공학박사학위논문

Evolutionary Algorithms Based on Effective Search Space Reduction for Financial Optimization Problems

(금융 최적화 문제를 위한
효과적인 탐색공간 축소에 기반을 둔
진화 알고리즘)

2015年 8月

서울대학교 대학원
전기·컴퓨터공학부
이 승 규

Evolutionary Algorithms Based on Effective Search Space Reduction for Financial Optimization Problems

指導教授 문 병 로

이 論文을 工學博士 學位論文으로 提出함.

2015年 4月

서울大學校 大學院
電氣·컴퓨터工學部
李 承 奎

李承奎의 工學博士 學位論文을 認准함.

2015年 6月

委 員 長 이 광 근 印

副委員長 문 병 로 印

委 員 엄 현 상 印

委 員 오 일 석 印

委 員 김 용 혁 印

Abstract

Evolutionary Algorithms Based on Effective Search Space Reduction for Financial Optimization Problems

Seung-Kyu Lee
School of Computer Science & Engineering
The Graduate School
Seoul National University

This thesis presents evolutionary algorithms incorporated with effective search space reduction for financial optimization problems. Typical evolutionary algorithms try to find optimal solutions in the original, or unrestricted search space. However, they can be unsuccessful if the optimal solutions are too complex to be discovered from scratch. This can be relieved by restricting the forms of meaningful solutions or providing the initial population with some promising solutions. To this end, we propose three evolution approaches including modular, grammatical, and seeded evolutions for financial optimization problems. We also adopt local optimizations for fine-tuning the solutions, resulting in hybrid evolutionary algorithms.

First, the thesis proposes a modular evolution. In the modular evolution, the possible forms of solutions are statically restricted to certain combinations of module solutions, which reflect more domain knowledge. To preserve the module solutions, we devise modular genetic operators which work on modular search space. The modular genetic operators and statically defined modules help genetic programming focus on highly promising search space.

Second, the thesis introduces a grammatical evolution. We restrict the possible forms of solutions in genetic programming by a context-free grammar. In the grammatical evolution, genetic programming works on more extended search space than

modular one. Grammatically typed genetic operators are introduced for the grammatical evolution. Compared with the modular evolution, grammatical evolution requires less domain knowledge.

Finally, the thesis presents a seeded evolution. Our seeded evolution provides the initial population with partially optimized solutions. The set of genes for the partial optimization is selected in terms of encoding complexity. The partially optimized solutions help genetic algorithm find more promising solutions efficiently. Since they are not too excessively optimized, genetic algorithm is still able to search better solutions.

Extensive empirical results are provided using three real-world financial optimization problems: attractive technical pattern discovery, extended attractive technical pattern discovery, and large-scale stock selection. They show that our search space reductions are fairly effective for the problems. By combining the search space reductions with systematic evolutionary algorithm frameworks, we show that evolutionary algorithms can be exploited for realistic profitable trading.

Keywords : Financial optimization problem, search space reduction, genetic programming, genetic algorithm, attractive technical pattern, stock selection.

Student Number : 2007-30834

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Financial optimization problems have attracted much attention due to their importance. In fact, they include any optimization problems arising in computational finance. Their common objective is to find optimal models that will consistently work on unseen data; it is quite similar to those of machine learning. Most popular financial optimization problems include optimizations of technical patterns, technical trading, trade execution, stock selection, and so forth.

In general, most financial optimization problems require effective processing and analyzing a huge amount of financial data such as prices, volumes, financial statements, and macroeconomic factors. The enormous dimensionality and non-stationarity of the data, however, prevent us from exploiting conventional exact optimization techniques. Recently, evolutionary algorithms motivated by natural evolution have been extensively exploited for solving such financial optimization problems. Although they are not guaranteed to find the exact optimal solution, it is known that they can find near-optimal solutions in a reasonable time budget. In this thesis, we apply genetic programming and genetic algorithm to financial optimization problems as the representatives of evolutionary algorithms.

According to the type of input data, there are two representative approaches to financial optimizations: *technical and fundamental analyses*. Technical analysis typically uses only price, volume, and open interest which can be obtained with diverse time granularity such as minutely, hourly, daily, and so forth. It implicitly

assumes that the technical data including price, volume, and open interest reflect all necessary information of financial objects such as stock, bond, and so forth. In other words, it needs only technical data which are believed to provide complete information for predicting the price movements. While it is still controversial whether its assumption is correct or not, technical analysis has been popular for building trading or prediction models due to its simplicity. Its common objective is to find profitable technical trading rules including buy and sell signals for short-term trading whose trading horizon is at most one month.

In contrast, fundamental analysis exploits financial statements which are announced quarterly. It tries to evaluate the *intrinsic value* of financial object, irrespective of short-term price movements possibly due to market noise. Owing to its focus on long-term intrinsic value, its common trading horizon is at least three months, or one quarter. One of the most popular methods in fundamental analysis is to calculate *financial ratio*, defined by the ratio between two financial variables. Using the financial ratios, the intrinsic values of stocks can be calculated and they are exploited for trading. For example, one of the simplest methods is to buy undervalued stocks and sell overvalued stock, which corresponds to long-short portfolio. The major limitation of fundamental analysis is the limited applicability due to its strong dependence on financial statements. Fundamental analysis is thus not applicable to markets without financial statements such as futures and foreign exchange markets. Recently, the clear discrimination between the two approaches has been blurred; several studies have exploited both technical and fundamental analyses to build more profitable trading systems. For example, technical analysis can be used to capture better timing for portfolio management based on fundamental analysis.

Most previous evolutionary algorithms for financial optimizations are limited in that they try to find optimal solution in the original search space. In other words, they do not use elaborate search space reductions, which is the major bottleneck for finding promising solutions. Among a number of search space reduction techniques to alleviate such problem, we focus on three techniques particularly tailored to financial optimization problems.

In this thesis, we present modular, grammatical, and seeded evolutions for ef-

fective search space reduction in financial optimization problems. The search space reduction techniques are adapted to reflect some important characteristics of the problems and are combined with local optimizations. They help our evolutionary algorithms focus on more meaningful, or promising search space. The effectiveness of each technique is empirically validated by one of financial optimization problems including attractive technical pattern discovery, its extended problem, and large-scale stock selection.

1.1 Search Methods

In recent years, evolutionary algorithms have been extensively used for financial optimization problems. They are inspired from natural evolution and exploit the encoding similarity among solutions in the population. Crossover, or also called recombination, is then used to generate an offspring by inheriting some features of high-quality parental solutions. In contrast, mutation operator introduces new features to the offspring; it compensates for the lack of introducing new features in recombination operator. The crossover and mutation are the main operators in most evolutionary algorithms such as genetic algorithm and genetic programming. Due to their generality, or less dependence on specific problem domain, most evolutionary algorithms belong to *metaheuristic*. Genetic algorithm and genetic programming are two famous examples of evolutionary algorithms. While the former uses a linear string for representation, the latter a nonlinear tree; the major difference between the two lies only in the representation of solution.

Evolutionary algorithms by themselves are neither effective nor efficient for many financial optimization problems with the large dimensionality and non-stationarity of data. In general, they are not good at fine-tuning around local optima; they are however greatly helped by local search's fine-tuning. Local search requires a neighborhood structure of a given solution and it chooses the best possible solution in the neighborhood. It typically uses a neighborhood designated by minimal or simple changes to a given solution. Choosing an appropriate neighborhood is quite important in that it eventually defines the set of solutions to be probed by a local

search. Evolutionary algorithms incorporated with local search are called *hybrid* ones.

1.2 Search Space Reduction

In a combinatorial optimization problem, a *search space*, or also called *solution space*, is defined by a finite set of feasible solutions and a cost function that maps a solution to a real value. Since many financial optimization problems, except for problems arising in continuous domain, are defined in discrete domain, the definition of search space is still applicable. However, the cardinality of the set of feasible solutions is generally too intractably large; the cost evaluation of all feasible solutions is thus impossible in a reasonable time.

Search space reduction is to restrict the original search space to some promising subspace. In fact, a wide range of techniques including modularization, typing, seeding, feature selection, dimensionality reduction, pruning, and so forth belong to search space reduction. In this thesis, we focus on modular, grammatical, and seeded evolutions which are exploited in evolutionary algorithms.

Modular evolution is to evolve the population based on modules which are encapsulated units preserved under genetic operators. There are a number of techniques for defining, discovering, and reusing the modules in evolutionary algorithms. Genetic programming, originally invented for evolving programs, is one of the most popular metaheuristics exploiting modular evolution. Although a number of different techniques are available for modular evolution, the common requirements for module include high frequency and fitness. In other words, modules should be defined so that they will be frequently reused and expected to be useful in a genetic run.

Grammatical evolution exploits a grammar to restrict the search space to the set of syntactically valid solutions. The grammar is different across problem domains, and it is defined typically by users, or programmers. However, it has an advantage of requiring less domain knowledge than other strict typing techniques such as modular evolution. This is because defining a grammar for valid solutions is generally easier

than identifying and exploiting promising solutions. In other words, typical grammatical evolution is based on valid solutions rather than promising ones. However, it can also be used for describing promising solutions indirectly, possibly using a large number of symbols and productions that reflect more domain knowledge.

The modular and grammatical evolutions are based on the static restrictions on solutions, which are considered to be typing mechanisms. They are used in genetic programming which uses unrestrictive, variable-length trees as its solution encoding. However, they are not directly applicable to genetic algorithms whose encoding is typically restricted to a string of fixed-length. In general, it is common that the solutions in genetic algorithms are already aware of type restrictions by incorporating them in the encoding stage. This motivates us to devise a space reduction technique tailored to genetic algorithms.

Seeded evolution is a candidate approach to search space reduction in genetic algorithms. Typical genetic algorithm uses a randomly generated population as its initial pool of solutions, whose motivation is to provide the initial population with diversity. However, many real-world problems require more improved initial population for more effective evolution. This is mainly due to the time budget for evolution, which limits the number of solutions to be searched. If the initial population is already pre-optimized, it is generally more helpful to the evolution at the cost of little degradation of population diversity and running time.

1.3 Main Contributions

In this thesis, we propose three search space reduction techniques for financial optimization problems. The major contributions of the thesis are listed in the following.

- Three effective search space reduction techniques [LM10, LM15, LMM15]:

  We present three effective search space reduction techniques: modular, grammatically, and seeded evolutions. Modular evolution restricts the search space into the combinations of module solutions which are predetermined by domain knowledge. To preserve the module solutions, modular genetic operators are devised. In this way, we can preserve atomic knowledge, which contained in the

modules, and exploit them to find better solutions. However, the preparation of the set of modules is not always straightforward since it requires too much domain knowledge. It can also overly restrict the possible forms of solutions unless the domain knowledge is carefully selected, organized, and reflected. To find less restricted but still meaningful solutions, we propose a grammatical evolution. We restrict the search space by a context-free grammar where the valid forms of solutions are described. Grammatically typed genetic operators are developed to search only syntactically valid solutions. In general, grammatical evolution requires less domain knowledge than modular one, since it typically designates only correct syntax. As a quite different search space reduction, we introduce a seeded evolution, where relatively simpler part of solution in terms of encoding complexity is pre-optimized by a heuristic. The pre-optimized solutions are provided to the initial population, which helps evolutionary algorithm focus on more meaningful search space.

- Formulations of attractive technical pattern discovery and its extension [LM10, LM15]:

  Previous work on technical trading typically focuses on simultaneous optimization of technical pattern and trading model. It severely increases the complexity of the problem, thereby typically producing profits worse than buy-and-hold. Instead, we approach the problem with a multi-stage optimization which attempts to find attractive technical patterns in the first stage and simulate trading with them in the later stage. We focus primarily on finding attractive technical patterns, which can be exploited by diverse trading models. To this end, we first formulate the problem in terms of profitability, simplicity, and frequency using modular search space. We also extend the problem using grammatically typed search space, which allows more free forms of technical patterns.

- Formulation of large-scale stock selection [LM15]:

  While typical stock selection uses only a small set of stocks with positive financial ratios, we extend the problem to a more general one. Large-scale

stock selection which uses the full universe of stocks and all types of financial ratios is first formulated. Using our formulation, all stocks can be consistently ranked, which enables us to select most profitable stocks without any selection bias.

- Realistic empirical results outperforming buy-and-hold [LM10, LM15, LMM15]:

  Our evolutionary algorithm frameworks are designed to simulate realistic markets as close as possible. This implies that our empirical results can be applied to real trading with some minor modifications. In addition, it also shows that our algorithms can outperform buy-and-hold, which is fairly promising for realistic profitable trading.

1.4 Organization

The thesis is organized as follows. In Chapter 2, we present the preliminaries for later chapters. Chapter 3 introduces a modular evolution with an application to attractive technical pattern discovery. In Chapter 4, we propose a grammatical evolution, where a context-free grammar directs valid solutions. Extended attractive technical pattern discovery is described as an example application. A seeded evolution is presented in Chapter 5. It is applied to large-scale stock selection, which is a generalized form of typical stock selection. Finally, we make our conclusions in Chapter 6.

Chapter 2

Preliminaries

2.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are metaheuristics that mimic the principle of natural evolution. They include genetic programming (GP) [Koz92], genetic algorithm (GA) [Hol75, Gol89], evolution strategies [Rec73], and evolutionary programming [FOW66].

In EA, each solution candidate, or *individual*, is encoded by a representation. It is also called *chromosome* since most EAs use *haploid* representation for the direct encoding of solution. For example, n-ary linear string and tree are the popular representations in GA and GP, respectively. The set of individuals is called *population*. Each individual is assigned a *fitness* which is defined such that a better individual obtains a higher fitness. By a *selection* operator, EA chooses individuals for crossover with a probabilistic bias toward fitter ones. It then recombines the individuals using *crossover* to produce an *offspring*. In general, crossover belongs to *exploitation* in that it searches new solution candidate by recombining the existing features encoded by the parental individuals.

In contrast, *mutation* provides the population with new unexplored features. It blindly mutates some encoded region of the offspring, which helps EA maintain a reasonable amount of genetic diversity. Mutation belongs to *exploration* in that it introduces new genetic features to the population.

The offspring produced by crossover and mutation replaces some individuals

in the population. One of the most common replacement methods is to select the worst individual in the population as the individual to be replaced. There are two popular schemes for replacement: *generational* and *steady-state*. The former replaces the whole or most of individuals in the population while the latter typically does only one individual. The ratio of the number of replaced individuals to the population size is called *generational gap*. The next generation begins if the replacement is applied to the population. A number of successive generations are created until a predefined stopping condition is satisfied. Typical stopping conditions include the number of maximum generations, the convergence of the population, and so forth. The final best individual is reported as the output of EA.

There are several advantages of using EAs over traditional optimization methods. First, they can be applied to intractable combinatorial optimization problems where only approximate heuristics are known. Due to *implicit parallelism* [Hol75], or the capability of evaluating a huge number of *schemata* [Hol75, PM03a, PM03b] simultaneously, they can explore the search space efficiently. In particular, they are known to be fairly attractive for non-differentiable problems with multiple local optima. Second, they do not require elaborate domain knowledge; they belong to black-box optimizations. They are quite suitable if some domain knowledge is not given at all, or it is hard to be obtained.

Despite these advantages, EAs have some drawbacks, or limitations as well. First, they require a considerable amount of time to evolve a population of individuals. They are not favored if several algorithms compete with one another to satisfy the time constraint of the problem domain. Second, they are not good at fine-tuning around local optima. Pure EAs, ones without any local searches, typically fail to improve upon the solutions around local optima which are more effectively fine-tuned by local searches. Recently, *hybrid* EAs [RB94a, WGM94, KM01, Kra01] have been proposed to alleviate the weakness of the pure EA.[1] Typical hybrid EAs apply a local optimization to the offspring produced by crossover and mutation. Figure 2.1 shows a typical steady-state hybrid evolutionary algorithm.

[1]Recently, hybrid EAs are also called *memetic algorithms* [Mos89, MC03] since the meaning of the term "hybrid" is too broad.

```
create an initial population of a fixed size;
do
        choose parent1 and parent2 from population;
        offspring ← crossover(parent1, parent2);
        mutation(offspring);
        local-optimization(offspring);
        replace(population, offspring);
until (stopping condition)
return the best individual;
```

Figure 2.1: A typical steady-state hybrid evolutionary algorithm

2.1.1 Genetic Algorithm

Genetic algorithm (GA) [Hol75, Gol89] is one of the most popular evolutionary algorithms. Each candidate solution, or chromosome, is typically encoded by a linear string of n-ary discrete or real values whose length is fixed. While binary encoding has been most popular due to its capability of producing diverse schemata, its popularity does not necessarily mean its universal dominance over other representations. This is due to the loss of some important information by the enforcement of too limited arity. The recent studies have alleviated such enforcement by using non-binary representations such as real-valued ones [Mic96].

Each chromosome is assigned a fitness representing the attractiveness of the corresponding solution. Selection for survival and reproduction is then applied to the population. Fitness-proportionate and tournament selections are the popular representatives.

The main operators of GA are crossover and mutation; crossover produces an offspring by recombination of two parental chromosomes and mutation introduces new genes into the offspring. There have been a number of crossovers such as multi-point [SD91], geographic [KM95], natural [JM00], and uniform [Sys89] crossovers. In particular, the choice among crossover candidates is generally dependent upon the underlying chromosomal representation. For example, geographic crossover can naturally be applied to two-dimensional representation of chromosome.

Mutation introduces new genes to the offspring with a low probability. The dis-

ruption by mutation is generally too small to affect the final performance of GA. However, it does guarantee, at least theoretically, that the optimal solution can be found if it exists.[2] In practical, replacing randomly chosen genes with random ones is enough for various applications.

The new offspring, produced by crossover and mutation, replaces one or more chromosomes in the population. There are also several candidates for replacement such as GENITOR-style [WK88], crowding [DeJ75], and so forth. Since population diversity is strongly affected by the replacement, choosing a good replacement is fairly important.

2.1.2 Genetic Programing

Genetic programming (GP) [Koz92] is also popular among evolutionary algorithms in particular for evolving variable-length expressions or programs. It is very similar to GA in that it evolves a population of candidate solutions by genetic operators. Traditionally, GPs have used nonlinear representations such as tree or graph of variable size. The nonlinearity[3] and variability in representation have been two major distinctive features of GP, compared with the other evolutionary algorithms.[4] Tree and graphs, for instance, are popular representations of typical GPs.

To encode solution candidates, GP uses two different sets of nodes: the sets of terminals and nonterminals. The terminal set consists of input features without arity. Each terminal can be a raw input, a pre-processed one, or even an abstracted combination of raw inputs. The nonterminal set includes several functions or operators representing the possible ways of combining the terminals.

Typical GP evolves a fixed size population of individuals by genetic operators including selection, crossover, mutation, and replacement. While selection and replacement are similar to those of other evolutionary algorithms, crossover and mutation are different from them. Due to the widespread tree encoding of most GPs,

[2] In fact, this is related to *ergodicity* in dynamical systems. The interested reader is referred to [NV92, SNF98].

[3] In the most strict sense, nonlinear representation is not included in the distinctive features of GPs. It is because linear representations have been used in the recent literature [BB07, WB10].

[4] In general, each evolutionary algorithm is discriminated from the others by representation.

subtree swapping is one of the most popular crossovers. It randomly chooses two subtrees from the two parents, and produces the offspring by swapping them. Typical mutation operators include replacing a subtree chosen at random with a randomly generated one.

2.2 Evolutionary Algorithms in Finance

The huge dimensionality and non-stationarity of financial data make evolutionary algorithms (EAs) attractive for solving a wide variety of financial optimization problems. In fact, these motivations were emphasized particularly in several studies [CKH08, KCT11]. Since the non-stationarity is naturally related to the evolving capability of EAs [CKH08], it can be one of the most appealing merits when applying EAs to the computational finance. According to different objectives, the past EA studies can be classified into three groups: forecasting [KH00, KM07], trading [NWD97, AK99, Wan00, HCCC12], and modeling [LAP99, CY01].[5] In general, the maximizations of precision and return are the objectives of forecasting and trading, respectively. Since the return is commonly considered with risk, multi-objective frameworks [CTAM09] have often been used in trading. Modeling is different from forecasting and trading, which inevitably involve optimizations, in that its main objective is to gain some insight into market dynamics [RCFM01]. A variety of applications using EAs in the computational finance can be found in [CK02, TC07].

2.3 Search Space Reduction

In this section, we briefly review three evolution approaches for search space reduction: modular, grammatical, and seeded evolutions.

2.3.1 Modular Evolution

Modular evolution is to evolve the population using modules, which are genes or chromosomes that are frequently used and expected to be promising. Its motivation

[5] In forecasting, EAs have often been used to improve upon some existing predictors based on artificial neural networks [KM07].

is to facilitate more effective and efficient evolution by reusing good reusable building blocks. It is thus very similar to modularization exploited in programming by human. Due to the enlarged search space of GP, incurred by representing solutions encoded by trees of variable depths, it has been one of the most important issues in GP. In fact, a large number of techniques for modular evolution have been proposed. They include encapsulation [Koz92], automatically defined function (ADF) [Koz94], module acquisition (MA) [AP93], adaptive representation through learning (ARL) [RB94b], and so forth. While they are different in discovering and reusing modules,[6] their common objective is to improve upon GP by introducing high-level representations. In the broadest context, any techniques exploiting modularity such as hierarchical or layered learning [JG07] can be classified into modular evolution. In recent years, more mathematically rigorous approaches [GGW04, GW07] to modular evolution were proposed, but they are applicable only to the well-known artificial problems including the one-max problem [SE91]. In general, designing effective and efficient modular evolution for complex real-world problems requires systematic algorithms for discovering and reusing modules.

However, modular evolution can be more direct when domain knowledge is readily available and abundant. If it is the case, discovering and reusing modules are degenerated to a restriction on the possible forms of solutions. Such evolution belongs to static module discovery, where the discovery is done before evolution.

2.3.2 Grammatical Evolution

Grammatical evolution is to exploit a grammar, commonly context-free one, for evolving solutions. In general, the possible forms of solutions are restricted to the language derived by the grammar. It is thus important to design a grammar that can generate only valid and possibly promising solutions. Since designing such grammar typically requires small sets of productions, nonterminals, and terminals, it is generally easier than selecting promising modules in modular evolution. That is, the major advantage of grammatical evolution over modular one lies in its less require-

[6] The scopes of module are different as well. For example, ADF is locally defined to an individual while MA is globally defined to the population.

ment for domain knowledge. Recently, a few studies [BO04, CBO10] have begun to use grammatical evolution. It is notable that grammatical evolution [OR01] is further extended using a linear genome representation with a developmental process, or a genotype-phenotype mapping.

2.3.3 Seeded Evolution

Seeded evolution is to provide the initial population with some promising solutions. It implicitly assumes that promising solutions, or at least parts of them, can be identified statically and exploited at runtime. Although it was proposed as one of methods for incorporating more domain knowledge [Gre87], its effectiveness has been validated by many successful applications [CIW92, Jul94, FH96, FM96, PF99, KK05]. Interestingly, some problems exhibiting strong optimal substructure can more naturally exploit the initial population seeded with the best solutions [Sas01] or the best matching ones [OC01] for their subproblems.

There are two major issues in seeded evolution: strength and granularity. Strength generally involves choosing the seeding algorithm, the ratio of seeded individuals, and so forth. It is notable that most seeding algorithms are relatively simple, thereby providing weakly optimized initial solutions. This is is because strongly optimized initial solutions do not always produce better performances, particularly in hybrid EAs. For instance, Reeves [Ree95] tried to solve flowshop sequencing problem with a seeded population but they found no significant improvements except for faster convergence of solutions. In terms of granularity, selecting the candidate for seeding is also essential; it can be either a set of genes or chromosomes. It is also important since excessively seeded population converges too quickly to find reasonably optimal solutions [OC01]. Balancing the search intensities between seeded population and entire genetic framework is thus important to obtain better results in a reasonable time budget.

2.3.4 Summary

Modular and grammatical evolutions are similar in that they restrict the possible forms of solutions using module set or grammar. In fact, they are systematic mecha-

nisms motivated by syntactic typing for solutions. Due to the similarity, any modular evolution can be converted into grammatical one by introducing a grammar whose language is the same as the module set, and vice versa. Since modular evolution requires the module set, which is preparable only by much domain knowledge, its applications are generally more restrictive than grammatical evolution. Seeded evolution is much simpler than modular and grammatical evolutions, since it modifies only the initialization of population. It provides the promising seeded genes or individuals and does not involve in the evolution after the seeded initialization. The seeded genes or individuals can also be replaced by better ones during the evolution. Its restriction on solutions is thus weaker than both modular and grammatical evolutions; it can thus be classified into *search space biasing*.

2.4 Terminology

2.4.1 Technical Pattern and Technical Trading Rule

It is common that the terms called *technical pattern* and *technical trading rule* are used without clear definitions. Typically, a technical pattern is a Boolean expression consisting of diverse price information, technical indicators, and operators. It is commonly associated with buy or sell. However, it can have no explicit trading implications when it is used as a pre-context for other technical patterns. For example, the pattern $MA_5(t) > MA_{20}(t) \wedge MA_{20}(t) > MA_{60}(t)$ represents short-term (5,20) moving averages are greater than long-term (20,60) ones for the current day. It is commonly interpreted as a buy signal but its meaning can also be interpreted as a pre-context for other patterns. In short, a technical pattern is typically interpreted as buy or sell signal but it has no inherent and context-free interpretations.

In contrast, a technical trading rule has explicit buy and sell signals. For example, consider a technical trading rule is as follows:

$$\begin{array}{l} MA_5(t) > MA_{20}(t) \wedge MA_{20}(t) > MA_{60}(t) \rightarrow BUY \\ MA_5(t) < MA_{20}(t) \wedge MA_{20}(t) < MA_{60}(t) \rightarrow SELL \end{array} . \tag{2.1}$$

In the technical trading rule, some trading days are assigned either BUY or SELL signal while the others are not associated with any signal. In a technical

trading rule, there are no ambiguities for interpretations. It is quite common that a typical technical trading rule includes several technical patterns according to its own interpretation.

2.4.2 Forecasting Model and Trading Model

The previous studies on financial data mining can be grouped by their objectives: forecasting and trading models. The objective of forecasting models is to find models for predicting the movement of financial objects including stocks, futures, and derivatives. There have been a number of studies can be grouped into forecasting models including artificial neural networks (ANNs) [KM07], support vector machines (SVMs) [CT01, Kim03], and so on. However, highly precise model does not necessarily mean profitable one, which was emphasized by [YCK05]. For example, most oscillator technical indicators such as stochastics produce high precisions but low returns. This is because the returns for correct predictions are small. In sum, forecasting models are interested in whether a financial object rises or falls.

In contrast, trading models focus on the returns. They typically focus on the rate by which a financial object rises or falls. Since the profitability is the main interest for traders, there have been a huge number of trading models including various machine learning techniques [AK99, PSV04]. It is notable that some trading models often produce trading rules with precisions less than 50%; their profits are made by the large moves of prices for correct predictions.

One can see that the two models seem to be closely related but they have quite different objectives. Forecasting models have often been used as good candidates for improving upon trading models.

There are also some preferences according to the machine learning methods. For technical analysis, most forecasting models have used learning methods on continuous space including ANNs and SVMs. Recently, SVMs have been more popular than other methods due to their generalization capabilities. In contrast, trading models have used learning methods on discrete space including GPs. The popularity of them is attributed to the interpretabilty of solutions which is one of the most appealing features of GPs.

2.4.3 Portfolio and Rebalancing

A *portfolio* is formulated as a real-valued column vector $(w_1, w_2, \ldots, w_N)^T$, where w_i and N represent the proportion of asset invested on the i-th financial object, called *weight*, and the number of financial objects, respectively. The sum of all proportions is typically one, meaning the entire asset that can be invested. Each proportion is generally nonnegative, but it can be negative if *short sale*, or simply sell of a borrowed financial object, is allowed for the corresponding financial object. The financial objects can be the same or different; our primary focus is on the homogeneous financial objects, i.e., stocks, or securities. The *portfolio return* is the weighted arithmetic mean of the returns of financial objects in a portfolio.

Portfolio consisting of different financial objects is commonly used in higher level asset managements; it generally includes both risky and risk-free assets to control the level of expected return and risk. By constructing a portfolio, an investor can minimize *non-systematic*, or *diversifiable risk*.[7] Once an initial portfolio is constructed, the weights in the portfolio are generally changed based on period or tolerance band of estimated asset [DY03, Mas03]; the readjustment of the weights in a portfolio is called *rebalancing*. The key idea to rebalancing is to increase the undervalued financial objects while decreasing the overvalued ones, thus providing a steady growth in the portfolio return.

2.4.4 Data Snooping Bias

Data snooping bias, or also called *data mining bias*, is a statistical bias by *ex post* selection of models. Technical trading rules and momentum/contrarian strategies [JT01, DT85] are common examples of the models. By ex post, we mean that a model is selected at the end of a given time series' period. It is thus possible that one can derive any profitable models by exploiting the past data extensively. However, the fitted models are likely to have little predictive power for previously unseen data.

In the finance literature, the data snooping bias is typically relieved by statistical tests including *reality checks* [Whi00, Han05] which use a full universe of

[7] Since it is related to the condition of a firm, it is also called *firm-specific risk*. Its counterpart is *systematic*, or *non-diversifiable risk* which results from the market condition.

trading models and *stationary bootstrap* samples [PR94]. The full universe means that any trading model's performance should be evaluated by a set of full possible configurations or parameterizations. For example, a filter rule using n-day local low price should be tested by all possible n's. The stationary bootstrap also has similar implications for the test data. Typical stationary bootstrap generates a number of samples, representing different possible scenarios, to be used for obtaining the distribution of test statistic.

While the statistical tests are statistically sound, they have several limitations as well. First, preparing a full universe of trading models requires a considerable amount of time. In addition, it can be biased toward the experimenter's experience as well. Second, choosing a set of parameters for stationary bootstraps is not standardized. Typically, it is done by following the earlier studies because the parameters are known to be insignificant for the statistical test [STW99].

Most statistical studies reported that they found almost no excess returns for popular trading rules. We think that this is due to severe performance degradation by including too large number of unmeaningful models or resampled data.

In the computer science literature, the most similar term to the data snooping bias is *overfitting*. Solutions to overfitting have a long history including the early stopping [Pre98], parsimony pressure [GSPT06] toward simpler models, and so forth. For technical trading, Allen and Karjalainen [AK99] summarized four general solutions to overfitting: 1) introducing a validation set, 2) increasing the amount of training data, 3) penalizing for the model complexity, and 4) minimizing information for describing the model and data. They used a validation set called a *selection period* for choosing the best validation rule from a sequence of the trained best rules during a run. Recently, GPs alleviating the data snooping bias with the reality check have also been proposed. For example, Agapitos *et al.* [AOB10] reported their out-of-sample returns were improved using Hansen's SPA test [Han05]. However, it is still unclear which method is better for obtaining stable out-of-sample returns with diverse data and learning frameworks.

