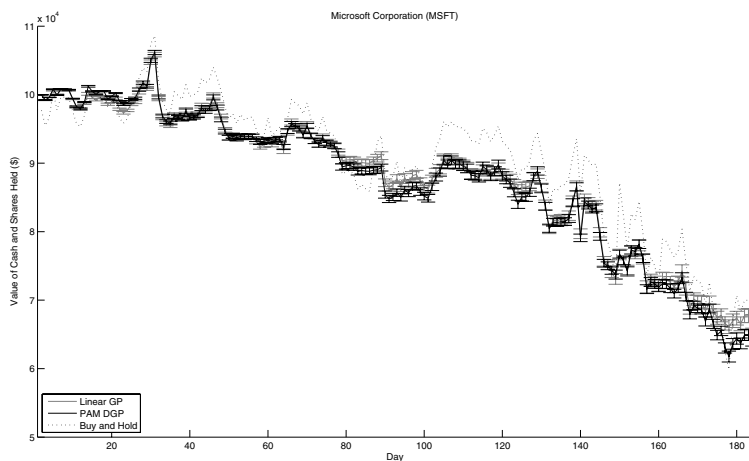


**Fig. 11.5.** Mean total worth for GOOG interday prices (value of cash and shares) of PAM DGP, LGP, and buy-and-hold strategies over 50 trials with standard error given initial \$100,000 cash value

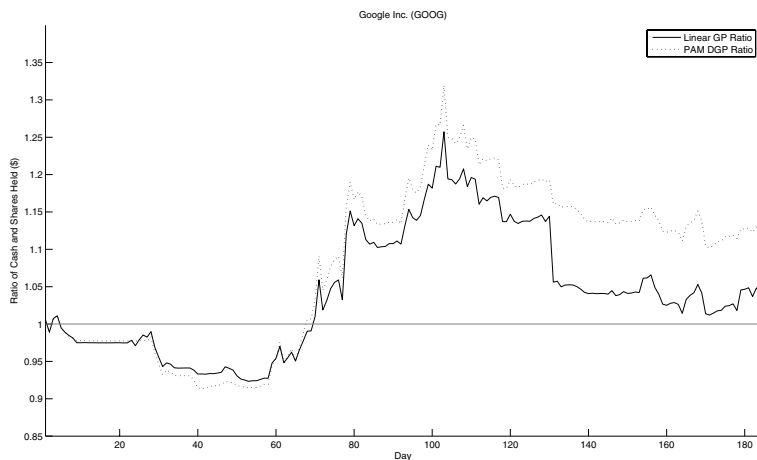


**Fig. 11.6.** Mean total worth for AAPL interday prices (value of cash and shares) of PAM DGP, LGP, and buy-and-hold strategies over 50 trials with standard error given initial \$100,000 cash value

of MSFT, there are no declines sharp enough or for long enough duration to cause significant withdrawal of investments. The MSFT example shows that a moderately volatile, gradual downward trend will have the algorithm gradually lose money as there is no consistent period of gain or loss. However, in this degenerate case buy-and-hold results in a similar loss of money—that is, one does not lose much even given such a deceptive scenario. Both algorithms achieve final profits better than buy-and-hold for



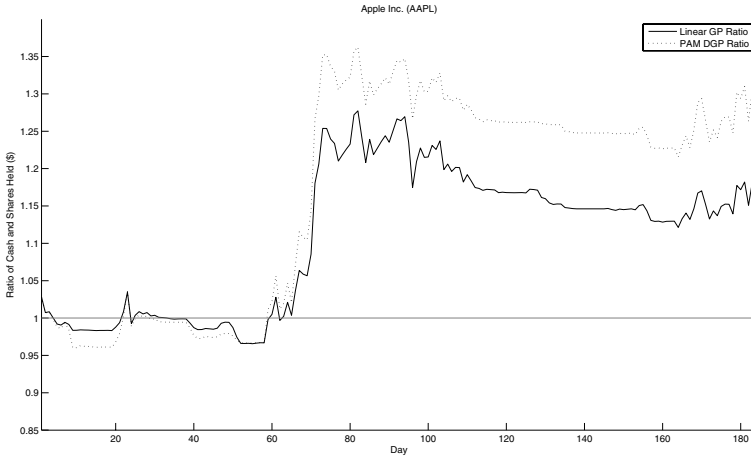
**Fig. 11.7.** Mean total worth for MSFT interday prices (value of cash and shares) of PAM DGP, LGP, and buy-and-hold strategies over 50 trials with standard error given initial \$100,000 cash value



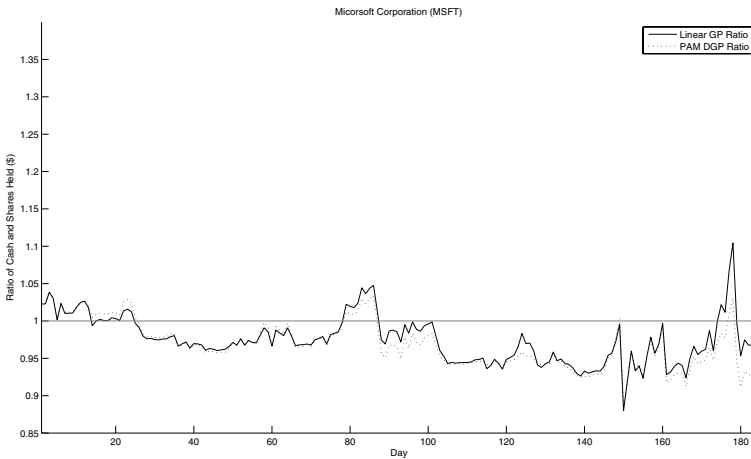
**Fig. 11.8.** Mean ratio of PAM DGP and LGP live trading system total worth to buy-and-hold over 50 trials for GOOG interday prices. Values greater than 1 indicate greater GP worth than buy-and-hold, values less than 1 *vice versa*

the remaining two stocks (GOOG and AAPL). Figures 11.8 to 11.10 provide a ratio of PAM DGP and LGP to buy-and-hold total worth for a finer comparison, with profit (final and cumulative measures) shown in Figure 11.11.

