Charles University in Prague
Faculty of Social Sciences
Institute of Economic Studies

Diploma thesis

Jaromír Malenko

Stock Market Trend Prediction Using Genetic Algorithms

2008

Supervisor: PhDr. Filip Žikeš, MSc.

Declaration

I hereby declare that I have written this diploma thesis on my own, have used only the cited sources and literature.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a použil pouze uvedené prameny a literaturu.

Jaromír Malenko

Abstract

In this thesis the genetic algorithm is considered as a method of computer learning and it is applied to stock market returns predicting. In the theoretical part we define the prediction task and summarize the results on stock market predictability. We present the statistical methods and measures to evaluate the accuracy of predictor and related computer learning approaches. We introduce the genetic algorithm as a method able to evolve an arbitrary algorithm. In the implementation part a tree representation of any algorithm is employed, the related genetic operators and user application are implemented. In the application part we first evolve predictor as an arbitrary program – the black-box predictor. The result does not differ significantly from liner model, but the economic measures improved. Next, we optimize the parameters of MACD and RSI technical indicators. Optimized indicators may generate profit on the contrary to unoptimized indicators. We find that genetic algorithm is a general technique that is not straightforward to be applied for prediction, but it is efficient in optimizing parameters of a specific model.

Abstrakt (in Czech)

V této práci uvažujeme genetický algoritmus jako způsob počítačového učení a aplikujeme jej na předpovídání výnosu na akciových trzích. V teoretické části definujeme předpovídání a shrneme výsledky o předpověditelnosti akciových trhů. Ukážeme statistické metody a míry pro vyhodnocování přesnosti předpovědi a související přístupy počítačového učení. Zavedeme genetický algoritmus jako metodu pro odvození libovolného algoritmu. V implementační části používáme stromovou reprezentaci algoritmu, naprogramujeme související genetické operátory a uživatelskou aplikaci. V aplikační části vyvineme předpovídač jako program fungující jako černá skříňka. Výsledek se významně neliší od předpovědi lineárního modelu, ale ekonomické ukazatele jsou lepší. Dále optimalizujeme parametry technických indikátorů MACD a RSI. Optimalizované indikátory mohou generovat zisk na rozdíl od neoptimalizovaných indikátorů. Shledali jsme, že genetický algoritmus je obecná technika, která se nedá přímočaře aplikovat, ale je účinná při optimalizaci parametrů konkrétního modelu.

Contents

1 Introduction

1.1 Motivation of this Thesis

As a student of financial markets and a enthusiastic fan of computers the topic of my diploma thesis was clear beforehand. I apply and an advanced computer approach to solve all the econometric challenges.

The genetic algorithms have been showed to be successful in wide areas. I generalized the idea of evolving binary codes to evolving algorithms.

Firstly, I defined a computer language that can encode such algorithms. I developed the SIMPLE programming language and implemented a framework to run the programs (the interpreter being the most difficult part of it).

Secondly, I implemented the genetic algorithm that performs its operators on the SIMPLE programs. A novel tree representation of SIMPLE source codes is used to implement the genetic operators of crossover and mutation.

Only the genetic algorithm and the tree representation is described in this thesis. The implementation of both SIMPLE language and genetic algorithm was similar in time and complexity. The total length of source code is 685 kilobytes in JAVA language.

During all phases of development I tested the program on stock data. I observed the huge impacts of little changes in program. There is no evidence how many error function I used, almost always arriving at a different optimal solution of the same (maximization) problem. The change of parameters of genetic algorithm and also a different implementation of a feature leads to substantial change in performance and optimality of solution.

In the application part I present only a representative few of the large amount of results I obtained.

I headed toward the noble task in which the predictor is simply an algorithm that works on input series by its own and outputs the (correct) predicted value. I must sadly admit, that I wasn't able to evolve such an algorithm. This is no wonder, but my expectation of genetic algorithm were high...

So the genetic algorithm is not able to evolve an universal and general model by itself. The evolved solution doesn't significantly differ from prediction linear model. My next

experiments focused on evolving just parts of a given model. I haven't evolved whole algorithms (source codes) but only the value of parameters of technical indicators. In this thesis I present only the standard MACD and RSI indicators. I played also with oter indicators: the WoodieCCI indicator has a huge and active community supporting (and, they claim, also trading) it. To my surprise this indicator is not better that MACD or RSI indicators!

My conclusion is that genetic algorithm is a useful method, that is *theoretically* able to find solution to many problem. The problem is that the search space is large and the available computer power is not sufficient to find the solution. The special model for predicting, like neural networks, are still worth discovering. The genetic algorithm may be effectively used to optimize its parameters.

1.2 Structure of Thesis

This thesis is organized as follows.

In chapter 2 the predictability of markets is discussed from the theoretical point of view as the efficient market hypothesis. Also the achieved practical predictability results of various models are presented. The prediction task is defined for time series prediction and stock market trend prediction. Then, the statistical test, overall measures and error functions of predictors are discussed. The approaches and problems related to computer learning are summarized.

In chapter 3 the introduction to genetic algorithms is presented. Genetic algorithms use genetic operators to evolve optimal solution. The genetic operators usually work with binary codes. We explain genetic algorithms on tree representation of algorithms. The algorithms are written in the SIMPLE computer language which is defined.

Together with this thesis an implementation of the described procedures was implemented in JAVA language. Our experiences and the results are reported in chapter 4. The suggestions for further research present a list of issues we came across.

In chapter 5 we summarize the lessons learned and the obtained results.

2 Markets and Predictability

The crucial assumption of predictability is that patterns of price development observed in the past will repeat in the future. The goal of prediction task is to identify such patterns. In the era of automation we let the computers do the hard work.

This problem is very demanding since it stimulates various approaches, there is (probably) no optimal solution and, finally, a solution that is just a bit better than others may lead to extensive profits.

Only the profit generated by prediction matters. The predictor does not need to give a prediction at each time point of a time series. The lack of prediction does not mean the loss of money, while the wrong prediction does. Although we present a list of statistical test and measures to evaluate predictors, the achieved profit is the only true rating.

As new prediction algorithms and techniques prove successful, the investors start including them in their investing decisions. The predicting power will decline until it becomes zero and the prediction algorithm becomes useless. That's the reason we must seek for novel approaches.

Although, the fixed costs of developing a new prediction method and an econometric software are high. The variable costs – transaction costs – are negligible. Even small investors may enjoy the benefits of applying novel predictors.

2.1 Stock Market Predictability

The stock prices are usually compared to a random walk process. In the the random walk, it is not possible (from definition) to predict the future price.

Earlier studies inclined to the *random walk hypothesis*: The stock market prices behave as random walk process. Hawawini (1995) found that the serial correlations in stock price time series were insignificant . Fama (undated) tested size filters (a simple indicator) and concludes that the simple buy-and-hold method consistently beats the profits produced by different size filters.

Recently many researchers published the evidence that the investigated market is predictable to some extent. See Lo and MacKinlay (1988) or Žikeš (2007) for valuable results or any other publication that presents a new prediction model (typically neural networks) or technical trading strategy to soaring slogans. Nygren (2004) successfully

predict the Swedish stock index with a neural network; the results of predicting stock are less convincing, nevertheless the network outperforms the naive strategy.

The following section discusses efficiency of markets. In low efficient markets, the more informed and knowledgeable investors can outperform the other ones. The *paradox of efficient markets* is that if every investor believed the market was efficient, then investors would not expect any profit, thus they would not be trading and no market would exist at all. To sum up, the the markets exist only because the participants believe the market is inefficient.

2.2 The Efficient Market Hypothesis

A market is said to be efficient if the prices fully reflect all the relevant information. According to Campbell (1997), depending on the information set that determine future prices, we distinguish the thee forms of market efficiency.

1. **Weak-form efficiency**: The information set includes only the past prices. In other words, technical analysis is of no use.
2. **Semistrong-form efficiency**: The information set includes all *publicly* available information known to all market participants. In other words, fundamental analysis is of no use.
3. **Strong-form efficiency**: The information set includes all *privately* available information known to any market participants. In other words, even insider information is of no use.

The weak and semistrong-form of efficient market hypothesis is confirmed on stock market by White (1988). On the other hand Hellstrom (1998) states that based on recent progress in computer science, the neural networks or genetic algorithms *can* predict effectively. This is not necessary a counterexample of the efficient market hypothesis, since the effective methods are instantly adopted by investors. There was an evidence of a drop in profit based on technical analysis the last but one decade of the 20th century observed by LeBaron (1991).

The strong-form efficiency tests are usually restricted to a subset of market participants. According to Trippi (1996), the performance of mutual fund managers is not superior to the performance of other investor types.

2.2.1 Price Models

We present several theoretical models that focus on the pricing of an asset. The common conclusion of all these models is that the expectation of price in next period equals the price in current period.

Martingale model

Martingale hypothesis by Girolamo Cardano (1565) states that the agents must have equal probability of winning or loosing to enter the game

$$E[P_{t+1}|I_t]=P_t$$

where P_t is stock price at time t and I_t is information set available at time t. The next price is sum of current price and change r_t (called return) in this period

$$E[P_{t+1}|I_t]=P_t+E[r_{t+1}|I_t].$$

Putting these two equations together we obtain

$$E[r_{t+1}|I_t]=E[P_{t+1}|I_t]-P_t=P_t-P_t=0.$$

In the martingale model the price changes are uncorrelated. From the point of view of current asset pricing models, this model does not allow trade-off between risk and return. The martingale hypothesis is considered to be a necessary condition for market efficiency.

Law of Iterated Expectations

The market is efficient with respect to some information if the prices remain unaffected by revealing this information to all market participants. We demonstrate this by *Law of Iterated Expectations* by Samuelson. Let's consider two information sets I and J. The information set J is superior to information set I in the sense that it contains some extra information, hence we assume $I \subset J$. We compare the expectation of a random variable X conditional on these information sets $E[X|I]$ and $E[X|J]$. The Law of Iterated Expectations states that if one has a limited information I, the best forecast one can make of a random variable X is the forecast of the forecast one would make if one would have a superior information J. In mathematical notation

$$E[X|I]=E[E[X|J]|I].$$

Samuelson applied the law of iterated expectations to the market analysis. Consider variables available at time t. The security price P_t is the expectation of its fundamental value V conditional on the information set I_t.

$$P_t=E[V|I_t]=E_t[V].$$

Similar reasoning holds for the next period

$$P_{t+1}=E[V|I_{t+1}]=E_{t+1}[V].$$

The expectation of price change in the next period is

$$E_t[P_{t+1}-P_t]=E_t[E_{t+1}[V]-E_t[V]]=E_t[E_{t+1}[V]]-E_t[E_t[V]].$$

From $I_t \subset I_{t+1}$ we have $E_t[E_{t+1}[V]]=E_t[V]$. Together with the fact that $E_t[E_t[V]]=E_t[V]$ we conclude

$$E_t[P_{t+1}-P_t]=E_t[V]-E_t[V]=0$$

with the natural reading that the price changes are unforeseeable given information in the set I_t.

Random Walk Model

The random walk model

$$p_{t+1}=\mu+p_t+\epsilon_t$$

where p_t is logarithm of price at time t, μ expected price change and ϵ_t a random variable independently and identically distributed $\epsilon_t \sim (0,\sigma^2)$. Campbel shows that the assumptions put on ϵ_t guarantee that the random walk is sufficient but not necessary condition for weak-form efficient market.

2.3 Trading strategies

The following two approaches are used for asset valuation. The distinction is based only on the type of known past data used to produce a prediction of future data.

2.3.1 Technical analysis

Predictions are based on the past stock data only.