2.5 Financial Optimization Problems

2.5.1 Attractive Technical Pattern Discovery and Its Extension

Most studies in technical analysis have focused on finding new profitable technical trading rules. They typically used genetic programming where each solution contains both buy and sell rules. The buy and sell conditions can be implicit, or not contained in the solution, if an independent automaton for trading is involved [AK99]. Since a technical trading rule contains one or more technical patterns, such studies are considered to implicitly optimize two objectives: optimizations of technical pattern and trading model. These studies collectively belong to *technical trading rule discovery*.

Although they have been popular and quite intuitive, they have several problems. One of the most severe problems is the increased complexity resulting from the simultaneous optimization of technical pattern and trading model. In fact, identifying optimal technical pattern involves a complex non-parametric optimization where both structure and parameters should be considered. Trading model is generally more easily optimized than technical pattern, but it is a non-trivial parametric optimization as well.

Recently, several studies have focused primarily on finding new technical patterns. They first try to find technical patterns that are attractive in terms of some criteria. For example, Lee and Moon [LM10] formulated *attractive technical pattern* based on profitability, simplicity, and frequency. They obtained notable excess returns over buy-and-hold with a commercial trading simulation tool, which corresponds to a trading model. The merit of finding attractive technical patterns is the separation of technical pattern and trading model. The newly discovered technical patterns can be exploited by other trading models; they are not strongly coupled with some trading models, resulting in an improved generality. These studies, focusing on finding new technical patterns, are referred to as *technical pattern discovery*. In general, they can virtually be called *attractive technical pattern discovery* since they are always associated with some criteria for attractiveness of technical pattern.

2.5.2 Stock Selection

Stock selection is to identify a set of stocks that are expected to produce high excess returns [HCCC12, Hua12, ZYH+06]. While modern portfolio theory [Mar52] uses a portfolio vector consisting of all stocks' invested asset proportions, stock selection scores all stocks with various criteria and select the top-ranked ones for a portfolio. It is more practical to use stock selection than modern portfolio theory due to its low computational cost and high scalability.

In general, stock selection is also called *stock picking* [Wer00, CB05, GNH11] or *stock screening* [GL99, TKC91, SD09].[8] It reduces the full universe of stocks to a set of stocks that have some attractive features typically by financial ratios [Cou78, Whi80, Bar87] including P/B (price-to-book), P/E (price-to-earnings), and D/E (debt-to-equity) ratios. This reduction makes stock selection attractive for both a prior step for portfolio construction [HZ95] and a direct portfolio by selecting the top-ranked stocks [HCCC12]. There have been a number of studies for stock selection including ANNs [QS99], GPs [CB05, BFL07], and GAs [HCCC12].

For stock selection, aggregating scores produced by the set of financial ratios is important; it can be linear or nonlinear. Linear models have been popular due to their simplicity but several studies showed that nonlinear models are promising as well. One of the most famous linear models is Piotroski's score, or also called F-Score, where nine binary financial signals are summed up for scoring stocks [Pio00].

By the degree of exploiting expert's evaluation, stock selection can be divided into two groups: *fully automated* and *partially automated*. While fully automated stock selection uses domain knowledge at the design stage, which is provided typically by experts, it is not helped by experts in the later stages. Most machine learning algorithms for stock selection belong to this class.

[8] In the most strict sense, stock screening typically refers to filtering out unattractive stocks. However, many studies have also used this term as synonym to stock selection. Stock picking is commonly exploited in the context of stock picking ability of funds.

2.6 Issues

2.6.1 General Assumptions

In fact, most financial optimization problems virtually belong to the general data mining with time series. They are generally provided with large data sets, commonly divided into training, validation, and test sets. Their objectives are commonly related to building forecasting or trading models that will consistently work on the test sets. Similar to the general data mining, solving financial optimization problems relies on three underlying assumptions: existence, identifiability, and exploitability of model [LC10]. We reintroduce these assumptions with our own view, focusing on financial optimization problems.

First, the existence of model assumes that there exists a model, irrespective of interpretability of the model, in the time series data. In fact, it is the most fundamental assumption since any data mining becomes futile when the data are closer to random ones, having no identifiable model. Since it cannot be validated easily, most studies implicitly assume that there are some models in their data. Recently, some pretests [CN06] were proposed to validate such model existence assumption, focusing on GP-evolved trading strategies. Second, the identifiability postulates that a model can be discovered if it exists in the data. While it seems to be quite natural, it cannot be easily verified as well; the identifiability involves an intractable number of testing possible algorithms when no formal proof is available. Instead, most studies on financial optimization problems have used popular methods such as classical regression models and evolutionary algorithms for model identification, or discovery. Finally, the exploitability means that an identified model will consistently work in the future. In general, it is much harder to satisfy in the financial domain than other general ones. This is mainly due to the different time in each data entry, meaning that the underlying features in the data are likely to be time-variant. The market context such as trend [LAOK10] can be used to improve upon such inconsistency, or degraded generality, but its availability is inherently limited in that it can only be determined *ex post*. Although there are also popular regression models for explain-

ing return and its volatility in the finance literature,[9] they are not quite useful for data-driven learning models; this is primarily due to many simplified assumptions for modeling time series data.

2.6.2 Performance Measure

In most financial optimization problems that focus on profitability, return and risk are two common measures for performance comparisons. In fact, these are the two fundamental performance factors in the computational finance. Arithmetic and geometric average returns are popular measures in aggregating the returns of stocks and periods, respectively. The aggregation of returns over a given period can use arithmetic average return and it is commonly used with risk measures[10] such as Sharpe ratio [Sha66, Sha94]. However, geometric average return is more natural than arithmetic average one, even with risk measures, since it is the real growth rate of invested asset [DE13].

Compared with other literature, where some famous benchmark problems are available, most financial optimization problems focus only on beating *buy-and-hold*, or market return. This is mainly due to different markets, periods, universes, and so forth. It is thus virtually impossible to compare among algorithms, except for some simple but standardized experimental studies. Fortunately, buy-and-hold can be used as the common benchmark for performance comparisons since it represents the market return. Since the possibility of outperforming buy-and-hold is still controversial, it is enough to compare the return with that by buy-and-hold for validating the usefulness of a system or strategy. It should be noted that each market has its most representative market return; in Korean stock market, KOSPI (Korea Composite Stock Price Index) return is commonly selected as a benchmark.

[9] Autoregressive moving average (ARMA) [Whi51] and generalized autoregressive conditional heteroskedastic (GARCH) [Bol86] models are popular return and volatility models, respectively.

[10] There are a number of risk measures including Jensen's alpha [Jen68], Treynor's ratio [Tre65], value at risk (VaR) [LP00], Sortino ratio [SVDM91], Sterling ratio [Kes96], and so forth. Despite of their differences, many risk measures showed an identical ranking in comparing among funds [ES07].

Chapter 3

Modular Evolution

Modular evolution is to evolve the population of candidate solutions with modules, or good reusable building blocks. It assumes that good building blocks appear repetitively in the evolution, thereby playing an important role in the subsequent evolution. This is quite similar to modular programming by human where reusable code chunks are modularized, and then reused by other similar programs. Since genetic programming has originally been invented for evolving programs, it has been the representative metaheuristic exploited in modular evolution. In this chapter, modular evolution is thus described only in the context of genetic programming.

In general, there are three fundamental issues in modular evolution: definition, discovery, and reuse. The definition of module is still not clear despite it can easily be defined intuitively. For example, a module can be defined by a frequently used subtree with a relatively high fitness in the most general definition. By such a broad definition, even an incomplete subtree consisting of only nonterminals such as arithmetic and Boolean operators can be a module; but its fitness cannot be defined explicitly. If it is the case, the fitness should be approximated for its availability, which is not straightforward for many real-world problems. Hence, it is common that only complete subtrees, which are meaningful and frequent ones, have been modularized in most studies.

Once the definition of module is clarified, the discovery of the module can be done in many ways. The most common classification for module discovery is based

on the timing to discover modules: static and runtime. In static module discovery, modules are identified before a genetic evolution begins. Due to the timing, they cannot exploit the runtime statistic of a genetic evolution such as the frequency of a subtree. Each module is thus discovered by some domain knowledge such as its popularity, complexity, and so forth. In contrast, modules can be discovered during a genetic evolution, which is based on runtime statistic. Typical runtime module discovery is based on several runtime statistic such as frequency and fitness of subtree. The choice between the two methods is commonly related to the amount of domain knowledge. In general, static module discovery is preferred if the domain knowledge is enough to give insight into the problem domain.

The discovered modules are then preserved and reused in a genetic evolution. Each module is encapsulated as an atomic unit and reused in other individuals, or solutions. Although there can be a number of possible ways to reuse modules, typical module reuse is implicitly done by crossover operator. To reuse the modules, a genetic programming should save them on a repository which includes individuals themselves, the set of terminals, some statistic data, and so forth. They can also be dynamically extended or shrunk by resource limitation, which is quite similar to popular cache replacement methods.

In this chapter, we propose a modular genetic programming for finding attractive and statistically sound technical patterns for stock trading. The search space is restricted to combinations of modules for more effective search. We carefully prepared the set of modules based on existing studies of technical indicators and our own experience. Our modular genetic programming successfully found unknown attractive technical patterns for the Korean stock market. A trading simulation with the generated patterns by a commercial tool showed significantly higher accumulative returns than buy-and-hold, or KOSPI.

3.1 Modular Genetic Programming

Typical, or unrestricted GPs are known to be very inefficient to find sensible solutions for highly complex problems [Mon95]. In fact, they spend most of their time on

finding only feasible solutions whose fitness values are much worse than the optimal ones [Mon95].

To give insight into our modular GP, let us consider the problem of discovering attractive technical patterns. The attractiveness of a technical pattern can be defined by several criteria such as profitability, simplicity, and frequency. Although there are a number of building block patterns for attractive technical patterns, they are generally too complex to be discovered from scratch by black-box optimizations such as genetic programming.

For example, a *white marubozu* with long body ($p_c(t) = p_h(t) \wedge p_o(t) = p_l(t) \wedge 1.07 * p_o(t) < p_c(t)$, where $p_o(t)$, $p_h(t)$, $p_l(t)$, and $p_c(t)$ are opening, high, low, and closing prices for the current day t, respectively) is known to be very effective for trading. It implies that a bullish market is strongly supported by buyers, thus it is a well-known signal for buy timing. However, it is hard to be discovered by typical GPs due to its complexity; it consists of three clauses which are tightly defined. The situation would become more serious if we attempt to find more attractive technical patterns containing one or more white marubozus. Type restrictive GPs including strongly typed and lambda abstracted ones have the same hardness as well.

Instead, we define a set of module patterns and try to find more attractive technical patterns by recombination of them. To define the set of modules requires some domain knowledge, but it is quite an efficient way to help GPs search on more sensible space.

More generally, let $M = \{m_1, m_2, \ldots, m_d\}$ be the set of module solutions, or expressions. Each m_i is restricted to a comparison expression or conjunctions of comparison expressions. In a genetic evolution, each m_i is preserved as a *module*. By module, we mean that it is never broken into individual clauses by any genetic operators including crossover, mutation, and local optimization. More formally, we define modular space and modular closure property as follows:

Definition 3.1.1 (Modular search space)**.** *A* modular search space C *induced by a module set* M *is* $C = \cup_{k=1}^{\infty} P_k$*, where* P_k *is the set of all* k *distinct conjunctive forms of* $m_i \in M$*.*

Definition 3.1.2 (Modular closure property)**.** *A genetic operator* $g\colon C^n \to C$ *is said to satisfy* modular closure property *on modular space* C *if and only if* $\forall(c_1, c_2, \ldots, c_n) \in C^n$, $g(c_1, c_2, \ldots, c_n) \in C$.

For example, a binary crossover satisfying modular closure property is regarded as a genetic operator g mapping two parents $(p_1, p_2) \in C^2$ to an offspring $g(p_1, p_2) \in C$. All genetic operators including crossover, mutation, and local optimization should satisfy the modular closure property on C. The modular closure property of genetic operators guarantees that modular GP works only on modular space. As an example, we show that a modular crossover, whose cut points are restricted to only the set of module root nodes and Boolean operator ones, satisfies modular closure property.

Fact 3.1.1. *A* modular crossover*, whose cut points are restricted to only the set of module root nodes and Boolean operator ones, satisfies modular closure property.*

Proof. Let $M(p_1) \subseteq M$ and $M(p_2) \subseteq M$ be the sets of modules in two parents p_1 and p_2, respectively. If a valid modular cut point is assigned to each parent, we have two sets of modules $S_1 \subseteq M(p_1)$ in p_1 and $S_2 \subseteq M(p_2)$ in p_2 to be swapped. After the crossover operator, we have two offspring o_1 and o_2 represented by $(M(p_1) \cap {S_1}^{\mathsf{c}}) \cup S_2$ and $(M(p_2) \cap {S_2}^{\mathsf{c}}) \cup S_1$, respectively. For the offspring o_1, $(M(p_1) \cap {S_1}^{\mathsf{c}}) \cup S_2 \subseteq M$ since $M(p_1) \cap {S_1}^{\mathsf{c}} \subseteq M(p_1)$ and $S_2 \subseteq M(p_2)$. By Definition 3.1.1, $o_1 \in P_{|(M(p_1) \cap {S_1}^{\mathsf{c}}) \cup S_2|} \subseteq C$, where $|(M(p_1) \cap {S_1}^{\mathsf{c}}) \cup S_2| < \infty$. The argument for o_2 is symmetric to o_1. □

It is notable that our modular search space is simpler than the existing studies [Woo03, GW07], but still captures several important aspects in modular evolution. The modular closure property is the most important aspect in that it enforces genetic operators to preserve the module solutions, which are atomic meaningful units. Our formulation also facilitates easier validation of the modular closure property for diverse genetic operators. For example, all genetic operators including crossover, mutation, and local optimization in the next section are designed so that they satisfy the modular closure property. The proofs of their modular closure properties are omitted since they are rather straightforward.

However, this naive set-theoretic definition of modular search space has limitations as well. One of the most important limitations is the underestimation of the

real search space of GP by excluding of isomorphic solutions, or trees. In fact, the search space of GP typically consists of trees which have a huge number of isomorphic ones. To give some insight into the problem, let us define a *finite modular search space* $C^d = \cup_{k=1}^{d} T_k$, where T_k is the set of full binary trees with k distinct modules in $M = \{m_1, m_2, \ldots, m_d\}$ and $k-1$ Boolean AND operators.

Fact 3.1.2. *The cardinality of a finite modular search space* C^d *is* $\sum_{k=1}^{d} \frac{(2(k-1))!d!}{(k-1)!k!(d-k)!}$.

Proof. The possible number of tree topologies with k distinct modules and $k-1$ Boolean AND operators is the same as the possible number of distinct binary tree topologies with k leaves; the modules and Boolean AND operators correspond to leaf nodes and internal ones, respectively. By this correspondence, it can be calculated by $(k-1)$th *Catalan number* $C_{k-1} = \frac{(2(k-1))!}{(k-1)!k!}$. For each tree topology, there exist $P(d,k) = \frac{d!}{(d-k)!}$ permutations of modules. Hence, the cardinality of T_k is the product of C_{k-1} and $P(d,k)$. The cardinality of C^d is then calculated as follows:

$$\begin{aligned} |C^d| &= \sum_{k=1}^{d} |T_k| \\ &= \sum_{k=1}^{d} C_{k-1} P(d,k) \\ &= \sum_{k=1}^{d} \frac{(2(k-1))!}{(k-1)!k!} \frac{d!}{(d-k)!} \\ &= \sum_{k=1}^{d} \frac{(2(k-1))!d!}{(k-1)!k!(d-k)!}. \end{aligned}$$

□

Fact 3.1.2 shows that even a finite modular space has a huge number of possible trees, or solutions in GP. The selection of appropriate module set is thus quite important for search space reduction.

3.2 Hybrid Genetic Programming

We use a steady-state genetic programming, where the offspring promptly replaces individuals in the same population.

- Representation and initialization

  Each individual is represented by a randomly generated tree. It is a parse tree representing a conjunctive expression of modules. The population size is set to 50.

- Selection, crossover, and mutation

  The tournament selection [GDK91] is used. The crossover chooses two subtrees from parents at random and swaps them. It is important that the crossover does not disrupt modules and preserves them as encapsulated units. For the preservation of modules, a cut point, or the root node of a subtree to be swapped, should be either a Boolean operator node or the root node of a module. Figure 3.1 shows an example individual and possible cut points colored by dark or light gray. This crossover satisfies the modular closure property as shown in Fact 3.1.1. We use a mutation that picks a module at random and replaces it with another module chosen at random.

- Replacement and stopping criterion

  The worst individual is first tried to be replaced. If it is the same as the best on the validation period, the worse of two randomly selected individuals is replaced with some probability. Otherwise, we replace the better with a small probability. The number of maximum generations is 1,000 and the maximum consecutive fails before termination is 300.

- Local optimization

  In the local optimization shown in Figure 3.2, we traverse each module in an individual and compute the gain if it is replaced with another module chosen at random. We apply the replacement of a module only if the maximum gain is positive. The neighborhood size N is set to 3.

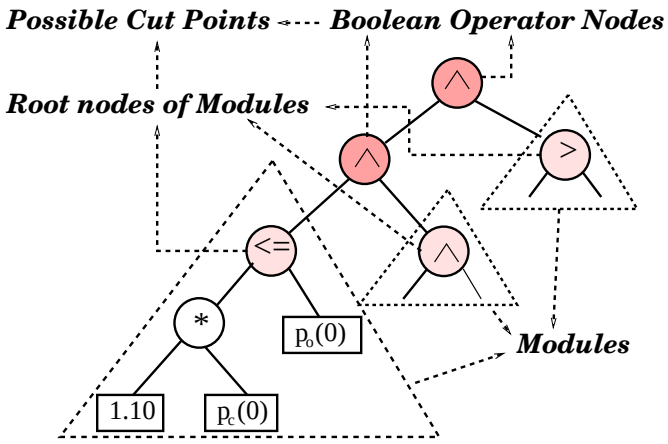


Figure 3.1: An example individual and possible cut points

3.3 Attractive Technical Pattern Discovery

3.3.1 Introduction

Technical pattern analyses [Nis91, EMB07, Mur99, LMW00] aim to capture appropriate timing in a stock market. There have been a huge number of technical patterns for predicting the trend of stock prices. They typically predict short-term movements: continuation, reversal, and no movement [Mor92]. However, most of them are based on investors' limited intuition and experience. In this sense, they are subjective and likely to have cognitive bias.

In Morris' study [Mor92], the average hit ratio (up or down) of 88 famous candle patterns was around 51%, which is just 1% over a random guess. Thirty three of them showed lower than 50% hit ratios. The result implies that the patterns based on human beings intuition are not statistically solid.

To alleviate the subjectiveness, evolutionary algorithms such as genetic algorithm [Hol75, Gol89] and genetic programming (GP) [Koz92, BNKF98] for finding technical patterns have been proposed. Some of them were successful to find technical patterns that give excess returns over buy-and-hold [Lip07] while others reported no excess returns over buy-and-hold [AK99, CY97, MYR07, MYC08]. The possibil-

```
Input:
T: tree, or individual
N: neighborhood size
Output:
T: locally optimized tree
Description:
LocalOptimization(T,N)
{
        i ← 0;
        for each root node of module m_i ∈ T;
                G_max ← 0;
                for j ← 1 to N
                        m'_i ← a random module from M;
                        T' ← T with m'_i in place of m_i;
                        G_j ← f(T') − f(T);
                        if G_j > G_max
                                G_max ← G_j;
                                m_max ← m'_i;
                if G_max > 0
                        T ← T with m_max in place of m_i;
        return T;
}
```

Figure 3.2: Local optimization for modular GP

ity of finding profitable technical patterns beating the efficient market hypothesis [Ale64, FB66, Sam65, JB70, Fam70] has been controversial.[1] Regardless of the profitability, most of them were unsuccessful to find sensible and interpretable technical patterns.

Since typical GPs require a considerable amount of time to find sensible and interpretable technical patterns, restrictive GPs have been popular alternatives. They include *strongly typed* GPs [Mon95] and *lambda abstracted* GPs [YCK05]. They are similar in restricting the set of input types, which permits GPs to evolve only on valid and sensible technical patterns. While they were successful for some domains, the restrictions are not strong. For example, we cannot expect such GPs to find well-

[1] In terms of technical analysis, the efficient market hypothesis can be interpreted that there is no exploitable patterns in financial markets.

known complex technical patterns such as white marubozu with long body [Mor92] in a reasonable time. It seems that the type restrictions by strongly typed and lambda abstracted GPs should be further refined to find sensible technical patterns in stock markets, which is the main motivation of our new modular genetic programming. Our new modular genetic programming works on *modules*, or predefined technical subpatterns to be combined, which reduces the search space to more promising ones.

In addition to our new genetic framework, our approach has two notable features.

First, we focus only on the buy side. Although most studies on technical trading have focused on both buy and sell sides, they have struggled with increased complexity. They managed to construct some successful technical trading models, but their models were too complex or hard to interpret. Instead, we focus on the buy side and try to find attractive technical patterns to capture buy timing. Of course, it would be symmetric and easy to apply our GP to the sell side.

Second, most of the previous studies are limited as well since they have focused mainly on profitability. They considered neither simplicity nor generality of technical patterns. If technical patterns are complex, they generally match few cases, which degrades the generality even if they are highly profitable. In this context, we try to find attractive technical patterns considering simplicity and generality as well as profitability.

3.3.2 Problem Formulation

Attractive technical patterns can be defined in terms of various criteria. One of the most intuitive criteria is profitability. However, highly profitable technical patterns may become too complex to be interpreted. Since too complex patterns disable us to interpret, they are not preferred in general. In other words, attractive technical patterns should be not only highly profitable but also simple enough to facilitate easy interpretation.

Another additional criterion for attractive technical patterns is frequency. Although a technical pattern is both highly profitable and simple enough, it may match few cases for a given data set. Its usefulness would be greatly degraded if it has too small matching cases.

In summary, we define a technical pattern r to be *attractive* if it satisfies all of the following three criteria:

1. r should be highly profitable. (profitable)

2. r should not be too complex. (simple)

3. r should have at least a given number of matching cases. (frequent)

More formally, let $R(r)$ be the set of matching cases by a technical pattern r as follows:

$$R(r) = \{(i, j) : r \text{ matches company } i \text{ on trading day } j\}. \tag{3.1}$$

To measure the profitability of a technical pattern r, we define *expected earning rate* of r after k trading days by

$$E_k(r) = \frac{1}{|R(r)|} \sum_{(i,j) \in R(r)} \frac{p_c(i, j + k)}{p_c(i, j)}, \tag{3.2}$$

where $p_c(i, j)$ is the closing price of company i on trading day j.

Intuitively, the expected earning rate of r after one trading day, or $E_1(r)$, can be used for finding profitable technical patterns. However, it is fairly sensitive to daily fluctuations of stock prices. To alleviate the sensitivity, we use the arithmetic average of n consecutive days' expected earning rates. This can be regarded as a form of *smoothing* to the expected earning rate.

To find simple and frequent patterns, we need additional constraints on technical patterns. We define a technical pattern r to be simple if $|r| < M$, where $|r|$ is the number of clauses in r. We also define a technical pattern r to be frequent if $|R(r)| \geq m$, where $|R(r)|$ is the cardinality of the set $R(r)$. The m and M are predetermined constants that require tuning by prior knowledge.

The fitness of a technical pattern r is defined by

$$f(r) = \begin{cases} \frac{1}{n} \sum_{k=1}^{n} E_k(r) & \text{if } |r| < M \text{ and } |R(r)| \geq m \\ 0 & \text{otherwise,} \end{cases} \tag{3.3}$$

where n is the number of consecutive days for smoothing the expected earning rates. We call n the *smoothing constant* for $f(r)$. Our problem is a maximization problem of finding a technical pattern r that maximizes $f(r)$.

Some useful facts on $f(r)$ can be drawn with a few assumptions. Let μ be the upper limit on daily price change. Differently from U.S. market, Korean stock market does not allow a daily price change exceeding some limit, which is enforced by a *price limit system* [LK95]. That is, we assume that $p_c(i, j+1) \leq \mu \times p_c(i, j)$ for all (i, j) cases.

Fact 3.3.1. *Given a daily price limit $\mu > 1$, $f(r) \leq \frac{1}{n}\frac{\mu(\mu^n - 1)}{\mu - 1}$.*

Proof. Suppose that $|r| < M$, $|R(r)| \geq m$, and for all $(i, j) \in R(r)$, $p_c(i, j+k) = \mu^k \times p_c(i, j)$ to maximize $f(r)$. The upper bound for $f(r)$ is calculated as follows:

$$\begin{aligned}
f(r) &= \frac{1}{n}\sum_{k=1}^{n} E_k(r) \\
&= \frac{1}{n}\sum_{k=1}^{n}\left(\frac{1}{|R(r)|}\sum_{(i,j)\in R(r)}\frac{p_c(i, j+k)}{p_c(i, j)}\right) \\
&\leq \frac{1}{n}\sum_{k=1}^{n}\left(\frac{1}{|R(r)|}\sum_{(i,j)\in R(r)}\frac{\mu^k \times p_c(i, j)}{p_c(i, j)}\right) \\
&= \frac{1}{n}\sum_{k=1}^{n}\mu^k \\
&= \frac{1}{n}\frac{\mu(\mu^n - 1)}{\mu - 1}.
\end{aligned}$$

□

Of course, the upper bound of $f(r)$ is dependent on both μ and n.

3.3.3 Modular Search Space

Let the original search space of technical patterns be the set of Boolean clauses. Each clause is an expression of comparison operators, arithmetic operators, functions, and constants. A technical pattern is restricted to a *conjunctive form* of expressions. The set of comparison operators and that of arithmetic operators are $\{<, >, =, \leq, \geq\}$ and $\{+, -, *, /\}$, respectively. A function is represented by a symbol which takes

Table 3.1: The set of functions

Name	Notation
Opening Price	$p_o(t)$
Closing Price	$p_c(t)$
Highest Price	$p_h(t)$
Lowest Price	$p_l(t)$
Trading Volume	$v(t)$
n-day Moving Average	$MA_n(t)$
n-day Volume Moving Average	$VMA_n(t)$
n-day Highest Price	$HP_n(t)$
n-day Lowest Price	$LP_n(t)$
Bollinger Upper Band	$BUB_n(t)$
Bollinger Lower Band	$BLB_n(t)$
n-day Disparity	$DISP_n(t)$
Stochastic(n,m,k) Fast $\%K$	$STOF_{(n,m,k)}\%K(t)$
Stochastic(n,m,k) Slow $\%K$	$STOS_{(n,m,k)}\%K(t)$
Stochastic(n,m,k) Slow $\%D$	$STOS_{(n,m,k)}\%D(t)$

an integer *relative offset* from the current trading day as an argument. Table 3.1 shows the set of functions. A constant is a real value which is typically multiplied to a variable or a constant. For example, the expression of "$1.1 * p_o(-1) < p_c(0) \wedge MA_{20}(0) > MA_{60}(0)$" means that the opening price of the previous trading day multiplied by 1.1 is smaller than the closing price of the current trading day and the 20-day moving average of the current trading day is greater than the 60-day one of the current trading day.

While the original search space is natural, it has a drawback of allowing a vast number of unmeaningful technical patterns. For example, a price can be compared with a trading volume, which has almost no meaning. Similar problems arise whenever an input is compared with less meaningful inputs. With the unrestricted GPs, these problems can only be alleviated by implicit penalization of such unmeaningful expressions, which is fairly inefficient.

Instead, we prepare the set of module solutions using our domain knowledge, which is the basis for our modular search space. Table 3.2 shows some representative modules. It includes various technical patterns consisting of popular candlesticks [Nis91] and technical indicators [BLL92]. The selection is based primarily on pat-

Table 3.2: Some representative modules: candlesticks and technical indicators

Category	Pattern	Constants†
White Body	$m * p_o(t) \leq p_c(t)$	$m \in \{1.07\}$, $t \in [-4, 0]$
	$m * p_o(t) \leq p_c(t) \wedge n * p_o(t) > p_c(t)$	$(m, n) \in \{(1.0, 1.02), (1.02, 1.05), (1.05, 1.07)\}$, $t \in [-4, 0]$
Black Body	$m * p_o(t) \geq p_c(t)$	$m \in \{0.93\}$, $t \in [-4, 0]$
	$m * p_o(t) \geq p_c(t) \wedge n * p_o(t) < p_c(t)$	$(m, n) \in \{(1.0, 0.98), (0.98, 0.95), (0.95, 0.93)\}$, $t \in [-4, 0]$
Gap Up	$m * p_h(t-1) < p_l(t)$	$m \in \{1.0, 1.02, 1.05, 1.07\}$, $t \in [-4, 0]$
Gap Down	$m * p_l(t-1) > p_h(t)$	$m \in \{1.0, 0.98, 0.95, 0.93\}$, $t \in [-4, 0]$
White Marubozu	$p_c(t) = p_h(t) \wedge p_o(t) = p_l(t) \wedge m * p_o(t) < p_c(t)$	$m \in \{1.0, 1.02, 1.05, 1.07\}$, $t \in [-4, 0]$
Black Marubozu	$p_o(t) = p_h(t) \wedge p_c(t) = p_l(t) \wedge m * p_o(t) > p_c(t)$	$m \in \{1.0, 0.98, 0.95, 0.93\}$, $t \in [-4, 0]$
Equalities of Prices	$P(t) = P'(t)$	P, P' $(P \neq P') \in \{p_o, p_c, p_h, p_l\}$, $t \in [-4, 0]$
Uptrend by MA	$MA_{20}(t-1) < MA_{20}(t)$	$t \in [-9, 0]$
Downtrend by MA	$MA_{20}(t-1) > MA_{20}(t)$	$t \in [-9, 0]$
Uptrend by Stochastic	$STOS_{(5,3,3)}\%D(t) < STOS_{(5,3,3)}\%K(t)$	$t \in [-9, 0]$
Downtrend by Stochastic	$STOS_{(5,3,3)}\%D(t) > STOS_{(5,3,3)}\%K(t)$	$t \in [-9, 0]$
Bullish Arrangement of MA	$MA_5(t) > MA_{10}(t) \wedge MA_{10}(t) > MA_{20}(t) \wedge MA_{20}(t) > MA_{60}(t)$	$t \in [-9, 0]$
Bearish Arrangement of MA	$MA_5(t) < MA_{10}(t) \wedge MA_{10}(t) < MA_{20}(t) \wedge MA_{20}(t) < MA_{60}(t)$	$t \in [-9, 0]$
Disparity	$DISP_{20}(t) > 105$	$t \in [-9, 0]$
	$DISP_{20}(t) < 95$	$t \in [-9, 0]$
Uptrend by VMA	$VMA_{20}(t-1) < VMA_{20}(t)$	$t \in [-9, 0]$
Downtrend by VMA	$VMA_{20}(t-1) > VMA_{20}(t)$	$t \in [-9, 0]$
Sell Signal by Stochastic	$STOS_{(5,3,3)}\%K(t) > 80$	$t \in [-9, 0]$
Buy Signal by Stochastic	$STOS_{(5,3,3)}\%K(t) < 20$	$t \in [-9, 0]$
Bollinger Band Break	$BUB(t) < p_c(t)$	$t \in [-9, 0]$
	$BLB(t) > p_c(t)$	$t \in [-9, 0]$
Trading Range Break	$HP_5(t) < p_c(t)$	$t \in [-9, 0]$
	$LP_5(t) > p_c(t)$	$t \in [-9, 0]$

† We denote by $[m, n]$ the set of integers from m to n (boundaries inclusive).

tern's popularity and complexity.