Comparing the ratio of PAM DGP and LGP worth across stocks in Figures 11.5 to 11.8, PAM DGP maintains higher worth than LGP for the large majority of trading days in the instances of GOOG and AAPL, with LGP dominating PAM DGP almost

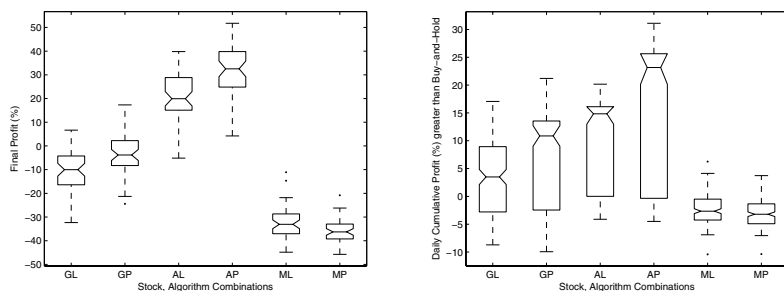


**Fig. 11.9.** Mean ratio of PAM DGP and LGP live trading system total worth to buy-and-hold over 50 trials for AAPL interday prices. Values greater than 1 indicate greater GP worth than buy-and-hold, values less than 1 *vice versa*

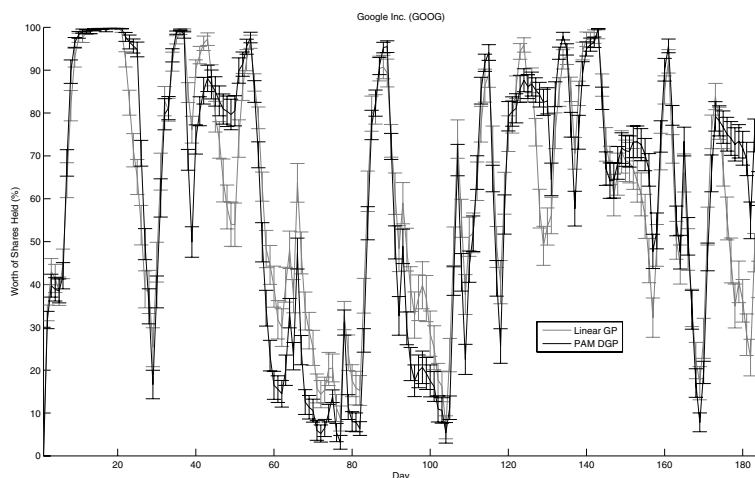


**Fig. 11.10.** Mean ratio of PAM DGP and LGP live trading system total worth to buy-and-hold over 50 trials for MSFT interday prices. Values greater than 1 indicate greater GP worth than buy-and-hold, values less than 1 *vice versa*

the entire period for MSFT (but not by a significant margin). PAM DGP outperforms buy-and-hold by almost 35% at times for GOOG and AAPL, with LGP outperforming buy-and-hold by over 25% at times on those stocks. LGP and PAM DGP perform very closely throughout MSFT (Figure 11.10). Both PAM DGP and LGP outperform buy-and-hold for the majority of the time period for GOOG and AAPL. When buy-and-hold outperforms the GP algorithms throughout MSFT it is by a lower margin (typically



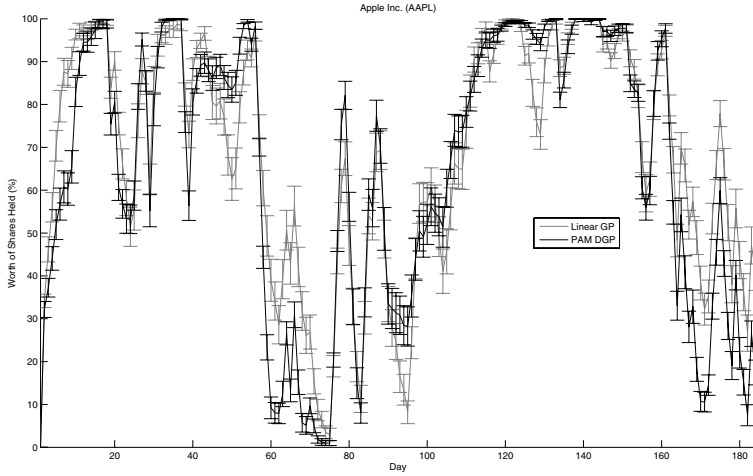
**Fig. 11.11.** Boxplot of mean final profit (%) and mean daily cumulative profit (%) greater than buy-and-hold for PAM DGP and LGP over 50 trials for interday prices. First letter of label indicates stock, second letter indicates algorithm. Value of 0 indicates the break even point



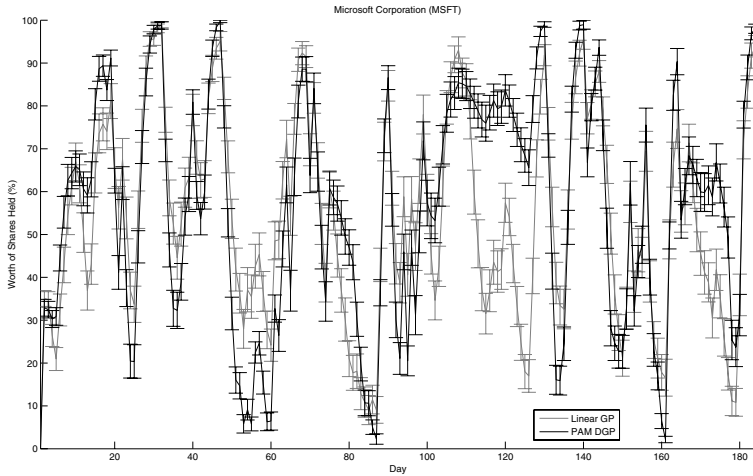
**Fig. 11.12.** Mean shares held by PAM DGP (black) and LGP (grey) live trading systems for GOOG interday prices as a percentage of total worth over 50 trials with standard error

5% or less). Comparing Figures [11.5](#) to [11.7](#) and [11.8](#) to [11.10](#), respectively, it is evident that PAM DGP provides increased robustness to market downturns and quickly capitalizes growth opportunities later in evolution.