The data for each asset contain the following values for each period (day, minute, etc.):

- Open, close, high and low values
- Volume

A basic analysis of stock market excess return data shows both linear and non-linear dependence present. This observation was used to argue that it must therefore be possible to predict future values. However, Manton (undated) shows that the linear and non-linear dependence can be explained by simply allowing the mean and variance of Gaussian noise to be modulated by a (typically 3 state) hidden Markov model. Attempting to fit a Markov modulated AR process proved fruitless; the conclusion is that there is no AR-predictability present in excess return data.

Technical indicators such as moving average convergence divergence (MACD), relative strength index (RSI), trend-lines, support and resistance, volatility, momentum etc. are used in technical analysis. As we show in the empirical section, the significance of technical indicators is negligible with the publicly recommended parameters. We also discuss optimizing the parameters.

2.3.2 Fundamental analysis

Predictions are based on the past stock data (as in technical analysis) and any other variables reflecting the company and market situation. Fundamental analysis aims at the true value of company's stock.

- Economic variables
  - Inflation
  - Interest rates
  - Foreign exchange rates
  - Imports, exports
- Industry variables
  - Stock market indexes – global indexes, sector indexes
  - Stock price of direct competitors
  - Prices of commodities
- Company variables – based on the financial statements of company
  - Number of shares
  - Dividends
  - Price/earnings ratio, book value, assets, liabilities
  - EBIT, total income, total costs
  - Prognoses of future profit, sales, etc. – issued by management or analysts

2.3.3 Derived data

From the technical and fundamental data artificial variables can be computed and also provided for analysis. This is done because the derived values may reveal new information (adjusting data for dividends).

The asset prices X_t have high variation and grow exponentially in time. Thus its difficult to create a valid model for prices. Instead, the return is usually used. The log-return

$$R_t = \log \frac{X_t}{X_{t-1}}$$

is used in theoretical papers. The relative increase in price

$$R_t = \frac{X_t - X_{t-1}}{X_{t-1}}$$

is mainly used in technical analysis. Both definitions are equally well since

$$\log \frac{a}{b} = \log(1 + \frac{a}{b} - 1) \approx \frac{a}{b} - 1 = \frac{a-b}{b}$$

where we used the theorem that $\log(1+x) \approx x$ for x close to 1 and the prices in adjoining periods to be similar $X_t / X_{t-1} \approx 1$.

The return as defined has many convenient properties. First, the values are small and around zero. Second, the return can be used to compare prices of one asset in distant periods. Finally different assets my be compared using returns.

Volatility is used to estimate the investment risk. The increase in volume often means a new information is published. Relative stock performance allows the spread trading.

There is a great deal of derived data and new ideas may prove useful. The genetic algorithms are constructed in such a way that derived data are computed automatically during the learning process. Thus the computer may find new relations in input data that a human may not exploit.

2.4 Specification of Computer Learning

We now define what prediction means in general models and the stock market application.

Given a set of N training examples $\{(X_i, Y_i)\}_{i=1}^{N}$ where X_i are inputs and Y_i expected prediction. The target function $f(X_i) = Y_i$ is to be approximated by predictor g . On each training input X_i the predictor computes the predicted values $Z_i = g(X_i)$.

An arbitrary error function e is forms the error vector $e_i = e(Y_i, Z_i)$. The goal of computer learning is to find a predictor that minimizes the norm of the error vector $E = (e_{1,}\ldots, e_N)$.

Time Series Approach

In the standard time series approach a time series of prices $\{P_t\}_{t=1}^{T}$ was observed. From these observation the training examples

$$X_i = (P_i, P_{i+1}, \ldots, P_{i+b-1})$$
$$Y_i = P_{i+b-1+h}$$
$$N = T - b - h + 1$$

where $b \geq 1$ is the number of past values used to compute prediction and $h \geq 1$ is prediction horizon. The error function

$$e_i = Z_i - Y_i$$

is used to derive norm of error vector

$$\|E\| = \sqrt{\frac{1}{T}\sum_{i=1}^{N} e_i^2}$$

As a typical example of time series consider AR(b) model. This predictor estimates coefficients $a_{1,}\ldots, a_b$ to minimize error of function

$$g_i = \sum_{j=1}^{b} a_j X_{i,j}$$

The AR model assumes that price at a time is a combination of b previous prices.

Trading Rule Approach

The time series approach predicts the precise price. In stock trading, the approach may be simplified. We are not interested in predicting the price but the up/down change is satisfactory.

The training examples X_i, Y_i, N are formed in the same way as in time series approach. The predictor output is used to create a market trade.

If $g(X_i) > 0$	BUY
If $g(X_i) < 0$	SELL
Otherwise	Do nothing

The error functions are discussed in section 2.5.2 Error Function. The usual norm $\|E\|$ is minimized during learning (estimating) the model.

In technical trading the predictor is called *indicator* and the recommended trade is called *signal*. Hellstrom (1998) study on correlations between commonly recommended indicators and stock return does not show a significant correlation.

2.5 Evaluation of Predictor

We discuss several properties that measure quality of an predictor. The error function measures the difference of one predicted value. The following section defines several measures of predictor behavior on a set of (training) data.

2.5.1 Overall Measures of an Predictor

The predictor is learned (trained, estimated) on input data. The input data are divided into two partitions: the *training set* and the *validation set*. While the genetic algorithm is learning, only the training set is used to evaluate individuals. The learning may be so well that it perfectly works on this training set but the learned information may not be generalized. This is called *overfitting*. The output of learning process is thus evaluated only on the validation set. The validation set is usually from 10% to 33% of the input data. Due to the overlaps in time series evaluation, the validation set is usually taken as last observations. The economic conditions may be different (change of strategy, economic reform, etc.) in the validation set than they were in training set. We must ensure that validation set has the same characteristic as the training set, otherwise the predictor may not be able to learn the time series at all.

Assume the predicted time series $\{\hat{X}_t\}_{t=1}^{T}$. For technical reasons define $X_0 = X_1$.

Test of *normality of residuals*. The Jarque-Bera statistics measures the difference of predicted values from the Gaussian distribution

$$JB(\vec{\epsilon}) = \frac{T}{6}\left(S^2 + \frac{(K-3)^2}{4}\right) \sim \chi^2(2)$$

where skewness $S = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{X_t - \bar{X}}{\hat{\sigma}}\right)^3$, kurtosis $S = \frac{1}{T}\sum_{t=1}^{T}\left(\frac{X_t - \bar{X}}{\hat{\sigma}}\right)^4$ and $\hat{\sigma}$ is estimator of standard deviation. Alternative tests for normality are Shapiro-Wilk test and Kolmogorov-Smirnov test.

Overall goodness of prediction. The R-squared (multiple correlation) coefficient

$$R^2 = 1 - \frac{\sum_{t=1}^{T}\left(\hat{X}_t - X_t\right)^2}{\sum_{t=1}^{T}\left(X_t - \bar{X}\right)^2}.$$

measures the proportion of variability in data. If $R^2=0$ the predictor fits the data no better than the mean of the data. If $R^2=1$ the predictor perfectly fits the data.

The problem with R^2 statistics is that it does not consider the number of input parameters k. As parameters are added the R^2 never decrease. The following statistics prefer predictor with lowest value.

The adjusted R^2, written $adjR^2$ penalizes R^2 for the number of parameters

$$adjR^2=1-(1-R^2)\frac{T-1}{T-k}.$$

where k is the number of free parameters to be regressed in the model. The adjusted R^2 increases only if the new parameter improves the model more than the model without this parameter. The adjusted R^2 is less than or equal R^2 and can be negative.

The *Schwartz information criterion* (also called Bayesian information criterion, BIC)

$$SC=\ln\left(\sum_{t=1}^{T}\frac{(\hat{X}_t-X_t)^2}{T}\right)+\frac{k\ln T}{T}$$

measures the difference of predicted values from expected values. The penalty term (the second summand) may be modified to $\frac{2k}{T}$ to obtain Akaike information criterion.

The *root mean square error* statistics under the quadratic loss function

$$RMSE=\sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{X}_t-X_t)^2}$$

and *normalized mean square error*

$$NMSE=\frac{\sum_{t=1}^{T}\left(\hat{X}_t-X_t\right)^2}{\sum_{t=1}^{T}\left(\bar{X}-X_t\right)^2}.$$

are the most common statistics for evaluating a predictor. The error is always nonnegative. The lower error, the better predictor.

The *Theil coefficient of inequality*, often called the information coefficient or t-test,

$$T=\frac{\sqrt{\sum_{t=1}^{T}\left(\hat{X}_t-X_t\right)^2}}{\sqrt{\sum_{t=1}^{T}\left(X_{t-1}-X_t\right)^2}}.$$

measures the predictor performance relative to naive price predictor. The naive prediction estimates the price in next period as today's return (increase since previous period) $P_{t+1}-P_t=P_t-P_{t-1}$ thus

$$P_{t+1}=P_t+(P_t-P_{t-1}).$$

The predictor is better than the naive predictor (predicting the mean) if an only if $T<1$.

The *hit rate* of predictor defines how often the sign of the return $R_t=P_t-P_{t-1}$ (and not the absolute value) is correctly predicted.

$$H=\frac{\left|\{t|R_t\hat{R}_t>0,t=1,\ldots,T\}\right|}{\left|\{t|R_t\hat{R}_t\neq 0,t=1,\ldots,T\}\right|}$$

Analogically to the Theil coefficient the measure for naive return predictor is defined

$$H_N=\frac{\left|\{t|R_tR_{t-1}>0,t=1,\ldots,T\}\right|}{\left|\{t|R_tR_{t-1}\neq 0,t=1,\ldots,T\}\right|}$$

The *relative hit rate* by Hellstrom (1998) provides the predictor's hit rate performance relative to the naive return predictor

$$H_0=\frac{H}{H_N}$$

For $H_0>1$ the predictor is making better predictions than the naive return predictor, while $H_0<1$ implies that the predictor is making worse predictions.

The *mean profit*

$$\Pi=\frac{1}{T}\sum_{t=1}^{T} sign(\hat{X}_t-X_{t-1})(X_t-X_{t-1})$$

measures the ability of predictor to produce a profit. A trade is assumed at every time point in the direction of predicted change. The higher is the mean or net profit achieved by a predictor, the better the predictor is.

2.5.2 Error Function

In this section we assume the predicted time series $\{\hat{X}_{t+h|t}\}_{t=1}^{T}$ where h is prediction horizon. Define the forecast error $\{\epsilon_{t+h|t}\}_{t=1}^{T}$ where $\epsilon_{t+h|t}=\hat{X}_{t+h|t}-X_{t+h|t}$. For technical reasons define $X_{h|0}=X_{1+h|1}$.

To measure the accuracy of predictor we consider h -step forecast, $h\geq 1$, and compute the error function at each time point.

$$L=\sum_{t=0}^{\infty} L_{t+h|t}$$

In practice the upper bound of the sum is given by the number of input data. We now discuss several loss functions.

The *quadratic loss function*

$$L_{t+h|t} = \epsilon_{t+h|t}^2$$

is symmetric and infinitely differentiable.

The *absolute loss function*

$$L_{t+h|t} = |\epsilon_{t+h|t}|$$

is symmetric and infinitely differentiable everywhere except $\epsilon_{t+h|t} = 0.$.

We may be more averse to loss than to profit. In this case the asymmetric loss functions are used.