3.3.4 Experimental Results

Parameter Settings

We set the smoothing constant n for $f(r)$ and the minimum cardinality m of $R(r)$ to 5 and 300, respectively.

Table 3.3: Data set and its division

Training years	# of stocks	Validation year	# of stocks	Test year	# of stocks
2000 - 2002	1636	2003	1589	2004	1616
2001 - 2003	1676	2004	1616	2005	1638
2002 - 2004	1704	2005	1638	2006	1653
2003 - 2005	1738	2006	1653	2007	1725
2004 - 2006	1775	2007	1725	2008	1768
2005 - 2007	1797	2008	1768	2009	1822
2006 - 2008	1796	2009	1822	2010	1846
2007 - 2009	1860	2010	1846	2011	1840
2008 - 2010	1949	2011	1840	2012	1805
2009 - 2011	2012	2012	1805	2013	1788
2010 - 2012	1964	2013	1788	2014	1820

Test Beds and Test Environment

We tested our GP with Korean stocks.[2] Table 3.3 shows our data set and its division. The numbers of stocks are different across training, validation, and test periods due to listing and delisting. For a training period, the number of stocks is larger than those of its validation and test periods due to its longer time span. We trained our GP with three consecutive years and validated it with the immediately following year. The best solution in the validation year was tested in the next year. This process was shifted year by year.

For easier interpretations, we define $E_k(r)_S$ and $f(r)_S$ to be the percentages of returns as follows:

$$\begin{aligned} E_k(r)_S &= (E'_k(r)_S - 1) \times 100 \\ f(r)_S &= (f'(r)_S - 1) \times 100, \end{aligned} \tag{3.4}$$

where $E'_k(r)_S$ and $f'(r)_S$ are the expected earning rate of r after k trading days on period S and the fitness of pattern r on period S. The S can be one of training t, validation v, and the test d periods. The E and f are called *expected profit* and *fitness*, both in percentage return, respectively. Throughout this section, the experimental

[2]We excluded both funds and preferred stocks from our study. It is because the prices of common stocks are believed to have different characteristics from funds and preferred stocks due to their own nature.

results will be shown in terms of $E_k(r)_S$ and $f(r)_S$. The best pattern on a validation period will be denoted by r_b and the size of matching set by $R(r)_S$ where $S \in \{t, v, d\}$. The number of modules in the best pattern r_b is denoted by $|r_b|$. Note that the minimum number of matching cases restriction is removed when evaluating $f(r)_d$ since it is too restrictive for test period d.

All programs were written in C# language and run on Intel Xeon E5620 2.40 MHz with Windows Server 2008 R2 Datacenter. They were compiled using Visual Studio 2010 Ultimate.

Performance

Table 3.4 shows the experimental results. The profit rates and the set sizes of the matching cases were averaged over 30 independent runs. They are denoted by $\mu(\cdot)$ to represent averaging.

As shown in Table 3.4, our GP found highly attractive technical patterns with average profits higher than 4% on all the test years; the average profit over the entire test period was 8.69%. They are in fact quite high considering our trading horizon, i.e., 5 trading days. Overall, the profit rates were consistently high for all data sets. The ratio of average return by each test period to that by the corresponding training one, i.e., $\mu(f(r)_d)/\mu(f(r)_t)$, was 0.995 on average, which implies a strong generalization capability. In particular, four test years out of the eleven test ones showed even higher generalized returns than their corresponding training years, as shown in Figure 3.3 The matching cases, closely associated with trading opportunity, appeared at least several hundreds times for all the test years. On average, they appeared 553.30 times per year. On the bounds, the year 2006 had 328.9 occurrences and the year 2011 had 865.53 occurrences.

Interestingly, the patterns found by our GP were not very complex; they were mostly the combinations of gap up patterns, moving averages, and so on. This is quite notable in that several hundreds of module patterns were provided to our GP. Table 3.5 shows the node complexity, simply measured by counting the number of nodes in an individual, of solutions for all years. Although the minimum and maximum node complexities were varied over years, the medians were quite stable

Table 3.4: Average profit rates and the set sizes of matching cases for training, validation, and test periods

Year	$\mu(f(r)_t)$	$\mu(f(r)_v)$	$\mu(f(r)_d)$	$max(f(r)_d)$	$\mu(\|R(r)\|_d)$
2004	13.3	8.08	9.25	16.16	622.87
2005	8.06	8.7	13	17.61	788.87
2006	8.45	14.54	13.58	20.99	328.9
2007	11.77	15.16	13.85	18.91	348.4
2008	11.23	12.12	6.73	11.53	455.7
2009	11.99	5.87	6.91	11.23	792.67
2010	7.55	6.26	4.83	10.18	490.1
2011	8.44	7.32	7.97	11.69	865.53
2012	5.93	6.39	5.59	10.28	653.6
2013	6.68	6.27	6.3	10.89	389.17
2014	6.26	5.73	7.57	13.71	350.47
Average	9.06	8.77	8.69	13.93	553.30

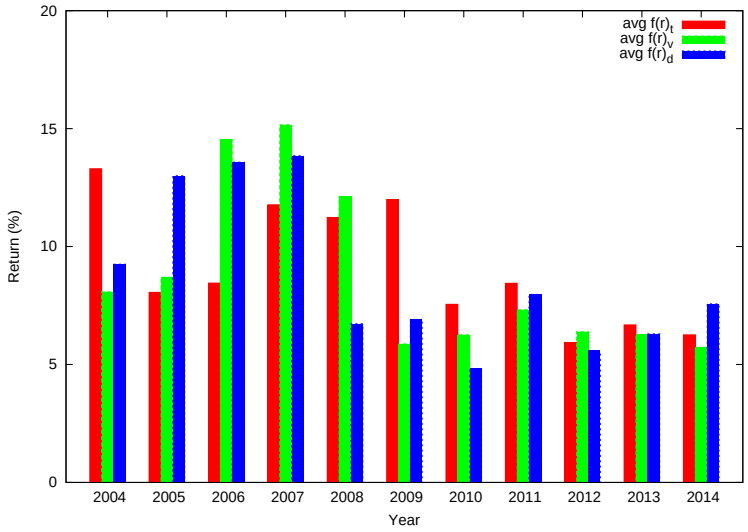


Figure 3.3: Comparison of returns between training, validation, and test periods

and less than 20 except for 2007, which are substantially lower than expected. It implies that several, not exceeding 4 or 5, module patterns were more effective than the others for our modular evolution.

Table 3.5: Node complexity of solution

Year	Minimum	Lower quartile	Median	Upper quartile	Maximum
2004	12	13	15	27	41
2005	7	13	18	25	31
2006	7	13	13	25	49
2007	11	15	27	27	44
2008	7	13	15	25	33
2009	7	13	13	27	55
2010	7	12	14	27	39
2011	11	13	13	23	33
2012	11	13	14	23	39
2013	7	13	14	23	33
2014	11	13	13	24	27

†Each value is rounded up or down to the nearest integers.

Comparison with Individual Modules

Since an individual module is a technical pattern as well, its performance can be evaluated by the fitness used in our GP. The performance of the individual modules is important since it can give insight into our genetic evolution.

Figure 3.4 shows the fitness of individual modules averaged over all years. Almost all modules showed slightly positive or negative fitness values, which are fairly close to zero; the average fitness of all modules was 0.02. Some modules, however, provided significantly large or small fitness values; most of them are related to gap up or down patterns. This implies that the final evolved patterns are likely to include some high-quality modules, which was empirically validated. It is thus important to select relevant or promising modules before a genetic run.

Figure 3.5 shows the maximum and average fitness values of the module set and our GP. The GP-evolved patterns showed larger maximum and average fitness values than those by the module set. Their significant improvement in performance is one of the key evidences for supporting the search effectiveness of our GP. It is also notable that the improvement factor begins to decrease from 2008, when the subprime mortgage crisis [DH09, DVH11] resulted in stock market crash.

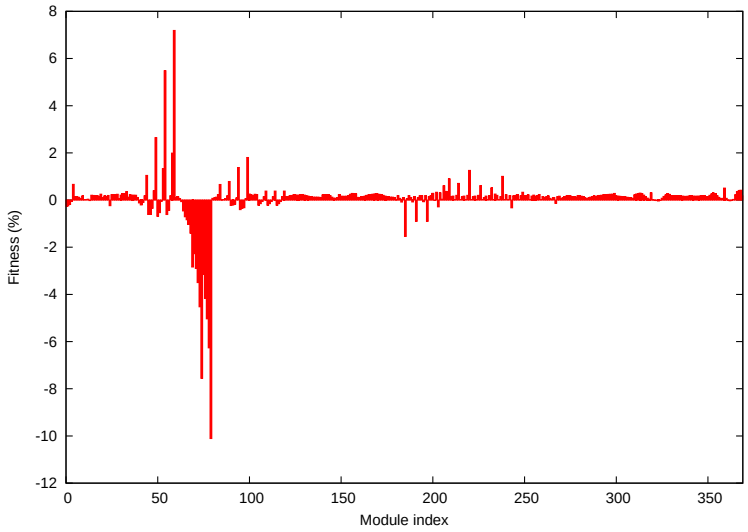


Figure 3.4: Fitness of individual modules

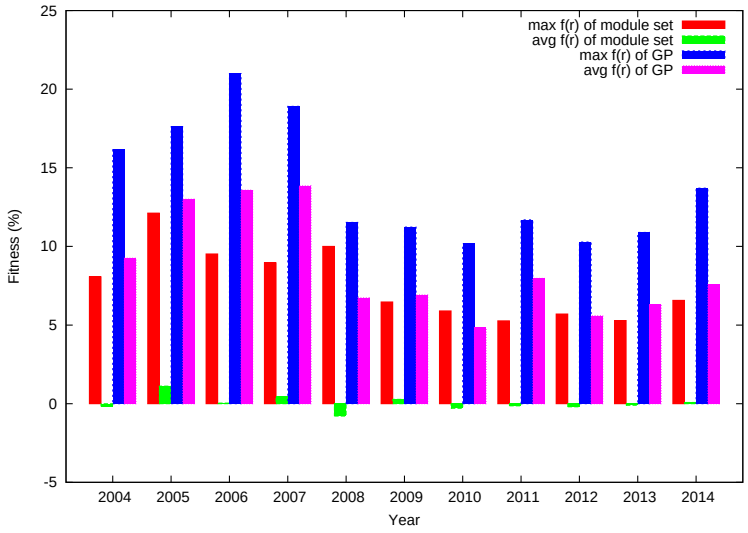


Figure 3.5: Comparison of fitness between module set and GP

Trading Simulation

We provided the final patterns to SimTrade, a simulation tool of Optus Investments Inc.[3] which is a company specialized in algorithm trading. Provided with a set of

[3]http://www.optus.co.kr

patterns, SimTrade simulates trading based on its own strategy starting with a given amount of money. The tax and transaction fee are parameters of SimTrade; we set them following the practice of the Korean stock market: 0.3% of tax per each sell and 0.015% of transaction fee per each sell or buy. SimTrade also has a parameter for the upper bound of daily trading volume; we restricted each trade not to exceed 2% of the total daily trading volume, which keeps one from overly affecting the price. We also used stop loss, or forceful liquidation of stock when its loss becomes greater than a certain level, to reflect one of the most important features in risky asset management.

Figure 3.6 shows the simulated returns by SimTrade with different initial cash and stop loss combinations. It showed that all combinations are profitable, but the simulated returns were degraded as the amount of cash increases. This is mainly due to the reduced trading amount enforced by the daily trading volume limit. Figure 3.7 shows that excess return, one deducted by KOSPI[4] return, can even become negative for too large initial cashes. However, the results are still fairly attractive; we found that the returns can be more improved by some ensembles with more attractive technical patterns.

Figure 3.8 shows accumulative normalized assets by our strategy and KOSPI. Our strategy used 500 million KRW and -10% as the initial cash and stop loss rate, respectively. One can see that we could obtain more consistent accumulative profits than the KOSPI index over the test years. This is notable in that our strategy with the attractive technical patterns shows less volatile as well as more profitable asset growth, even with the realistic limitations such as transaction cost and trading volume limit.

3.3.5 Summary

We proposed a modular genetic programming for finding attractive technical patterns. We defined the properties for attractive technical patterns and tried to find them using a new modular GP. Experimental results showed that our GP success-

[4]The KOSPI is a capitalization-weighted index of 200 big Korean stocks. It consists of over 70% of the total market value of the Korea Stock Exchange.

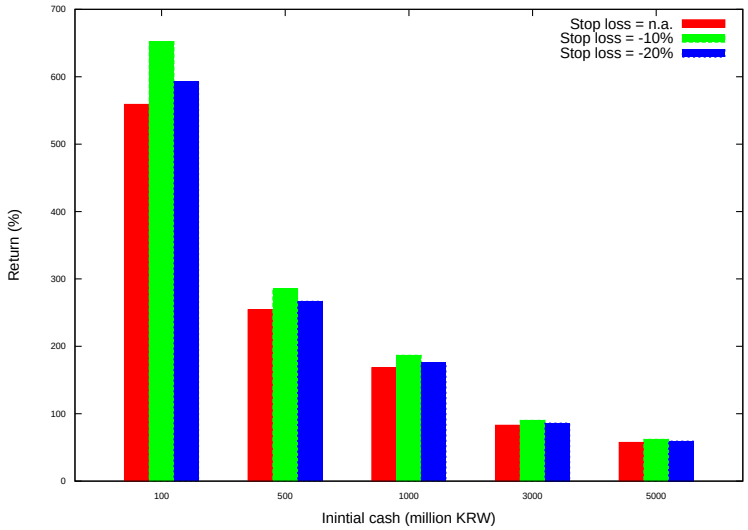


Figure 3.6: Simulated absolute returns using different initial cash and stop loss rate combinations

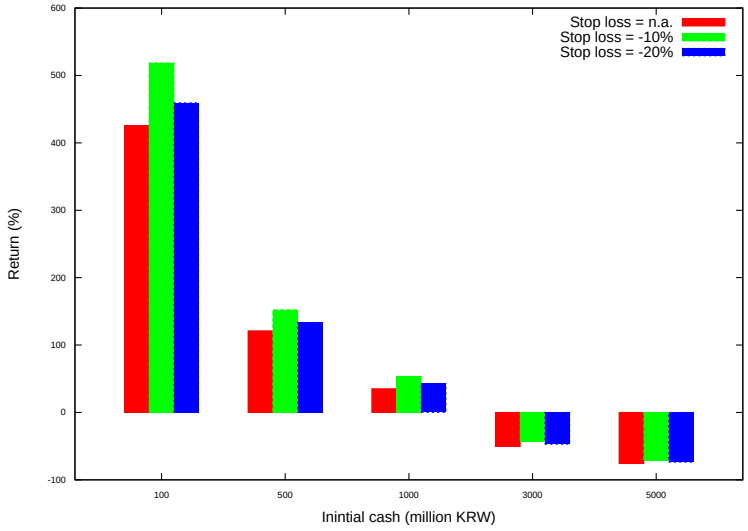


Figure 3.7: Simulated excess returns using different initial cash and stop loss rate combinations

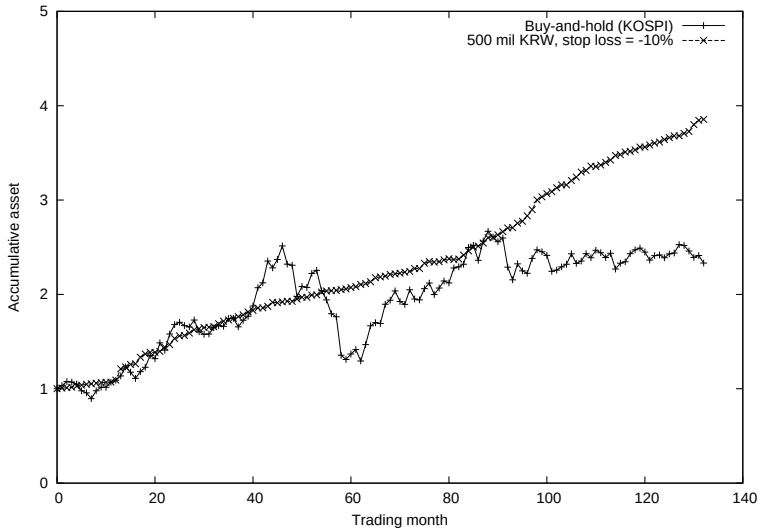


Figure 3.8: Accumulative normalized assets

fully found attractive technical patterns for the Korean stock market. It is notable that the patterns were not very complicated, which means that they are practical enough. The comparison with individual modules showed that our GP can find more profitable technical patterns than individual modules. We also ran a trading simulation tool based on the attractive technical patterns. The simulation resulted in significantly higher balances than the strategy following the KOSPI index fund.

Although our modular genetic programming found notable attractive technical patterns, we believe that there still remains considerable room for improvement. In terms of modular search space, we need more elaborate module patterns to incorporate more domain knowledge. Since there are a huge number of technical subpatterns that are known to work, the selection of module patterns is fairly important. In fact, it is another optimization problem as well.

Future work would include refining the definition of attractive technical patterns. For example, incorporating the risk concept such as Sharpe ratio [Sha66, Sha94] is a candidate. Finding attractive technical patterns is also multi-objective [CTAM09], where pareto optimization is a good appropriate approach.

Chapter 4

Grammatical Evolution

In the broadest context, grammatical evolution is to exploit a grammar for describing the search space. It is thus also called *grammar-based* or *grammar-guided* evolution, where a grammar provides type restrictions on the search space. Recently, it is further extended to grammatical evolution, where a mapping from genotype to phenotype by a grammar is exploited; it is quite similar to the biological process of protein synthesis. In this thesis, grammatical evolution is restricted to any evolution exploiting a grammar for type restrictions.

The grammar in grammatical evolution is considered to be a rule-based knowledge by type restrictions. Even the earliest studies in genetic programming (GP) have already emphasized that such type restrictions are important for solving complex problems. They enforced some syntactic structures on their candidate solutions, which guarantees that GP always produces syntactically valid solutions. The syntactic structures can be provided by some typing methods as well; strongly typed and lambda abstracted GPs are popular examples. Such typing methods for inputs and output for a function is, however, generally not enough to restrict the search space reasonably. They are frequently suffered from less meaningful inputs and output, which are not effectively used in the problem domain.

Grammatical evolution alleviates such problem by exploiting a grammar of describing the search space. Typically, it generates a population of initial solutions that are derived from the grammar. It then evolves them by genetic operators which

are also directed by the grammar. Only syntactically valid solutions are searched owing to the grammar during its evolution.

Designing effective grammar is one of the most important issues in grammatical evolution. Intuitively, the grammar should incorporate the key concept of domain knowledge in a succinct and unambiguous form. Context-free grammar is frequently used for the purpose since it is widely used for representing knowledge in machine learning. Typically, it is defined statically before a genetic run but it can also be evolved in the genetic run. If it is evolved at runtime, its productions are associated with selection probabilities which are adapted based on some statistic in the genetic run.

In this chapter, we propose a grammatically typed genetic programming for discovering attractive technical patterns. Typical genetic programming approaches for finding technical trading rules have optimized both technical pattern and trading strategy simultaneously. They have induced trading rules using common or popular technical patterns which have not been verified to be statistically sound. The strong coupling between technical pattern and trading strategy has been a major bottleneck for identifying attractive technical patterns. To this end, we try to find attractive technical patterns which can be exploited by diverse trading strategies or models. We describe the problem of finding attractive technical patterns and present a hybrid genetic programming incorporated with a grammatical type system. Extensive experimental results using all Korean stocks showed that our algorithm successfully found a number of attractive technical patterns. Trading simulations also showed that systematic trading strategies using the attractive technical patterns can outperform buy-and-hold.

4.1 Grammatical Type System

We use a context-free grammar $G = (V, \Sigma, P, S)$, where V, Σ, P, and S are the set of nonterminals, the set of terminals, the set of production rules, and the start symbol, respectively. Table 3.1 shows the set of terminal functions that are selected after some experiments. Figure 4.1 shows our context-free grammar G for describing technical

patterns. In the grammar, T is a relative offset designating where the technical pattern r is evaluated. For example, $MA_5(-1)$ returns the previous day's 5-day moving average. The nonterminals O_b, O_c, and O_a represent Boolean, comparison, and arithmetic operators, respectively.

Our context-free grammar is elaborately designed by a compromise between sensibility and generality. By sensibility, we mean that a technical pattern is syntactically and semantically correct. For example, consider a technical pattern $MA_5(0) > VMA_5(0)$. The pattern is syntactically correct because each function returns a real value. However, it is semantically incorrect since $MA_5(0)$ and $VMA_5(0)$ return price and volume, respectively. It is also hard to find out what is implied by the pattern.[1] Such semantically incorrect patterns are not generated by our grammar. By generality, we mean that our grammar allows new patterns that have not been popular. For example, $BLB(0) < 1.05 * p_o(-1) + BUB(0)$ is a valid pattern that can be derived by our context-free grammar. It is quite general since a typical Bollinger band (i.e., BLB or BUB) is compared only with the closing price.

Compared with other GPs using grammatical type systems [AOB10, KT10], our grammar has three notable features. First, it allows type compatible but complex patterns. Using the intermediate nonterminals (i.e., C_p, C_v, C_d, and C_s), which are called *group nonterminals*, the possible pattern forms are enforced to be type compatible. The C_v, for instance, restricts its possible functions to only F_v's which are then derived to only volume moving averages VMA_n's. The production rule $S_r \rightarrow S_r\ O_b\ S_r \mid C$ can build an indefinitely long and complex pattern as needed. In our GP, the indefinite derivation is implicitly blocked by decreasing the probability of selecting such derivation. Second, each literal is aware of its belonging group and it is then used in its appropriate context. Agapitos *et al.* [AOB10], for example, used a union of two literal sets for two different types of technical indicators. Their union set is inefficient since a literal can be compared with an inappropriate technical indicator. In contrast, our technical pattern can use its most meaningful literals by using the derivations from the group nonterminals. For example, $DISP_5$ and $STOF_{(5,3,3)}\%K$

[1] Wang [Wan00] managed to explain such pattern by large price movement with low volume. However, most studies do not allow patterns with incompatible comparisons [BPZ02].

use L_d and L_s, respectively. Finally, the grammar has some normalization effects [CM03, CM08]. The production $C_v \rightarrow C_{vl}\ O_c\ C_{vr}$, for example, designates the left tree C_{vl} and the right one C_{vr}. The C_{vl} is derived to a pattern containing an arithmetic operator while C_{vr} is not. It implies that only left subtrees contain arithmetic expressions. Every parse tree generated by our grammar is thus a normalized form, which enables us to reduce the search space effectively.

4.2 Hybrid Genetic Programming

We use a steady-state genetic programming, where the offspring replaces an individual in the same population.

- Representation and initialization

  Each individual is represented by a randomly generated tree with our context-free grammar G. The population size is set to 50. The minimum and maximum internal depths, considering only logical operator nodes, for creating the initial population are 0 and 2, respectively.

- Selection, crossover, and mutation

  The tournament selection [GDK91] is used. The crossover chooses two subtrees from parents at random and it swaps them. The cut points are restricted to Boolean and comparison operators. Crossover between different types of operators, e.g., Boolean operator between comparison one, is allowed. Such more free crossover produces more diverse trees than the strict typing among operators [AK99]. Mutation is performed by replacing a subtree chosen at random with a randomly generated one. It is also directed by our context-free grammar G.

- Replacement and stopping criterion

  We replace the worst individual on the training period. If the worst is the same as the best on the validation period, two individuals are randomly selected as replacement candidates. One of the replacement candidates is replaced with

$// V = \{S_r, C, C_p, C_{pl}, C_{pr}, F_p, C_v, C_{vl}, C_{vr}, C_d, C_{dl}, C_{dr},$
$//\quad C_s, C_{sl}, C_{sr}, F_{p1}, F_{p2}, F_{p3}, F_{p4}, L_p, F_v, L_v,$
$//\quad F_d, L_d, F_s, L_s, T, O_b, O_c, O_a\}$
$// \Sigma =$ all symbols excluding V
$// S =$ the start symbol
$// P =$ the set of production rules in the following

$S \rightarrow S_r\ O_b\ S_r$
$S_r \rightarrow S_r\ O_b\ S_r \mid C$
$C \rightarrow C_p \mid C_v \mid C_d \mid C_s$
$C_p \rightarrow C_{pl}\ O_c\ C_{pr}$
$C_{pl} \rightarrow L_p\ O_a\ F_p(T)$
$C_{pr} \rightarrow F_p(T)$
$F_p \rightarrow F_{p1} \mid F_{p2} \mid F_{p3} \mid F_{p4}$
$C_v \rightarrow C_{vl}\ O_c\ C_{vr}$
$C_{vl} \rightarrow L_v\ O_a\ F_v(T)$
$C_{vr} \rightarrow F_v(T)$
$C_d \rightarrow C_{dl}\ O_c\ C_{dr}$
$C_{dl} \rightarrow F_d(T)$
$C_{dr} \rightarrow L_d \mid F_d(T)$
$C_s \rightarrow C_{sl}\ O_c\ C_{sr}$
$C_{sl} \rightarrow F_s(T)$
$C_{sr} \rightarrow L_s \mid F_s(T)$
$F_{p1} \rightarrow p_o \mid p_c \mid p_h \mid p_l$
$F_{p2} \rightarrow MA_5 \mid MA_{10} \mid MA_{20} \mid MA_{60} \mid MA_{120} \mid MA_{250}$
$F_{p3} \rightarrow HP_5 \mid HP_{10} \mid HP_{20} \mid HP_{60} \mid HP_{120} \mid HP_{250} \mid$
$\quad LP_5 \mid LP_{10} \mid LP_{20} \mid LP_{60} \mid LP_{120} \mid LP_{250}$
$F_{p4} \rightarrow BUB \mid BLB$
$L_p \rightarrow 0.85 \mid 0.90 \mid 0.95 \mid 1.00 \mid 1.05 \mid 1.10 \mid 1.15$
$F_v \rightarrow VMA_5 \mid VMA_{10} \mid VMA_{20} \mid VMA_{60} \mid VMA_{120} \mid VMA_{250}$
$L_v \rightarrow 0.33 \mid 0.50 \mid 1.00 \mid 3.00 \mid 5.00$
$F_d \rightarrow DISP_5 \mid DISP_{10} \mid DISP_{20} \mid DISP_{60} \mid DISP_{120} \mid DISP_{250}$
$L_d \rightarrow 90 \mid 95 \mid 100 \mid 105 \mid 110$
$F_s \rightarrow STOF_{(5,3,3)}\%K \mid STOS_{(5,3,3)}\%K \mid STOS_{(5,3,3)}\%D \mid$
$\quad STOF_{(10,6,6)}\%K \mid STOS_{(10,6,6)}\%K \mid STOS_{(10,6,6)}\%D \mid$
$\quad STOF_{(20,12,12)}\%K \mid STOS_{(20,12,12)}\%K \mid STOS_{(20,12,12)}\%D \mid$
$\quad STOF_{(60,36,36)}\%K \mid STOS_{(60,36,36)}\%K \mid STOS_{(60,36,36)}\%D$
$L_s \rightarrow 20 \mid 25 \mid 30 \mid 70 \mid 75 \mid 80$
$T \rightarrow -10 \mid -9 \mid -8 \mid -7 \mid -6 \mid -5 \mid -4 \mid -3 \mid -2 \mid -1 \mid 0$
$O_b \rightarrow \wedge \mid \vee$
$O_c \rightarrow = \mid < \mid >$
$O_a \rightarrow + \mid - \mid \times \mid \div$

Figure 4.1: Context-free grammar for technical patterns

a probabilistic bias toward worse individuals. Our GP stops if the number of consecutive fails to update the best solution reaches 300 or the number of generations exceeds 1,000.

- Local optimization

We devise a local optimization using two different neighborhoods. Figure 4.2 shows a neighborhood search by node change. It defines a neighborhood of a tree by replacing a node with compatible ones, which is a single node version of shaking search [MHF08]. The solutions in the neighborhood are examined by an approximate fitness function $\hat{f}$[2] which is evaluated on a sample of 30 random companies. The approximate fitness function helps significantly reducing the running time of fitness evaluations. The neighborhood size is determined by N_d and the entire search process is directed by the direction of maximal gain in fitness, i.e., fitness differential.

Figure 4.3 shows a neighborhood search by clause change. It is quite similar to the one by node change, but uses a larger unit, i.e., clause, to generate neighbor solutions. It uses the neighborhood size determined by N_c and its search process is also directed by the fitness differential.

Figure 4.4 shows our local optimization using these neighborhood searches.[3] It applies the two different neighborhood searches sequentially until no gain is obtained from any of them or the maximum trial count, T_{max}, is reached. We set all N_d, N_c, and T_{max} to 3 after some experiments.

- Best update and elitism

  After the initialization of population, the best individual on the training period is evaluated on its validation one. Its fitness on the validation period is then saved as the best fitness. Whenever a better solution on the training period is found, its fitness is evaluated on its validation one. If it also improves upon the best fitness, it is accepted as the new best individual. This best update scheme has been extensively used in the literature [AK99].

  Elitism [DeJ75] is also used to further improve upon our GP. The best indi-

[2]In the approximate fitness, the minimum set size, β, should be adjusted to a smaller value, reflecting the ratio of sample size to the number of all companies. In the fitness approximations literature, there have been a huge number of techniques. The interested reader is referred to [Jin05].

[3]This local optimization using sub-layer heuristics can be regarded as a special case of hyper-heuristics [Kam13]. Dynamic selection by the performance of each sub-layer heuristic is a candidate for further improvements.