In the boxplots of Figure [11.11](#), each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times the interquartile range. PAM DGP outperforms LGP at the end of the time period (Figure [11.11](#), left) for GOOG and AAPL, with no statistically significant difference in final profits for MSFT (all at the 95% confidence interval). Figure [11.11](#) (left) shows impressive final profit for AAPL, where the algorithm took advantage of market gains and losses (Figure [11.6](#)). There was a general loss for both PAM DGP and

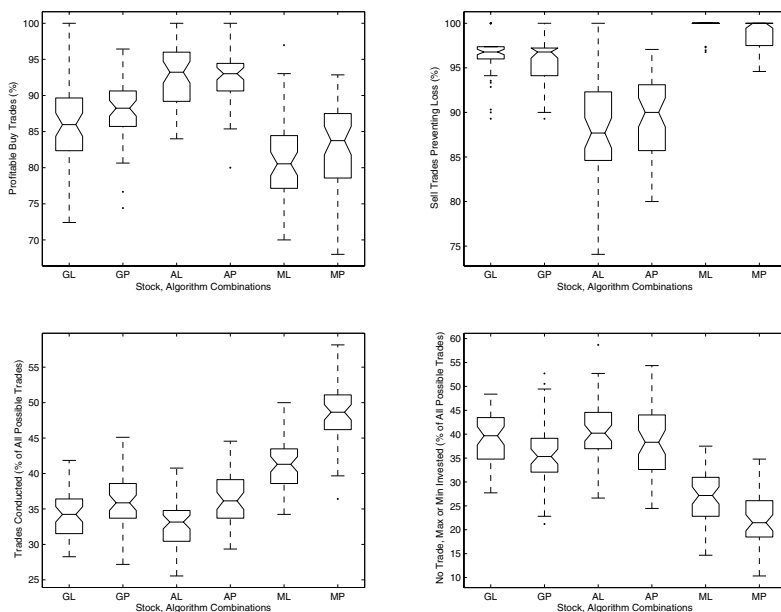


**Fig. 11.13.** Mean shares held by PAM DGP (black) and LGP (grey) live trading systems for AAPL interday prices as a percentage of total worth over 50 trials with standard error



**Fig. 11.14.** Mean shares held by PAM DGP (black) and LGP (grey) live trading systems for MSFT interday prices as a percentage of total worth over 50 trials with standard error

LGP considering final profit for GOOG. GOOG incurred losses during most of the time period and was thus not profitable overall. Note that time period end is arbitrary and profits are a direct reflection of underlying market trend. Figure 11.11 (right) shows the mean daily cumulative profit (%) greater than buy-and-hold for the LGP and PAM DGP live trading systems over all trading days. Figure 11.11 (right) indicates that both PAM DGP and LGP were generally more profitable than buy-and-hold at any given time for GOOG and AAPL, but not for the degenerate case of MSFT where all GP algorithms



**Fig. 11.15.** Percentage of profitable buy trades, sell trades preventing losses, percentage of trades executed overall for each stock, and percentage of trades not conducted while maximally or minimally invested for each algorithm combination over 50 trials for interday prices. First letter of label indicates stock, second letter indicates algorithm

and naïve buy-and-sell incur similar losses. PAM DGP was more profitable than LGP at any given time by a large margin for GOOG and AAPL, and there was no statistically significant difference (at the 95% confidence interval) between LGP and PAM DGP for MSFT. Number of shares retained daily as a percentage of live trading total worth is shown in Figures 11.12 to 11.14.

Comparing Figures 11.5 to 11.7 and 11.12 to 11.14, it is evident that both PAM DGP and LGP are impressively reactive in that they will sell stock if a market downturn starts and buy when the market appears to be experiencing gains. Figures 11.5 to 11.6 and 11.12 to 11.13 also indicate that both algorithms are effective at staying maximally invested during profitable periods. In the instance of MSFT (Figure 11.7), however, where frequent gains and losses of non-substantial amounts occur, Figure 11.14 indicates that the GP never fully invests or is out of the market. Indeed, given the trend of Figure 11.7, it is not prudent to fully invest or sell all shares at any particular point. Proportion of profitable trades is a common metric for evaluation of trading activity, although it is deceptive: it does not even reflect the overall ability of an algorithm in terms of actual profit generated [1]. Many trades, although not profitable, are beneficial in preventing loss during market downturns. Thus, rather than percentage of profitable trades, the percentage of profitable buy trades and percentage of sell trades preventing loss for each algorithm are shown in the top left and right boxplot of Figure 11.15, respectively. A profitable buy is defined as a buy where the total value of shares and cash held at a



time prior to the next sell exceeds the total value at the time of purchase (less transaction cost). Similarly, a sell preventing further losses is defined as a sell where the total value of shares and cash held at a time prior to the next buy is less than the total value at the time of sale (less transaction cost). The percentage of trading opportunities where action was taken is shown in Figure 11.15 (bottom left). Out of all possible trades, the number of trades not conducted when the system was maximally or minimally invested is shown in Figure 11.15 (bottom right).