The *Linex loss function*

$$L_{t+h|t}^{Linex} = \exp(a\,\epsilon_{t+h|t}) + a\,\epsilon_{t+h|t} - 1$$

is for $a < 0$ almost exponential for negative differential and almost linear for positive differential. For positive differential the Linex function is negative, so we need to use

$$L_{t+h|t} = |L_t^{Linex}|.$$

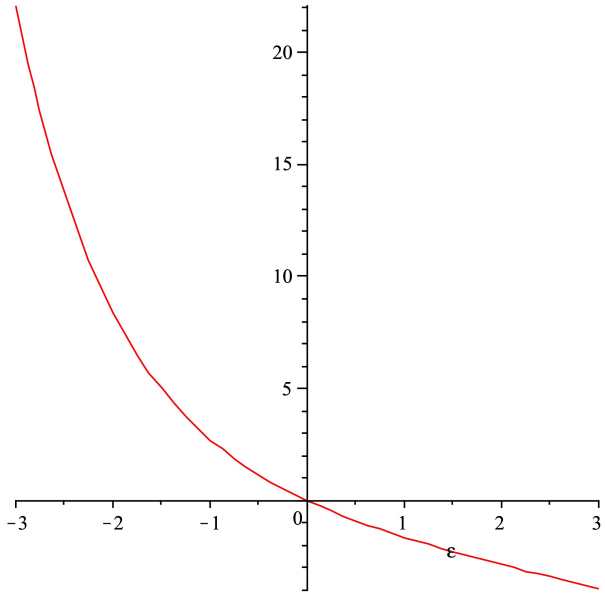

Figure 1: The graph of the Linex asymmetric loss function for $a = -0.2$.

The general *piecewise asymmetric function*

$$L_{t+h|t} = \begin{matrix} a\,L_{t+h|t}^1 & for & \hat{X}_{t+h} - X_{t+h} \geq 0 \\ b\,L_{t+h|t}^2 & & otherwise \end{matrix}$$

where typically

$$L_{t+h|t}^1 = L_{t+h|t}^2 = |\epsilon_{t+h|t}{}^{\rho}|$$

and $a, b, \rho > 0$. The piecewise asymmetric loss function is typically used with parameters displayed in figure 2.

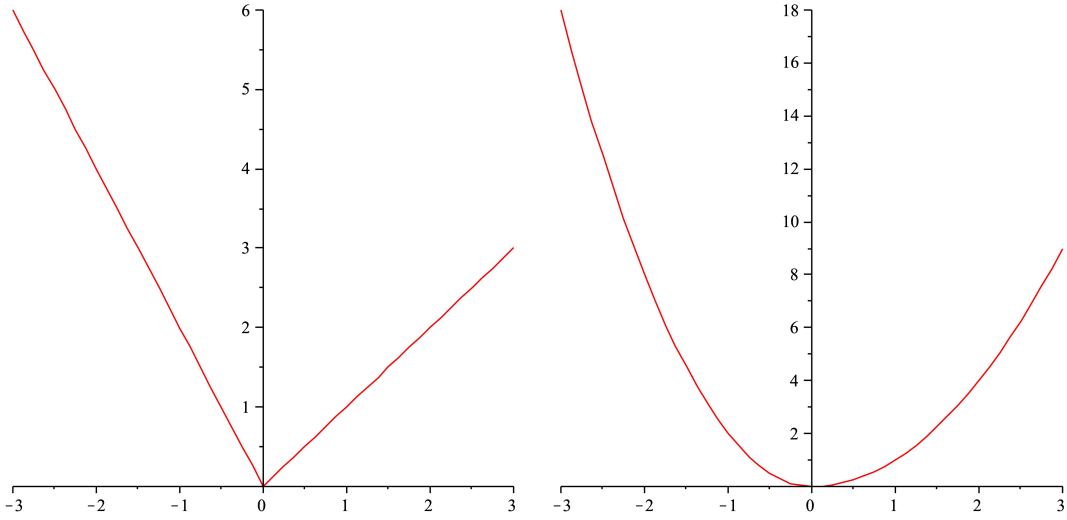

Figure 2: The graph of the piecewise asymmetric loss functions for $a=1, b=2, \rho=1$ *and* $a=1, b=2, \rho=2$.

2.5.3 Comparing Two Predictors

To compare the prediction quality of two predictors we use the *Diebold-Mariano test*.

Assume two predictors $\{\epsilon^{i}_{t+h|t}\}_{t=1}^{T}$ where $i \in \{1,2\}$. Define the loss differential

$$d_t = L(\epsilon^{1}_{t+h|t}) - L(\epsilon^{2}_{t+h|t})$$

We test the null hypothesis of equal predictive ability against the the alternative of non-equal predictive ability.

$$H_0 : E[d_t] = 0$$
$$H_0 : E[d_t] \neq 0$$

The Diebold-Mariano statistics is

$$DM_\tau = \frac{\frac{1}{T}\sum_{t=1}^{T} d_t}{\sqrt{\frac{1}{T}\sum_{\tau=-(T-1)}^{T-1} 1\left\{\frac{\tau}{S(T)}\right\}\hat{\gamma}(\tau)}} \sim N(0,1)$$

where $\hat{\gamma}(\tau) = \frac{1}{T}\sum_{t=|\tau|+1}^{T} (d_t - \bar{d})(d_{t-|\tau|} - \bar{d})$, the $1\left\{\frac{\tau}{S(T)}\right\}$ is the lag window and $S(T)$ is truncation lag.

2.6 Computer learning

In this section we show the predicting task from pragmatical point of view. Then an overview of methods from the area of computer science are discussed. The search for an ideal algorithm is concluded with motivation to the genetic algorithm.

2.6.1 Prediction as Black Box

The eternal puzzle is to predict future values. The scientific contribution is to develop a procedure that takes the observed recent data as input and produces future predicted data.

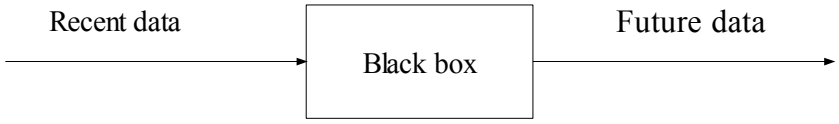

Figure 3: The prediction task is to find the black box.

The procedure may take several forms. The statisticians favor the time series analysis in recent decades. Developing the autoregressive moving average (ARMA) process includes creating a theory, proving theorems and, finally, estimating the model. The crucial assumption of ARMA(1,1) process is that the current value is lagged

$$X_t = \mu + \phi X_{t-1} + \epsilon_t + \theta \epsilon_{t-1}$$

where X_t are data values at time t, $t \geq 0$, ϵ_t is white noise and μ, ϕ, θ are parameters. Given an arbitrary sequence of data X_t the parameters are computed (estimated) such that the error function is minimized. The error function usually is the quadratic norm of difference between real data X_t and predicted data $\hat{X}_t$

$$error = \sum_{t=0}^{\infty} (X_t - \hat{X}_t)^2.$$

This ARMA model is generalized and a comprehensive time series theory Hamilton (1994).

Another scientist may prefer another theory with different assumptions, another researcher relaxes different assumptions or develops the theory a little bit further. And so on, and so forth. The ultimate solution requires human labor and may never be found.

The computer scientists rely on computers to make the hard work. Let the ultimate solution where no human interaction is needed and only the computer power is used be the goal that we attempt to sew up. The computer may be working hard to go through all the steps in creating a black box, including the theory.

2.6.2 Approaches to Computer Learning

Exhaustive Search

Every theory leads to an algorithm computed in the black box that produces the output. The first attempt may be to generate *all* algorithms and use the algorithm with the best quality for prediction. There are several problems with such attempt.

The number of all algorithms is infinite, thus the exhaustive search will never end. This problem may be proceed by considering only algorithms up to some length. If the algorithm is written in some computer language then only the algorithms with code shorter than limit will be considered. If the length limit is 100 then the number of considered codes is $256^{100} \approx 6 \times 10^{241}$. (Each character in computer is a byte. A byte consists of eight bits. A bit is a binary digit. Thus there are $2^8 = 256$ different characters at each position.) If a computer tested 10^9 codes per second (very upside scenario) and each years had $60 \times 60 \times 24 \times 356 \approx 3 \times 10^7$ seconds, the computation would take

$\frac{6 \times 10^{241}}{10^9 \times 3 \times 10^7} \approx 2 \times 10^{225}$ years. The nominator grows exponentially in the length limit while the denominator is constant, given the assumptions on computer power. This value is independent on a particular computer language.

This example shows that the practical attempts must *cut off* some algorithms to produce output in a reasonable time.

Heuristics

A *heuristics* is a function that for a given individual (algorithm in this case) *quickly* estimates its quality. This information may be used to consider the more promising individuals first; when the later individuals are considered, the information about the best individual may be used to skip (cut) the similar (subsequent) individuals.

We are not aware of any reasonable heuristics that allows to pick the more promising algorithms first and skip the less promising. The measures described in section 2.5.1 Overall Measures of an Predictor may be used to navigate the search.

The exhaustive search is often employed with heuristic. The result is referred to as informed search algorithms, a proved implementation is called A* algorithm.

Expert Systems

In past decades a wide range of areas emerged in computer science that analyze data.

The *expert systems* are trained by supervised learning. Given a set of training examples the system learns rules to derive the output. The early systems assumed rules in form of an implication: If feature A or C hold then D happen. Recent systems such as neural networks use more complex forms of rules.

The goal of *data mining* is to sort large sets of data and identify the relevant information. This type of learning is called unsupervised learning since the output is a priori unknown. Methods from data mining may be used to identify the most relevant sources of information for fundamental analysis.

2.6.3 Search Space

Given a problem, we are usually looking for an optimal solution, which is the best among other solutions. The class of all feasible solutions is called *search space*. Each member of a search space represents one feasible solution. Each feasible solution can be assigned a fitness value – a measure how well this solution solves the problem. The task is to find the solution with extreme (maximal or minimal) fitness.

The complication is that the search space can be very large or even infinite. Due to time limitation it is not possible to compute fitness for all feasible solutions. The algorithm that looks for the solution must skip less promising solution and direct the search towards solutions with higher fitness.

We illustrate that there exist hard task with the traveling salesperson problem. Solution to such problems may be obtained by approximate algorithms. In predicting stock market prices even the specification of the task is problematic and we practically restrict to searching for an approximate solution.

Traveling Salesperson Problem

To illustrate this issue on an example, consider the famous Traveling Salesperson Problem (TSP):

Given a number of cities and the costs of traveling from any city to any other city, what is the cheapest round-trip route that visits each city exactly once and then returns to the starting city?

Note that the formulation of the problem imposes no restrictions on the costs – the real values are allowed.

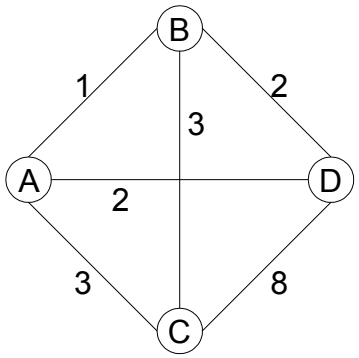

Figure 4: The graph shows distances between each pair of cities. If a salesperson starts at A, what is the shortest route which visits all cities and returns to A?

This problem was shown to be hard: To find the solution an algorithm must (in the every case) consider all feasible routes. The number of these routes is *n!*, where *n* is the number of cities. Since the number of routes grows very fast with number of cities, this problem is practically insolvable if *n* is higher than hundreds. It was proved that no better algorithm exists.

Various approximations that quickly yield good solutions with high probability, have been devised. Modern methods can find solutions for extremely large problems (with millions of cities) within a reasonable time. Such an approximate solution is just 2–3% longer than the optimal solution with high probability.

The approximations use various heuristics that cut off the less promising routes using techniques such as nearest neighbor algorithm, iterative improvement heuristic and randomized algorithm.

The basic feature of the TSP is that the set of all feasible solution is finite and the optimal solution can be found by checking all the feasible solutions. However, this is not the case of out main goal: predicting the trend at stock market.