Input:
τ: parse tree for the technical pattern r
n: a node in the tree τ
$\hat{f}_o$: approximate fitness of the original tree
N_d: neighborhood size
S: the set of sampled companies for approximate fitness
$\hat{f}(\tau)$: approximate fitness of the tree τ
$\Gamma(p)$: the set of possible terminal symbols generated by all productions whose left symbol is p
Output:
T: a structure containing τ and g_{max}
Description:
NSNC(τ,n,$\hat{f}_o$,N_d,S)
{
    $g_{max} \leftarrow 0$;
    $n_o \leftarrow n$;
    $n_{max} \leftarrow n$;
    for $i \leftarrow 1$ **to** N_d
        if $n \in \Gamma(O_i)$ ($i = b$ or c or a)
            $n \leftarrow$ a random operator $\in \Gamma(O_i)$;
        else if n is a literal and its parent is a function
            $n \leftarrow$ a random offset literal $\in \Gamma(T)$;
        else if n is a function and $n \in \cup_{i=p1}^{p4}\Gamma(F_i)$
            $j \leftarrow$ a random symbol $\in \{p1, p2, p3, p4\}$;
            $n \leftarrow$ a random symbol $\in \Gamma(F_j)$;
        else if n is a function and $n \in \Gamma(F_i)$ ($i = v$ or d or s)
            $n \leftarrow$ a random symbol $\in \Gamma(F_i)$;
        else
            $k \leftarrow$ a nonterminal ($\in \{C_p, C_v, C_d, C_s\}$) that generated the literal n;
            if $k = C_i$ ($i = p$ or v or d or s)
                $n \leftarrow$ a random symbol $\in \Gamma(L_i)$;
        $g \leftarrow \hat{f}(\tau) - \hat{f}_o$;
    if $g > g_{max}$
        $g_{max} \leftarrow g$;
        $n_{max} \leftarrow n$;
    $n \leftarrow n_o$;
    if $g_{max} > 0$
        $n \leftarrow n_{max}$;
    $T.g_{max} \leftarrow g_{max}$;
    $T.\tau \leftarrow \tau$;
    return T;
}

Figure 4.2: Neighborhood search by node change

vidual is replaced only if it performs worse than the new best individual on the validation period.

Input:
τ: parse tree for the technical pattern r
n: a node in the tree τ
$\hat{f}_o$: approximate fitness of the original tree
N_c: neighborhood size
S: the set of sampled companies for approximate fitness
$\hat{f}(\tau)$: approximate fitness of the tree τ
$\Gamma(p)$: the set of possible terminal symbols generated by all productions whose left symbol is p
Output:
T: a structure containing τ and g_{max}
Description:
NSCC(τ,n,$\hat{f}_o$,N_c,S)
{
    $g_{max} \leftarrow 0$;
    $n_o \leftarrow n$;
    $n_{max} \leftarrow n$;
    for $i \leftarrow 1$ **to** N_c
        $n \leftarrow$ a random operator $\in \Gamma(O_c)$ (subtree's root);
        $j \leftarrow$ a random group nonterminal subscript $\in \{p, v, d, s\}$;
        $n.left \leftarrow$ a subtree generated by the symbol C_{jl} as the start symbol;
        $n.right \leftarrow$ a subtree generated by by the symbol C_{jr} as the start symbol;
        $g \leftarrow \hat{f}(\tau) - \hat{f}_o$;
        if $g > g_{max}$
            $g_{max} \leftarrow g$;
            $n_{max} \leftarrow n$;
        $n \leftarrow n_o$;
    if $g_{max} > 0$
        $n \leftarrow n_{max}$;
    $T.g_{max} \leftarrow g_{max}$;
    $T.\tau \leftarrow \tau$;
    return T;
}

Figure 4.3: Neighborhood search by clause change

4.3 Extended Attractive Technical Pattern Discovery

4.3.1 Introduction

Trading with technical patterns [Nis91, EMB07, Mur99, LMW00] has been extensively used for diverse markets including stock markets, foreign exchange markets, and so on. It tries to predict short-term price trends typically using price and volume information [Mor92]. Chart patterns [LMW00], candlesticks [Mor92, LJ99, MYR06], and technical indicators [Kau98] are well-known building blocks for technical trading.

The technical pattern studies can be classified into two distinctive groups: sta-

Input:
τ: parse tree for the technical pattern r
n: a node in the tree τ
N_d: neighborhood size for NSNC()
N_c: neighborhood size for NSCC()
T_{max}: maximum number of trials
$\hat{f}_o$: approximate fitness of the original tree
$DFS(\tau)$: a sequence of nodes in τ by the depth first search
$DFS'(\tau)$: a sequence of comparison or logical operator nodes in $DFS(\tau)$
$\Gamma(p)$: the set of possible terminal symbols generated by all productions whose left symbol is p
S: the set of sampled companies for approximate fitness
Output:
T: a structure containing τ and g_{max}
Description:

```
LocalOptimization(τ,n,N_c,N_d,T_max,S)
{
    i ← 0;
    g_n ← 0;
    g_c ← 0;
    while i < T_max and g_n ≥ 0 and g_c ≥ 0
        g_n ← 0;
        f̂_o ← f̂(τ);
        foreach node n in DFS(τ)
            T ← NSNC(τ,n,f̂_o,N_d,S);
            τ ← T.τ;
            g_n ← g_n + T.g_max;
        g_c ← 0;
        f̂_o ← f̂(τ);
        foreach node n in DFS'(τ)
            T ← NSCC(τ,n,f̂_o,N_c,S);
            τ ← T.τ;
            g_c ← g_c + T.g_max;
        i ← i + 1;
    return T;
}
```

Figure 4.4: Local optimization

tistical analysis approach and evolutionary approach. The main objective of the statistical analysis approach is to find out whether technical patterns have statistical predictive power or not. For example, Alexander [Ale61] examined filter rules which advise a trader to buy or sell if price moves out of a given filtered range; the filter rules were found to be almost unprofitable for Dow Jones Industrial Average (DJIA) stocks after adjusting transaction costs [FB66]. There have also been other studies using famous technical indicators or candlestick patterns, which failed to find profitable trading rules [VHP68, CKH08, MYC08, CTAM09]. These negative results including the

famous *efficient market hypotheses* (EMH) [Fam70] have resulted in the widespread skepticism about the usefulness of technical trading rules [PI07]. However, some recent studies found evidences against EMH [DT85, JT01, GMS$^+$09, EM11], which makes it harder to reach consensus among researchers.

Most early statistical studies are limited in that they have *data snooping biases* [JB70, LM90]. Data snooping bias is a statistical bias by *ex post* selection of technical trading rules. It states that profitable technical trading rules can be identified by extensive mining with past data but they are likely to have little predictive power. For example, Brock *et al.* [BLL92] found a set of profitable technical trading rules but they were not able to eliminate the data snooping bias. To alleviate the data snooping bias, new statistical tests called *reality checks* [Whi00, Han05] have been developed. They tested a number of technical trading rules with a full universe of parameterizations on *stationary bootstrap* [PR94] time series. They showed that the profitability of most popular technical trading rules are attributable to data snooping bias.

Recently, evolutionary approaches such as genetic programmings (GPs) [Koz92, BNKF98] have been extensively used for finding technical trading rules. Allen and Karjalainen [AK99] applied GP to find profitable technical trading rules for S&P 500 index, which is the first evolutionary approach in the literature. They used a small set of terminals and excess return over buy-and-hold including risk-free interest rate for no position as the fitness function. The data snooping bias was relieved by incorporating validation sets. Their results reported almost no excess return over buy-and-hold, which was confirmed by other studies [Nee03, FMT05, HLV10] with a few exceptions [YCK05]. Some studies using individual stocks [PSV04] rather than composite indexes also failed to find profitable technical trading rules. Interestingly, most GPs for other markets such as foreign exchange markets [NWD97, DJ01, BPZ02] have been successful. Wang [Wan00] suspected that the conflicting results might be due to different transaction costs, base returns, or testing methods for various markets [YCK05]. However, it has not been validated which factors are really affecting the conflicting results across different markets.

Most GPs have tried to find profitable technical trading rules using a set of simple

technical patterns. They typically encoded a solution with patterns for either buy or sell. It implies that the ordinary encoding optimizes both technical pattern and trading strategy simultaneously. It is thus not surprising that most studies struggled with increased complexity and they were not able to find profitable technical trading rules; some of them even replicated buy-and-hold or its variants [Wan00, DJ01]. The strong coupling between technical pattern and trading strategy also prevents us from exploiting the patterns for other trading strategies or models.

To find profitable, statistically sound, and more exploitable technical patterns, Lee and Moon [LM10] formulated attractive technical patterns. Using the formulation, they tried to find attractive technical patterns consisting of predefined module patterns using GP. Trading strategies using the patterns showed they can outperform buy-and-hold. However, their strong modularization restricts the search space into a considerably limited region consisting of some predefined module patterns. The preparation of module patterns also requires extensive domain knowledge which is likely to have strong selection biases. These motivate us to devise a GP without the predefined module patterns.

4.3.2 Problem Formulation

We extend the formulation of attractive technical pattern [LM10] to include more meaningful features.

Let R be the set of ordered pairs (i, j) where i and j represent indices for company and trading day, respectively. The indices i and j are nonnegative integers whose upper bounds are determined by the data set.

A *technical pattern* r is an element of $L(G)$ by our context-free grammar $G = (V, \Sigma, P, S)$ as follows:

$$L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}, \tag{4.1}$$

where * and $\overset{*}{\Rightarrow}$ represent the *Kleen star* operation and repetitive production applications, respectively.

The *matching set* of a technical pattern r is defined by

$$R(r) = \{(i,j) \in R : r \text{ is evaluated to be true for } (i,j)\}. \tag{4.2}$$

Each $(i,j) \in R(r)$ is called a *matching case* for r. The *matching set by date* that depends only on a calendar date d is defined as follows:

$$R_d = \{(i,j) \in R : (i,j)\text{'s calendar date} = d\}. \tag{4.3}$$

In a matching set, most of the matching cases can occur at only a few number of dates by sudden market crashes or rallies. If it is the case, any statistic on the matching set is biased due to the large number of matchings for such dates. To filter out the effects, we define the *filtered matching set* by

$$R^f(r) = \{(i,j) \in R(r) : |R_{\delta(i,j)} \cap R(r)\ | \leq \alpha |R(r)|\}, \tag{4.4}$$

where $\delta(i,j)$ is a function that maps a matching case (i,j) into its corresponding calendar date and $0 < \alpha \leq 1$ is a predefined constant.

For a given matching set $M(\subset R)$, the *expected return* after k trading days is defined by

$$E_k(M) = \frac{1}{|M|} \sum_{(i,j)\in M} \left(\frac{p_c(i,j+k)}{p_c(i,j)} - 1 \right), \tag{4.5}$$

where $p_c(i,j)$ is the closing price of company i on trading day j.

The k^{th} element of fitness function $f_k(r)$ is defined as follows:

$$\begin{array}{rcl} f_k(r) & = & \frac{1}{|R^f(r)|} \sum_{(i,j)\in R^f(r)} \left(\frac{p_c(i,j+k)}{p_c(i,j)} - 1 - E_k(R_{\delta(i,j)}) \right) \\ & = & E_k(R^f(r)) - \frac{1}{|R^f(r)|} \sum_{(i,j)\in R^f(r)} E_k(R_{\delta(i,j)}). \end{array} \tag{4.6}$$

The $E_k(R_{\delta(i,j)})$ represents the expected return of all companies on the day $\delta(i,j)$ after k trading days. It is called the *base expected return* for the day $\delta(i,j)$. By subtracting the base expected return from $\frac{p_c(i,j+k)}{p_c(i,j)} - 1$, $f_k(r)$ measures the average *relative expected return* compared to all companies after k trading days. The *fitness vector* $\vec{f^r}$ is defined by a n-dimensional vector whose element is $f_k(r) + 1$.

To enforce a constraint of the number of yearly matching cases, we define the *product* operator $\otimes$ by

$$W \otimes y = \{(i,j) \in W : y = \varphi(\delta(i,j))\}, \tag{4.7}$$

where W is a matching set and $\varphi(d)$ is the function returns the year of a calendar date d.

To alleviate possible daily fluctuations of stock prices, the fitness function f is defined as follows:

$$f(r) = \begin{cases} \frac{1}{||w||}\vec{w} \cdot \vec{f^r} - 1 & \text{if } |R^f(r) \otimes y| \geq \beta \text{ for all } y \in Z \\ \eta & \text{otherwise,} \end{cases} \tag{4.8}$$

where Z, $\vec{w}$, β, and η are $Z = \{z : \varphi(\delta(i,j)) = z \text{ for all } (i,j) \in R\}$, a weight vector $\vec{w} = (w_1, w_2, \ldots, w_n)$, a constant for the minimum set size, and a constant for the minimum fitness, respectively. More intuitively, $f(r)$ is a weighted average of $\{f_1(r), f_2(r), \ldots, f_n(r)\}$ with the weight $\{w_1, w_2, \ldots, w_n\}$.

4.3.3 Experimental Results

Parameter Settings

We set the constant α for $R^f(r)$ and n for $f(r)$ to 0.05 and 5, respectively. The β for $f(r)$ is set to 200. There are several candidates for choosing the weight vector $\vec{w}$ whose element w_k gives an weight to $f_k(r)$. We choose five representative weight models as follows:

- Equal (EQ) : $w_k = \frac{1}{n}$
- Linearly Decreasing (LD): $w_k = \frac{n-k+1}{\sum_{i=1}^{n} n-i+1}$
- Linearly Increasing (LI): $w_k = \frac{k}{\sum_{i=1}^{n} i}$
- Exponentially Decreasing (ED): $w_k = \frac{e^{n-k+1}}{\sum_{i=1}^{n} e^{n-i+1}}$
- Exponentially Increasing (EI): $w_k = \frac{e^k}{\sum_{i=1}^{n} e^i}$

where e is the natural logarithm constant. The η in $f(r)$ is set to -1.0 which is small enough for the invalid fitness.

Test Beds and Test Environment

We tested our GP with daily data of Korean stocks[4] including all listed and delisted stocks[5] from January 1999 to December 2014.

To alleviate the data snooping bias, we divided the data into nine overlapping sequences and used the *rolling forward method* [PT95]. For each sequence, three consecutive years[6] were used as the training period and the next year as the validation one. The year following the validation period was used as the test year.

For the complete availability of all technical indicators, a period preceding the training one, called *windup period* [Kau98], is also necessary. The windup period is needed since the calculations of n-day technical indicators require the previous n trading days. By rolling forward, the first sequence, for instance, uses 1999, 2000-2002, 2003, and 2004 as the windup, training, validation, and test periods, respectively; this process was shifted year by year. We performed 30 runs for each sequence. In later sections, we use subscripts t, v, and d for training, validation, and test periods, respectively. For example, the training fitness of a technical pattern r is denoted by $f(r)_t$.

It should be noted that a form of bias can be included unless we carefully filter out some subperiod from training, validation, and test periods. Assume that a matching case (i, j) is included in $R^f(r)$ and its corresponding date $\delta(i, j)$ is the last trading day of 2002 in the training period 2000-2002. In calculating $f_k(r)$, the term $p_c(i, j+k)$ then uses some data in 2003 which are beyond the scope of the training data. Exactly the same problem also occurs at the ends of its validation and test periods. To this end, we removed the last month from each training, validation, and test period.

[4] We excluded funds, preferred stocks, and convertible ones from our study. It is because the prices of common stocks are believed to have different characteristics from funds and preferred stocks due to their own nature. Convertible stocks are typically very short lived, which is not meaningful for investments. All data were provided by Optus Investments Inc.

[5] The inclusion of delisted stocks is important to eliminate *survivorship bias* [GT92, EGB96].

[6] In general, the length of training period also affects the characteristics of technical patterns [MB04].

Another problem arises when we calculate fitness on test period. The $|R(r)|$ is required for constructing $R^f(r)$, but it is not available until the end of a given test period. It is thus impossible for the fitness on test period to use $R^f(r)$. In our GP, we used $R(r)$ instead of $R^f(r)$ in calculating $f_k(r)$ on test period.

These problems are closely related in that they require the unavailable future data from a given trading day. We call such problems *look-ahead biases*.[7] The look-ahead bias is often hard to be identified if it appears in a complex pattern or function.

All programs were written in C# language and run on Intel Xeon E5620 2.40 MHz with Windows Server 2008 R2 Datacenter. They were compiled using Visual Studio 2010 Ultimate.

Comparisons among Different Weight Models

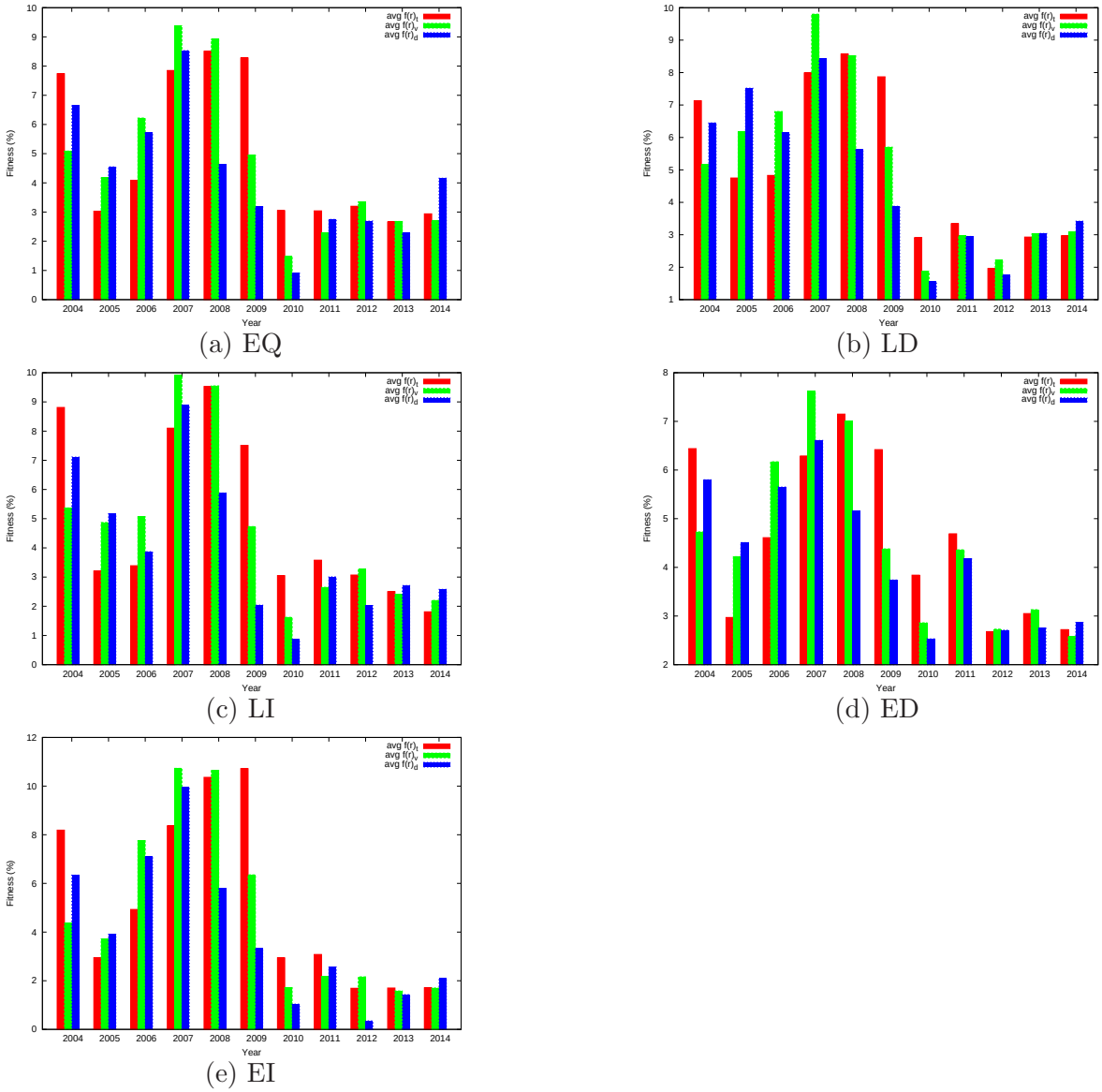
Figure 4.5 shows the fitness values using different weight models: EQ, LD, LI, ED, and EI. They cannot be compared with each other due to their different calculations. However, they showed that the weight models produce quite generalized results, which was confirmed by high values of the generalization factor $\mu(f(r)_d)/\mu(f(r)_t)$. The generalization factor averaged over all years ranged from 0.82 in EI weight model to 0.96 in ED one. Figure 4.6 shows the generalization factors of different weight models for all years. It shows that LD and ED have comparable generalization factors to the others before the year 2008 but they obtain higher ones thereafter. Since both LD and ED give more weights on the near future, it implies that shorter term technical patterns are more consistently identified since 2008.

To find out which weight model is the best, we provided the final best patterns to SimTrade,[8] a commercial simulation tool of Optus Investments Inc.[9] which is an investment company specialized in algorithm trading. The SimTrade begins with a

[7] In the finance literature, it is often referred to as survivorship bias which is caused by ex post selection of survived stocks [HNV01]. However, we think that this term should be extended to include any ex post actions to financial data. In the most strict sense, Yeh *et al.* [YLT11] even discriminated the model design stage from the model validation ones.

[8] We are not able to present its detailed algorithm by business contract. For the same reason, we also do not provide the found patterns explicitly. However, the interested readers can replicate our results using popular asset allocation algorithms [OLLZ05].

[9] http://www.optus.co.kr



(a) EQ

(b) LD

(c) LI

(d) ED

(e) EI

Figure 4.5: Fitness values using different weight models

given amount of the initial cash. It uses the best pattern for each year and switches it to that for the next year after the end of each year. With the patterns, it simulates trading by its own strategy with a number of parameters. The parameters also

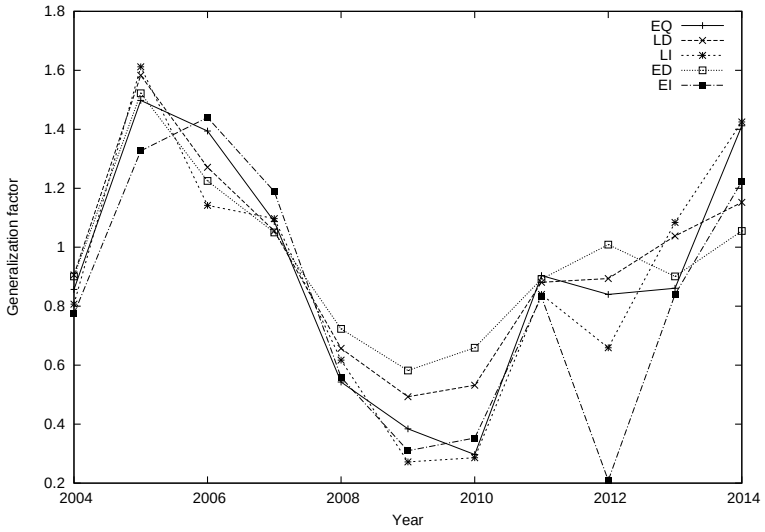


Figure 4.6: Generalization factors of different weight models

include the tax and transaction fee; we set them following the practice of the Korean stock market: 0.3% of tax per each sell and 0.015% of transaction fee per each sell or buy. Compared with common trading simulations without the upper bound of trading volume [PSV04, WYC12], the SimTrade has a parameter for the upper bound of trading volume; the daily accumulative trading volume is restricted up to 2% of the total daily trading volume. The restriction on daily trading volume is quite important to obtain realistic simulation results. It is because one cannot trade a huge amount of volumes without affecting the price.

Table 4.1 and 4.2 show the simulated absolute and excess returns using the different weight models with the SimTrade, respectively. The excess return was calculated by deducting buy-and-hold[10] return from the simulated absolute one. The EI weight model outperformed the other ones across diverse parameters. Both absolute and excess returns were degraded as the amount of the initial cash increases, which is mainly due to the daily trading volume limit. Hereafter, we describe the experimen-

[10]In this section, buy-and-hold return was calculated by KOSPI which is a capitalization-weighted index of 200 large Korean stocks. During the entire period, i.e., from January 2004 to December 2014, buy-and-hold return was 133.25%.

Table 4.1: Simulated absolute returns using different weight models

Parameter		EQ	LD	LI	ED	EI
Initial cash	Stop loss rate	Return	Return	Return	Return	Return
100	n.a.	686.20	627.30	554.64	604.94	**736.03**
500	n.a.	315.17	294.43	272.21	286.02	**337.45**
1000	n.a.	209.01	197.39	186.96	194.42	**225.46**
3000	n.a.	99.37	96.85	93.91	95.96	**109.95**
5000	n.a.	67.84	67.23	66.46	65.68	**75.49**
100	-10	897.18	796.76	711.68	766.97	**919.25**
500	-10	387.22	356.24	328.99	348.35	**401.70**
1000	-10	249.14	231.83	219.43	230.45	**262.39**
3000	-10	114.66	109.39	106.17	109.53	**122.60**
5000	-10	77.90	75.12	74.02	74.19	**83.20**
100	-20	773.60	698.46	618.29	671.64	**816.48**
500	-20	346.12	321.93	294.38	313.54	**366.94**
1000	-20	226.19	212.16	198.88	210.20	**242.27**
3000	-20	105.70	101.85	97.66	101.81	**115.79**
5000	-20	71.78	70.27	68.53	69.30	**78.89**

The best pattern on validation period among 30 runs was selected for each test year.
Bold-faced number represents the best return.
Monetary unit for initial cash is million KRW.
Returns and stop loss rates were written in percentage.

tal results only by the best weight model, i.e., the EI weight model, unless otherwise noted explicitly.

Features of Attractive Technical Patterns

Table 4.3 shows the fitness values for all years. On average, both training and validation periods showed about 5 percent fitness values. The fitness was degraded on the test period to about 4 percent which is still very close to those of the training and validation periods. Table 4.4 shows the sizes of matching sets for all years. The size of matching set varied with year; it ranged from 485.73 in 2008 to 24283.57 in 2010. The average size of the matching sets was 7263.93, which means 29.06 matching companies per day assuming each year has 250 trading days. It is quite adequate in that common investors trade at most tens of companies per day.[11] This is be-

[11] Even with longer trading horizons, it is common to restrict the number of companies in a portfolio to at most 20 or 30 [ZYH+06, HCCC12].

Table 4.2: Simulated excess returns using different weight models

Parameter		EQ	LD	LI	ED	EI
Initial cash	Stop loss rate	Return	Return	Return	Return	Return
100	n.a.	552.95	494.05	421.38	471.69	**602.78**
500	n.a.	181.92	161.18	138.96	152.77	**204.20**
1000	n.a.	75.76	64.14	53.71	61.17	**92.21**
3000	n.a.	-33.88	-36.40	-39.34	-37.29	**-23.30**
5000	n.a.	-65.41	-66.02	-66.79	-67.57	**-57.76**
100	-10	763.93	663.51	578.43	633.72	**786.00**
500	-10	253.97	222.99	195.74	215.10	**268.45**
1000	-10	115.89	98.58	86.18	97.20	**129.14**
3000	-10	-18.59	-23.86	-27.08	-23.72	**-10.65**
5000	-10	-55.35	-58.13	-59.23	-59.06	**-50.05**
100	-20	640.35	565.21	485.04	538.39	**683.23**
500	-20	212.87	188.68	161.13	180.29	**233.69**
1000	-20	92.94	78.91	65.63	76.95	**109.02**
3000	-20	-27.55	-31.40	-35.59	-31.44	**-17.46**
5000	-20	-61.47	-62.98	-64.72	-63.95	**-54.36**

The best pattern on validation period among 30 runs was selected for each test year.
Bold-faced number represents the best return.
Monetary unit for initial cash is million KRW.
Returns and stop loss rates were written in percentage.

Table 4.3: Results on fitness values

Year	$\mu(f(r)_t)$	$\mu(f(r)_v)$	$\mu(f(r)_d)$	$max(f(r)_d)$	$\sigma(f(r)_t)/\sqrt{n}$	$\sigma(f(r)_v)/\sqrt{n}$	$\sigma(f(r)_d)/\sqrt{n}$
2004	8.19	4.36	6.34	17.34	0.90	0.64	0.90
2005	2.95	3.72	3.91	16.85	0.55	0.79	1.12
2006	4.93	7.76	7.10	21.49	0.70	1.16	1.24
2007	8.37	10.73	9.95	16.30	0.73	0.99	0.78
2008	10.36	10.64	5.79	15.57	1.01	0.99	0.84
2009	10.72	6.35	3.32	9.43	1.24	0.89	0.72
2010	2.94	1.72	1.04	8.15	0.64	0.41	0.45
2011	3.08	2.17	2.56	12.11	0.72	0.51	0.66
2012	1.69	2.16	0.35	7.69	0.33	0.41	0.44
2013	1.70	1.58	1.43	6.76	0.43	0.28	0.35
2014	1.71	1.70	2.09	15.17	0.37	0.31	0.59
Average	5.15	4.81	3.99	13.35	0.69	0.67	0.74

For each year, fitness values were averaged over 30 runs.
The subscripts t, v, and d represent training, validation, and test periods, respectively.
$\mu(\cdot)$ and $\sigma(\cdot)/\sqrt{n}$ represent average and group standard deviation of n runs, respectively.
All fitness values were written in percentage.

Table 4.4: Results on sizes of matching sets

Year	$\mu(\|R(r)\|_d)$	$\sigma(\|R(r)\|_d)/\sqrt{n}$	$max(\|R(r)\|_d)$	$min(\|R(r)\|_d)$
2004	3588.63	2394.38	72667	211
2005	3301.23	1912.76	58255	66
2006	879.03	150.26	3471	210
2007	974.33	356.72	11216	219
2008	485.73	119.08	3280	100
2009	1739.57	852.57	26288	298
2010	24283.57	10846.27	251408	186
2011	14040.27	8809.81	204562	133
2012	20188.97	10590.06	250522	251
2013	9021.5	7363.33	221914	151
2014	1400.4	326.25	8517	148
Average	7263.93	3974.68	101100	179.36

For each year, sizes of matching sets were averaged over 30 runs.
The subscript d represents test period.
$\mu(\cdot)$ and $\sigma(\cdot)/\sqrt{n}$ represent average and group standard deviation of n runs, respectively.