Figure 11.15 reveals that both algorithms are extremely accurate at buying to gain profit, with medians of approximately 80% - 95% profitable buys. There is no statistical difference (at the 95% confidence interval) in the ability of PAM DGP or LGP to buy for profit for any of the stocks examined. In terms of protecting investment through selling to prevent loss, the median for PAM DGP and LGP was very good (typically 95% to 100%) for GOOG and AAPL. For MSFT, the percentage of sells preventing loss was lower (87 to 90%), where the trend in price was volatile with few sudden, extreme changes (Figure 11.7). LGP had better, or no statistical difference in, performance compared to PAM DGP when selling to prevent losses. Any outliers in either buying for profit or selling to prevent loss were acceptably high percentages. These beneficial transactions are also the result of trading levels with medians of 30% to 40% of possible trades for GOOG and AAPL, with higher medians for MSFT (more trades were conducted due to the volatility of MSFT). PAM DGP generally conducted more trades (based on spread of data) than LGP for all stocks. Figure 11.15 (right, bottom) indicates medians of approximately 35-40% of trades where the system wished to maintain a maximally or minimally invested position for GOOG and AAPL. Compared with Figures 11.12 and 11.13 it is evident that most of these positions were maximal investment to generate profit. Neither algorithm maximally or minimally invested for a high percentage of trades for MSFT in Figure 11.15 (right, bottom), also seen previously in Figure 11.14 due to the volatility of MSFT. Overall, Figure 11.15 indicates that the percentage of beneficial trades that were made to generate profit or protect from losses were impressively high, where this occurred in the context of moderate levels of trading.

## 11.5 Intraday Trading Results

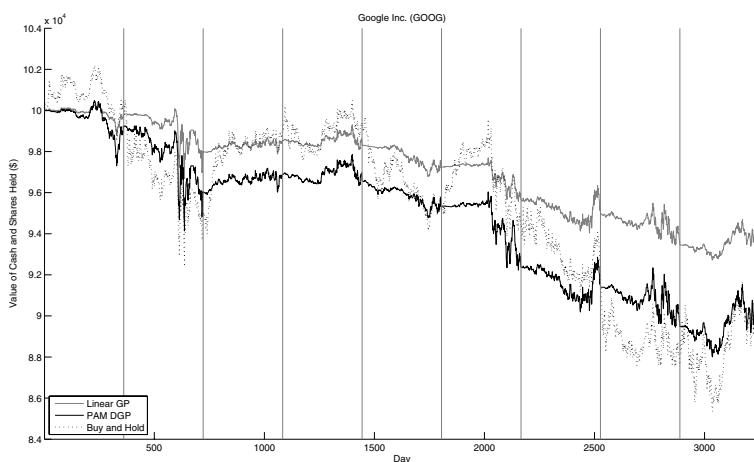
Both PAM DGP and LGP systems were applied to intraday data on the same three technology stocks (GOOG, AAPL, and MSFT) in a different time period. The worth of the assets held by the live trading system for each of 9 consecutive days of trading is analyzed. For each of nine days, 370 stock ticks from the beginning of the trading day are used where the value of the stock is requested from a server every minute (giving 3330 ticks total). As in [1], we opted not to trade during the final minutes of the trading day due to extreme values (daily high/low points) and high volatility in early and late trading. The last 20 minutes of trading each day are simply not used, as there are 390 minutes of trading on the NASDAQ (where all three stock prices are traded) exchange per day. To avoid early trading but not lose too much trading time, the first 8 minutes of trading are used to generate preprocessed values and then seed technical indicators. It is also worth noting that the data was taken from a period during which the markets were not doing well, namely September to October 2008, so the challenge was to trade for

**Table 11.1.** Intraday stock trading final profit (%)

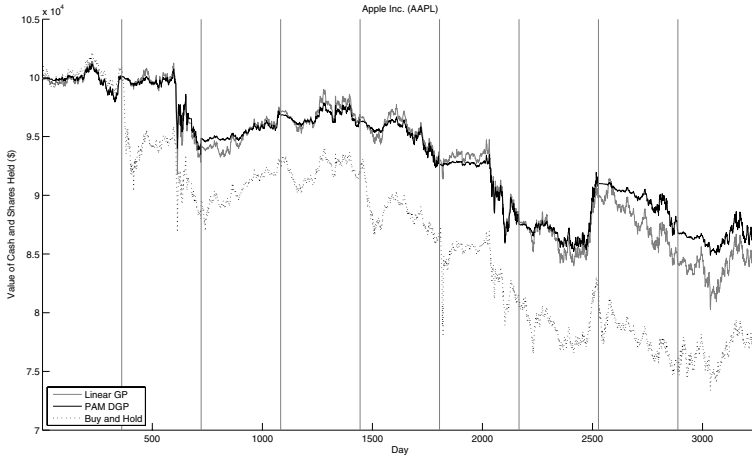
	GOOG	AAPL	MSFT
PAM DGP	-11.52	-14.76	-7.84
LGP	7.07	-16.64	-3.97

profit on generally declining stocks. After each trading day, all shares were cashed out; thus no overnight positions were held (a common practice for many intraday traders to avoid the pitfalls of volatility in late trading).

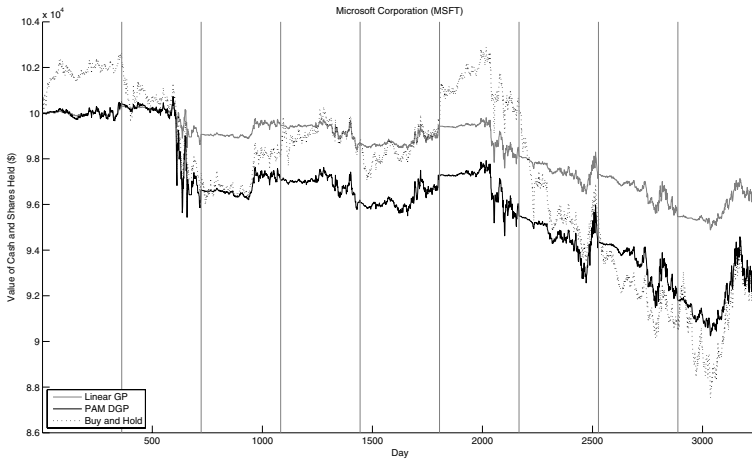
At the start of the next trading day, the cash value left after the previous trading day was used to invest. The nature of intraday data trends compared to interday trends also necessitated some changes to the parametrization of the algorithm, since the intraday data was much more volatile. Thus, a smaller window for technical indicators was adopted (5 ticks), but the size of the moving window used by the GP was kept the same (also 5 ticks). Since there were considerably more opportunities to trade during each day, with increased volatility of prices, the maximum trade allowed was reduced to \$1000. An Apple iMac Intel Core 2 Duo 2.8 GHz CPU and 4GB RAM using OS X Leopard v10.5.4 was again used to conduct intraday trading, as for interday data. In real time, the analysis took approximately 4 hours to analyze 3330 ticks for one stock, giving a speed of 4.32 seconds per tick. Starting trading with \$100,000, the mean worth (with standard error) of the live trading system for PAM DGP, LGP, and naive buy-and-hold strategies is given in Figures 11.16 to 11.18. Ratio of PAM DGP and LGP performance to buy-and-hold is shown in Figures 11.19 to 11.21. Final profit is shown in Table 11.1 with cumulative profit (%) greater than buy-and-hold in Figure 11.22.