2.6.4 Preprocessing the Data

The models typically used in time series require the data to be in a special form. During the preprocessing phase the input data are adjusted (scaling), the derived data are computed, non-stationarity is removed, etc.

The input data are typically large. The principal component analysis identifies the redundant data that may be removed and thus the dimension of input data be reduced. The Karhunen-Loeve transformation or neural networks (with the number of output neurons lower than the number of input neurons) may be used for reduction of dimensionality problem.

The time series are typically time dependent. To gain covariance *stationarity* the data must be differentiated. The stationarity test was proposed by Dickey and Fuller. In financial time series the log-return time series (as discussed in section 2.3.3 Derived data)

$$R_t = \log \frac{X_t}{X_{t-1}}$$

is stationary and is typically used instead of price time series.

Data *scaling* is used to map the input values to a required range of values. This transformation is needed to efficiently use all bits to store the value. The most simple transformation is linear scaling that scales values from the set $X = \{X_1, \ldots, X_n\}$ to range $(0; 1)$

$$\tilde{X}_t^1 = \frac{X_t - min(X)}{max(X) - min(X)}$$

Scaling function that maps values to range $(-k; k)$ is defined as follows

$$\tilde{X}_t^2 = 2k\,\tilde{X}_t^1 - k$$

An instance of nonlinear method of scaling is

$$\tilde{X}_t^3 = \frac{1}{1 + \exp(-Z_t)}$$

$$Z_t = \frac{X_t - \bar{X}}{\sigma_X}$$

where $\bar{X}$ is mean, σ_X is standard deviation and Z_t is standardization to which a nonlinear transformation is applied.

It often problematic what scaling method to use. We later describe a genetic algorithm that is able to develop scaling or any other data processing function which best suits the problem.

3 Genetic Algorithms

A *genetic algorithm* is a search technique used to find solution to optimization problems. Genetic algorithms are inspired by evolutionary biology which is an area of artificial intelligence research.

Genetic algorithms were presented as an abstraction of biological evolution. In this method we move from one population of organisms to a new population using natural selection together with crossover and mutation. The solution is evolved from a primitive to a more sophisticated.

Early on, the scientists believed that it would be straightforward to encode the rules that lead to intelligent behavior. Nowadays, it is believed that the rules underlying intelligence are too complex to be encoded by hand in a top-down fashion. Instead, the bottom-up approach attracts more attention – humans write only very simple rules and the more complex behavior emerge from massive automatic combination of these rules.

3.1 Biological background

Even in computer approach we use the biological terminology.

3.1.1 Evolution

Evolution is a method of searching for optimal solutions among and enormous number of possibilities. In biology the possibilities are all the possible genetic sequences and the solutions are the organisms that are well suited to live and reproduce in the environment. Alternatively, evolution is a method for inventing new solutions to complex problems.

The environment is constantly changing and the *fitness* of organism depends on how well it behaves in the environment and how well it competes or cooperates with other organisms.

Evolution is massively parallel: The organism are evolving all at once.

The rules of evolution are very simple: the organisms evolve by *random variation* (crossover, mutation and possibly other techniques) and produce *offspring*. In the natural selection the organisms with higher fitness tend to survive and reproduce, spreading its genetic information.

3.1.2 Terminology

All living organisms consist of cells. Each cell contains the same set of *chromosomes* A chromosome is encoded by a string of DNA. A chromosome consists of *genes*. Each gene consist of *nucleotides* (elementary bits of DNA) that encode a *trait*, such as color of eyes. The different possible settings for a gene are called an *alleles*. The position of a gene in a chromosome is called *locus*.

The set of all chromosomes in a cell is called *genome*. A particular set of genes in genome is called *genotype*. The genotype gives rise to a *phenotype* – a particular physical and mental characteristics such as color of eyes and intelligence.

During reproduction a *crossover* (recombination) occurs: the genes are exchanged between the two parents to form the offspring's chromosome. Offspring are subject to *mutation*, in which single nucleotides are randomly changed, often due to the copying errors.

The *fitness* of an organism is measured by success of the organism in its life. Fitness is the probability that the organism will reproduce.

3.2 Representation of a Chromosome

Genetic algorithm is usually explained on a binary encoding of chromosomes. In our implementation a chromosome is an algorithm so we need an representation of algorithms. We decided for a tree representation of algorithm.

3.2.1 Binary encoding

In computers every information is stored in a variable. This value of a variable is a binary number. Every digit of a binary number is called a *bit*. For each variable the length of its representation (number of bits) is fixed, thus each variable can hold only a limited range of different values. The programmer specifies the length of variable from a predefined list; the lengths are usually multiples of 8.

Representing a decimal number. Suppose we want to represent a decimal number from interval 0..100. This number can be represented in a 7 bit binary variable (The number of different values 101 is less than $2^7 = 128$). The smallest variable type that can hold this range has length 8. For example, the decimal value 85 is represented as a sequence of bits

```
01010101
```

since

$$85 = 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0.$$

Similarly, the decimal value 15 is represented as

```
00001111.
```

The real numbers with decimal fraction are represented analogously. Some part of bits represent the integer part (possibly with sign) and the rest bits represent the decimal fraction.

Representing an algorithm as a number. The *Turing hypothesis* states that every algorithm can be represented by a program in some (reasonable) programming language. All the present computer science is build on this hypothesis. As a reasonable programming language the Turing machine, Random Access Machine (RAM) or all practically used programming languages are assumed.

Given an algorithm in human speech, it can be written in a programming language. This is called a source code or simply a code. We want to assign each source code (and thus an algorithm) an integer number. Then we will not talk about a concrete source code but we will talk only about this number.

We define an ordering on the class of all source codes as follows. Define sets

$$S_n = \{C | C \textit{ is source code of length } n\}, n \geq 1.$$

Sort every S_n in the lexicographical order. From now on, consider S_n as an ordered set (list). Let $C_{i,n}$ is the i-th source code in the set S_n. We assign each source code $C_{i,n}$ an integer number

$$o(C_{i,n}) = \sum_{j=1}^{n-1} |(S_j)| + i.$$

The ordering o is bijection. Thus we can freely skip between source codes and their numbers. The genetic algorithm may work with binary numbers that could be translated to source codes and further evaluated. This is decreases the clearness of implementation of genetic algorithm.

The genetic algorithm is described using binary encoding of every entity. In case of algorithms the encoding is unintuitive. Due to this reason we use the tree encoding written in the SIMPLE language.

3.2.2 The SIMPLE programming language

A chromosome refers to a candidate solution of the problem. In our application a chromosome is a computer program. We developed the SIMPLE language to encode

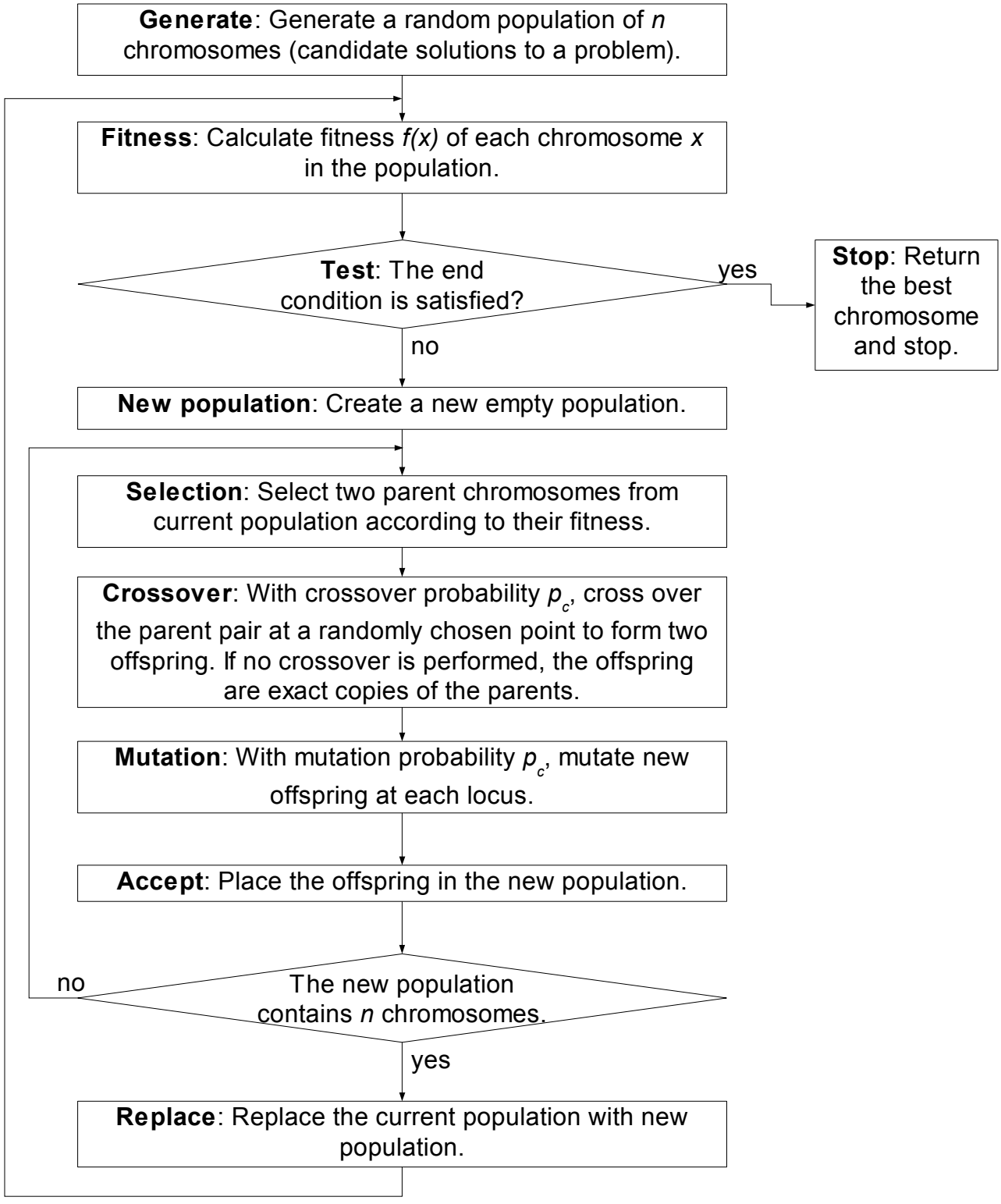

Figure 5: Schema of a basic genetic algorithm.

algorithms. The special language is needed to include only the features that guarantee full computation abilities and a simple syntax for visualization. We now describe the language.

The input to the program is an array *data* representing the values. A particular value is accessible via an numeric index. The value from first period is thus denoted *data[0]*, value from the second period s denoted *data[1]*, etc. For technical reasons the variable *data.length* is set to the number of items in array *data*. The last accessible index (holding value from the last period) in the array is *data[data.length-1]*.

The variables that can be used by the program are either input variables *data* (with index) or *data.length* or local variables. Local variables are denoted by symbol *$* (dollar) followed by an integer number. This form of local variables is required to implement genetic operators with SIMPLE programs (especially mutation in which a new variable is introduced).

The syntax of the program is similar to C language. More specifically, a program is a sequence of statements. A statement is is either a mathematical expression or a control structure such as *if* condition or *while* cycle. A mathematical expression is represented as a tree in which the inner nodes are labeled by operators and the leafs are labeled by constants or variables. The parameters of an operator are the siblings of the respective nodes. We often unify a node with its label.