Table 4.5: Statistical significance by t-test

Period			$f(r)_t$		$f(r)_v$		$f(r)_d$	
Training	Validation	Test	t-value	p-value	t-value	p-value	t-value	p-value
2000-2002	2003	2004	9.094	< 0.000001	6.859	< 0.000001	7.056	< 0.000001
2001-2003	2004	2005	5.372	0.000005	4.688	0.000030	3.504	0.000754
2002-2004	2005	2006	7.035	< 0.000001	6.687	< 0.000001	5.745	0.000002
2003-2005	2006	2007	11.485	< 0.000001	10.796	< 0.000001	12.723	< 0.000001
2004-2006	2007	2008	10.251	< 0.000001	10.69	< 0.000001	6.889	< 0.000001
2005-2007	2008	2009	8.638	< 0.000001	7.118	< 0.000001	4.603	0.000038
2006-2008	2009	2010	4.612	0.000037	4.183	0.000121	2.306	0.014220
2007-2009	2010	2011	4.296	0.000089	4.282	0.000093	3.909	0.000256
2008-2010	2011	2012	5.075	0.000010	5.237	0.000007	0.802	0.214503
2009-2011	2012	2013	3.933	0.000240	5.597	0.000002	4.13	0.000141
2010-2012	2013	2014	4.599	0.000039	5.462	0.000004	3.539	0.000689

cause too large number of trades degrade excess return; they also increase implicit management cost for trading. In practice, the number of daily trades is restricted to manageably small even with automated trading executions.

To validate our results statistically, we conducted t-test to the fitness values on training, validation, and test periods. The null and alternative hypotheses are H_0: $\mu(f(r)) = 0$ and H_1: $\mu(f(r)) > 0$, respectively. The statistical results by one-

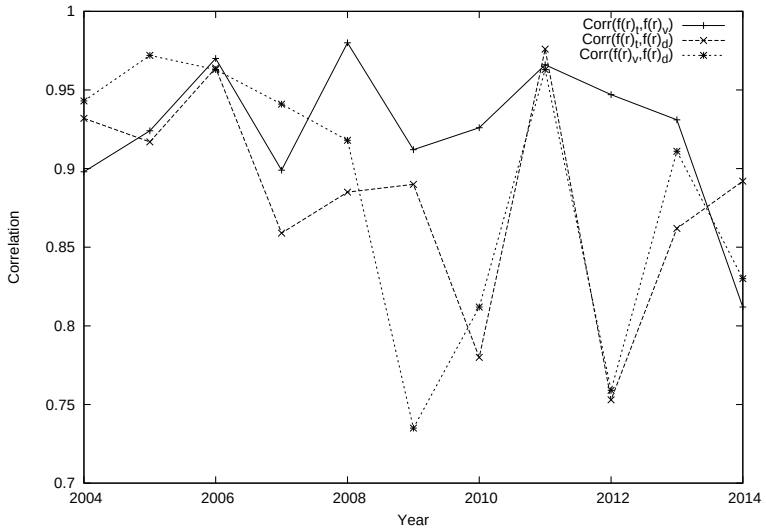


Figure 4.7: Fitness correlations between data sets

tailed t-test are shown in Table 4.5. They showed that all fitness values, except for $f(r)_d$ in 2012, are statistically significant with $p < 0.05$.

Figure 4.7 shows the fitness correlations between data sets. The fitness values for training, validation, and test periods are denoted by $f(r)_t$, $f(r)_v$, and $f(r)_d$, respectively. The figure shows that our GP performed consistently on the tested years. On the bounds, the correlation between the training period and the test one was 0.75 in 2012 and 0.98 in 2011. It implies that if our GP can improve upon a solution on training period, it is likely to obtain a better solution on test period. However, the high correlation does not necessarily mean that more profitable technical patterns can be found easily. Highly correlated, but low fitness values for both training and test periods are possible as indicated in 2011. As expected, the correlation between validation and test periods was fairly low since they are both out-of-samples.

Although profitability of technical patterns is the main interest for common traders, precision is an important feature as well. We measured precisions using two different methods: absolute and relative. Absolute precision measures the proportion of matching cases with positive returns to all matching ones. In contrast, relative

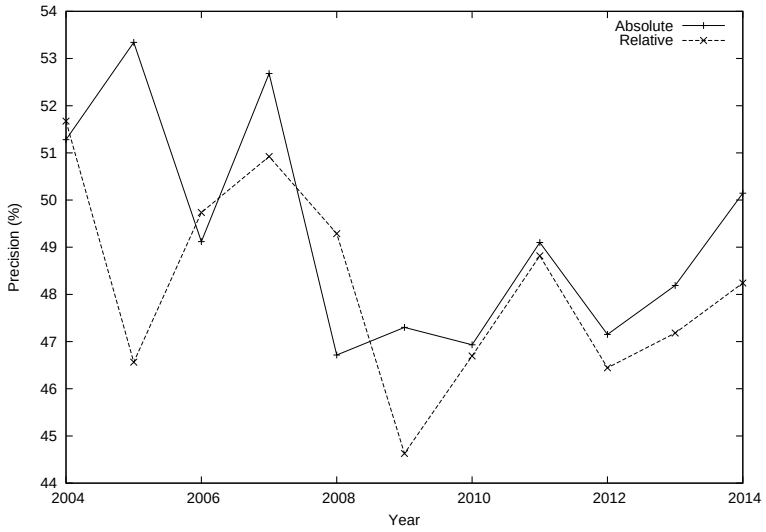


Figure 4.8: Absolute and relative precisions

precision is measured by the proportion of matching cases with positive excess return over the base return, i.e., $E_k(R_{\delta(i,j)})$, to all matching ones. Figure 4.8 shows the average absolute and relative precisions for all years after five trading days. Both absolute and relative precisions showed similar performance patterns to the fitness values in Figure 4.5(e); they were significantly degraded after 2008. In general, absolute precisions performed better than relative ones for most years. Most notable feature is that both precisions were generally lower than 50% except for a few years. It implies that our patterns obtain large profits for correct predictions, which is a feature of most trend following patterns.

Figure 4.9 shows the box-and-whisker plot of node complexities [CKH08] for all years. The medians, or 50% quartiles, of the node complexities were quite small; they were around 20, which means that a pattern typically consists of only 4-5 clauses. We think that this is due to the selection of best patterns using validation periods. Since the performances of too complex patterns are generally degraded in their validation periods, they are not likely to be selected as the final best patterns. This is the main reason for excluding the restriction on the number of nodes or clauses from

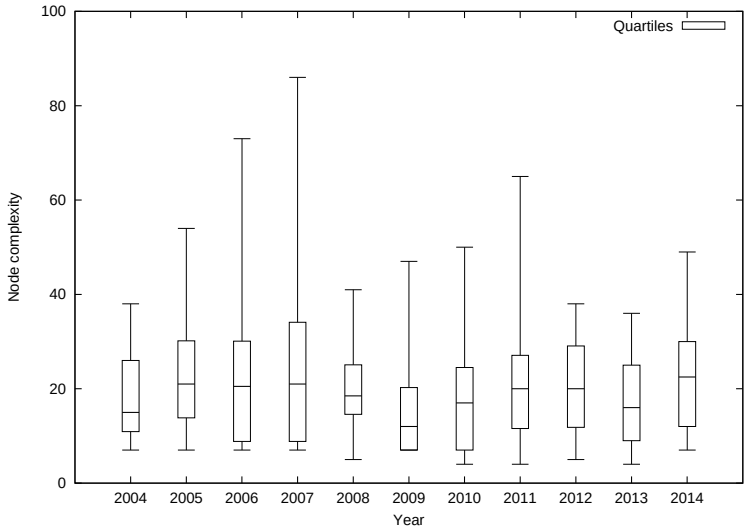


Figure 4.9: Node complexities

our formulation of attractive technical patterns.

Visualization of Similarity Networks

Due to multiple runs for each year, there are many patterns for each year to be analyzed. In particular, we have 30 best patterns per year and 330 ones for all eleven years in total by our parameter settings. To give more insight into the patterns, we constructed a graph for each year.

Let $G = (V, E)$ be an undirected weighted graph, where V is the set of vertices and E is the set of weighted edges. The set V consists of the best patterns in a test year and E has edges whose weight represents a similarity between two patterns. Each pattern in V is converted into its corresponding postfix expression of token IDs. For example, $1.2 * MA_{20}(-1) > p_c(0)$ is converted into $(L, *, L, MA_{20}, L, p_c, >)$ where the first L(literal) corresponds to 1.2 while the second and the final ones to relative offsets -1 and 0, respectively. Using the conversion, each pattern is represented by a sequence of token IDs which can also be regarded as a document in the information retrieval literature. Similarly, each token ID corresponds to a term in a document.

These correspondences enable us to use famous document similarity measures for calculating the similarities among patterns.

To this end, TF-IDF(Term Frequency, Inverse Document Frequency) [SJ72, SM86] was selected. In the similarity measure, $TF(t, p)$ represents the frequency of token ID t's in pattern p. $IDF(t)$ is calculated by

$$IDF(t) = log\left(1 + \frac{|P|}{DF(t)}\right), \tag{4.9}$$

where $DF(t)$ is the pattern frequency of token id t and $|P|$ is the number of patterns.

An element V_i^p of vector $\vec{V}^p$ representing pattern p with distinct token ID t_i's is defined by

$$V_i^p = \frac{TF(t_i, p)}{\sum_{t_i \in p} TF(t_i, p)} \times IDF(t_i). \tag{4.10}$$

The $\vec{V}^p$ is called the *pattern vector* of pattern p.

The similarity between two pattern vectors $\vec{V}^p$ and $\vec{V}^q$ is defined by

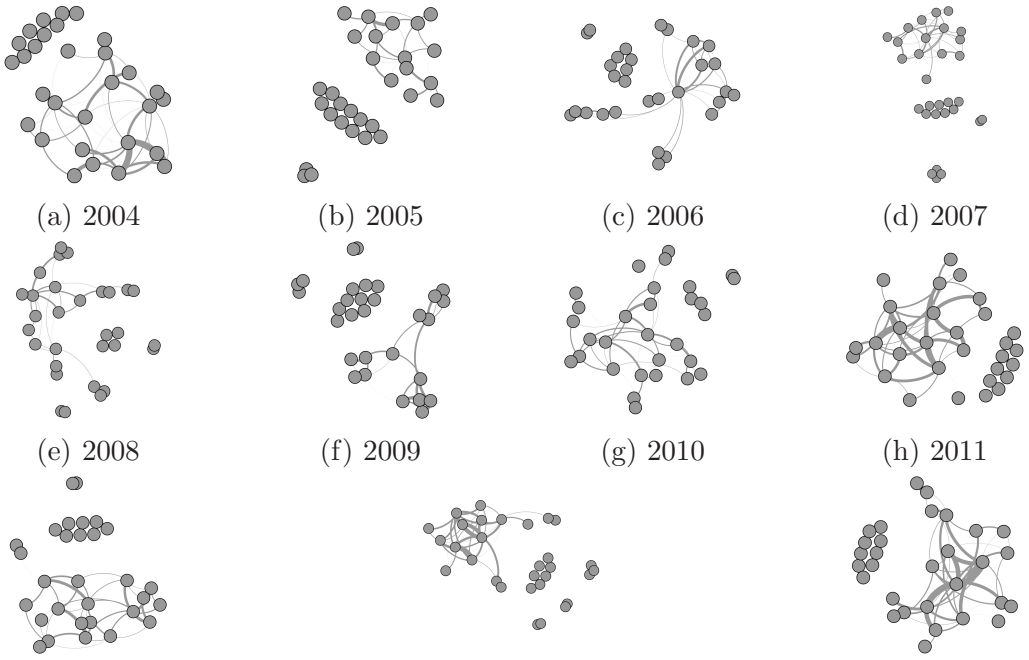
$$SIM(\vec{V}^p, \vec{V}^q) = \frac{\vec{V}^p \cdot \vec{V}^q}{||\vec{V}^p||||\vec{V}^q||}, \tag{4.11}$$

where $||\cdot||$ represents the Euclidean norm.

Using this similarity measure as a weight function, a complete weighed graph G was constructed for each year. However, we found that it is hard to interpret the graph G due to its large number of edges. To include only statistically significant edges, we calculated the average μ_E and sample standard deviation σ_E of weights in E. A new graph $G' = (V, E')$ was then built by selecting edges whose weights, or similarities, are greater than $\mu_E + \sigma_E$. It means that only statistically heavier edges, assuming Gaussian distribution, are selected [KKM03]. The graph G' is called *similarity network* for clarity.

Figure 4.10 shows the similarity networks[12] for each year. The years 2004-2007 showed that they have several connected components with a few patterns. In contrast, the networks in 2008-2012 revealed connected components with more scattered edges. Figure 4.11 shows average clustering coefficients and modularities for

[12]They were visualized by Gephi [BHJ09], a tool for visualizing large networks. We used the Force Atlas algorithm [Noa09] for drawing the networks.



(a) 2004 (b) 2005 (c) 2006 (d) 2007

(e) 2008 (f) 2009 (g) 2010 (h) 2011

(i) 2012 (j) 2013 (k) 2014

Figure 4.10: Similarity networks using TF-IDF

each year. It showed that average clustering coefficient begins to drop suddenly in 2008 and remains considerably low until 2012; the local peak in 2009 is the only exception possibly due to strong trend reversal between 2008 and 2009. In contrast, modularity provided a bitonic pattern showing the highest peak in 2009, which implies that several solution communities were formed. We suspect that these results are related to the sharp fitness degradation after 2008 as shown in Figure 4.5(e). In 2008, there were a few important events such as the currency swap agreement between U.S. and Korea[13] which incurred a dramatic bullish market. Their effects were sharply contrasted with the severe bearish market[14] in 2008. In this sense, the year 2008 seems to add fairly large noises to the learning process of our GP.

[13]KOSPI has risen by +11.95%, which is the highest record in its history, on Dec. 30, 2008.

[14]In 2008, Korean stock market experienced a dramatic meltdown incurred by the subprime mortgage crisis [DH09, DVH11].

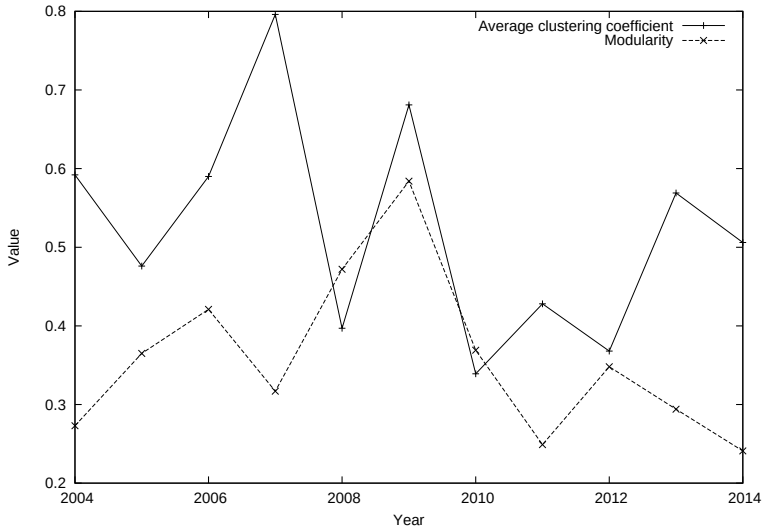


Figure 4.11: Average clustering coefficients and modularities

Ensemble Methods for Improving Returns

In general, ensemble [Bre96] is a method of combining the outputs of multiple models to build a better model. The model can be a classifier, a neural network, and so forth. Typical ensemble aggregates outputs in a model [KM07], but it is also possible to combine different underlying models with various layers to obtain better outputs [HYC08]. In our GP, we built three ensembles combining the outputs of the multiple runs for each year as follows:

1. *I-Ensemble*

- by intersection among the matching sets of the best n solutions.

2. *M-Ensemble*

- by majority voting among the matching sets of the best n solutions.

3. *U-Ensemble*

- by union among the matching sets of the best n solutions.

By majority voting, a matching case is included in *M-Ensemble* if it occurs in more than or equal to $\lceil \frac{n}{2} \rceil$ matching sets of the best n solutions. We set the number

Table 4.6: Simulated returns using ensembles

Parameter		w/o Ensemble		*I-Ensemble*		*M-Ensemble*		*U-Ensemble*	
Initial cash	Stop loss rate	Abs	Exc	Abs	Exc	Abs	Exc	Abs	Exc
100	n.a.	736.03	602.78	180.13	46.88	797.20	663.95	**1450.36**	**1317.11**
500	n.a.	337.45	204.20	109.96	-23.29	368.00	234.75	**614.90**	**481.65**
1000	n.a.	225.46	92.21	77.91	-55.34	243.98	110.73	**398.52**	**265.27**
3000	n.a.	109.95	-23.30	39.91	-93.34	113.21	-20.04	**185.46**	**52.21**
5000	n.a.	75.49	-57.76	27.33	-105.92	76.84	-56.41	**128.29**	**-4.96**
100	-10	919.25	786.00	198.66	65.41	992.94	859.69	**2099.57**	**1966.32**
500	-10	401.70	268.45	120.38	-12.87	438.29	305.04	**819.55**	**686.30**
1000	-10	262.39	129.14	85.31	-47.94	284.22	150.97	**517.07**	**383.82**
3000	-10	122.60	-10.65	43.31	-89.94	129.23	-4.02	**238.13**	**104.88**
5000	-10	83.20	-50.05	29.61	-103.64	88.06	-45.19	**166.01**	**32.76**
100	-20	816.48	683.23	186.08	52.83	871.80	738.55	**1708.33**	**1575.08**
500	-20	366.94	233.69	113.18	-20.07	396.12	262.87	**692.56**	**559.31**
1000	-20	242.27	109.02	80.18	-53.07	260.00	126.75	**440.67**	**307.42**
3000	-20	115.79	-17.46	41.00	-92.25	119.13	-14.12	**202.19**	**68.94**
5000	-20	78.89	-54.36	28.13	-105.12	80.73	-52.52	**139.22**	**5.97**

The best five patterns on validation period among 30 runs consist of an ensemble pattern for each test year.
Bold-faced number represents the best return.
"Abs" and "Exc" represent absolute and excess returns, respectively.
Monetary unit for initial cash is million KRW.
Returns and stop loss rates were written in percentage.

of the best solutions n to 5. The ensemble matching sets were then provided to the SimTrade.

Table 4.6 shows the simulated returns using the ensembles and Figure 4.12 summarizes them in terms of excess return. *I-Ensemble* and *M-Ensemble* were not useful; *I-Ensemble* severely degraded the return and *M-Ensemble* performed marginally better than the strategy without ensemble. In contrast, the returns by *U-Ensemble* were dramatically increased for all parameters. They were further improved with -10% stop loss rate. Their absolute returns were about two times than those without ensemble. We suspect that the different returns among the ensembles are due to the different sizes of matching sets in the ensembles. Since *I-Ensemble* and *M-Ensemble* produce too small matching sets, they lose most opportunities for profitable trades. The significant improvement by *U-Ensemble* implies that the best patterns found by our GP are so diverse that they can be combined to obtain better returns.

Figure 4.13 shows the normalized accumulative assets without ensemble and with

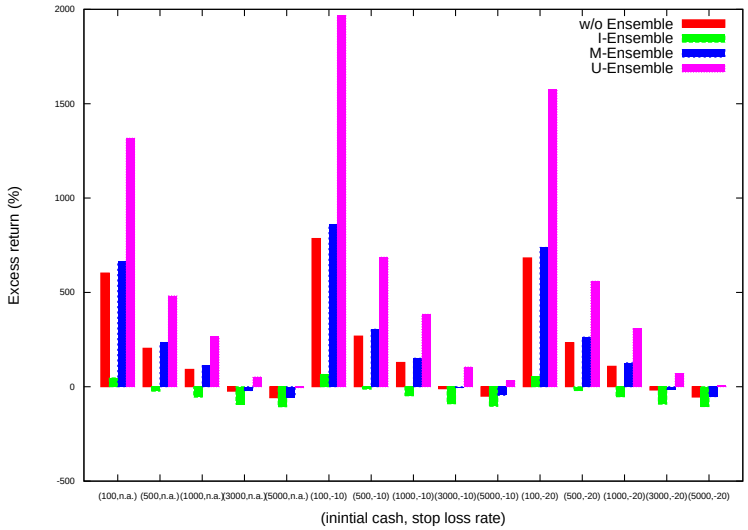


Figure 4.12: Simulated excess returns of ensembles with different parameters

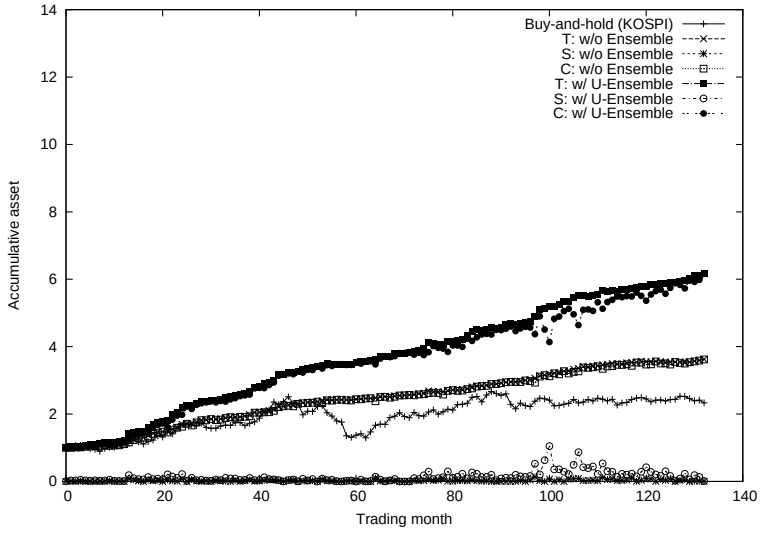


Figure 4.13: Ensemble effect on accumulative assets

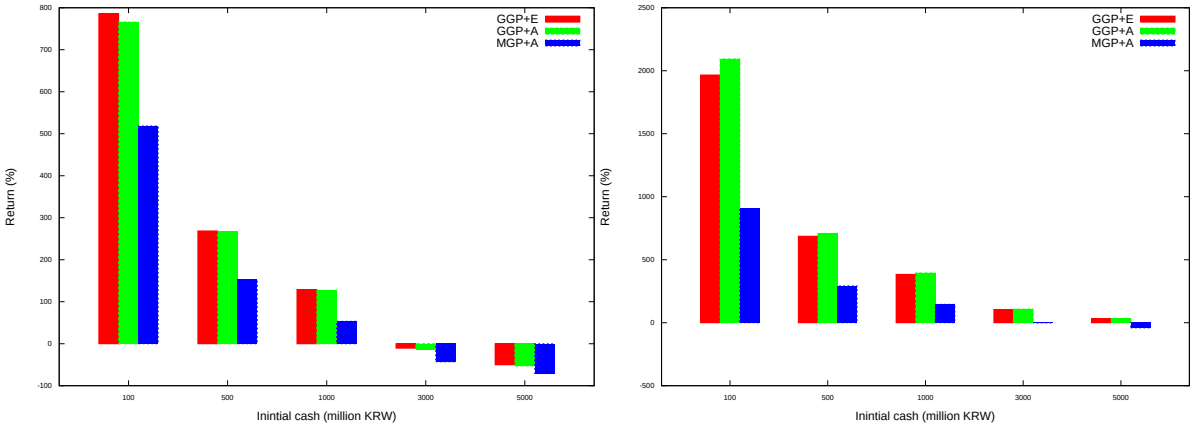
U-Ensemble, where 1000 million KRW and -10% were used as the initial cash and the stop loss rate, respectively. We denote by T, S, and C total, stock, and cash assets,

respectively. For easier comparisons, all assets were normalized to 1 in the beginning date. We found that both the strategies without ensemble and with *U-Ensemble* dominated buy-and-hold; they showed stable and consistent total asset growths as well. In particular, the growth rate of total asset with *U-Ensemble* significantly dominated that without ensemble. Since *U-Ensemble* invests more assets on stock than the strategy without ensemble, it seems to have stable *risk premium* over no ensemble. Its high cash asset rate means that its performance is also related to the effect of time diversification [AR00]. Due to the stability and profitability, our ensemble patterns would be readily exploited by other trading systems.

Comparison with Other Frameworks

It is also necessary to compare our results with those of other previous frameworks. We denote by "A" and "E" the attractive technical pattern discovery (ATPD) [LM10] and its extension in this section, respectively. For fair comparisons, we reimplemented the modular GP [LM10] with validation period and restructured the set of modules. The new modular GP and our grammatical GP are denoted by MGP and GGP, respectively. Among all possible combinations, the case of MGP+E was not included since the set of modules needs to be carefully rebuilt using our new technical inputs. The excess returns without ensemble and with *U-Ensemble* were selected to represent the base and best performances, respectively. Both returns were simulated with -10% stop loss rate, which results in the best return.

Figure 4.14 shows the simulated excess returns of the three framework combinations: GGP+E, GGP+A, and MGP+A. Without ensemble, GGP+E only marginally outperformed GGP+A but significantly did MGP+A. However, GGP+E was slightly dominated by GGP+A under trading with *U-Ensemble*; the return gap was quickly diminished as the amount of cash increases. MGP+A provided the worst returns irrespective of the application of ensemble. These results imply that one of the drawbacks in ATPD, overfitting to some specific period, is significantly relieved by our new implementation. We suspect that the introduction of validation period to the modular GP [LM10] is the key factor for such improvement. In sum, the difference between GPs was significant while the different problem formulations provided



(a) w/o Ensemble (b) *U-Ensemble*

Figure 4.14: Simulated excess returns of three framework combinations

little difference. Since our extension to ATPD, however, uses the filtered matching set, which decreases the trading opportunities substantially by removing too large number of matching cases for a specific day, it has more potential than ATPD for practical trading. We think that more systematic comparisons with more diverse parameters are needed to give further insight into this issue.

4.3.4 Summary

We proposed a hybrid genetic programming for finding attractive technical patterns. To find more free forms of attractive technical patterns, we provided a grammatical type system for describing valid and promising search space of technical patterns. In addition, the formulation of attractive technical patterns was extended in a more systematic way including filtered matching set, base return, and weighted average. By using the extension, we constructed not only more clear but also more meaningful model for describing attractive technical patterns. We also presented an efficient local optimization exploiting two different neighborhoods to fine-tune around local optima.

Various weight models including EQ, LD, LI, ED, and EI were tested on diverse parameters with a commercial simulation tool by Optus Investments Inc. It was

found that EI weight model performed better than the others in terms of simulated returns, which is attributable to its more weights on the distant future.

We studied diverse features of the attractive technical patterns: size of matching set, correlation between data sets, precision, and node complexity. The average size of matching sets over all years was neither too large nor too small, which is appropriate for realistic trading. High correlation between data sets showed that our problem formulation is robust in that a consistent fitness measure can be used for training, validation, and test periods. Our patterns typically showed low precisions which are lower than 50 percent except for a few years. This is related to the characteristics of patterns; our patterns seem to include trend following patterns which typically show low precisions. The small node complexities indicated that our patterns are not very complicated, which is another notable feature of our patterns. We suspect that this is due to out-of-sample validation where the fitness of an overtrained complex pattern is implicitly penalized.

The similarity networks among patterns showed that strongly connected components were found in most years. To improve upon returns, we presented ensembles aggregating the best patterns. The ensemble using union among the matching sets outperformed the others. We think that it is partly related to the strong size restriction on the matching set for each year. The relative performance among ensembles can be different if the size restriction is relieved. Our ensemble results showed that systematic trading strategies with the attractive technical patterns significantly outperform KOSPI which is the buy-and-hold in the Korean stock market.

The comparison with the previous frameworks showed that our GP is more attractive than modular GP, but the difference between problem formulations is not significant. More systematic comparisons are needed since each framework has a number of specific methods, parameters, and so forth.

Future studies would include extension of the set of inputs and trading simulations. Financial statements and their related indicators [Has08] are good candidates for inputs. If the number of inputs becomes too large, it would be necessary to parallelize GPs using popular parallel platforms such as CUDA [CJHNL12, MW12]. More elaborate inputs possibly coupled with band filter [Kau10] or market context

[KM07, Res09, Kau10] can be very meaningful for further improvements. In terms of performance comparisons, more trading simulation models are needed to compare among various formulations of attractive technical patterns.

Finding attractive technical patterns for sell is also quite interesting. Portfolio rebalancing with high-frequency signals [IK11], for instance, can exploit both buy and sell patterns to refine its trading decisions. Besides popular financial objectives such as return and its variance, there are also a number of objectives including utility [HL70, KS02], robustness across various parameters [Kau98], and so on. Modeling our problem as a multi-objective [CTAM09], or pareto optimization problem is left for the further study.

Chapter 5

Seeded Evolution

Typical evolutionary algorithm begins its evolution with the initial population of candidate solutions which are generated at random. In other words, it does not assume any domain knowledge on the initial solutions, which is one of the main advantages of evolutionary algorithms. However, the advantage becomes no longer evident when domain knowledge is available and it can be exploited to reduce the search space. One of the most effective methods to reflect domain knowledge is to provide, or seed, the initial population with some promising solutions. Since any evolutionary algorithm uses a finite population, its performance is likely to be affected by the quality of the initial population.

In general, there are three fundamental issues in seeded evolution: the strength of seeding which involves the selection of heuristic, the ratio of seeded individuals, and the genes to be seeded. First, the performance of seeded evolution is strongly dependent upon the selected heuristic for generating promising solutions. If the heuristic is too strong, the subsequent evolution is expected to suffer from premature convergence due to the loss of diversity among individuals. The opposite case is not effective as well; the heuristic has little effect on the initial population. Second, the ratio of seeded individuals is also important in that it designates the number of seeded individuals, indirectly controlling the strength of seeding. In fact, it is the major candidate for tuning the strength of seeding since modifying the selection of heuristic requires more effort than changing the ratio of seeded individuals. Finally,

the selection of the genes to be seeded has a substantial effect on the performance as well. When more than one encodings are used for solution, each encoding can have its own meaning and its hardness can also be different. Ideally, more important but cheaply optimizable genes are considered to be the best choice for seeding. The best seeded evolution can be achieved by the optimal balancing between exploitation and exploration, similar to other techniques used in more improved evolution.

In this chapter, we propose a hybrid genetic algorithm for large-scale stock selection. We present a heuristic seeding for the initial population. The heuristic seeding provides the initial population with partially optimized individuals generated by a sampling-based heuristic. A ranking with partitions for penalization of unattractive and invalid financial ratios is also presented for further improvements. In the ranking algorithm, all stocks are assigned consistent rank scores with partitioned penalization of negative and invalid financial ratios. Experimental results showed that our methods significantly improve upon the return of portfolio.

5.1 Heuristic Seeding

While a linear string of binary, integer, or real values is the most popular encoding used in genetic algorithms, a mixture of them can be natural according to problem domain. For example, stock selection problem [HCCC12] used a linear string of real values followed by that of binary values as its chromosomal encoding. In fact, it involves a simultaneous optimization of two heterogeneous gene parts in the encoding, that is, real and binary vectors. However, such simultaneous optimization is not the best when the importance of each gene part is quite different.