**Fig. 11.16.** Mean total worth for GOOG intraday prices (value of cash and shares) of PAM DGP, LGP, and buy-and-hold strategies given initial \$100,000 cash value. Vertical lines separate trading days

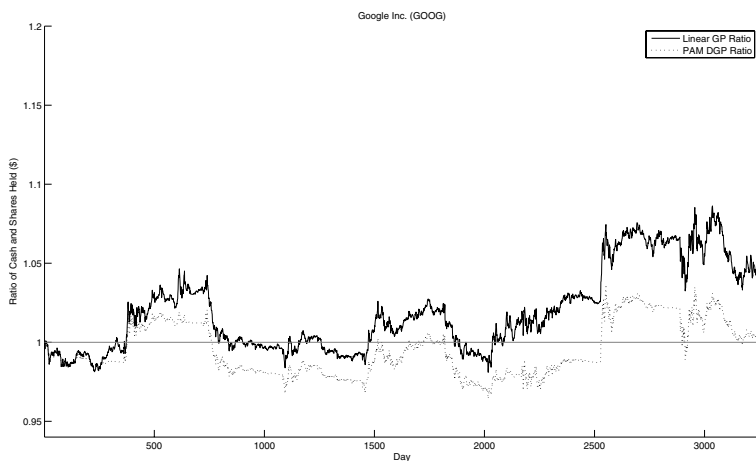


**Fig. 11.17.** Mean total worth for AAPL intraday prices (value of cash and shares) of PAM DGP, LGP, and buy-and-hold strategies given initial \$100,000 cash value. Vertical lines separate trading days

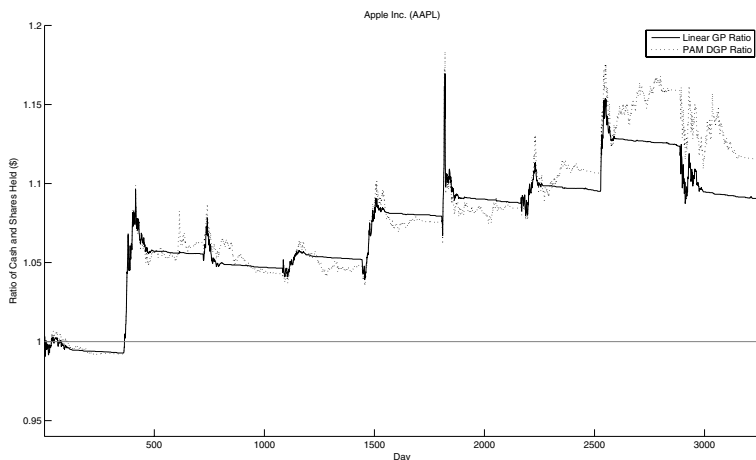


**Fig. 11.18.** Mean total worth for MSFT intraday prices (value of cash and shares) of PAM DGP, LGP, and buy-and-hold strategies given initial \$100,000 cash value. Vertical lines separate trading days

Figures 11.16 to 11.18 show Linear GP outperforming PAM DGP and buy-and-hold for most portions of the time period for GOOG (Figure 11.16) and MSFT (Figure 11.18). Moreover, for both GOOG and MSFT, LGP outperforms buy-and-hold for much of the time period, whereas PAM DGP does not. LGP also seems to be better able to sell to prevent loss during market downturns for GOOG and MSFT (Figure 11.16 and 11.18). For AAPL (Figure 11.17), however, the performance of LGP and PAM DGP are closely

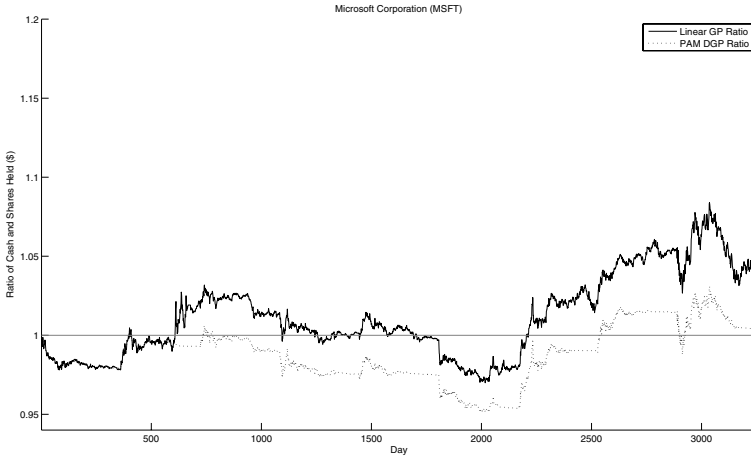


**Fig. 11.19.** Mean ratio of PAM DGP and LGP live trading system total worth to buy-and-hold for GOOG intraday prices. Values greater than 1 indicate greater GP worth than buy-and-hold, values less than 1 vice versa