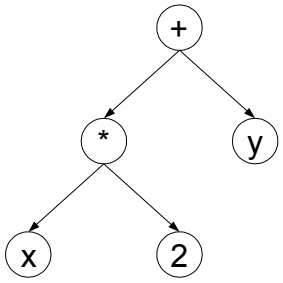

*Figure 6: The representation of expression x*2+y. Note that the tree representation correctly reflects priority of operators since the tree is evaluated from leafs.*

The following code shows a more advanced example program that computes an average of all input variables.

```
{
      $1 = 0;    // variable $1 traverses indexes
      $2 = 0;    // variable $2 stores sum of values
      while ($1 < data.length) { // cycle all indexes
```

```
            $2 = $2 + data[$1];   // add current value to sum
            $1 = $1 + 1;          // shift to next index
        }
        $0 = $2 / data.length;    // compute average and
}                                 // store it in variable $0
```

The result is stored in a local variable. By convention, it's variable *$0*.

For every source code there exist a tree representation. Furthermore every tree representation may be serialized to source code. Thus we have bijection on source codes and tree representations. It should be clear that we can freely switch between source code and representation.

The following picture shows the tree representation of previous program.

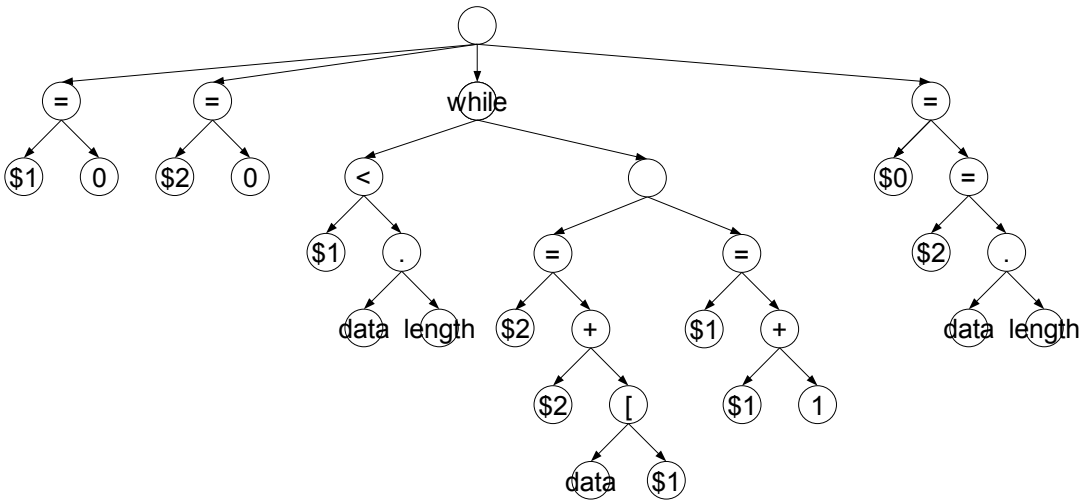

Figure 7: The tree representation of program that computes average of a time series.

Note that the tree has 33 nodes including 15 inner nodes with operators and 18 leaves with values. The children nodes of each inner node represent the (ordered) set of arguments. The condition in *if* and *while* statements is represented as the first child and the statement is represented as the second child. The blank nodes enclose a set of statements.

How difficult is to evolve such program?

This is one of many programs/trees that compute the intended result. The genetic algorithm may in fact evolve a different program that might use less nodes and that also solves the problem. This is again a nice example of genetic algorithm: to find a solution

that is better in some sense (in this case this means that it has a shorter representation). It is not our goal to find the shortest program now but a criterion (error function) involving number of nodes might do the job.

Each inner node can be labeled with operator. The set of operators used in the simple language is closed. The following table list all operators in order of their priorities.

=	assignment
&& ||	logical conjunction, disjunction
== !=	equality, inequality
> >= < <=	greater, greater or equal, less, less or equal
+ - * / %	plus, minus, times, divide, modulo
+ - !	unary plus, unary minus, logical negation
.	object property (ex: `data.length` where data is an array)
[]	array index (ex: `data[5]` where data is an array)
()	method call (ex: `fib(5)` where fib is a name of a method with one argument)
if while	statements (ex: `if (a<0) a=0` where a is an variable)

The total number of operators is 22.

To systematically evolve a program that computes average we need to consider all $22^{15} \approx 1.4 * 10^{20}$ trees with 22 inner nodes plus all trees with less than 22 inner nodes. If we were able to try 1 billion of programs per 1 second the search would take

$$\frac{22^{15}}{10^9} * \frac{1}{60 * 60 * 24 * 365} \approx 4.3 * 10^3 \text{ years.}$$

Up till now we were considering only the inner nodes. The situation with leaves is more difficult. What value should a leaf with integer value hold? Theoretically the count of integers is infinite, although the technical maximum is $2^{32} \approx 4.3 * 10^{10}$. It is clear that the systematic trying of each value would considerably increase the complexity.

The main message of this example is that the search must prune some branches to return a result in a satisfactory time.

3.3 Genetic Algorithm

The schema of genetic algorithm is depicted in figure 8. The schema is very general and allows for various implementations and modifications.

We now describe our implementation of the genetic algorithm. We start with explaining the operations with binary representation and continue with demonstrating the operations with tree representations

3.3.1 Generate

In the generation step *n* random chromosomes are created. These chromosomes are the candidates for solution and will be refined in further generations. The generated chromosomes should be uniformly distributed on the space of all possible chromosomes (up to some length limit).

In the binary representation this step involves generating a random number. This number in binary representation is exactly the chromosome. All the chromosomes may have the same fixed length or variable length.

In the tree representation the syntactically valid SIMPLE programs are generated. The tree representation generates a program as a list of expressions and statements.

An expression is generated by the following recursive function. The parameter *level* denotes height of the tree to be generated.

```
generateExpression(level)
if (level > 0):
      create a new node op
      label the node op by a random (binary) operator
      tree1 = generateExpression(level-1)   // generate all
      tree2 = generateExpression(level-1)   // subtrees
      point the left sibling of node op to node tree1
      point the right sibling of node op to node tree2
      return node op
else:
      create new node x
      label the node x by a random constant or variable
      return node x
```

The real implementation takes care about the arity of operators and *data.length* variable.

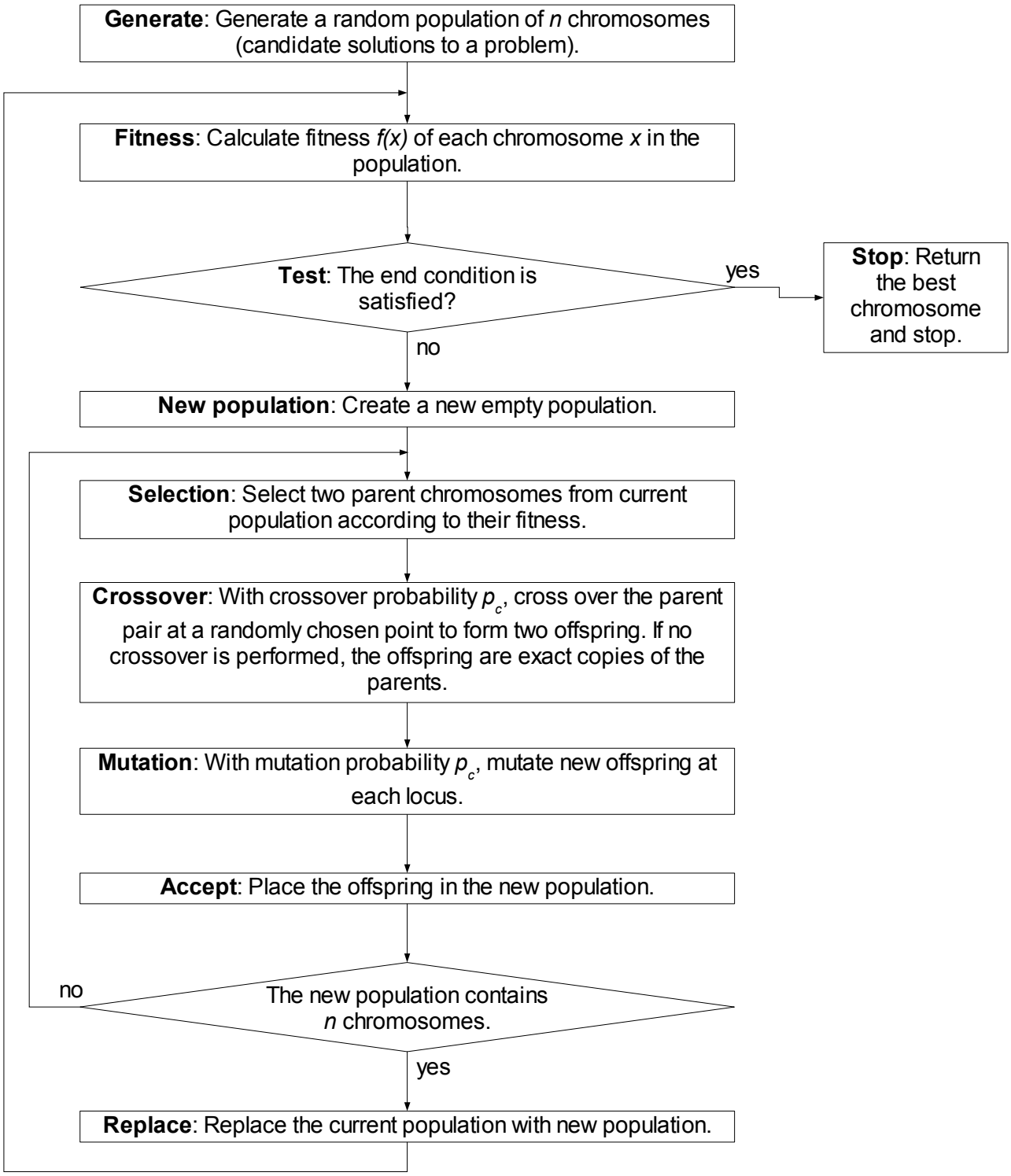

Figure 8: Schema of a genetic algorithm.

A statement may be generated with analogous approach. The statement condition is an expression and the statement command is (recursion) a statement with probability p or (terminal condition) an expression with probability $1-p$.

In the generation of a chromosome we may include parts that we think may be useful in computation. This includes data preprocessing and other techniques described in section 2.6.4 Preprocessing the Data. In our experiments with generation step we included code that computes average of input data, moving average, their ratios etc. This method quite quickly evolves an algorithm that is too similar to the generated chromosomes. Since we use genetic algorithm to evolve an solution that is new and differ from "people approach", we believe this is a short-sighted optimization.

3.3.2 Fitness

Given a chromosome, fitness is a measure of its quality. Fitness is later used to pick the better chromosomes.

In the time series approach, the fitness is the norm of error vector as defined in section 2.4 Specification of Computer Learning. The chromosome (algorithm) is used to compute a predicted value at each time point. An error function, as discussed in section 2.5.2 Error Function, measures its error from expected value. The norm of error vector is used as fitness. The lower fitness the better is the chromosome.

In the trading rule approach, the fitness of a chromosome is the calculated by running the program stored in chromosome. The output is the future value stored in variable $0. If a program run does not finish in a certain time limit (possible infinite cycling) or the value *$0* is not set, then the DO NOTHING signal is assumed. At each input data the signal is computed and the total realized profit is taken as the fitness.

In our experiments we evaluated further restriction such as at least 10% of signal must be BUY or SELL signals and other measures from section 2.5.1 Overall Measures of an Predictor but restrained from them since the most representative measure of quality is the profit. We are not saying such restriction are worthless. They may very useful in guiding the search of genetic algorithm: the predictor with good overall measures but lower profit may contain superior features that might be later improved (via mutation and crossover) to yield higher profit. Thus fitness might also be a function of a measure.

3.3.3 End Condition

If the end condition holds, then the genetic algorithm found a solution and the computation terminates.