Let us suppose that we use a mixed chromosomal encoding using a linear string of K w-values followed by that of K b-values, where the number of possible values in w-value is quite larger than that in b-value. The former values are called w-vector and the latter b-vector in a vector notation. The most common examples of w-value and b-value are real and binary ones, respectively. Intuitively, the b-values can more easily be optimized than the w-values due to their fairly smaller number of possible values. In addition, if they are believed to be more important in performance, they

are good candidates for pre-optimization by fixing w-values to some representative ones. That is, we try to pre-optimize b-values, which are less complex in encoding and believed to be more important, using fixed w-values. The pre-optimized b-values are used to generate some individuals in the initial population, thereby providing the initial pool of promising solutions.

Figure 5.1 shows our heuristic seeding algorithm in detail. First, we select the set of genes that are encoded in a more complex form, or w-vector, which is the candidate for fixation. The vector is fixed to a *representative vector* to reduce the dimension of the problem. We simply choose the *geometric center*, i.e., $(w_1, w_2, \ldots, w_K)^T$ where $w_i = \frac{1}{K}$ $(i = 1, 2, \ldots, K)$ for K inputs, as the representative vector. Second, a number of random solutions, whose w-vectors are fixed by the representative vector, are generated and evaluated with the fitness function. In other words, we evaluate diverse b-vectors for the representative w-vector. The number of samples to be evaluated is parameterized by h_n. After the sampling, the most attractive b-vector is determined by the majority voting among $\lfloor h_n \times h_v \rfloor$ samples, where h_v is the ratio of voting samples. Finally, the most attractive b-vector is then provided to the initial population with probability h_p. The ratio of seeded individuals to the entire initial population is controlled by h_p.

Our heuristic seeding has two notable features. First, it optimizes only part of a chromosome that is expected to be more easily optimized. Since tuning w-vector, which involves optimization of real-valued genes, seems to be much harder than that of b-vector, it tunes only b-vector. This is notable in that most seeded GAs use fully-optimized chromosomes, meaning all genes are optimized, as the promising initial seeds. Second, it is quite general due to its sole dependence upon fitness; it can be applied to general problems where the cost of fitness evaluation is reasonably cheaper than the entire GA framework.

5.2 Hybrid Genetic Algorithm

We use a steady-state genetic algorithm [WK88, Sys89], where the offspring replaces an individual in the same population. In this section, w-vector and b-vector corre-

spond to the weight and sorting indicator vectors, respectively (see Section 5.3.2 for details).

- Representation and initialization

  Each individual is represented by a linear chromosome of two distinct vectors: weight and sorting indicator vectors. The weight vector consists of K weights, where each weight $0 \leq w_i \leq 1$ represents the importance of the i-th financial ratio. The sum of all weights is always constrained to 1. The i-th element of the sorting indicator vector represents the ranking function used by the i-th financial ratio. It is zero or one, meaning that we consider only two ranking methods indicated by the binary sorting indicator. We use an unbiased random sampling, which was introduced by [Rub81],[1] for generating a random weight vector.[2] Figure 5.2 shows the unbiased sampling for choosing a random weight vector. The population size is set to 50.

- Fitness function

  The fitness function is defined by the average compound return $\varphi_p^{t_b,t_e}$. The portfolio size p is set to 30.

- Selection, crossover, mutation, and repair

  The tournament selection [GDK91] is used. The crossover exploits two different crossovers: arithmetic and uniform crossovers. The arithmetic crossover [Mic96] is used for weight vector and uniform crossover [Sys89] for sorting indicator vector. Mutation for weight vector is performed by increasing or decreasing a randomly chosen gene by 0.1 with the bound $[0, 1]$. For sorting indicator vector, a randomly chosen sorting indicator, having 0 or 1, is flipped. After the crossover and mutation, a repair by normalization of weight is performed so that the sum of all weights is 1.

- Replacement and stopping criterion

[1] http://cs.stackexchange.com/questions/3227/uniform-sampling-from-a-simplex

[2] In fact, this is identical to uniform sampling on the unit *simplex*, which is also closely related to the Dirichlet distribution.

We first try to replace the worst individual on the training period. If the worst is the same as the best on the validation period, two individuals are randomly selected as replacement candidates. One of the replacement candidates is replaced with a probabilistic bias toward worse individuals. Our GA stops if the number of consecutive fails to update the best solution reaches 100 or the number of generations exceeds 500.

- Local optimization

  We use a local optimization that sequentially optimizes the weight and sorting indicator vectors. Figure 5.3 shows our local optimization algorithm. It first begins with the sorting indicator vector and samples n_h solutions by flipping the value of a randomly chosen gene index. The fitness gains are computed for all sample solutions and the gene index and its value with the best gain, i.e., s_{max} and b_{max} are saved. If the best gain is positive, its corresponding gene, i.e., $c.si[s_{max}]$, is modified to b_{max}. The optimization for the weight vector is similar but it uses two different neighborhoods: by increase and decrease. The size for both increase and decrease is determined by the constant d. The weight vector is then normalized by dividing each weight by the sum of all weights so that the sum of all weights is constrained to 1. Note that the search intensity can be controlled by adjusting n_h; the larger value for n_h means more neighborhood solutions to be investigated. After some experiments, we set the parameters n_h and d to 3 and 0.1, respectively.

- Best update and elitism

  After the initialization, the best individual on its training period is selected and its fitness on the validation period is saved as the best fitness. The best fitness is updated if new trained best individual also improves upon the fitness on the validation period. This update scheme has been extensively used in the literature [AK99].

  Elitism [DeJ75] is also used to further improve upon our GP. The best individual is replaced only if it performs worse than the new best individual on

the validation period.

5.3 Large-Scale Stock Selection

5.3.1 Introduction

In stock markets, identifying attractive stocks [QS99, HCCC12] is a critical topic for investments. Typical objectives for selecting attractive stocks include famous features such as high return and low risk. Among the features, one of the most common features is return, which is to be maximized for any traders or investors.

There have been two common approaches to choose attractive stocks: *modern portfolio theory* and *stock selection*. For modern portfolio theory, Markowitz [Mar52] proposed a method for maximizing expected return of a combination of stocks, or *portfolio*, for a given level of risk. Although the framework is theoretically attractive, it has a drawback of requiring a huge amount of computations particularly in calculating the covariance between the returns of all stocks. To alleviate the computational complexity, Sharpe [Sha63] proposed a single index model assuming that the risk of any individual stock is dependent only upon a common market's variance. These portfolio theories have resulted in *market equilibrium models* including capital asset pricing model (CAPM) [Sha64, Lin65a, Lin65b, Mos66] and arbitrage pricing theory (APT) [Ros76]. The modern portfolio theories, however, have common drawbacks in that they cannot reflect the practical constraints including desired cardinality for portfolio and minimum transaction lots; such constraints have been alleviated by some recent studies [CMBS00, MS99]. Instead of using classical optimization methods, a number of evolutionary algorithms [Bäc96] such as genetic algorithms (GAs) [Hol75] for portfolio optimization [AI09, LL08] have also been proposed.

Compared with the modern portfolio theory, which optimizes a portfolio vector where each element represents the proportion of asset invested on the corresponding stock, stock selection is to score stocks with various criteria and select the top-ranked ones for a portfolio [HCCC12]. Although it can be used as a prior stage for limiting the universe of portfolio [HZ95], it has typically been used for directly constructing a portfolio by selecting the top-ranked stocks. The criteria for stock

selection include financial, technical, and macroeconomic variables. One of the most common variables is financial ratio [Hor68, Cou78] such as P/E (price-to-earnings) ratio in relation to quarterly or annual financial statements.

Due to the recent advances of computer technologies, several machine learning methods have been proposed for stock selection. Quah and Srinivasan [QS99] used an artificial neural network (ANN) for choosing attractive stocks with several financial ratios including yield, liquidity, risk, growth, and momentum factors. They showed that a simple ANN was enough to construct portfolios with significant excess returns in Singapore equity market. To find more interpretable models, some studies used genetic programmings (GPs) [CB05, BFL07] for stock selection. Fuzzy systems [YS13] and hybrid ones [Che13] have also gained in public favor for improving upon the naive stock selection models. Recently, Huang *et al.* [HCCC12] proposed a fuzzy-based genetic algorithm with feature selection. They adjusted the final stock ranking with a triangular fuzzy membership function whose parameters were adaptively tuned. It is notable that they evolved even the sorting orders for financial ratios which have been determined by financial experts [XMP09].

Most studies on stock selection are limited in that they use the naive sorting algorithm for converting from financial ratios into rank scores. For example, Huang *et al.* [HCCC12] used a binary sorting indicator which directs only ascending or descending order of a given financial ratio. Since their data set was restricted to only large-cap 200 stocks, which have good, or positive financial ratios, they could obtain significant excess returns. We strongly suspect that their model will not work well for the full universe of stocks which includes stocks with bad, or negative financial ratios. In addition, an extension of their model to include delisted stocks would also require some modifications.

Another important problem is the heterogeneous complexity of subproblems. The stock selection model by Huang *et al.* [HCCC12] optimizes both weight and sorting indicator simultaneously. From an encoding complexity perspective, it consists of two intertwined problems: optimizations of real-valued weight and binary sorting indicator. They are not completely independent but it is clear that the latter has less complexity due to its binary encoding, which facilitates some pre-optimization.

5.3.2 Problem Formulation

We extend Huang *et al.*'s stock selection model [HCCC12] in a systematic way to include more general features. For simplicity, we assume that the number of stocks is constant over time; it equals to the number of currently available all stocks. This assumption simplifies the problem but it incurs two types of invalid stocks: delisted and not-yet-listed stocks. Both delisted and not-yet-listed stocks have no available data for prices and financial statements. We call them *invalid* stocks at a given time t. Stocks that are not invalid, i.e., listed stocks, are called *valid*.

Let N be the number of all stocks including valid and invalid stocks and K be the number of financial ratios. The sets of valid and invalid stocks at time t are represented by L^t and D^t, respectively. Note that N is time-invariant, meaning that N is equal to the sum of $|L^t|$ and $|D^t|$ irrespective of time t. Let $V^t = [v_{ij}^t]$ be the $N \times K$ value matrix whose element represents the value of financial ratio j of stock i at time $t = \{0, 1, \ldots, Q-1\}$.

A *ranking function* r_j^t ($\in R$) for financial ratio j at time t is a function that maps a column vector $(v_{1j}^t, v_{2j}^t, \ldots, v_{Nj}^t)^T$ to a *rank score* column vector $(s_{1j}^t, s_{2j}^t, \ldots, s_{Nj}^t)^T$, where $0 \leq s_{ij}^t < 1$ for $i \in L^t$, $s_{ij}^t = 1 + \delta$ ($\delta > 0$) for $i \in D^t$, and R is the set of all possible ranking functions. By applying the ranking function r_j^t to all columns of V^t, a score matrix $S^t = [s_{ij}^t]$ is obtained.

The *score* of stock i at time t is defined by

$$S_i^t = (s_{i1}^t, s_{i2}^t, \ldots, s_{iK}^t)(w_1^t, w_2^t, \ldots, w_K^t)^T, \tag{5.1}$$

where $(w_1^t, w_2^t, \ldots, w_K^t)^T$ is the *weight vector* for assigning nonnegative weights w_i's ($0 \leq w_i \leq 1$) to the corresponding K financial ratios. The sum of weights, or $\sum_{i=1}^K w_i$ is constrained to unity, or 1.

To construct a portfolio with size p, it is common to choose stocks with the smallest p scores. Let $\sigma_i^t(r^t, w^t)$ denote the index of stock with the i-th smallest score at time t by the vector of ranking functions $r^t = (r_1^t, r_2^t, \ldots, r_K^t)$ and the weight vector $w^t = (w_1^t, w_2^t, \ldots, w_K^t)^T$. For simplicity, we write σ_i^t instead of $\sigma_i^t(r^t, w^t)$.

Let the indices for the stocks with the smallest scores be $\{\sigma_1^t, \sigma_2^t, \ldots, \sigma_p^t\}$, where σ_i^t represents the original index of stock i. The return of the portfolio at time t is

defined by

$$\varphi_p^t = \frac{1}{p} \sum_{i=1}^{p} \left(\frac{P_{\sigma_i^t}^t}{P_{\sigma_i^t}^{t-1}} - 1 \right), \tag{5.2}$$

where $P_{\sigma_i}^t$ is the total value, or simply the price, of stock σ_i^t at time t.

For a period from t_b to t_e, the *average compound return* of the portfolio is calculated by

$$\varphi_p^{t_b,t_e} = \left(\prod_{t=t_b}^{t_e} (\varphi_p^t + 1) \right)^{\frac{1}{n}} - 1, \tag{5.3}$$

where $n = t_e - t_b + 1$ is the number of portfolio returns.

Definition 5.3.1 (Stock selection)**.** *Let R and $W = [0, 1]$ be the sets of ranking functions and weight values, respectively. For a given period (t_b, t_e), K financial ratios, and portfolio size p, stock selection is to find the optimal sequence of (r, w)'s S^* such that*

$$S^* = \underset{(r^{t_b}, w^{t_b}), \ldots, (r^{t_e}, w^{t_e})}{\arg\max} \varphi_p^{t_b,t_e},$$

where $r^t = (r_1^t, r_2^t, \ldots, r_K^t) \in R^K$ is the vector of ranking functions, which are selected a priori, and $w^t = (w_1^t, w_2^t, \ldots, w_K^t) \in W^K$ is the weight vector.

However, finding the optimal vector of ranking functions r by nonparametric optimizations is computationally intractable in general. To reduce the problem to a tractable one, it is customary to choose a small set of ranking functions from all possible ones. The most common set for ranking function is the set of two linear rankings by sort: by ascending and descending orders, which reduces R to the set of two ranking functions (i.e., $|R| = 2$). The linear ranking by ascending sort, for example, assigns $\frac{i-1}{|L^t|}$ score to the i-th smallest value for L^t stocks. In later sections, we use two linear ranking functions, which are indexed by the value of *sorting indicator*.

Finally, we verify the safety condition which states our portfolio never chooses invalid stocks under reasonable assumptions.

Fact 5.3.1. *Given $0 < p \leq |L^t|$, any portfolio of p stocks with the smallest p rank scores does not have invalid stock.*

Proof. It is easily verified by contradiction. Suppose that an invalid stock i is included a portfolio of p stocks and its rank score is S_i^t. From the definition of score for invalid stock, $S_i^t = \sum_{k=1}^{K} w_k^t s_{ik}^t = (1+\delta)\sum_{k=1}^{K} w_k^t = 1 + \delta(\delta > 0)$ (i.e., $S_i^t > 1$). Since any valid stock j has its rank score $S_j^t < 1$, any invalid stock with $S_i^t > 1$ cannot be selected for a portfolio unless $p > |L^t|$. This causes a contradiction to our assumption that invalid stock i is included in a portfolio with size $0 < p \leq |L^t|$. □

Note that this fact holds if the sum of weights is unity irrespective of weights and ranking functions.

5.3.3 Ranking with Partitions

Linear ranking by naive sorting is one of the most popular methods for scoring financial ratios. Although it is enough for some restricted universe of stocks, it is not appropriate for a large or the full universe of stocks which includes bad financial ratios as well. Note that bad financial ratio typically means negative one for most well-known financial ratios. For example, P/E (price-to-earnings) ratio measures the price-level, or market capitalization, divided by net income or profit. In stock selection, stocks with smaller P/E ratios are generally preferred since they are undervalued; their prices are believed to rise to their fair values, which has been supported by many empirical studies [Bas77, Bas83].[3] However, if the P/E ratio is negative, its interpretation is not equivalent to that of positive value; smaller P/E ratio means larger net loss, thereby implying less attractive. This inconsistency for the sign makes rankings by native sorting less meaningful for a large or the full universe of stocks.

There have been two popular methods to deal with the inconsistency: *restriction of universe* and *explicit filtering*. The restriction of universe [HCCC12] typically limits the universe, or all stocks under evaluation, to stocks with large market capitalizations. It implicitly filters out stocks with negative financial ratios since the large-cap stocks are typically likely to have positive ones. In contrast, explicit fil-

[3]These results are strong counter-evidences to the earlier work [BB68] which supported the futility of analyzing announced, or publicly available, financial statements for obtaining excess returns.

tering [Gil90, JKW89, MDK97, HLY07, YS13] excludes stocks with some negative financial ratios, i.e., negative P/E ratios, from its experimental study. One common drawback of these two methods is their lack of ranking for negative financial ratios. Moreover, most studies did not use delisted stocks, which are useful for eliminating *survivorship bias* [GT92, EGB96].

For more generalized ranking, we propose a ranking algorithm with partitions, where all financial ratios can be consistently ranked. Figure 5.4 shows our ranking algorithm with partitions for a financial ratio. In the algorithm, Partition($\vec{v}_j^t$,m,n,ψ) reorders the vector elements from v_{mj}^t to v_{nj}^t with the pivot ψ so that $\forall i < q$, $v_{ij}^t < \psi$ and $\forall i \geq q$, $v_{ij}^t \geq \psi$ and it returns q as the largest index of the lower partition. The binary values of the sorting indicator I represent two predefined ranking functions. The $index[a, b]$ returns the index of a in the sorted list b.

The ranking algorithm first divides all stocks into two partitions: valid and invalid. The valid stocks have the values for a financial ratio; each value can be positive or negative. The partitioning is then applied only to valid stocks and it divides them into two further partitions: positive and negative. The ranking algorithm assigns rank scores to all stocks by the corresponding sorting indicator. Note that the partitions for negative and invalid values are not affected by the sorting indicator. It is because they are not very useful for constructing profitable portfolio. Only positive values are affected by the sorting indicator.

It should be noted that our ranking algorithm first seems to be similar to the explicit filtering in that it assigns higher scores to positive financial ratios than negative and invalid ones by the partitioning. However, it leaves the possibility of selecting the negative ones while the explicit filtering does not. The difference would be evident if some financial ratios have skewed distributions of their signs. In addition, the explicit filtering raises a selection problem when several competing financial ratios are present for the filtering.

5.3.4 Experimental Results

Parameter Settings

The sets of h_n, h_v, and h_p values are $\{500, 1000, 2000\}$, $\{0.05, 0.5\}$, and $\{0.2, 0.5, 0.8\}$, respectively. The number of financial ratios K is set to 15 and the number of all stocks to 2471. We set the number of times Q to 54, which corresponds to the number of quarters in our study. The special values δ and ζ in Figure 5.4 are set to 0 and the maximum floating-point number,[4] respectively.

Test Beds and Test Environment

We used all Korean stocks including listed and delisted stocks from 2000 to 2014. All quarterly financial statements were used for calculating the values of financial ratios. The data were provided by Optus Investments Inc.

The time lag between the fiscal quarter and its actual announcement date was reflected as well; we assumed that the quarterly financial statements for each fiscal year-end are available after 90 days and those for other quarters after 45 days.[5] By these time lags, the four quarters in each year begin from April 1, May 16, August 16, and November 16, respectively. For simplicity, they are denoted only by their months in later sections.

The set of financial ratios was adapted from Huang *et al.*'s study [HCCC12] with minor modifications. To alleviate possible seasonal effects, some financial variables used aggregate values summed over the most recent four quarters. They include net income, net sales, operating income, inventory, cash flow from operating activities, and accounts receivable. The others used the values at the most recent quarter. Compared with most financial variables, which are completely determined by financial

[4] This can be different across the underlying compilers or systems.

[5] This is rather conservative since most companies release their annual reports within 90 days. For example, Ball and Brown [BB68] showed that most firms release their annual reports about 40 days before the filing deadline in 1965. They also showed that the time lag between the fiscal year-end and the announcement date of the annual report had been declining over their sample period. Recently, U.S. Securities and Exchange Commission (SEC) has reduced the filing deadlines for annual and quarterly reports to 60 and 35 days, respectively [Sen04]. Korean firms have similar tendency of such early announcements particularly in their annual reports.

Table 5.1: The set of financial ratios

Name	Description
PBR (P/B Ratio)	Price-to-book ratio = market capitalization / shareholders' equity
PSR (P/S Ratio)	Price-to-sales ratio = market capitalization / net sales
PER (P/E Ratio)	Price-to-earnings ratio = market capitalization / net income
ROE	Return on equity = net income / shareholder's equity
ROA	Return on asset = net income / total assets
OPM	Operating profit margin = operating income / net sales
NPM	Net profit margin = net income / net sales
DER (D/E Ratio)	Debt-to-equity ratio = total liabilities / shareholder's equity
CFR (CF Ratio)	Cash flow ratio = cash flow from operating activities / current liabilities
CR	Current ratio = current assets / current liabilities
QR	Quick ratio = quick assets / current liabilities
ITR	Inventory turnover rate = net sales / inventory
RTR	Receivables turnover rate = net sales / accounts receivable
OIG	Operating income growth rate = $\sum_{j=t-3}^{t} OI_j$ / $\sum_{j=t-7}^{t-4} OI_j$
NIG	Net income growth rate = $\sum_{j=t-3}^{t} NI_j$ / $\sum_{j=t-7}^{t-4} NI_j$

OI_j and NI_j are operating and net incomes at quarter j, respectively.
Note that OIG and NIG were calculated differently from the common equation for growth rate of income. Both net and operating incomes were adjusted after taxes.

statements, market capitalization requires the price and number of shares outstanding at a specific date. It was calculated at the beginning date of each quarter; it also included preferred stocks for more exact valuation. Table 5.1 shows the set of financial ratios. Each financial ratio belongs to one of six groups as follows:

- Share price rationality : PBR, PSR, and PER
- Profitability : ROE, ROA, OPM, and NPM
- Leverage : DER
- Liquidity : CFR, CR, and QR
- Efficiency : ITR and RTR
- Growth : OIG and NIG

To obtain more consistent out-of-sample performance, we used the *rolling forward method* [PT95]. For a given test quarter t, the four quarters from $t-5$ to $t-2$ were used as training period and the quarter at $t-1$ as validation one. This process was

shifted quarter by quarter for all overlapping sequences. We performed 30 runs for each sequence.

Dividends were reflected in calculating returns due to their significance. Transaction fees, however, were not included in our results; they were found to be too small to significantly affect the quarterly returns of our portfolio. The portfolio size was set to 30, which is common to both academic experiments and practical fund managements.

All programs were written in C# language and run on Intel Xeon E5620 2.40 MHz with Windows Server 2008 R2 Datacenter. They were compiled using Visual Studio 2010 Ultimate.

Effects of Diverse Factors

Table 5.2 shows the average quarterly compound returns of our portfolio with respect to diverse factors[6] including the ranking with partitions, heuristic seeding, and parameters of the heuristic seeding. The total number of generations for each run is the sum of all generations required for running all sequences. Inclusions of the ranking with partitions and the heuristic seeding are denoted by "R" and "H," respectively. The columns "Best," "Average," and "$\sigma/\sqrt{n}$" represent the best, average, and group standard deviation of 30 runs, respectively. The statistical significance of the average returns over HGA and HGA+R was evaluated by two tailed t-test.[7] The columns "P_{HGA}" and "P_{HGA+R}" represent the corresponding p-values.

The table showed that HGA+H performed better than HGA for all heuristic seeding parameters in average returns. In particular, 14 out of the 18 cases were significant with $p < 0.05$. In addition, the best returns were notably improved as well. It seems that HGA+H has little discernible patterns in performance across various heuristic seeding parameters.

In contrast, HGA+R showed a dramatic improvement over HGA; it increased the

[6]Pure GAs, or ones without any local optimizations, were also experimented but they are not included in this thesis. They showed consistently degraded results compared with their hybrid GA counterparts.

[7]The null and alternative hypotheses were H_0: $\mu_1 = \mu_2$ and H_1: $\mu_1 \neq \mu_2$, respectively; μ_1 and μ_2 are the average quarterly compound returns.

average quarterly return from 2.20% to 8.42%. It seems to have better capability for selecting attractive stocks due to the ranking with partitions which penalizes bad, or negative financial ratios. Among various methods, HGA+HR outperformed all the other GA variants using diverse heuristic seeding parameters. The statistical test of HGA+HR against HGA+R showed that 8 out of the 18 cases were significant with $p < 0.05$. In particular, the average quarterly return tends to be improved as increasing h_v and h_p values while it does not for increasing h_n. In fact, too large h_n was not quite meaningful for obtaining higher returns, which is possibly due to over-tuning of the heuristic seed.

It is interesting that the heuristic seeding is more meaningful when the ranking with partitions is incorporated. In other words, it is more useful only when relatively better *schemata* [Hol75], i.e., the ones which have better sorting indicator vector provided by the heuristic seeding, are rather consistently identified. Hereafter, we present the experimental results only by HGA+HR with $(h_n, h_v, h_p) = (2000, 0.5, 0.8)$ which performed best in terms of average quarterly compound return.

Effectiveness of Heuristic Seeding

Our GA is provided with the population of partially seeded solutions. The partially seeded solutions are used by the genetic operators to find better ones. If they are meaningful for improvement, the partially seeded genes would remain largely unchanged after a genetic run.

To investigate the effect of the heuristic seeding, we computed the Hamming distance [Ham50], or the number of different bits between two binary strings, between the initial sorting indicator vector and the final one contained in the best solution after a genetic run. For each quarter, the distances of 30 runs were averaged. Figure 5.5 shows the average distance between the initial and final sorting indicator vectors. It shows that the average distances were very small compared to the dimension of the sorting indicator vector, i.e., 15, for all quarters. In fact, the distance averaged over all quarters was around 2.00, which means that the seeded sorting indicator vectors were almost unchanged in the genetic run. This is attributable to the high-quality

Table 5.2: Average quarterly compound returns with respect to diverse factors

Method	h_n	h_v	h_p	Best	Average	$\sigma/\sqrt{n}$	P_{HGA}	P_{HGA+R}	Gen†	CPU‡
HGA	-	-	-	3.994	2.202	0.216	1.000	0.000	7628	343
HGA+H	500	0.05	0.2	5.024	2.889	0.193	0.025	0.000	7398	421
	500	0.05	0.5	4.981	3.003	0.170	0.007	0.000	7315	416
	500	0.05	0.8	5.120	3.188	0.155	0.001	0.000	7314	424
	500	0.5	0.2	4.969	2.697	0.199	0.102	0.000	7391	431
	500	0.5	0.5	4.898	3.039	0.216	0.010	0.000	7483	429
	500	0.5	0.8	4.781	2.704	0.213	0.109	0.000	7274	422
	1000	0.05	0.2	6.117	3.044	0.175	0.005	0.000	7509	524
	1000	0.05	0.5	5.044	3.019	0.177	0.007	0.000	7392	517
	1000	0.05	0.8	5.308	3.274	0.170	0.001	0.000	7328	513
	1000	0.5	0.2	4.940	3.065	0.192	0.006	0.000	7398	521
	1000	0.5	0.5	4.810	2.848	0.194	0.034	0.000	7420	521
	1000	0.5	0.8	4.393	3.084	0.119	0.001	0.000	7338	518
	2000	0.05	0.2	4.221	2.422	0.160	0.420	0.000	7408	711
	2000	0.05	0.5	4.841	3.164	0.144	0.001	0.000	7461	704
	2000	0.05	0.8	4.572	3.158	0.154	0.001	0.000	7202	700
	2000	0.5	0.2	4.358	2.723	0.159	0.062	0.000	7517	709
	2000	0.5	0.5	5.293	2.853	0.185	0.030	0.000	7315	708
	2000	0.5	0.8	5.296	3.217	0.187	0.001	0.000	7247	696
HGA+R	-	-	-	9.687	8.418	0.145	0.000	1.000	7705	347
HGA+HR	500	0.05	0.2	10.201	8.778	0.132	0.000	0.078	7516	435
	500	0.05	0.5	9.968	8.539	0.130	0.000	0.540	7420	432
	500	0.05	0.8	10.329	8.666	0.139	0.000	0.227	7429	435
	500	0.5	0.2	9.786	8.694	0.130	0.000	0.168	7536	439
	500	0.5	0.5	11.163	8.883	0.136	0.000	0.027	7298	425
	500	0.5	0.8	**11.337**	9.102	0.139	0.000	0.002	7306	427
	1000	0.05	0.2	9.746	8.600	0.100	0.000	0.310	7531	535
	1000	0.05	0.5	9.469	8.755	0.073	0.000	0.047	7526	533
	1000	0.05	0.8	9.899	8.654	0.120	0.000	0.220	7287	520
	1000	0.5	0.2	9.990	8.713	0.118	0.000	0.125	7496	533
	1000	0.5	0.5	10.315	9.104	0.128	0.000	0.001	7562	528
	1000	0.5	0.8	10.259	9.007	0.113	0.000	0.003	7334	520
	2000	0.05	0.2	9.855	8.602	0.120	0.000	0.335	7663	730
	2000	0.05	0.5	9.760	8.513	0.111	0.000	0.607	7510	720
	2000	0.05	0.8	9.647	8.791	0.107	0.000	0.048	7378	724
	2000	0.5	0.2	10.006	8.630	0.121	0.000	0.270	7514	722
	2000	0.5	0.5	10.895	9.073	0.119	0.000	0.002	7348	715
	2000	0.5	0.8	10.160	**9.253**	0.100	0.000	0.000	7354	716

† Average of 30 runs.
‡ CPU seconds on Intel Xeon E5620 2.40 MHz.

of the seeded sorting indicator vectors.

Quarterly Analyses of Returns

It is important to analyze how the compound return was achieved over the entire period as well. Table 5.3 shows the quarterly returns of all quarters. To compare each quarterly return with buy-and-hold, we used two benchmark returns by "KOSPI" and "All stocks." The former is a capitalization-weighted index, which is the most famous market index in Korea, while the latter an equal-weighted index[8] that averages the returns of all stocks in the same country. The excess returns over the two benchmarks are denoted by "Exc_K" and "Exc_A," respectively.

For most quarters, our portfolio showed positive excess returns; average winning rates by Exc_K and Exc_A were 78% and 89%, respectively. The quarters with lower Exc_K overlapped several bullish quarters driven only by large-cap stocks. However, the average excess returns by Exc_K and Exc_A were quite similar; they were 7.73% and 7.36%, respectively. It implies that our portfolio has several quarters with fairly large excess returns, which compensates for its lower winning rate by Exc_K.

To give insight into the compensation effect, Figure 5.6 compares the quarterly returns of our portfolio and buy-and-hold (KOSPI). It shows that a notable asymmetry exists between the returns of our portfolio and the market. For most bullish quarters, identified by large buy-and-hold returns, our portfolio outperformed buy-and-hold significantly. In November 2004, for example, KOSPI showed 12.01% return while our portfolio did 56.78%, which causes the largest excess return over all periods. Similarly, one can easily identify several quarters with large buy-and-hold returns, but with even larger portfolio returns. In contrast, our portfolio provided little, or even slightly negative excess returns for most bearish quarters.