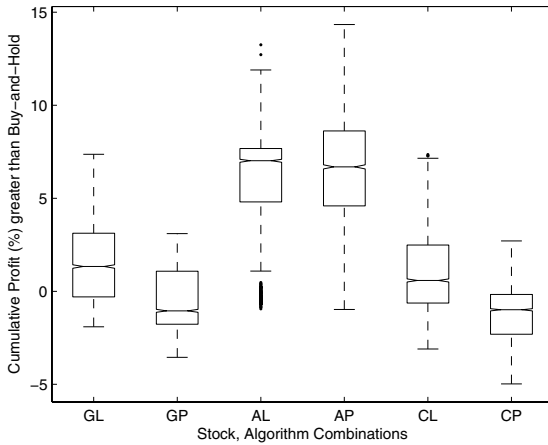


**Fig. 11.20.** Mean ratio of PAM DGP and LGP live trading system total worth to buy-and-hold for AAPL intraday prices. Values greater than 1 indicate greater GP worth than buy-and-hold, values less than 1 vice versa

matched, with PAM DGP outperforming LGP (albeit by a small margin) during a significant portion of the time period. The intraday trend of AAPL differs from GOOG and MSFT in that it presents a downward slope with less severe downturns and climbs, with the exception of start of trading on the second day. The intraday AAPL trend is similar to, but slightly more volatile than, MSFT interday data in Figure 11.7 where LGP and PAM DGP performed closely throughout the time period. Figures 11.19 to 11.21 echo



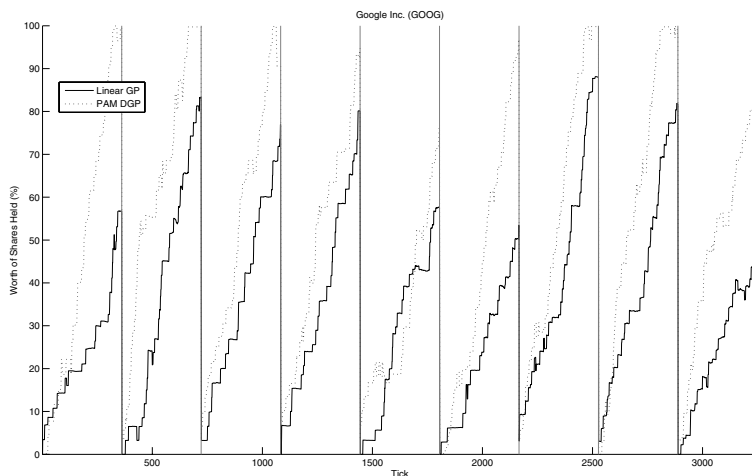
**Fig. 11.21.** Mean ratio of PAM DGP and LGP live trading system total worth to buy-and-hold for MSFT intraday prices. Values greater than 1 indicate greater GP worth than buy-and-hold, values less than 1 vice versa



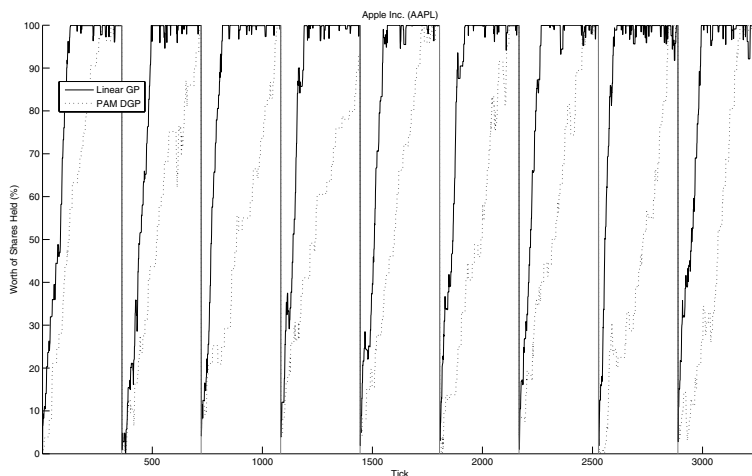
**Fig. 11.22.** Intraday cumulative profit (%) greater than buy-and-hold for PAM DGP and LGP over 50 trials. First letter of label indicates stock, second letter indicates algorithm. Value of 0 indicates the break even point

the observations of Figures 11.16 to 11.18. For GOOG and MSFT (Figures 11.19 and 11.21), LGP outperforms buy-and-hold the majority of the time whereas PAM DGP lags behind simple buy-and-hold. In the case of AAPL, Figure 11.20, LGP and PAM DGP perform closely throughout the data set. However, both more significantly outperform buy-and-hold than was the case for either GOOG or MSFT.

Table 11.1 reflects that trading occurred during a difficult time for the markets. That is, no final profit was made due to the general downward trend of all stocks. Greater

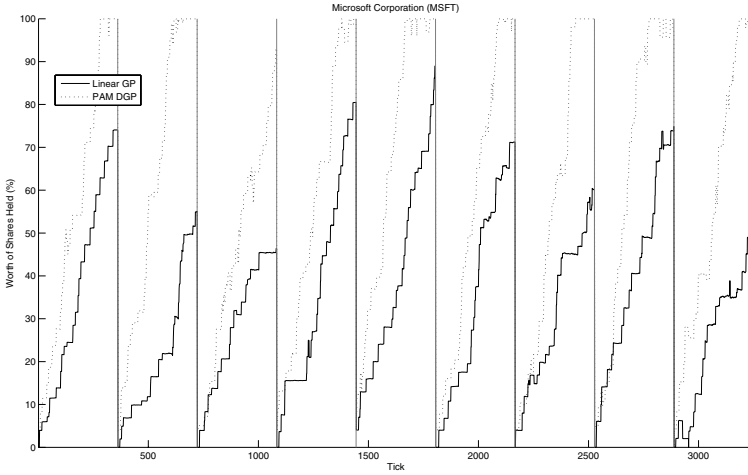


**Fig. 11.23.** Mean shares held by PAM DGP (black) and LGP (grey) live trading systems for GOOG intraday prices as a percentage of total worth



**Fig. 11.24.** Mean shares held by PAM DGP (black) and LGP (grey) live trading systems for AAPL intraday prices as a percentage of total worth

losses were incurred by PAM DGP than for LGP for GOOG and MSFT, but AAPL incurred greater (but similar) losses using LGP as when using PAM DGP. Considering cumulative profit over buy-and-hold (Figure 11.22) for all intraday data, LGP outperformed buy-and-hold for every stock. LGP outperformed PAM DGP for GOOG and MSFT, but performed on par with PAM DGP for AAPL (as seen also in Figures 11.16 to 11.18). The shares held by the algorithm as a percentage of total worth is shown in Figures 11.23 to 11.25.