In the optimization problem (the time series approach) the optimal solution perfectly predict the output and the error vector contains only zeros.

In the maximization problem (maximization of profit) the end condition is not obvious and the genetic algorithm works until it is stopped by user.

Another possibility is to stop the algorithm when the fitness of all chromosomes do not change between populations for several generations.

3.3.4 Selection

The chromosomes in the next generation are created analogously to the biological evolution: Two parent chromosomes form a new offspring chromosome. This section describes how the parents are selected.

Two parent chromosomes are selected proportionally to their fitness. The better fitness, the higher probability that the chromosome is selected. This method is called *roulette wheel*.

The problem with this selection is that if there were a chromosome with fitness much higher than is the fitness of other chromosomes then it would be always selected. To guarantee variability we use the following optimization: When a chromosome is selected then its fitness which will be used in subsequent selections is divided by a factor $f > 1$.

The selection is without replacement: when a parent is selected it stays in the population and may be selected in further selections.

3.3.5 Crossover

In the crossover step, the chromosomes of parents are combined to create offspring. Due to the symmetrical nature of crossover methods, the crossover produces two offspring in each step.

There are several crossover algorithms for the binary representation.

The *single point crossover* cuts the parent chromosomes in the middle and uses the the left part and the right part to form offspring. The offspring is formed from the left part of one parent and the right part of another parent.

In the following example the parent chromosomes `01010101` and `00110011` were selected. The left part `0101` of parent 1 and right part `0011` of parent 2 form one offspring `01010011`. The second offspring is formed from the right part `0101` of parent 1 and left part `0011` of parent 2 to obtain `00110101`.

```
Parent 1:          01010101
Parent 2:          00110011
Offspring 1:       01010011
Offspring 2:       00110101
```

The randomized version of single point crossover uses a random position (and not the middle) to cut the chromosomes to parts.

The *dual point crossover* is similar to single point crossover, but uses two positions. Each chromosome is thus cut into three parts. The offspring is formed by taking the odd parts from one parent and the even parts from other parent.

```
Parent 1:          01010101
Parent 2:          00110011
Offspring 1:       01110001
Offspring 2:       00010111
```

The dual point algorithm may further be extended to cut chromosomes into more parts where each position is random.

In the *position based crossover* the nucleotide (bit) from parent one is used with a probability p and the nucleotide from parent two is used with complementary probability $1-p$.

The crossover in tree representation is similar to dual point crossover. The operation is illustrative – random subtrees are swapped.

The following procedure is used to crossover two chromosomes *a* and *b*. Note that the procedure modifies the chromosomes (and does not create a new two chromosomes).

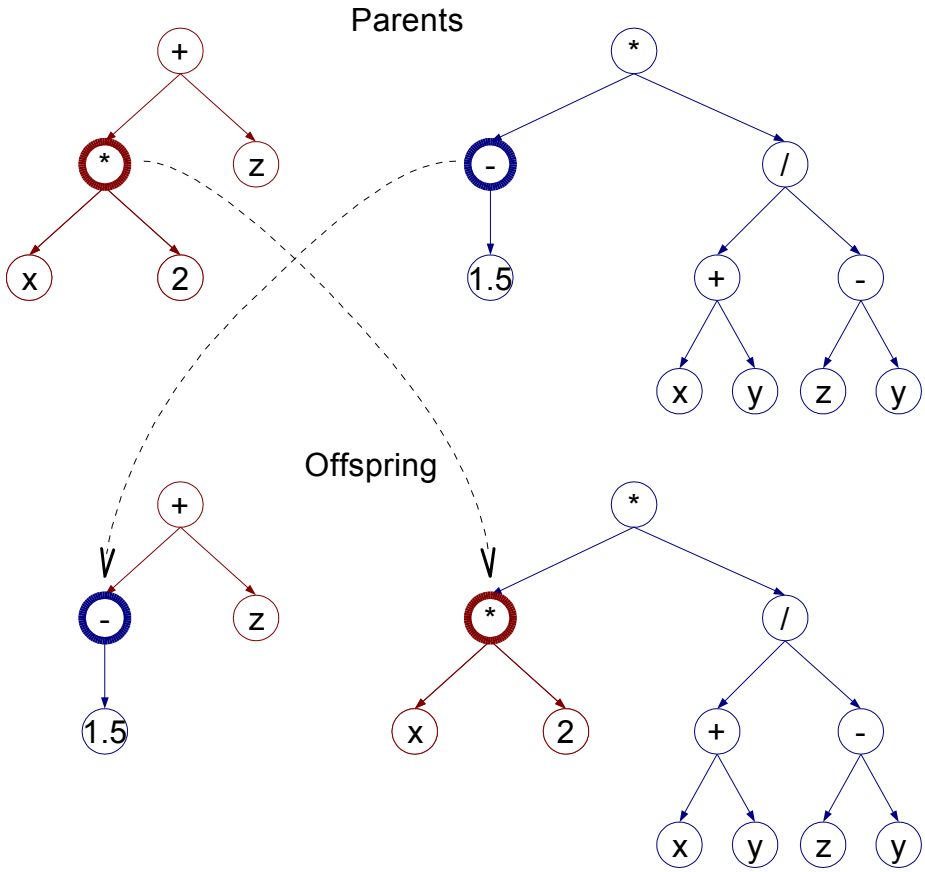

Figure 9: An example of crossover operation. The two parent chromosomes (top) with selected nodes (thick circles) are transformed into two offspring (bottom) in which the selected subtrees are swapped.

crossover(p_1, p_2)

select a node n_1 in tree p_1

select a node n_2 in tree p_2

swap the subtrees of n_1 and n_2

The tree crossover is illustrated by the figure 9.

3.3.6 Mutation

In the crossover operation only the features of parents are reproduced. The mutation allows for creating innovative features.

The mutation of binary representation involves switching some bits. The *single point mutation* switches one random bit. The *k-point mutation* switches k random bits. The *position based mutation* switches each bit with a non-zero probability.

The mutation of tree representation is analogous to single point mutation – a node is selected and its label (or value) is changed. We discuss firstly the mutation of inner nodes that contain operators only and secondly the mutation of leaves that contain variables and numerical values.

The inner node is labeled with an operator. The mutation changes the operator to another operator with the same arity.

The leaf node is labeled with a variable or numerical value. If it is labeled with a variable this variable name is changed to another. With non-zero probability a new variable name is created, otherwise one of already introduced variables is used. The numerical values are mutated as was described in the binary representation.

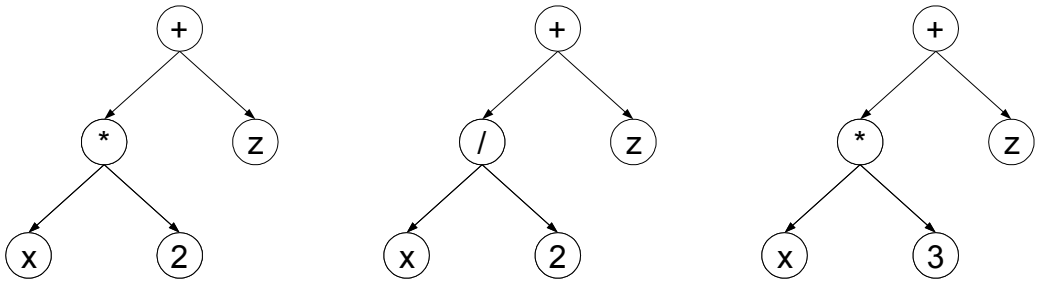

Figure 10: An example of mutation operation. The original chromosome (left) may be mutated either in a inner node from "" to "/" (middle) or in a leaf node from "2" to "3" (right).*

3.3.7 Extensions

The genetic algorithm may be extended in various manners.

Hybrid crossover and mutation operators

The crossover and mutation operators are designed in a way to possibly evolve to any chromosome.

Given any two chromosomes (one be from the initial generation, the other be the optimal chromosome) there must (1) exist a sequence of crossover and mutation operations that transforms one chromosome to the other and (2) each such operation must be performed with a nonzero probability. Hartl (1990) proves that under this condition the genetic algorithm converges to global optimum.

To direct the algorithm toward any (even nonglobal) optima the genetic operators are used with operators that locally improve the chromosomes. This combination of genetic and classical search is called hybrid approach.

The *hill-climbing* (also known as highest descent or complete search) variants of operators are similar to genetic operators described before. The hill-climbing operator tries to locally improve the chromosome in the following way. The randomized operators worked at a random position. The hill-climbing operator applies the genetic operator at each position thus forming all possible offspring of the parents (and especially including the best one).

The downside of the hybrid approach is a high number of generated offspring. Also the search may end up in a local optimum rather than in global optimum.

If the fitness function of the best chromosome does not change for a certain number of generation the chromosome stuck in (local) optimum. The *simulated annealing* is motivated by its use in metallurgy. While cooling, the metal product is warmed up to higher temperature and leaved to keep cooling. The warm up changes the structure of atoms and the binding of atoms strengthen. In the genetic algorithm simulated annealing is implemented by random changes of chromosomes when the fitness does not change for several generations.

Elitism

Elitism ensures that the best chromosomes from current generation are copied to the next generation without any changes. This implies that the best found solution survives to the last generation of the algorithm.

3.4 Further References

The theoretical background and computer science analysis of the genetic algorithms are based on the *schema theorem* by Holland (1998).

Mitchell (1999) is a bible of genetic algorithms and works out many applications. He states that “systematic experiments evaluating the usefulness of tree encodings [...] is just beginning”. The promising results of Koza (1992) show that genetic algorithm found a solution in electrical engineering area that outperformed a recent patent method that was commonly used.

4 Applications

In this chapter we use the genetic algorithm to empirically predict stock market returns. Our own implementation of genetic algorithm based on presented theory is used for evolving the predictors and for evaluation.

4.1 Input Data

In the following we consider a sample of 2500 daily returns of the Dow Jones Industrial Average index. The data are from the January 1997 until September 2007. The index values were preprocessed to remove stationarity by return function $R_t = \frac{X_t - X_{t-1}}{X_{t-1}}$.

The data were downloaded from Yahoo Finance on the date of last sample.

First, we perform an analysis of the input data. The following table summarizes the descriptive statistic.

Minimum	-0.071290
Maximum	0.063480
Median	0.000418
Mean	0.000301
Std. deviation	0.010876
Skewness	-0.060527
Kurtosis	3.607032

To test the date for normal distribution, we perform the Shapiro-Wilk normality test

```
W = 0.9641, p-value < 2.2e-16
```

and Kolmogorov-Smirnov test

```
D = 0.0516, p-value = 3.254e-06
```

Based on p-value we reject the null hypothesis that returns are normally distributed. The histogram and Epanechnikov kernel density estimator with Sheater's bandwidth is depicted in figure 11. The kurtosis grater then 3 is line with the observations that financial time series are leptokurtic and thus not distributed according to Gaussian distribution.
For the following prediction tasks the input data were split into a training set and a validation set of last 10% of input data. Thousands of predictors were evolved on the training data. The predictor that generated highest net profit on validation data was the result of learning process. The validation set measures are done with respect to this

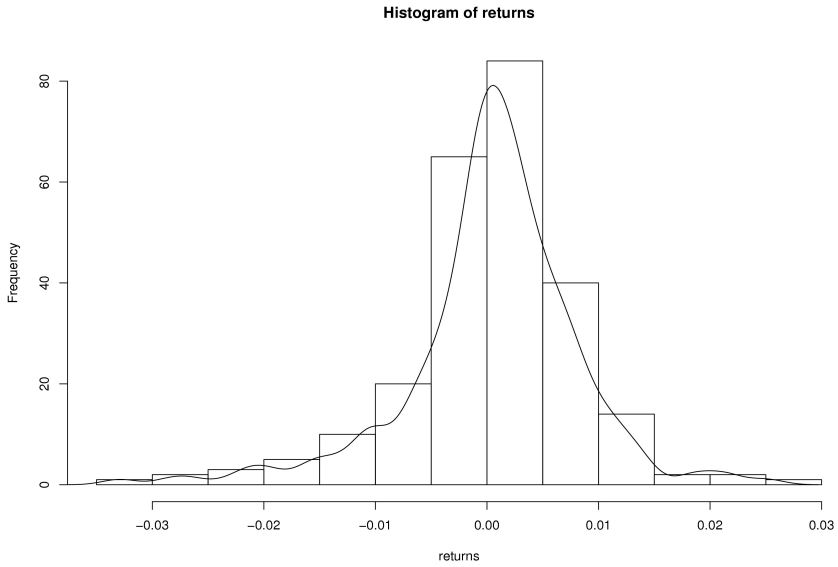

Figure 11: Histogram and kernel density of input data.

evolved predictor. The buy-and-hold net profit was 1.908766 for the training set and 1.118659 for the validation set.