[8]Despite of its less popularity, it is meaningful because our portfolio is an equal-weighted portfolio. Ohlson and Rosenberg [OR82] emphasized that the first index fund [BS74] was to trace an equal-weighted index.

Table 5.3: Quarterly returns of all quarters

Quarter	Training		Validation		Test		Benchmark		Excess return	
Date	Avg	Std	Avg	Std	Avg	Std	KOSPI	All stocks	Exc_K	Exc_A
May 2001	9.80	3.24	13.56	1.32	21.04	3.86	1.49	−2.18	19.55	23.22
Aug 2001	14.45	2.91	28.62	3.70	10.16	3.41	5.15	−1.6	5.01	11.76
Nov 2001	20.00	2.67	12.46	2.57	44.98	8.62	43.37	26.38	1.61	18.60
Apr 2002	22.84	1.37	48.09	6.79	2.66	10.95	−2.03	−8.32	4.69	10.98
May 2002	29.98	1.58	7.72	11.71	−13.72	2.27	−16.29	−19.96	2.57	6.24
Aug 2002	32.22	2.48	−11.15	2.84	−5.80	2.48	−6.31	−14.78	0.51	8.98
Nov 2002	17.38	2.72	−3.87	2.22	−13.34	3.33	−19.97	−19.93	6.63	6.59
Apr 2003	13.77	2.07	−9.92	1.77	20.55	2.91	13.42	17.09	7.13	3.46
May 2003	1.36	4.22	23.47	2.72	9.70	5.66	19.02	8.72	−9.32	0.98
Aug 2003	0.78	1.10	16.06	4.54	14.71	4.84	11.4	−3.46	3.31	18.17
Nov 2003	7.75	0.97	18.61	3.57	13.02	4.86	9	−1.73	4.02	14.75
Apr 2004	13.52	1.22	16.33	3.10	−4.53	2.24	−12.95	−3.56	8.42	−0.97
May 2004	20.97	1.20	−0.72	1.91	5.57	4.10	0.7	−8.16	4.87	13.73
Aug 2004	14.93	1.46	10.35	2.82	31.81	3.94	13.28	9.86	18.53	21.95
Nov 2004	12.49	2.20	33.35	3.55	56.78	8.63	12.01	40.87	44.77	15.91
Apr 2005	17.82	1.79	64.07	5.42	−3.18	2.07	−5.38	−1.23	2.20	−1.95
May 2005	25.68	1.75	3.38	2.48	40.12	15.14	20.22	30.62	19.90	9.50
Aug 2005	26.02	1.67	47.63	14.45	37.20	7.25	13.45	31.04	23.75	6.16
Nov 2005	43.74	4.23	50.62	3.15	1.88	4.14	7.3	8.08	−5.42	−6.20
Apr 2006	47.59	4.97	8.41	3.30	3.31	2.61	1.66	1.19	1.65	2.12
May 2006	34.89	4.43	8.78	4.74	−11.70	2.84	−4.81	−13.48	−6.89	1.78
Aug 2006	33.42	2.50	−8.17	1.82	20.13	4.26	7.23	13.38	12.90	6.75
Nov 2006	18.05	1.29	21.48	1.83	26.28	4.19	2.96	13.83	23.32	12.45
Apr 2007	12.61	1.59	35.71	4.69	19.54	2.80	10.19	12.48	9.35	7.06
May 2007	16.00	1.56	23.23	2.45	17.50	5.93	5.71	5.64	11.79	11.86
Aug 2007	18.92	1.95	24.44	5.20	13.13	9.16	13.84	2.72	−0.71	10.41
Nov 2007	28.40	2.70	17.96	8.93	−4.52	3.99	−11.63	−11.06	7.11	6.54
Apr 2008	27.92	2.82	0.90	2.72	9.40	2.18	10.96	5.25	−1.56	4.15
May 2008	17.36	1.73	12.79	1.74	−12.07	2.05	−16.77	−14.58	4.70	2.51
Aug 2008	15.13	1.29	−8.84	2.79	−39.42	3.78	−30.78	−38.88	−8.64	−0.54
Nov 2008	7.01	1.32	−32.87	3.26	33.21	6.91	13.33	27	19.88	6.21
Apr 2009	−7.39	1.22	40.07	3.64	33.02	4.44	12.84	31.96	20.18	1.06
May 2009	3.02	1.27	32.62	4.73	5.56	3.67	14.35	4.75	−8.79	0.81
Aug 2009	8.17	1.93	10.07	2.54	−2.86	3.01	0.07	−8.11	−2.93	5.25
Nov 2009	14.52	2.10	0.46	2.84	22.91	4.65	7.96	11.72	14.95	11.19
Apr 2010	27.24	1.37	26.90	3.09	7.05	7.30	−1.37	0.95	8.42	6.10
May 2010	21.45	1.60	6.60	3.11	−1.81	1.98	2.81	−2.86	−4.62	1.05
Aug 2010	14.67	1.75	1.23	1.72	19.17	3.30	8.94	5.27	10.23	13.90
Nov 2010	10.16	1.64	17.97	3.28	10.10	7.12	11.68	5.12	−1.58	4.98
Apr 2011	14.84	1.55	17.54	5.91	6.23	2.34	−0.79	−2.12	7.02	8.35
May 2011	13.18	1.22	9.28	2.04	−0.34	6.29	−10.66	−3.73	10.32	3.39
Aug 2011	15.00	1.17	6.87	3.22	−3.13	4.82	−1.27	−0.97	−1.86	−2.16
Nov 2011	18.27	1.82	3.45	3.21	35.82	9.82	8.51	17.78	27.31	18.04
Apr 2012	13.65	2.60	42.15	6.97	−6.20	1.88	−8.62	−9.04	2.42	2.84
May 2012	17.30	3.84	−3.69	1.11	9.41	2.55	6.38	7.28	3.03	2.13
Aug 2012	13.82	2.25	13.76	2.46	12.43	3.70	−4.96	−0.07	17.39	12.50
Nov 2012	14.17	2.22	15.95	2.53	23.48	4.52	7.26	16.48	16.22	7.00
Apr 2013	16.97	2.24	31.65	3.26	10.32	5.86	−0.46	5.39	10.78	4.93
May 2013	15.45	3.21	20.58	3.87	3.21	3.76	−3.36	−2.42	6.57	5.63
Aug 2013	22.97	2.84	6.84	6.38	1.53	2.03	4.45	−4.49	−2.92	6.02
Nov 2013	23.03	4.64	2.60	2.23	25.90	6.37	−0.68	9.95	26.58	15.95
Apr 2014	29.85	5.16	28.82	6.15	10.78	4.70	1.08	3.26	9.70	7.52
May 2014	18.72	2.02	11.69	2.60	13.27	5.01	2.47	2.99	10.80	10.28
Aug 2014	19.17	3.14	15.64	6.11	−3.02	1.92	−5.72	−0.32	2.70	−2.70
Average†	18.17	2.26	15.40	3.84	10.69	4.67	2.96	3.33	7.73	7.36

For each quarter, returns were averaged over 30 runs.
† Arithmetic average.

Interpretation of Solutions

Due to our direct encoding of weight and sorting indicator vectors, analyzing the solutions is rather straightforward. To give more insight into our problem, the solutions for each quarter were arithmetically averaged over 30 runs; they were then averaged over the entire period, i.e., 54 quarters. Figure 5.7 shows the average weight and sorting indicator vectors. PBR, PSR, and PER showed the highest weights, which is notable due to their simplicity and popularity. This is roughly consistent with the previous studies for Korean stock markets [MDK97, BRZ08].[9] In contrast, QR, CR, and DER showed the lowest weights; the differences among them were not significant.

For sorting indicator, an average value close to zero or one is more consistent than one close to 0.5. The sorting indicators close to 0 and 1 mean ascending and descending orders in our ranking function, respectively. For example, the average sorting indicator of PBR was around 0.013, which strongly suggests that the ranking by ascending order is more appropriate. The results showed that PBR, PSR, and PER strongly suggest ascending orders while NIG and OIG do descending ones. It is notable that the value for NIG is larger than OIG, which means that NIG is more consistent than OIG in terms of directionality over a long period of time; this is quite contrary to most fund managers' intuition that net income is less credible than operating one. The sorting indicators closest to 0.5 were CFR, NPM, and ITR whose values were 0.529, 0.535, and 0.465, respectively. Their values imply that the sorting direction is not very meaningful for them.

To find out the change of quarterly values over time, we plotted the quarterly weight and sorting indicator vectors as heatmaps. Figure 5.8 shows the weights and sorting indicators over time. PBR, PSR, and PER showed consistently large weights for most quarters; PBR was the most dominant factor among them. Most notable financial ratios were CR and QR which are related to liquidity or short-term solvency; they showed the recent increase in weight. We think that more profitable

[9] However, the significance of financial ratios was different across studies. For example, Mukherji *et al.* [MDK97] showed that PBR, PSR, and DER were significant for explaining Korean stock returns, while PER was not.

portfolios can be constructed by exploiting such recent changes in weight.

The results for sorting indicator vector were similar to those of weight vector. The sorting indicator of PBR was almost time-invariant; it strongly favors ascending order. While PBR, PSR, and PER supported ascending order, NIG and OIG directed descending one. Compared with the weight vectors, they showed more consistent results possibly due to their simpler binary encoding. It is notable that ITR and RTR, which are related to turnover rate, showed the recent increase in directionality. However, they showed fairly noisy patterns over time, which can also be explained by other factors such as periodicity.

Comparisons with Commercial Portfolios

Despite many studies in the finance literature reported their excess returns over buy-and-hold, most of them did not compare their returns with those by commercial portfolio algorithms. Such comparisons have been virtually impossible since most investment companies hide the details of asset management.

Fortunately, we could obtain a commercial preliminary portfolio from Optus Investments Inc.[10] Dynamic Portfolio^{TM},[11] one of the subsystems exploited in the first stage of portfolio selection at Optus Investments Inc., was used for constructing the portfolio.[12] It uses several hundreds of inputs including macroeconomic, fundamental, and technical variables. The inputs can be modularized, feedbacked, or replaced by some criteria; they can have even general expressions including arithmetic, Boolean, and so forth. Table 5.4 shows some representative settings of Dynamic Portfolio^{TM}.

The average quarterly compound returns by our GA, Dynamic Portfolio^{TM}, and buy-and-hold (KOSPI) were 9.36%,[13] 15.44%, and 2.29%, respectively. It seems that

[10] http://www.optus.co.kr

[11] In fact, it is a preliminary portfolio universe screening system. Note that its output is not the operating portfolio in Optus Investments Inc. In general, operating portfolio is constructed by interactions among more complex, multi-layered systems.

[12] We are not able to provide its detailed algorithm due to business contract.

[13] This return is slightly larger than the best average return in Table 5.2 due to its arithmetic averaging of 30 runs' portfolio returns for each quarter, which contributes to a smaller variance of quarterly returns, resulting in the larger average compound return.

Table 5.4: Representative settings of Dynamic Portfolio^{TM}

Feature	Setting
Inputs	macroeconomic, financial, and technical variables
- Number of inputs	several hundreds
- Modularization of inputs	N
- Feedback inputs	N
- Replacement for inputs	N
- General expression for inputs	N
Initial seed	approximation
Genetic operators	-
Local optimization	iterative improvement with feature selection
Time complexity	$\Theta(K^2)$ to $\Theta(K)$ (reducible to even sublinear)
Training method	extending window
Intra-quarter transactions	N
Sectoral filters	N
Liquidity filters	N
Additional information	N
- Sectoral information	N
- Holding company information	N

The symbol "-" means the feature is not supported.
The number of inputs is represented by K.

beating the commercial portfolio is very hard due to the different degrees of abstraction level, optimizations, coverages of inputs, and so forth. However, it is notable that our GA was able to provide a large excess return over buy-and-hold, in particular with much smaller number of inputs and less complex framework. In addition, the performance pattern to the market context was similar to that of Dynamic Portfolio^{TM}, which implies that our portfolio captured important features of obtaining excess returns in the commercial preliminary portfolio.

5.3.5 Summary

We presented a hybrid genetic algorithm for large-scale stock selection. To include all types of financial ratios including positive, negative, and invalid ones, we formulated the stock selection in a systematic way. Due to the interpretational inconsistency of negative financial ratios, the ranking with partitions was proposed. It partitions financial ratios into positive, negative, and invalid ones, where each stock is assigned

a rank score based on its matching partition. The ranking with partitions provided significant improvements over the linear ranking using a naive sort.

To further improve upon the performance, a heuristic seeding algorithm was presented. In our problem, the sorting indicator vector was heuristically seeded due to its less complex encoding in comparison to the weight vector. The initial population seeded by our seeding algorithm showed significant improvements, which substantially dominated buy-and-hold. Our results imply that typical stock selection with binary sorting indicator can be efficiently solved by a multistage optimization framework.

By analyzing the solutions, we found that the popular financial ratios including PBR, PSR, and PER have larger weights than the others. The least significant financial ratios were QR, CR, and DER. It is notable that CR and QR showed the recent increase in weight. For sorting indicator, PBR showed an almost time-invariant consistency, which implies that its direction for attractiveness is more consistently identified. It is interesting that the directionalities of ITR and RTR were increased in the recent quarters.

Despite of our significant excess returns, we found that our portfolio is not comparable to a commercial preliminary portfolio provided by Optus Investments Inc. We suspect that the different levels of abstraction, optimizations, and coverages of inputs are key factors describing the performance gap. However, the similar performance pattern to the market context implies that we were able to capture core features of obtaining excess returns in the commercial portfolio. We believe that our GA is still promising due to its scalability and efficiency.

Future work would include multi-objective modeling [BFF08] and stock scoring reflecting accounting particularities [XMP09]. In particular, we think that reflecting accounting particularities is quite important for more effective stock selection. Extending our framework to include a large number of stock markets in various countries is also promising. The market differences across countries, however, would require more elaborate financial ratios that incorporate information asymmetries [SHH12] among different countries.

Input:
K: the length of each vector
h_n: the number of samples
h_v: the ratio of voting samples
h_p: the ratio of seeded individuals in the population
P: the initial population
Output:
P: the seeded initial population
Description:
HeuristicSeeding(P,h_n,h_v,h_p,K)
{
    // c: a temporary chromosome
    // $c.w$: w-vector in c
    // $c.b$: b-vector in c
    // L_s: a sorted list of $< f, c.b >$ with descending order of f
    for $i \leftarrow 1$ **to** K
        $c.w[i] \leftarrow \frac{1}{K}$;   // geometric center
    for $i \leftarrow 1$ **to** h_s
        for $j \leftarrow 1$ **to** K
            $c.b[j] \leftarrow$ a random integer from $\{0, 1\}$;
        $f \leftarrow$ calculate fitness of chromosome c;
        insert $< f, c.b >$ to L_s;
    $v \leftarrow \lfloor h_n \times h_v \rfloor$;   // voting samples
    for $i \leftarrow 1$ **to** K
        $s[i] \leftarrow 0$;
    for $i \leftarrow 1$ **to** v
        $c.b \leftarrow c.b$ in $L_s[i]$;   // select from the largest f
        for $j \leftarrow 1$ **to** K
            $s[j] \leftarrow s[j] + c.b[j]$;
    for $i \leftarrow 1$ **to** K // majority voting
        if $s[i] > \lfloor \frac{v}{2} \rfloor$
            $b[i] \leftarrow 1$;
        else
            $b[i] \leftarrow 0$;
    $P_s \leftarrow \lfloor h_p \times |P| \rfloor$ random indices from $\{1, 2, \ldots, |P|\}$;
    for each $c \in P_s$
        for $j = 1$ to K
            $c.b[j] \leftarrow b[j]$;
    return P;
}

Figure 5.1: Heuristic seeding

Input:
K: dimension, or the number of financial ratios
Output:
$\vec{w}$: a random n-dimensional weight vector
Description:
UnbiasedSamplingRandomWeight(K)
{
    for $i \leftarrow 1$ **to** $K+1$ // w^r: temporary vector
        if $i = 1$
            $w_i^r \leftarrow 0$;
        else if $i = K+1$
            $w_i^r \leftarrow 1$;
        else
            $w_i^r \leftarrow$ a random real number from $[0, 1]$;
    $\vec{w}^r \leftarrow$ Sort $\vec{w}^r$ in ascending order;
    for $i = 1$ to K
        $w_i \leftarrow w_{i+1}^r - w_i^r$;
    return $\vec{w}$;
}

Figure 5.2: Unbiased sampling for random weight

Input:
c: chromosome
n_h: neighborhood size
d: step size for increase/decrease
Output:
c: locally optimized chromosome
Description:

```
LocalOptimization(c,n_h,d)
{
    f_o ← calculate fitness of chromosome c;
    g_max ← 0;
    for i ← 1 to n_h
        s ← a random integer from [1,K];
        t ← c.si[s];
        c.si[s] ← 1 − c.si[s];
        f ← calculate fitness of chromosome c;
        g ← f − f_o;
        c.si[s] ← t;
        if g > g_max
            g_max ← g;
            b_max ← 1 − c.si[s];
            s_max ← s;
    if g_max > 0
        c.si[s_max] ← b_max;
        f_o ← f_o + g_max;
    g_max ← 0;
    t_arr ← copy the array c.w;
    for i ← 1 to n_h
        s ← a random integer from [1,K];
        c.w[s] ← min(1.0, c.w[s] + d);      // increase
        Normalize c.w so that Σ_{j=1}^{K} c.w[j] = 1;
        f ← calculate fitness of chromosome c;
        g ← f − f_o;
        c.w ← copy the array t_arr;
        if g > g_max
            g_max ← g;
            w_max ← min(1.0, c.w[s] + d);
            s_max ← s;
        c.w[s] ← max(0.0, c.w[s] − d);      // decrease
        Normalize c.w so that Σ_{j=1}^{K} c.w[j] = 1;
        f ← calculate fitness of chromosome c;
        g ← f − f_o;
        c.w ← copy the array t_arr;
        if g > g_max
            g_max ← g;
            w_max ← max(0.0, c.w[s] − d);
            s_max ← s;
    if g_max > 0
        c.w[s_max] ← w_max;
    Normalize c.w so that Σ_{j=1}^{K} c.w[j] = 1;
    return c;
}
```

Figure 5.3: Local optimization

Input:
$\vec{v}_j^t$: a column vector of the values of financial ratio j at time t
δ: special constant denoting the invalid rank score ($\delta > 0$)
ζ: special constant denoting the invalid financial ratio
($\zeta \gg v_{ij}^t$ for all $i \in L^t$, where L^t is the set of valid stocks)
I: sorting indicator (0: ascending, 1: descending) only for positive values
Output:
$\vec{s}_j^t$: a column vector of the rank scores of financial variable j at time t
Description:
RankingWithPartitions($\vec{v}_j^t$,δ,ζ,I)
{
    $q \leftarrow$ Partition($\vec{v}_j^t$,1,N,ζ);
    $p \leftarrow$ Partition($\vec{v}_j^t$,1,q,0);
    // $[1\,..\,p]$: negative, $[p+1\,..\,q]$: positive, $[q+1\,..\,N]$: invalid
    if $I = 0$
        $v_p \leftarrow$ sort $v_{(p+1)j}^t, v_{(p+2)j}^t, \ldots, v_{qj}^t$ in ascending order;
    else
        $v_p \leftarrow$ sort $v_{(p+1)j}^t, v_{(p+2)j}^t, \ldots, v_{qj}^t$ in descending order;
    // v_n: does not dependent upon I
    $v_n \leftarrow$ sort $v_{1j}^t, v_{2j}^t, \ldots, v_{pj}^t$ in descending order;
    for $i \leftarrow 1$ **to** N
        if $v_{ij}^t \in v_p$
            $s_{ij}^t \leftarrow (index[v_{ij}^t, v_p] - 1)/q$;
        else if $v_{ij}^t \in v_n$
            $s_{ij}^t \leftarrow (index[v_{ij}^t, v_n] - 1 + p)/q$;
        else
            $s_{ij}^t \leftarrow 1 + \delta$;
    return $\vec{s}_j^t$;
}

Figure 5.4: Ranking with partitions

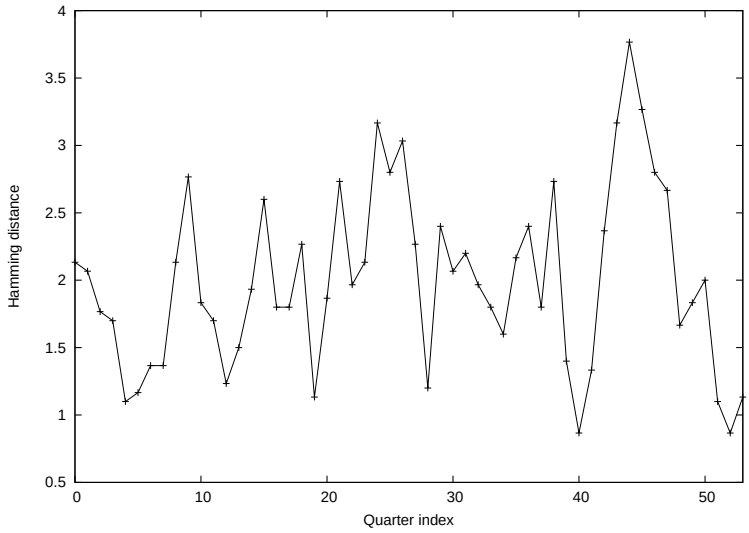


Figure 5.5: Distance between the initial and final sorting indicator vectors

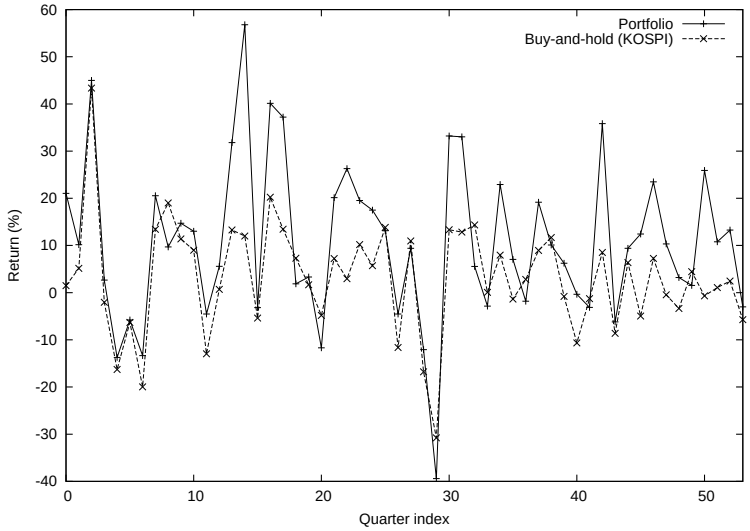
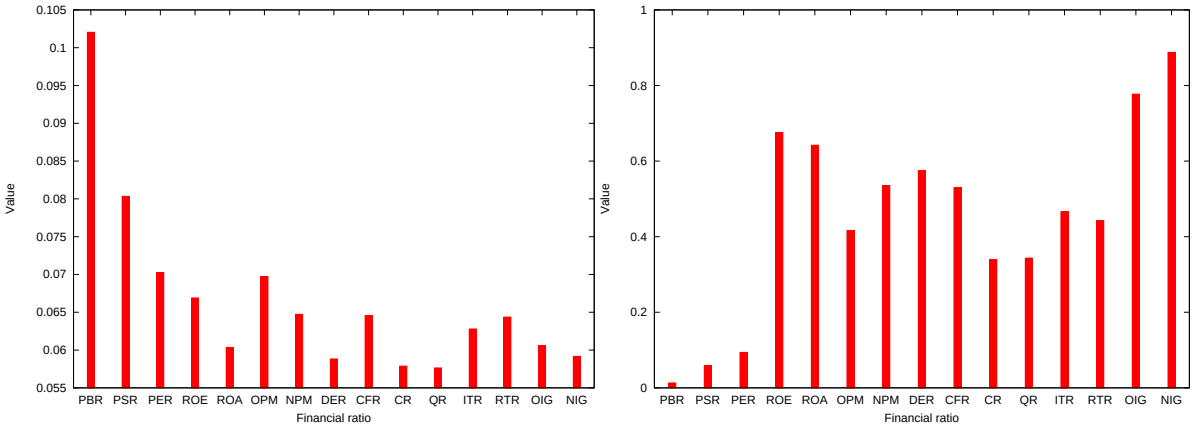
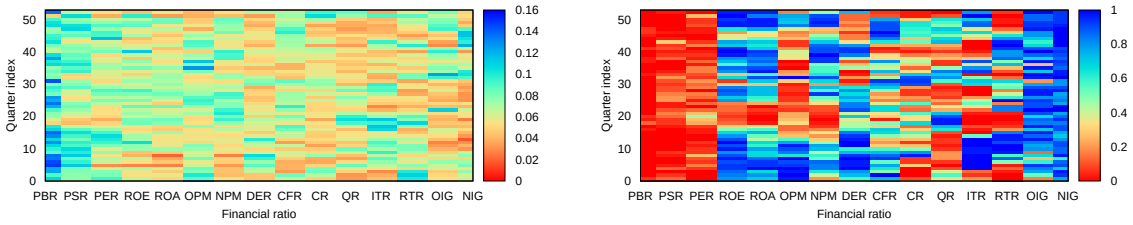


Figure 5.6: Comparison of quarterly returns between portfolio and buy-and-hold



(a) Weight (b) Sorting indicator

Figure 5.7: Average weight and sorting indicator over the entire period



(a) Weight (b) Sorting indicator

Figure 5.8: Heatmaps for weights and sorting indicators

Chapter 6

Conclusions

In this thesis, we presented evolutionary algorithms using effective search space reduction for financial optimization problems. Due to the large dimensionality of such problems, we devised three effective search space reductions: modular, grammatical, and seeded evolutions. Modular and grammatical evolutions restrict the possible forms of solutions using prior domain knowledge by using modules and context-free grammar, respectively. They were exploited in genetic programming where the forms of solutions are generally not restricted. In contrast, seeded evolution provides the initial population with partially optimized solutions, which is exploited in genetic algorithm. While all of these evolutions are different in reflecting domain knowledge, they have the common principle of focusing on more promising solutions in their evolutions.

In Chapter 3, we described a modular evolution, where the possible forms of solutions are restricted by module patterns. The restriction was statically determined by the set of module patterns using prior domain knowledge. Genetic operators which can disrupt the module patterns were modified to modular ones, which always preserve the module patterns. We showed that our modular genetic programming is quite effective for attractive technical pattern discovery.

In Chapter 4, we proposed a grammatical evolution, where solutions are generated by a context-free grammar. The grammar also requires domain knowledge, but it defines more free forms of solutions than modular evolution. Similar to modular

evolution, genetic operators were modified to be aware of the underlying grammar. We also extended attractive technical pattern discovery using more relevant features. The grammatically typed genetic programming was fairly effective for the extended problem.

In Chapter 5, we introduced a seeded evolution, where partially optimized solutions are provided to the initial population. Since most genetic algorithms use a fixed-length encoding, where typing of solutions is not meaningful, we directly seeded the initial population with promising solutions. To maintain the diversity in the population, the seeding solutions were partially optimized. The candidate genes for such optimization were selected primarily by domain knowledge including encoding complexity. The usefulness of this seeded evolution was empirically validated by large-scale stock selection problem.

While our search space reductions were quite effective for the selected problems, there are still a wide variety of problems to be investigated, even in the computational finance. Since their common idea is to focus on promising solutions using domain knowledge, they are more readily applicable only to problems with abundant domain knowledge. They need to be further studied to find out their fundamental benefits and disadvantages using various problems. There are also a number of issues in finding, organizing, and reusing domain knowledge. In financial optimization problems, they become more complex due to the huge dimensionality and non-stationarity of time series data. Devising more effective and efficient evolutionary algorithms, which are knowledge-directed, for diverse financial optimization problems is left for further study.

Bibliography

[AI09] C. Aranha and H. Iba. The memetic tree-based genetic algorithm and its application to portfolio optimization. *Memetic Computing*, 1(2):139–151, 2009.

[AK99] F. Allen and R. Karjalainen. Using genetic algorithms to find technical trading rules. *Journal of financial Economics*, 51(2):245–271, 1999.

[Ale61] S. S. Alexander. Price movements in speculative markets: trends or random walks. *Industrial Management Review*, 2(2):7, 1961.

[Ale64] S. S. Alexander. Price movements in speculative markets: trends or random walks, number 2. *Industrial Management Review*, 5(2):25–46, 1964.

[AOB10] A. Agapitos, M. O'Neill, and A. Brabazon. Evolutionary learning of technical trading rules without data-mining bias. In *Proceedings of the Conference on Parallel Problem Solving from Nature*, pages 294–303. Springer, 2010.

[AP93] P. J. Angeline and J. Pollack. Evolutionary module acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–163, 1993.

[AR00] S. P. Abeysekera and E. S. Rosenbloom. A simulation model for deciding between lump-sum and dollar-cost averaging. *Journal of Financial Planning*, 13(6):86–96, 2000.

[Bäc96] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[Bar87] P. Barnes. The analysis and use of financial ratios: a review article. *Journal of Business Finance & Accounting*, 14(4):449–461, 1987.

[Bas77] S. Basu. Investment performance of common stocks in relation to their price-earnings ratios: a test of the efficient market hypothesis. *Journal of Finance*, 32(3):663–682, 1977.

[Bas83] S. Basu. The relationship between earnings' yield, market value and return for NYSE common stocks: further evidence. *Journal of Financial Economics*, 12(1):129–156, 1983.

[BB68] R. Ball and P. Brown. An empirical evaluation of accounting income numbers. *Journal of Accounting Research*, 6(2):159–178, 1968.

[BB07] M. F. Brameier and W. Banzhaf. *Linear Genetic Programming*. Springer, 2007.

[BFF08] Y. L. Becker, H. Fox, and P. Fei. An empirical study of multi-objective algorithms for stock ranking. In *Genetic Programming Theory and Practice V*, pages 239–259. Springer, 2008.

[BFL07] Y. L. Becker, P. Fei, and A. M. Lester. Stock selection: an innovative application of genetic programming methodology. In *Genetic Programming Theory and Practice IV*, pages 315–334. Springer, 2007.

[BHJ09] M. Bastian, S. Heymann, and M. Jacomy. Gephi: an open source software for exploring and manipulating networks. In *Proceedings of the Third International ICWSM Conference*, pages 361–362, 2009.

[BLL92] W. Brock, J. Lakonishok, and B. LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, 47(5):1731–1764, 1992.

[BNKF98] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction*. Morgan Kaufmann, 1998.