**Fig. 11.25.** Mean shares held by PAM DGP (black) and LGP (grey) live trading systems for MSFT intraday prices as a percentage of total worth

Figures 11.23 and 11.25 show that for GOOG and MSFT, respectively, on any given trading day LGP will not maximally invest. Note that at the end of each trading day, the shares are all sold so no overnight positions are held: the abrupt selling of all shares is not a result of GP solutions. However, PAM DGP will maximally invest. Given the overall performance of both algorithms compared to buy-and-hold (see Figure 11.22), the LGP tendency not to maximally invest yields better performance. In contrast, for AAPL in Figure 11.24, LGP shows a tendency to maximally invest prior to PAM DGP taking a similar market position. In this instance, though, neither strategy significantly

**Table 11.2.** Profitable buy trades (% of all buys), protective sell trades that prevented loss (% of all sells), trades conducted (% of all possible trades), and no trade with maximum or minimum invested (% of all possible trades)

		GOOG	AAPL	MSFT
Profitable Buys(%)	PAM DGP	95.64	96.63	98.10
	LGP	97.12	97.76	96.19
Protective Sells(%)	PAM DGP	100.00	98.46	98.36
	LGP	100.00	100.00	100.00
Conducted Trades(%)	PAM DGP	12.22	15.88	11.60
	LGP	8.09	20.62	7.02
Max/Min No Trade(%)	PAM DGP	7.02	8.77	10.40
	LGP	1.42	51.77	1.79

outperforms the other (see Figure 11.22). Further analysis of the success of the trades is provided in Table 2 including: profitable buy trades, protective sell trades, trades conducted, and number of trades not made while maximally or minimally invested. Profitable buys and protective sells were determined in the same way as interday data (see previous Section).

Table 11.2 shows that both algorithms were very successful at choosing trades that led to profitable buys and sells to prevent further losses across every stock. In this respect, the interday and intraday performance of the algorithms was very similar. However, compared to interday data (Figure 11.15), the algorithms traded much less frequently when operating on intraday data. As there were many more opportunities to trade, but the price per trade was kept constant, the algorithms naturally had to be more selective regarding the trades to be conducted. Regarding the number of trades not conducted to allow maximum or minimum investment, it is clearly evident that PAM DGP had a much greater tendency than LGP to stay entirely in or out of the market for GOOG and MSFT, with the converse being true for AAPL.

## 11.6 Conclusions and Future Work

This work examined the trading performance of a co-evolutionary developmental GP model (PAM DGP) using a genotype-phenotype mapping and a more traditional LGP on four stocks. For interday data, PAM DGP was found to better adapt to guard investments during market downturns and readily take advantage of market gains than LGP. However, for the more volatile intraday data where trading had to be more selective to attempt to be profitable, LGP was found to perform better by not investing as reactively as PAM DGP. For both interday and intraday data, the algorithm was found to not perform optimally in the degenerate case of a moderately volatile, gradual downward slope. However, even in such a deceptive scenario, the GP algorithms performed comparably with buy-and-hold. In both interday and intraday data sets, both algorithms exhibited impressive accuracy in choosing beneficial trades, both for profitable buys and selling to protect investments. The accurate trading ability occurred with moderate levels of trading for interday and lower levels for intraday (indicating more selective trading). Future work will examine options for risk adjusted fitness and portfolio management.

## References

1. Brabazon, A., O'Neill, M.: *Biologically Inspired Algorithms for Financial Modelling*. Springer, Berlin (2006)
2. Brameier, M., Banzhaf, W.: *Linear Genetic Programming*. Springer, New York (2007)
3. Drezewski, R., Sepielak, J.: *Evolutionary System for Generating Investment Strategies, Applications of Evolutionary Computing*. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) *EvoWorkshops 2008*. LNCS, vol. 4974, pp. 83–92. Springer, Heidelberg (2008)
4. Grosnan, C., Abraham, A.: *Stock Market Modeling Using Genetic Programming Ensembles*. *Studies in Computational Intelligence* 13, 131–146 (2006)



5. Wilson, G., Heywood, M.: Introducing Probabilistic Adaptive Developmental Genetic Programming with Redundant Mappings. *Genetic Programming and Evolvable Machines* 8, 187–220 (2007)
6. Wilson, G., Banzhaf, W.: Prediction of Interday Stock Prices Using Developmental and Linear Genetic Programming, Applications of Evolutionary Computing. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (eds.) *EvoWorkshops 2009*. LNCS, vol. 5484, pp. 172–181. Springer, Heidelberg (2009)
7. Yan, W., Sewell, M., Clack, C.: Learning to Optimize Profits Beats Predicting Returns—Comparing Techniques for Financial Portfolio Optimisation. In: *Proceedings of the 2008 Genetic and Evolutionary Computation Conference (GECCO 2008)*, pp. 1681–1688. ACM Press, New York (2008)

---

# Index

- adaptive strategy, [50](#)
- agent-based artificial stock market, [45](#), [52](#)
- agent-based co-evolutionary algorithms, [63](#)
- agent-based co-evolutionary systems, [64](#), [66](#)
- agent-based co-operative co-evolutionary system, [67](#)
- agent-based computational economics, [52](#)
- agent-based modeling, [47](#), [52](#), [63](#)
- algorithmic trading, [45](#)
- antecedent clause, [149](#)
- arbitrage pricing theory, [9](#)
- ARIMA model, [95](#)
- artificial market model, [86](#)
- artificial stock market, [47](#)
- ask side, [46](#)
- asymmetric information, [89](#)
- aversion coefficient, [115](#)
  
- beta values, [10](#)
- bid side, [46](#)
- bid-ask spread, [47](#)
- breakdown value, [10](#)
- breeder genetic algorithm, [155](#)
  