4.2 Index Prediction

We predict the data using ARIMA model and model evolved by genetic algorithm. The time series approach which predicts the exact value of future data is used. The analysis is done both in context of time series and trading rules (economic profit measures).

We perform the Ljung-Box Q statistics for examining the null hypothesis of independence in data.

```
X-squared = 0.4635, df = 1, p-value = 0.496
X-squared = 3.8013, df = 2, p-value = 0.1495
X-squared = 3.9239, df = 3, p-value = 0.2698
X-squared = 4.7807, df = 4, p-value = 0.3106
X-squared = 8.7489, df = 5, p-value = 0.1195
```

We cannot reject the hypothesis of randomness. The data may contain serial correlation and forecasting with ARIMA model must be done with care (we use it as benchmark for GA since we are not aware of a better model).
We use ARIMA(5,0,5) model to explain the data.

	ARIMA model	GA model
	Training set	Training set
Adjusted R^2	0.022402	0.122781
Bayesian information criterion (BIC)	-6.119261	-9.871495
Akaike information criterion (AIC)	-6.147226	-7.460432

Adjusted R^2 statistics tell us that the ARIMA model explains 2% of data, while the GA model explains 12% of data. Both Akaike and Bayes information criteria also show that GA model is better in explaining the data than ARIMA model.

We perform the Diebold-Mariano test.

```
Standard Normal = -1.9585, Lag = 0, p-value = 0.15017
Standard Normal = -1.8684, Lag = 1, p-value = 0.16171
Standard Normal = -1.7824, Lag = 2, p-value = 0.17468
Standard Normal = -1.6892, Lag = 3, p-value = 0.19118
Standard Normal = -1.6579, Lag = 4, p-value = 0.19734
```

We cannot reject the null hypothesis of equal predictive ability of ARIMA and GA models. Therefore GA do not have statistically significant different (lower) errors than ARIMA model.

We compare overall measures of the models.

	ARIMA model		GA model	
	Training	Validation	Training	Validation
Root mean square error (RMSE)	1.397699	1.419905	1.276912	1.372917
Normalized mean square error (NMSE)	1.950762	2.054613	2.402355	2.523608
T-test	1.218513	0.980581	1.107569	1.098457
Relative hit rate	1.068383	0.964599	1.079585	1.010799
Mean profit	0.531037	0.356275	0.699239	0.456736

We include both training set and validation set measures. The training set measures are always better than validation set measures. This was expected since models were optimized on training data. In the following we discuss only the measures of validation set.

The root and normalize mean square errors each favor different model. The RMSE is higher for ARIMA model than for GA model, while NMSE is lower for ARIMA than for GA model.

The Theil coefficient of inequality (T-test) shows that ARIMA is better at learning the training set than GA model. There is more inequality in GA model than in ARIMA model. The relative hit rate and mean profit are better for GA model.

In summary we cannot say that GA model explains the data better than ARIMA model. Though the GA model has most (all but NMSE and T-test) overall measures better. Genetic algorithm model data the time series a little better than linear model.

4.3 Indicator Prediction

We implemented and optimized two commonly used technical indicators. The implementation is highly parametrized to allow (genetic) optimization of parameters. We indicate the typical or recommended values of parameters in the description of indicators. We do not list the optimized values of parameters since there is no clear relationship to the recommended values – the optimized values are obtained by maximizing the objective (profit) function.

4.3.1 Moving Average Convergence/Divergence Indicator

MACD indicator uses moving averages to generate trend-following characteristics. From the moving averages the momentum oscillator MACD is computed by subtracting the longer moving average from the shorter moving average. For a complete description of MACD indicator see reference [MACD].

MACD generates signals from three main sources: First is the divergence between asset price change and MACD. Second, moving average crossover is present when the graphs of two moving averages (of length 16 and 26) cross. Third, centerline crossover occurs when MACD crosses the zero.

The table summarizes statistics of MACD with typically recommended parameters and parameters optimized using genetic algorithm.

	MACD indicator		Optimized MACD indicator	
	Training set	Validation set	Training set	Validation set
Root mean square error (RMSE)	1.050182	1.053029	1.183182	1.214827
Normalized m. s. e. (NMSE)	9.392720	5.969887	2.677255	2.583333
T-test	2.457534	2.617250	1.415834	1.478387
Relative hit rate	1.085535	1.060844	1.164776	1.003312
Mean profit	1.016413	0.971659	1.035642	0.971659
Net profit	0.890042	1.015424	2.741483	1.198434

The root mean square error and normalized mean square errors are contradictory. While RMSE is lower for (typical) MACD, then NMSE is lower for optimized MACD. The NMSE is significantly higher than RMSE.

The result of T-test shows that optimized MACD really fits data better. This was the goal of optimizing parameters. The inequality is lower for optimized parameters.

The relative hit rate is better for (typical) MACD than for optimized MACD. The mean profit is the same for both indicators. As expected, the net profit generated by optimized MACD indicator is higher than net profit of MACD indicator with typical parameters.

From the results we see that, on training set, the optimized indicator has better overall measures than indicator with typical measures. When we evaluate indicators on validation set the results are not so convincing. The measures are more consistent for typical indicator on both training and validation sets. The measures get worse for optimized indicator. We first attributed this to overfitting of parameters. But since the training data covers 9 year period of both economic growth and decline we must question if the validation set is representative. There might be an undiscovered pattern in stock behavior or the validation set does not cover a representative period (or period is too short).

4.3.2 Relative Strength Index Indicator

The RSI is a momentum oscillator. It compares the magnitude of recent gains to the magnitude of recent losses and turns that information into a number that ranges from 0 to 100. For a complete description of RSI indicator see reference [RSI].

The RSI signal is generated from the three sources: First, the asset is overbought or oversold if RSI is higher or lower than a level (70 and 30, respectively). Second source

comes from divergence between the RSI and the underlying asset. Third, the centerline crossover is present when RSI passes a value (50).

Each source is assigned its weight. If a source is present, its weight is added or subtracted from the signal value depending on the bullish or bearish orientation of source.

The indicator generates trade signals as follow: If the signal value is higher than zero then a BUY signal is generated. If the signal value is less than zero then a SELL signal is generated.

There are more sources are identified, but not all of them have the same importance. That is why the sources are weighted. We implemented RSI_{min} indicator as an extension of RSI indicator, where the signal value must pass a threshold: If the signal value is higher than threshold then a BUY signal is generated. If the signal value is less than threshold then a SELL signal is generated. If the absolute value of signal value is lower or equal than threshold, do nothing.

The table summarizes statistics of RSI with typically recommended parameters. The optimized RSI and RSI_{min} indicators use parameters optimized using genetic algorithm.

	RSI		Optimized RSI		Optimized RSI_{min}	
	Training	Validation	Training	Validation	Training	Validation
RMSE	1.415345	1.470133	1.391959	1.307546	1.215394	1.198117
NMSE	1.983872	1.868226	2.051152	2.016569	2.883942	2.585754
T-test	2.112885	2.071309	5.877317	3.640055	3.758006	3.845994
Relative hit rate	1.042406	0.894018	1.076762	1.113602	1.096258	0.953740
Mean profit	0.604325	0.574899	0.534241	0.510121	0.798158	0.817814
Net profit	0.497162	0.848270	2.345297	1.316947	2.367124	1.251906

The root mean square error favors optimized RSI and even more optimized RSI_{min}. The normalized mean square error gives opposite results as was the case of MACD indicator.

From T-test we see that optimization lead to higher inequality. This result is counterintuitive since we would expect the optimized predictor to give results that are more similar (equal) to expected values.

The relative hit rate of optimized RSI indicators is higher than rate of the (typical) RSI indicator. Although relative hit rate is lower for optimized RSI_{min} than for optimized RSI. The mean profit is consistently better in optimized models. Optimized RSI_{min} indicator generates higher profit than optimized RSI indicator, but only on the training data. On the

validation data optimized RSI_{min} indicator generated lower net profit than optimized RSI indicator. We attribute this to overfitting of optimized RSI_{min} indicator.

4.4 Conclusion of Predictions Experiments

At the beginning of this chapter we described and analyzed the data. The distributional properties of DJI index were in line with expectations of properties of financial time series.

Using genetic algorithm we evolved an GA predictor which was a SIMPLE language source algorithm. We compared it to benchmark linear model. The predictions of GA predictor are not significantly different from linear model, but some of the overall economic measures improved.

We optimized parameters of MACD and RSI indicators. Optimized parameters of both indicators generate higher net profit. The optimized RSI_{min} indicator was overfitting the training data and led to lower net profit on validation data than the optimized RSI indicator.

4.5 Lesson Learned During Implementation

Implementation of both SIMPLE language and genetic algorithm is the largest programming task we have ever participated in.

SIMPLE Programming Language

Implementation of the framework for SIMPLE language includes following subprograms. When a source code is run, it must be first parsed and stored in memory. The code must successfully pass all the following analyzes. The *lexicographic analysis* check if the source code satisfies grammar rules. Example of such rule is that the binary operator divide is applied to two arguments (both two arguments are present). The *semantic analysis* guarantees that the commands are purposeful. Example of semantic error happen if a result of an expression is assigned to anything else than a variable (another expression, for instance). For the syntactic and semantic analysis the ANTLR Parser Generator was used. Its product is an abstract syntax tree (AST) representing the given source code.

When a program is stored in AST tree it can be performed. We have written an *interpreter* of AST tree. The interpreter makes no optimization during its execution. The

interpreted programs are generally from definition slower than compiled programs or programs using just-in-time compilation. The compilation is a hard and time consuming task. The trade-off between time spent with compilation and running the program faster and interpreting the program may be an interesting topic for further computer science research. During the compilation advanced optimization techniques may be performed to again speedup the execution time. In real programming and especially in programs evolved by genetic algorithm it often happen that a variable is assigned a value, but this variable is never read. Thus such assignment (including the time need to compute the value) may be left out.

Genetic Algorithm

The largest issue in developing and debugging genetic algorithm is the nondeterminism. The algorithm uses randomness and thus every run is unique and may not be exactly reproduced. Thus, locating the source of an unwanted behavior is complicated.

The visualization of chromosomes is not needed in the application of genetic algorithms, but it is necessary for a developer to have a clear picture of the state of computation.

The computation time of genetic algorithm is long. Each generation, a large amount of new chromosomes is created. The AST representation of algorithms is long, thus large memory space is required. During evaluation, fitness is computed at each time point of past data, again requiring time. During debugging the developer does not have much time to running the program and observe the economic implications of the results.