[BO04] A. Brabazon and M. O'Neill. Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Computational Management Science*, 1(3):311–327, 2004.

[Bol86] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.

[BPZ02] S. Bhattacharyya, O. V. Pictet, and G. Zumbach. Knowledge-intensive genetic discovery in foreign exchange markets. *IEEE Transactions on Evolutionary Computation*, 6(2):169–181, 2002.

[Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[BRZ08] S. Brown, S. G. Rhee, and L. Zhang. The return to value in Asian stock markets. *Emerging Markets Review*, 9(3):194–205, 2008.

[BS74] F. Black and M. Scholes. From theory to a new financial product. *Journal of Finance*, 29(2):399–412, 1974.

[CB05] M. Caplan and Y. Becker. Lessons learned using genetic programming in a stock picking context. In *Genetic Programming Theory and Practice II*, pages 87–102. Springer, 2005.

[CBO10] W. Cui, A. Brabazon, and M. O'Neill. Evolving efficient limit order strategy using grammatical evolution. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 1–6, 2010.

[Che13] S.-H. Cheng. An intelligent stock-selecting system based on decision tree combining rough sets theory. In *Recent Trends in Applied Artificial Intelligence*, pages 501–508. Springer, 2013.

[CIW92] A. L. Corcoran III and R. L. Wainwright. A genetic algorithm for packing in three dimensions. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1021–1030, 1992.

[CJHNL12] I. Contreras, Y. Jiang, J. I. Hidalgo, and L. Núñez-Letamendia. Using a GPU-CPU architecture to speed up a GA-based real-time system for trading the stock market. *Soft Computing*, 16(2):203–215, 2012.

[CK02] S.-H. Chen and T.-W. Kuo. Evolutionary computation in economics and finance: a bibliography. In *Evolutionary Computation in Economics and Finance*, pages 419–455. Springer, 2002.

[CKH08] S.-H. Chen, T.-W. Kuo, and K.-M. Hoi. Genetic programming and financial trading: how much about "what we know". In *Handbook of Financial Engineering*, pages 99–154. Springer, 2008.

[CM03] S.-S. Choi and B.-R. Moon. Normalization in genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 862–873, 2003.

[CM08] S.-S. Choi and B.-R. Moon. Normalization for genetic algorithms with nonsynonymously redundant encodings. *IEEE Transactions on Evolutionary Computation*, 12(5):604–616, 2008.

[CMBS00] T.-J. Chang, N. Meade, J. E. Beasley, and Y. M. Sharaiha. Heuristics for cardinality constrained portfolio optimisation. *Computers & Operations Research*, 27(13):1271–1302, 2000.

[CN06] S.-H. Chen and N. Navet. Pretests for genetic-programming evolved trading programs: "zero-intelligence" strategies and lottery trading. In *Neural Information Processing*, pages 450–460, 2006.

[Cou78] J. K. Courtis. Modelling a financial ratios categoric framework. *Journal of Business Finance & Accounting*, 5(4):371–386, 1978.

[CT01] L. Cao and F. E. Tay. Financial forecasting using support vector machines. *Neural Computing & Applications*, 10(2):184–192, 2001.

[CTAM09] S. Chiam, K. C. Tan, and A. Al Mamun. Investigating technical trading strategy via an multi-objective evolutionary platform. *Expert Systems with Applications*, 36(7):10408–10423, 2009.

[CY97] S.-H. Chen and C.-H. Yeh. Toward a computable approach to the efficient market hypothesis: an application of genetic programming. *Journal of Economic Dynamics and Control*, 21(6):1043–1063, 1997.

[CY01] S.-H. Chen and C.-H. Yeh. Evolving traders and the business school with genetic programming: a new architecture of the agent-based artificial stock market. *Journal of Economic Dynamics and Control*, 25(3):363–393, 2001.

[DE13] R. DeSantiago and J. Estrada. Geometric mean maximization: an overlooked portfolio approach. *The Journal of Investing*, 22:109–119, 2013.

[DeJ75] K. A. DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, Ann Arbor, MI, 1975.

[DH09] M. Dooley and M. Hutchison. Transmission of the US subprime crisis to emerging markets: evidence on the decoupling–recoupling hypothesis. *Journal of International Money and Finance*, 28(8):1331–1349, 2009.

[DJ01] M. A. H. Dempster and C. M. Jones. A real-time adaptive trading system using genetic programming. *Quantitative Finance*, 1(4):397–413, 2001.

[DT85] W. F. DeBondt and R. Thaler. Does the stock market overreact? *Journal of Finance*, 40(3):793–805, 1985.

[DVH11] Y. Demyanyk and O. Van Hemert. Understanding the subprime mortgage crisis. *Review of Financial Studies*, 24(6):1848–1880, 2011.

[DY03] C. Donohue and K. Yip. Optimal portfolio rebalancing with transaction costs. *The Journal of Portfolio Management*, 29(4):49–63, 2003.

[EGB96] E. J. Elton, M. J. Gruber, and C. R. Blake. Survivor bias and mutual fund performance. *Review of Financial Studies*, 9(4):1097–1120, 1996.

[EM11] A. Esfahanipour and S. Mousavi. A genetic programming model to generate risk-adjusted technical trading rules in stock markets. *Expert Systems with Applications*, 38(7):8438–8445, 2011.

[EMB07] R. D. Edwards, J. Magee, and W. H. C. Bassetti. *Technical Analysis of Stock Trends*. CRC Press, 2007.

[ES07] M. Eling and F. Schuhmacher. Does the choice of performance measure influence the evaluation of hedge funds? *Journal of Banking & Finance*, 31(9):2632–2647, 2007.

[Fam70] E. F. Fama. Efficient capital markets: a review of theory and empirical work. *Journal of Finance*, 25(2):383–417, 1970.

[FB66] E. F. Fama and M. E. Blume. Filter rules and stock-market trading. *Journal of Business*, 39(1):226–241, 1966.

[FH96] H. Furuya and R. T. Haftka. Combining genetic and deterministic algorithms for locating actuators on space structures. *Journal of Spacecraft and Rockets*, 33(3):422–427, 1996.

[FM96] B. Freisleben and P. Merz. New genetic local search operators for the traveling salesman problem. In *Proceedings of the Conference on Parallel Problem Solving from Nature*, pages 890–899. Springer, 1996.

[FMT05] C. Fyfe, J. P. Marney, and H. Tarbert. Risk adjusted returns from technical trading: a genetic programming approach. *Applied Financial Economics*, 15(15):1073–1077, 2005.

[FOW66] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.

[GDK91] D. E. Goldberg, K. Deb, and B. Korb. Don't worry, be messy. In *Proceedings of the International Conference on Genetic Algorithms*, pages 24–30, 1991.

[GGW04] I. I. Garibay, O. O. Garibay, and A. S. Wu. Effects of module encapsulation in repetitively modular genotypes on the search space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1125–1137, 2004.

[Gil90] S. L. Gillan. An investigation into CAPM anomalies in New Zealand: the small firm and price-earnings ratio effects. *Asia Pacific Journal of Management*, 7(2):63–78, 1990.

[GL99] S. C. Gold and P. Lebowitz. Computerized stock screening rules for portfolio selection. *Financial Services Review*, 8(2):61–70, 1999.

[GMS+09] A. Ghandar, Z. Michalewicz, M. Schmidt, T.-D. Tô, and R. Zurbrugg. Computational intelligence for evolving trading rules. *IEEE Transactions on Evolutionary Computation*, 13(1):71–86, 2009.

[GNH11] A. Gorgulho, R. Neves, and N. Horta. Applying a GA kernel on optimizing technical analysis rules for stock picking and portfolio composition. *Expert Systems with Applications*, 38(11):14072–14085, 2011.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[Gre87] J. J. Grefenstette. Incorporating problem specific knowledge into genetic algorithms. In *Genetic Algorithms and Simulated Annealing*, pages 42–60. Morgan Kaufmann, 1987.

[GSPT06] C. Gagné, M. Schoenauer, M. Parizeau, and M. Tomassini. Genetic programming, validation sets, and parsimony pressure. In *Genetic Programming*, pages 109–120. Springer, 2006.

[GT92] M. Grinblatt and S. Titman. The persistence of mutual fund performance. *Journal of Finance*, 47(5):1977–1984, 1992.

[GW07] O. O. Garibay and A. S. Wu. Analyzing the effects of module encapsulation on search space bias. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1234–1241, 2007.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[Han05] P. R. Hansen. A test for superior predictive ability. *Journal of Business & Economic Statistics*, 23(4):365–380, 2005.

[Has08] G. Hassan. Non-linear factor model for asset selection using multi objective genetic programming. In *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation*, pages 1859–1862, 2008.

[HCCC12] C.-F. Huang, B. R. Chang, D.-W. Cheng, and C.-H. Chang. Feature selection and parameter optimization of a fuzzy-based stock selection model using genetic algorithms. *International Journal of Fuzzy Systems*, 14(1):65–75, 2012.

[HL70] G. Hanoch and H. Levy. Efficient portfolio selection with quadratic and cubic utility. *Journal of Business*, 43(2):181–189, 1970.

[HLV10] J. How, M. Ling, and P. Verhoeven. Does size matter? a genetic programming approach to technical trading. *Quantitative Finance*, 10(2):131–140, 2010.

[HLY07] J. How, J. Lam, and J. Yeo. The use of the comparable firm approach in valuing Australian IPOs. *International Review of Financial Analysis*, 16(2):99–115, 2007.

[HNV01] J. Horst, T. Nijman, and M. Verbeek. Eliminating look-ahead bias in evaluating persistence in mutual fund performance. *Journal of Empirical Finance*, 8(4):345–373, 2001.

[Hol75] J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[Hor68] J. O. Horrigan. A short history of financial ratio analysis. *Accounting Review*, 43(2):284–294, 1968.

[Hua12] C.-F. Huang. A hybrid stock selection model using genetic algorithms and support vector regression. *Applied Soft Computing*, 12(2):807–818, 2012.

[HYC08] C.-J. Huang, D.-X. Yang, and Y.T. Chuang. Application of wrapper approach and composite classifier to the stock trend prediction. *Expert Systems with Applications*, 34(4):2870–2878, 2008.

[HZ95] C. Hurson and C. Zopounidis. On the use of multi-criteria decision aid methods to portfolio selection. *Journal of Euro-Asian Management*, 1(2):69–94, 1995.

[IK11] R. Israelov and M. Katz. To trade or not to trade? informed trading with short-term signals for long-term investors. *Financial Analysts Journal*, 67(5):23–36, 2011.

[JB70] M. C. Jensen and G. A. Benington. Random walks and technical theories: some additional evidence. 25(2):469–482, 1970.

[Jen68] M. C. Jensen. The performance of mutual funds in the period 1945–1964. *Journal of Finance*, 23(2):389–416, 1968.

[JG07] D. Jackson and A. P. Gibbons. Layered learning in Boolean GP problems. In *Genetic Programming*, pages 148–159. Springer, 2007.

[Jin05] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1):3–12, 2005.

[JKW89] J. Jaffe, D. B. Keim, and R. Westerfield. Earnings yields, market values, and stock returns. *Journal of Finance*, 44(1):135–148, 1989.

[JM00] S. Jung and B.-R. Moon. The natural crossover for the 2D Euclidean TSP. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1003–1010, 2000.

[JT01] N. Jegadeesh and S. Titman. Profitability of momentum strategies: an evaluation of alternative explanations. *Journal of Finance*, 56(2):699–720, 2001.

[Jul94] B. A. Julstrom. Seeding the population: improved performance in a genetic algorithm for the rectilinear steiner problem. In *Proceedings of the ACM Symposium on Applied Computing*, pages 222–226, 1994.

[Kam13] M. Kampouridis. An initial investigation of choice function hyper-heuristics for the problem of financial forecasting. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 2406–2413, 2013.

[Kau98] P. J. Kaufman. *Trading Systems and Methods*. Wiley, 1998.

[Kau10] M. Kaucic. Investment using evolutionary learning methods and technical rules. *European Journal of Operational Research*, 207(3):1717–1727, 2010.

[KCT11] M. Kampouridis, S.-H. Chen, and E. Tsang. Investigating the effect of different GP algorithms on the non-stationary behavior of financial markets. In *Proceedings of the IEEE Conference on Computational Intelligence for Financial Engineering & Economics*, pages 1–8, 2011.

[Kes96] L. N. Kestner. Getting a handle on true performance. *Futures*, 25(1):44–46, 1996.

[KH00] K.-J. Kim and I. Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*, 19(2):125–132, 2000.

[Kim03] K.-J Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003.

[KK05] E. Keedwell and S.-T. Khu. A hybrid genetic algorithm for the design of water distribution networks. *Engineering Applications of Artificial Intelligence*, 18(4):461–472, 2005.

[KKM03] Y.-H. Kim, Y.-K. Kwon, and B.-R. Moon. Problem-independent schema synthesis for genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1112–1122, 2003.

[KM95] A. B. Kahng and B.-R. Moon. Toward more powerful recombinations. In *Proceedings of the International Conference on Genetic Algorithms*, pages 96–103, 1995.

[KM01] Y.-K. Kwon and B.-R. Moon. Personalized email marketing with a genetic programming circuit model. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1352–1358, 2001.

[KM07] Y.-K. Kwon and B.-R. Moon. A hybrid neurogenetic approach for stock forecasting. *IEEE Transactions on Neural Networks*, 18(3):851–864, 2007.

[Koz92] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Koz94] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[Kra01] K. Krawiec. Genetic programming with local improvement for visual learning from examples. In *Proceedings of the 9th International Conference on Computer Analysis of Images and Patterns*, pages 209–216, 2001.

[KS02] Y. Kabanov and C. Stricker. On the optimal portfolio for the exponential utility maximization: remarks to the six-author paper. *Mathematical Finance*, 12(2):125–134, 2002.

[KT10] M. Kampouridis and E. Tsang. EDDIE for investment opportunities forecasting: extending the search space of the GP. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 1–8, 2010.

[LAOK10] S. J. Lee, J. J. Ahn, K. J. Oh, and T. Y. Kim. Using rough set to support investment strategies of real-time trading in futures market. *Applied Intelligence*, 32(3):364–377, 2010.

[LAP99] B. LeBaron, W. B. Arthur, and R. Palmer. Time series properties of an artificial stock market. *Journal of Economic Dynamics and Control*, 23(9):1487–1516, 1999.

[LC10] D. Lohpetch and D. Corne. Outperforming buy-and-hold with evolved technical trading rules: daily, weekly and monthly trading. In *Applications of Evolutionary Computation*, pages 171–181. Springer, 2010.

[Lin65a] J. Lintner. Security prices, risk, and maximal gains from diversification. *Journal of Finance*, 20(4):587–615, 1965.

[Lin65b] J. Lintner. The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *The Review of Economics and Statistics*, 47(1):13–37, 1965.

[Lip07] P. Lipinski. ECGA vs. BOA in discovering stock market trading experts. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 531–538, 2007.

[LJ99] K.H. Lee and G.S. Jo. Expert system for predicting stock market timing using a candlestick chart. *Expert Systems with Applications*, 16(4):357–364, 1999.

[LK95] S.-B. Lee and K.-J. Kim. The effect of price limits on stock price volatility: empirical evidence in Korea. *Journal of Business Finance & Accounting*, 22(2):257–267, 1995.

[LL08] C.-C. Lin and Y.-T. Liu. Genetic algorithms for portfolio selection problems with minimum transaction lots. *European Journal of Operational Research*, 185(1):393–404, 2008.

[LM90] A. W. Lo and A. C. MacKinlay. Data-snooping biases in tests of financial asset pricing models. *Review of Financial Studies*, 3(3):431–467, 1990.

[LM10] S.-K. Lee and B.-R. Moon. A new modular genetic programming for finding attractive technical patterns in stock markets. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1219–1226, 2010.

[LM15] S.-K. Lee and B.-R. Moon. A hybrid genetic programming for finding attractive technical patterns in stock markets. Submitted, 2015.

[LMM15] S.-K. Lee, S.-H. Moon, and B.-R. Moon. A hybrid genetic algorithm for large-scale stock selection. In preparation, 2015.

[LMW00] A. W. Lo, H. Mamaysky, and J. Wang. Foundations of technical analysis: computational algorithms, statistical inference, and empirical implementation. *Journal of Finance*, 55(4):1705–1765, 2000.

[LP00] T. J. Linsmeier and N. D. Pearson. Value at risk. *Financial Analysts Journal*, 56(2):47–67, 2000.

[Mar52] H. Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77–91, 1952.

[Mas03] S. J. Masters. Rebalancing. *The Journal of Portfolio Management*, 29(3):52–57, 2003.

[MB04] K. Mehta and S. Bhattacharyya. Adequacy of training data for evolutionary mining of trading rules. *Decision Support Systems*, 37(4):461–474, 2004.

[MC03] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, pages 105–144. Springer, 2003.

[MDK97] S. Mukherji, M. S. Dhatt, and Y. H. Kim. A fundamental analysis of Korean stock returns. *Financial Analysts Journal*, 53(3):75–80, 1997.

[MHF08] E. Mabrouk, A.-R. Hedar, and M. Fukushima. Memetic programming with adaptive local search using tree data structures. In *Proceedings of the 5th International Conference on Soft Computing as Transdisciplinary Science and Technology*, pages 258–264, 2008.

[Mic96] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[Mon95] D. J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995.

[Mor92] G. Morris. *Candlestick Charting Explained*. McGraw-Hill, 1992.

[Mos66] J. Mossin. Equilibrium in a capital asset market. *Econometrica*, 34(4):768–783, 1966.

[Mos89] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program, California Institute of Technology, 1989.

[MS99] R. Mansini and M. G. Speranza. Heuristic algorithms for the portfolio selection problem with minimum transaction lots. *European Journal of Operational Research*, 114(2):219–233, 1999.

[Mur99] J. J. Murphy. *Technical Analysis of the Financial Markets*. New York Institute of Finance, 1999.

[MW12] D. McKenney and T. White. Stock trading strategy creation using GP on GPU. *Soft Computing*, 16(2):247–259, 2012.

[MYC08] B. R. Marshall, M. R. Young, and R. Cahan. Are candlestick technical trading strategies profitable in the Japanese equity market? *Review of Quantitative Finance and Accounting*, 31(2):191–207, 2008.

[MYR06] B. R. Marshall, M. R. Young, and L. C. Rose. Candlestick technical trading strategies: can they create value for investors? *Journal of Banking & Finance*, 30(8):2303–2323, 2006.

[MYR07] B. R. Marshall, M. R. Young, and L. C. Rose. Market timing with candlestick technical analysis. *Journal of Financial Transformation*, 20(1):18–25, 2007.

[Nee03] C. J. Neely. Risk-adjusted, ex ante, optimal technical trading rules in equity markets. *International Review of Economics & Finance*, 12(1):69–87, 2003.

[Nis91] S. Nison. *Japanese Candlestick Charting Techniques: A Contemporary Guide to the Ancient Investment Techniques of the Far East*. New York Institute of Finance, 1991.

[Noa09] A. Noack. Modularity clustering is force-directed layout. *Physical Review E*, 79(2):026102, 2009.

[NV92] A. E. Nix and M. D. Vose. Modeling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence*, 5(1):79–88, 1992.

[NWD97] C. Neely, P. Weller, and R. Dittmar. Is technical analysis in the foreign exchange market profitable? a genetic programming approach. *Journal of financial and Quantitative Analysis*, 32(4):405–426, 1997.

[OC01] S. Oman and P. Cunningham. Using case retrieval to seed genetic algorithms. *International Journal of Computational Intelligence and Applications*, 1(1):71–82, 2001.

[OLLZ05] J. O, J. Lee, J. W. Lee, and B.-T. Zhang. Dynamic asset allocation for stock trading optimized by evolutionary computation. *IEICE Transactions on Information and Systems*, 88(6):1217–1223, 2005.

[OR82] J. Ohlson and B. Rosenberg. Systematic risk of the CRSP equal-weighted common stock index: a history estimated by stochastic-parameter regression. *Journal of Business*, 55(1):121–145, 1982.

[OR01] M. O'Neil and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[PF99] P. Ponterosso and D. S. Fox. Heuristically seeded genetic algorithms applied to truss optimisation. *Engineering with Computers*, 15(4):345–355, 1999.

[PI07] C.-H. Park and S. H. Irwin. What do we know about the profitability of technical analysis? *Journal of Economic Surveys*, 21(4):786–826, 2007.

[Pio00] J. D. Piotroski. Value investing: the use of historical financial statement information to separate winners from losers. *Journal of Accounting Research*, 38:1–41, 2000.

[PM03a] R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: part I. *Evolutionary Computation*, 11(1):53–66, 2003.

[PM03b] R. Poli and N. F. McPhee. General schema theory for genetic programming with subtree-swapping crossover: part II. *Evolutionary Computation*, 11(2):169–206, 2003.

[PR94] D. N. Politis and J. P. Romano. The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313, 1994.

[Pre98] L. Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[PSV04] J.-Y. Potvin, P. Soriano, and M. Vallée. Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, 31(7):1033–1047, 2004.

[PT95] M. H. Pesaran and A. Timmermann. Predictability of stock returns: robustness and economic significance. *Journal of Finance*, 50(4):1201–1228, 1995.

[QS99] T.-S. Quah and B. Srinivasan. Improving returns on stock investment through neural network selection. *Expert Systems with Applications*, 17(4):295–301, 1999.

[RB94a] J.-M. Renders and H. Bersini. Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 312–317, 1994.

[RB94b] J. P. Rosca and D. H. Ballard. Hierarchical self-organization in genetic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 251–258, 1994.

[RCFM01] M. Raberto, S. Cincotti, S. M. Focardi, and M. Marchesi. Agent-based simulation of a financial market. *Physica A: Statistical Mechanics and its Applications*, 299(1):319–327, 2001.

[Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[Ree95] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[Res09] M. Resta. Seize the (intra) day: features selection and rules extraction for tradings on high-frequency data. *Neurocomputing*, 72(16-18):3413–3427, 2009.

[Ros76] S. A. Ross. The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, 13(3):341–360, 1976.

[Rub81] D. B. Rubin. The bayesian bootstrap. *The Annals of Statistics*, 9(1):130–134, 1981.

[Sam65] P. A. Samuelson. Proof that properly anticipated prices fluctuate randomly. *Industrial Management Review*, 6(2):41–49, 1965.

[Sas01] K. Sastry. Efficient cluster optimization using a hybrid extended compact genetic algorithm with a seeded population. In *Workshop Proceedings of the Genetic and Evolutionary Computation Conference*, pages 222–225, 2001.

[SD91] W. M. Spears and K. A. DeJong. An analysis of multi-point crossover. In *Foundations of Genetic Algorithms*, pages 301–315. Morgan Kaufmann, 1991.

[SD09] P. Sevastjanov and L. Dymova. Stock screening with use of multiple criteria decision making and optimization. *Omega*, 37(3):659–671, 2009.

[SE91] J. D. Schaffer and L. J. Eshelman. On crossover as an evolutionarily viable strategy. In *Proceedings of the International Conference on Genetic Algorithms*, pages 61–68, 1991.

[Sen04] P. Sengupta. Disclosure timing: determinants of quarterly earnings release dates. *Journal of Accounting and Public Policy*, 23(6):457–482, 2004.

[Sha63] W. F. Sharpe. A simplified model for portfolio analysis. *Management Science*, 9(2):277–293, 1963.

[Sha64] W. F. Sharpe. Capital asset prices: a theory of market equilibrium under conditions of risk. *Journal of Finance*, 19(3):425–442, 1964.

[Sha66] W. F. Sharpe. Mutual fund performance. *Journal of Business*, 39(1):119–138, 1966.

[Sha94] W. F. Sharpe. The sharpe ratio. *The Journal of Portfolio Management*, 21(1):49–58, 1994.

[SHH12] C.-H. Shen, Y.-L. Huang, and I. Hasan. Asymmetric benchmarking in bank credit rating. *Journal of International Financial Markets, Institutions and Money*, 22(1):171–193, 2012.

[SJ72] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[SM86] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1986.

[SNF98] L. M. Schmitt, C. L. Nehaniv, and R. H. Fujii. Linear analysis of genetic algorithms. *Theoretical Computer Science*, 200(1):101–134, 1998.

[STW99] R. Sullivan, A. Timmermann, and H. White. Data-snooping, technical trading rule performance, and the bootstrap. *Journal of Finance*, 54(5):1647–1691, 1999.

[SVDM91] F. A. Sortino and R. Van Der Meer. Downside risk. *The Journal of Portfolio Management*, 17(4):27–31, 1991.

[Sys89] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms*, pages 2–9, 1989.

[TC07] M. G. C. Tapia and C. A. C. Coello. Applications of multi-objective evolutionary algorithms in economics and finance: a survey. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, pages 532–539, 2007.

[TKC91] K. Y. Tam, M. Y. Kiang, and R. T. Chi. Inducing stock screening rules for portfolio construction. *Journal of the Operational Research Society*, 42(9):747–757, 1991.

[Tre65] J. L. Treynor. How to rate management of investment funds. *Harvard Business Review*, 43(1):63–75, 1965.

[VHP68] J. C. Van Horne and G. G. Parker. Technical trading rules: a comment. *Financial Analysts Journal*, 24(4):128–132, 1968.

[Wan00] J. Wang. Trading and hedging in S&P 500 spot and futures markets using genetic programming. *Journal of Futures Markets*, 20(10):911–942, 2000.

[WB10] G. Wilson and W. Banzhaf. Interday foreign exchange trading using linear genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1139–1146, 2010.

[Wer00] R. Wermers. Mutual fund performance: an empirical decomposition into stock-picking talent, style, transactions costs, and expenses. *Journal of Finance*, 55(4):1655–1703, 2000.

[WGM94] D. Whitley, V. S. Gordon, and K. Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *Proceedings of the Conference on Parallel Problem Solving from Nature*, pages 5–15. Springer, 1994.

[Whi51] P. Whitle. *Hypothesis Testing in Time Series Analysis*. Almqvist & Wiksells, 1951.

[Whi80] G. Whittington. Some basic properties of accounting ratios. *Journal of Business Finance & Accounting*, 7(2):219–232, 1980.

[Whi00] H. White. A reality check for data snooping. *Econometrica*, 68(5):1097–1126, 2000.

[WK88] D. Whitley and J. Kauth. GENITOR: a different genetic algorithm. In *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, pages 118–130, 1988.

[Woo03] J. R. Woodward. Modularity in genetic programming. In *Genetic Programming*, pages 254–263. Springer, 2003.

[WYC12] F. Wang, P. L. Yu, and D. W. Cheun. Complex stock trading strategy based on particle swarm optimization. In *Proceedings of the IEEE Conference on Computational Intelligence for Financial Engineering & Economics*, pages 1–6, 2012.

[XMP09] P. Xidonas, G. Mavrotas, and J. Psarras. A multicriteria methodology for equity selection using financial analysis. *Computers & Operations Research*, 36(12):3187–3203, 2009.

[YCK05] T. Yu, S.-H. Chen, and T.-W. Kuo. Discovering financial technical trading rules using genetic programming with lambda abstraction. In *Genetic Programming Theory and Practice II*, pages 11–30. Springer, 2005.

[YLT11] I.-C. Yeh, C.-H. Lien, and Y.-C. Tsai. Evaluation approach to stock trading system using evolutionary computation. *Expert Systems with Applications*, 38(1):794–803, 2011.

[YS13] M. G. Yunusoglu and H. Selim. A fuzzy rule based expert system for stock evaluation and portfolio construction: an application to Istanbul stock exchange. *Expert Systems with Applications*, 40(3):908–920, 2013.

[ZYH+06] C. Zhou, L. Yu, T. Huang, S. Wang, and K. K. Lai. Selecting valuable stock using genetic algorithm. In *Simulated Evolution and Learning*, pages 688–694. Springer, 2006.

국문 초록

본 논문은 금융 최적화 문제를 위해 효과적인 탐색 공간 축소와 결합한 진화 연산을 제시한다. 전형적인 진화 연산은 제한이 없는 원래의 탐색 공간에서 최적해를 찾으려 노력한다. 그러나 최적해가 사전 지식없이 찾기에 지나치게 복잡한 경우, 이러한 진화 연산은 성공적이지 않다. 이러한 문제는 해의 형태를 의미있는 형태로 제한하거나 초기 해집단에 유망한 해를 공급함으로써 완화될 수 있다. 이를 위해 모듈화된, 문법적인, 파종된 진화를 포함하는 세 가지 진화 접근을 제안한다. 또한 해들의 미세 조정을 위한 지역 최적화 알고리즘도 채용하여, 혼합형 진화 연산을 구현한다.

첫번째로, 모듈화된 진화를 제안한다. 모듈화된 진화는 가능한 해의 형태를 더 많은 영역 지식을 반영하는 모듈해의 조합만으로 정적으로 제한한다. 모듈해를 보존하기 위해 모듈화된 탐색공간에서 동작하는 모듈화된 유전 연산자를 고안한다. 모듈러 연산자와 정적으로 정의된 모듈들은 유전 프로그래밍이 매우 유망한 탐색 공간에 집중하도록 돕는다.

두번째로, 문법적인 진화를 소개한다. 문법적인 진화는 유전 프로그래밍에서 가능한 해의 형태를 문맥 자유 문법을 사용하여 제한한다. 유전 프로그래밍은 모듈화된 진화보다 확장된 문법적인 탐색공간에서 해를 탐색한다. 이러한 진화에서 사용되는 문법적으로 타입화된 유전 연산자들을 소개한다. 모듈화된 진화에 비해, 문법적인 진화는 영역 지식을 적게 요구한다.

마지막으로, 파종된 진화를 제안한다. 파종된 진화는 초기 해집단에 부분적으로 최적화된 해를 공급한다. 부분적인 최적화 대상이 되는 유전자는 인코딩 복잡도 측면을 고려하여 선택한다. 부분적으로 최적화된 해들은 유전 알고리즘이 보다 유망한 해를 효율적으로 찾을 수 있도록 돕는다. 이러한 해들은 과도하게 최적화되어 있지는 않으므로, 유전 알고리즘은 여전히 더 좋은 해들을 찾을 수 있다.

본 논문은 매력적인 기술적 패턴 발견과 그의 확장된 문제, 대규모 주식 선택을 포함한 세 가지 금융 최적화 문제를 대상으로 광범위한 실험 결과를 제시한다. 이러한

문제들에 대해 제안된 탐색 공간 축소 방법들이 상당히 효과적임을 보인다. 체계적인 진화 연산과 탐색 공간 축소를 결합하여, 진화 연산이 실제 수익을 내는 거래에 이용될 수 있음을 보인다.

주요어 : 금융 최적화 문제, 탐색 공간 축소, 유전 프로그래밍, 유전 알고리즘, 매력적인 기술적 패턴, 주식 선택.
학번 : 2007-30834