- calculus of fuzzy rules, [150](#)
- calendar based rebalancing, [113](#)
- calendar based strategy, [109](#)
- Capital Asset Pricing Model, [9](#), [85](#)
- centroid method, [151](#)
- co-evolution, [63](#)
- co-evolutionary multi-agent system (CoEMAS), [64](#)
- co-evolutionary algorithms, [63](#)
- co-evolve, [194](#)
  
- co-operative co-evolutionary multi-agent-based algorithm, [68](#)
- cognitive psychology, [173](#)
- cointegration based index tracking, [110](#)
- consequent clause, [149](#)
- coupled Markov chain, [31](#)
- coupled Markov chain model, [32](#)
- credit risk, [31](#)
  
- data mining, [149](#)
- decision support system, [133](#)
- default probabilities, [31](#)
- defuzzification, [151](#)
- determination puzzle of foreign exchange, [169](#)
- developmental co-evolutionary genetic programming, [191](#)
- developmental genetic programming, [193](#)
- differential evolution, [9](#), [14](#), [19](#), [109](#), [115](#), [116](#)
- DTLZ test functions, [74](#)
- dual tree structure, [174](#)
  
- economic utility theory, [173](#)
- efficient market hypothesis, [169](#)
- electronic order book, [46](#), [48](#)
- emergent properties, [52](#)
- entry strategies, [175](#)
- evolutionary algorithm, [37](#), [151](#)
- evolutionary algorithms, [63](#), [131](#)
- evolutionary artificial neural networks, [152](#)
- evolutionary decision support system, [131](#), [134](#)
- execution style, [46](#)
- exit strategies, [175](#)
- extreme observations, [9](#)

- foreign exchange market, [169](#)
- fund tracking problem, [111](#)
- fuzzy logic, [147](#)
- fuzzy rule-based systems, [149](#)
- fuzzy rules, [149](#), [150](#)
  
- Gaussian copula model, [32](#)
- genetic algorithm, [51](#), [86](#), [90](#), [170](#)
- genetic programming, [169](#)-[171](#), [192](#)
- grammatical evolution, [131](#), [192](#)
  
- heuristic methods, [13](#)
- high-frequency tick data, [169](#)
- hourglass structures, [154](#)
  
- index fund replication problem, [110](#)
- index fund return replication, [110](#)
- index tracking, [109](#), [110](#)
- interday trading, [192](#)
- inverted market, [85](#)
- island-based distributed EA, [151](#)
  
- knowledge pattern, [136](#)
  
- least median of squares estimator, [11](#)
- least squares, [9](#), [10](#)
- least trimmed squares estimator, [12](#)
- limit order, [45](#)
- linear genetic programming, [191](#), [193](#)
- linear regression, [9](#)
- linear regression model, [11](#)
- linguistic variables, [150](#)
- lognormal model of stock prices, [95](#)
- loss aversion, [115](#), [173](#)
  
- Mamdani model, [150](#)
- Mamdani system, [150](#)
- market impact, [46](#), [47](#)
- market microstructure, [47](#), [86](#)
- market order, [45](#)
- mature technology, [21](#)
- maximum likelihood function, [33](#)
- mean variance problem, [10](#)
- migration, [152](#)
- minimum variance portfolio, [10](#)
- money management, [169](#), [174](#)
- multi-layer perceptrons, [153](#)
- mutual fund replication, [109](#)
  
- neural networks, [147](#)
  
- neuro-genetic approach, [153](#)
- NSGA2, [71](#)
  
- objective function, [173](#)
- opcodes, [193](#)
- opportunity cost, [46](#), [48](#)
- order statistics, [12](#)
  
- particle swarm algorithm, [35](#)
- particle swarm optimisation, [9](#), [14](#), [31](#)
- performance evaluation of trade execution systems, [50](#)
- portfolio optimisation, [63](#), [76](#)
- portfolio replication, [9](#)
- predicting trend reversals, [147](#)
- price adjustment function, [89](#)
- price formation process, [48](#)
- principal components factor model, [110](#)
- probabilistic adaptive mapping developmental genetic programming, [191](#), [194](#)
  
- quad tree structure, [169](#), [174](#)
  
- rating class, [32](#)
- rating transition, [31](#), [32](#)
- rebalancing strategies, [113](#)
- reference dependence, [173](#)
- resistant estimation, [10](#)
- resistant estimators, [9](#)
- risk management, [9](#)
- robust estimation, [10](#), [12](#)
- robust estimators, [9](#), [10](#)
  
- Santa Fe artificial stock market, [52](#)
- Sharpe Ratio, [9](#)
- Sharpe ratio, [134](#), [173](#)
- simple genetic algorithm, [135](#)
- simulated annealing, [15](#)
- SPEA2, [71](#)
- static strategy, [50](#)
- stock market trading expert, [133](#)
- stock market trading rule, [132](#)
- style analysis, [10](#)
- subset approach, [13](#)
- subsymbolic models, [157](#)
  
- technical analysis, [169](#), [191](#)
- technical indicators, [191](#)
- ternary decision problems, [170](#)
- threshold accepting, [9](#), [15](#)

- tolerance triggered rebalancing, [114](#)
- tolerance triggered strategy, [109](#)
- tournament selection, [171](#)
- tracker portfolio, [111](#)
- tracker rebalancing, [113](#)
- tracking error, [109](#), [111](#)
- tracking error minimisation, [111](#)
- trade execution, [45](#)
- trade execution strategy, [48](#)
- trading cost, [47](#)
- trading experts, [131](#)
- trading rule, [86](#), [87](#)
- trading strategies, [169](#)
- trading system, [173](#)
- trinary Boolean variable, [174](#)
- truncation selection, [156](#)
- turning points, [147](#)
- USD/EUR series, [172](#)
- utility functions, [176](#)
- VWAP, [50](#)
- ZDT test functions, [72](#), [73](#)
- zero-intelligent agents, [52](#)
- zig-zag filter, [158](#)
- zig-zag indicator, [148](#)