As a conclusion, debugging the genetic algorithm is difficult both due to large computation time requirements and the lack of visualization tools.

4.6 Future Work

During our experiments we identified several areas that substantively influence the results and may be worth further analyzing.

4.6.1 Transaction costs

The transaction cost are usually omitted in scientific publications analyzing predictors. To address this issue we included commission per trade (0.2% of invested amount) and interest rate per annum (10% p.a.) in evaluation of the developed indicators. The

commission minimizes number of trades, while the interest rate minimizes the holding time of investment. We summarize our preliminary results.

The indicators optimized without transaction cost behave badly when evaluated with transaction costs. In fact, using such indicators lead to loss. The initial amount of money quickly dropped close to zero. The basic reason is that commission per trade substantively decreases the profit from trade, thus favoring strategies in which the investment last for more periods.

The indicators optimized with transaction costs outperform the buy-and-hold strategy. Such an indicator is found quickly. Since we have not run the learning of such indicators for long, we cannot compare the profitability of indicators with and without transaction costs.

4.6.2 Integration with Trading Programs

We developed a user application to command the evolution of genetic algorithm. This application can be easily used to produce new indicators. The analysis of such indicators is not user friendly (running another statistical program, etc.). The integration of our application into programs with advanced back-testing features and visualization may be necessary to practically use the output of our work.

The indicators described in this section were hard coded in the application. Adding new indicators requires programming skills and should also be simplified. Integration of genetic algorithm into existing trading programs does not mean that their build-in indicators may be also evolved (requires advanced communicating interface).

4.6.3 Learning Time

The learning process takes a lot of computer time. There is no method that can decide whether an algorithm ever ends on input data or it ends. This theorem is called halting problem. Knowing this theorem, the only way how to evaluate an SIMPLE program is to run it on each time point and observe the behavior of the program: The program is computing and when a result is computed it ends. But if the program is computing for more than a time limit (1 second) the program is terminated and no prediction is computed (interpreted as do nothing signal).

If the predictor is tested for longer periods then the total time may sum to long time. Assume every year has 250 trading days, and the predictor is to be evaluated for 10 years.

Then one predictor is evaluated for $250 \times 10 = 2500$ second, that's approximately 43 minutes. This is unrealistically high value and a work-around technique must be invented. We propose to start evaluating predictors on shorter intervals (1 year) and slowly prolong this interval. An predictor that fails to generate profit in first year hardly generates a profit in more years. In each generation a different interval may be used to evolve algorithm with best generalization properties.

It might be interesting to compare the same predictor that was trained on different training set. What decade is the most representative in the sense that the predictor trained on this interval performs best on all data? If the predictor was trained on year intervals, are there any similarities between predictors? (For instance, predictors for growing years might have learned the BUY signals only.) These are the questions that might be insightful to question before tuning the computer learning algorithms.

4.6.4 Language Constructs

The development of our own SIMPLE language allows to implement specific features that are not present in one particular computer language, but that are common in different areas.

Theoretical computer science consider simple yet fully computationally powerful models of computer. The models include Turing Machine (with the only instruction operating on a tape) and Random Access Machine (RAM). The RAM contains an infinite number of registers indexed by integer numbers. No such feature is present in commonly used computer languages, but it may be simulated by an array. We included this feature and allowed registers *$0*, *$1*, etc. Since the evolved program does not use array, the registers are needed to achieve computability.

The usual code to compute average was displayed in section 3.2.2 The SIMPLE programming language. The code uses a cycle with one variable to traverse all the indexes and at each index the value is added to the sum. Such program uses 33 nodes and may be difficult to evolve. Although the while cycle is general, its practical use on time series is single-purpose: to traverse indexes. We propose a new command *fold*, that traverses an array and performs an operation at each index. The greatest feature of fold command is that it always terminates. The fold command takes two parameters. The first is the initial value of temporary result. The second parameter is an expression that uses variables $1 and $2. The variables are set at each iteration to the temporal result and current value, respectively.

The length of the average example that uses fold command is then substantively reduced. Another advantage is that the fold command finishes after it traverses all the indexes. Contrary, the while command may run infinitely (this may happen when a variable in cycle condition is not changed in the cycle body).

```
sum = fold(0, $1 + $2);
avg = sum / data.length;
```

To clearly explain the fold command, the following code shows the expansion of it that is actually used by interpreter during computation.

```
$1 = 0;                          // initial value
$3 = 0;                          // traverse all indexes
while ($3 < data.length) {
      $2 = data[$3];
      $1 = $1 + $2;              // actual operation
      $3 = $3 = 1;
}
sum = $1;
avg = sum / data.length;
```

The code length can substantively be reduced whereas allowing a simplified form of cycles. Bramaier (2004) disallow cycles at all in his implementation of linear genetic algorithm. Without cycles the evolved programs are longer and must be used on inputs of the same length.

4.6.5 Missing Data

Different markets have different trading days and holidays. This is leads to problems in comparing time series of two stock prices that were traded on different days. The data must be approximated.

The most of the data used in fundamental analysis is publicly available in the company's annual report. To our knowledge there is no database that contains such data in a form that can be easily processed by computer.

4.6.6 Challenge of Predictors

During the implementation of genetic algorithm we observed huge effects of a small difference in a particular implementation. This opens space for tuning almost any algorithm (and commercial software) for a specific task.

To our knowledge, there is no open challenge of predictors. The challenge is present – traders decide which program they use. But a challenge with open, clear and simple rules and evaluation (done in future) does not take place. Such competition might truly rank the software and help traders in deciding.

5 Conclusion

In this thesis we present a genetic algorithm approach to predict the stock market trend. Our implementation of genetic algorithm performs genetic operators with a tree representation of an algorithm. Using the genetic algorithm we evolved a black-box predictor and optimized parameters of MACD and RSI indicators. We performed empirical test on the Dow Jones Industrial Average index in period 1996–2007.

The theoretical background section discussed the efficient market hypothesis and its truthfulness. Supported by promising previous results we specified the prediction task and defined statistical tests and overall economic measures to evaluate predicting models. From a wide range of computer learning approaches we picked genetic algorithm as a method that is generally able to develop any model.

In our application the model is a source code in the SIMPLE language. To evolve such code we represent it in memory in the tree representation. We presented novel genetic operators of crossover and mutation operating on the tree representation. The tree genetic operators were suggested for evolutionary algorithms in several sources, but no prior implementation was described and no worthy presentation of achieved result were published. We implemented the SIMPLE language and genetic algorithm in the JAVA language. We summarized the lessons learned during implementation and issues related to evaluating time series predictors (time complexity and termination being the most important). The empirical part list the results achieved by genetic algorithm with tree representation of algorithms.

In the empirical application we developed a black-box algorithm predicting the values of DJIA index. This black-box predictions do not significantly differ from predictions of linear model. Although economic measures like relative hit rate and net profit improved in our model. As we argue in the introductory chapter, the generated profit is the only true measure of any predictor. Thus, as the fitness function in genetic algorithm we use the profit function and not a norm of differences between expected and predicted value. This makes comparison between different model complicated and demonstrates that genetic algorithm may be adjusted to solve any task.

We optimized technical trading indicators. We implemented a highly parametric version of MACD and RSI indicators and used genetic algorithm to optimize the parameters of indicators. The main conclusion of our experiments is that the indicators, if used with

recommended parameters, do generate loss instead of profit. After optimization of parameters the indicators generate higher profit than buy-and-hold strategy.

In our experiments we used only historical values of a stock index. The results show that genetic algorithm could generate some profits. Although, we believe that asset prices are not based only on past prices (as pure technical analysis assumes). Further analysis needs to be done to uncover other variables affecting the asset prices.

6 References

[ANTLR] ANTLR Parser Generator. Version 3. http://www.antlr.org/ [Accessed 1 Dec 2007].

Baltagi, Badi H.: *Econometrics*, Springer-Verlag, 2002.

Baruník, J.: *On the predictability of Central European Stock returns*. Diploma thesis, Charles University in Prague, Faculty of Social Sciences, 2006.

Brameier, M.: *On Linear Genetic Programming*. Disertation Thesis, Dortmund University, 2004.

Campbell, John Y. and Lo, Andrew W. and MacKinlay, A. Craig: *The Econometrics of Financial Markets*. Princeton University Press, 1997.

Diebold, Francis X. and Mariano, Robert S.: *Comparing Predictive Accuracy*. Journal of Business and Economic Statistics, 1995, vol. 13, pp. 253-265.

Fama, Eugene F.: *Random Walks in Stock Market Prices*. University of Chicago. Selected papers No. 16.

Hamilton, James D.: *Time series analysis*. Princeton University Press. Princeton, 1994.

Hartl, Richard F.: *A Global Convergence Proof for a Class of Genetic Algorithms*. University of Technology, Vienna, 1990.

Hawawini, G. and Keim, D. B.: *On the predictability of common stock returns: Worldwide evidence*. In R. A. Jarrow, V. Maksimovic, and W. T. Ziemba, editors, Handbooks in Operations Research and Management Science, volume 9: Finance, chapter 17, pages 497–544. Amsterdam, The Netherlands, 1995.

Hellstrom, Thomas and Hellstrom, Kenneth: *Predicting the Stock Market*. Thecnical Report Series Ima-TOM-1977-07, Malardalen University, 1998.

Holland, John H. : *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, 1992.Hsieh, David A.: *Chaos and Nonlinear Dynamics: Application to Financial Market*. Journal of Finance, vol. 46, pp. 1839—1877, 1991.

Kalyvas, Efstathios: *Using Neural Networks and Genetic Algorithms to Predict Stock Market Returns*. 2001. http://citeseer.ist.psu.edu/kalyvas01using.html [Accessed 1 Dec 2007].

Koza, J. R.: *Genetic Programming: On Programming of Computers by Means of Natural Selection*. MIT press, 1992.

LeBaron, B.: *Technical trading rules and regime shifts in foreign exchange*. Technical report, Department of Economics, University of Wisconsin - Madison, Madison, WI, October 1991.

Lo, A. W. and MacKinlay, A. C.: *Stock market prices do not follow random walks: evidence from a simple specification test*. Review of Finantial Stuies, vol. 1, pp. 41-66, 1988.

Malkiel, Burton G.: *The Efficient Market Hypothesis and its Critics*. Princeton University, CEPS Working Paper No. 91, 2003.

Manton, Jonathan and Muscatelli, Anton and Krishnamurthy, Vikram and Hurn, Stan: Modeling *Stock Market Excess Returns by Markov Modulated Gaussian Noise*. Working Papers 9806, Department of Economics, University of Glasgow, undated.

Mitchell, Melanie : *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1999.

[MACD] Moving Average Convergence/Divergence (MACD). http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:moving_average_conve [Accessed 1 Dec 2007.]

Nygren, K.: *Stock Prediction—A Neural Network Approach*. Diploma thesis, Royal Institute of Technology, Sweden, 2004.

Peters, Edgar E.: *Fractal Market Analysis: Applying Chaos Theory to Investment and Economics*. Willey, 1994.

[RSI] Relative Strength Index (RSI). http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:relative_strength_in [Accessed 1 Dec 2007.]

Trippi, R. R. and Lee, J. K.: *Artificial Intelligence in Finance and Investing*. Irwin Professional Publishing, Chicago, London, Singapore, revised edition, 1996.

White, H.: *Economic prediction using neural networks: The case of IBM daily stock returns*. In IEEE International Conference on Neural Networks, San Diego, pages 451–459, San Diego, 1988.

Žikeš, Filip: *The Predictability of Asset Returns: An Empirical Analysis of Central-European Stock Markets*. 2003. http://ies.fsv.cuni.cz/storage/work/101_filip_zikes.pdf [Accessed 1 Dec 2007.]