ELSEVIER

Journal of Financial Economics 51 (1999) 245–271

JOURNAL OF Financial ECONOMICS

Using genetic algorithms to find technical trading rules[1]

Franklin Allen[a,*], Risto Karjalainen[b]

[a] *The Wharton School, University of Pennsylvania, Philadelphia, PA 19104, USA*
[b] *Merrill Lynch & Co., Inc., Mercury Asset Management, 33 King William Street, London, EC4R 9AS, UK*

Received 12 October 1995; received in revised form 7 April 1998

Abstract

We use a genetic algorithm to learn technical trading rules for the S&P 500 index using daily prices from 1928 to 1995. After transaction costs, the rules do not earn consistent excess returns over a simple buy-and-hold strategy in the out-of-sample test periods. The rules are able to identify periods to be in the index when daily returns are positive and volatility is low and out when the reverse is true. These latter results can largely be explained by low-order serial correlation in stock index returns.

JEL classification: C61; G11; G14

Keywords: Trading rules; Genetic algorithms; Excess returns

1. Introduction

Genetic algorithms belong to a class of machine learning algorithms that have been successfully used in a number of research areas. There is a growing interest

* Corresponding author. Tel.: 215 898 3628; fax: 215 573 2207; e-mail: allenf@wharton.upenn.edu.

[1] Helpful comments were made by Adam Dunsby, Lawrence Fisher, Steven Kimbrough, Paul Kleindorfer, Michele Kreisler, James Laing, Josef Lakonishok, George Mailath, and seminar participants at Institutional Investor, J.P. Morgan, the NBER Asset Pricing Program, Ohio State University, Purdue University, the Santa Fe Institute, Rutgers University, Stanford University, University of California, Berkeley, University of Michigan, University of Pennsylvania, University of Utah, Washington University (St. Louis), and the 1995 AFA Meetings in Washington, D.C. We are particularly grateful to Kenneth R. French (the referee), and G. William Schwert (the editor) for their suggestions. Financial support from the National Science Foundation is gratefully acknowledged by the first author and from the Academy of Finland by the second and from the Geewax-Terker Program in Financial Instruments by both. Correspondence should be addressed to Franklin Allen, The Wharton School, University of Pennsylvania, Philadelphia, PA 19104-6367.

0304-405X/99/$ – see front matter
PII: S0304-405X(98)00052-X

in their use in financial economics but so far there has been little formal analysis. The purpose of this paper is to demonstrate how they can be used to find technical trading rules. The techniques can be easily extended to include other types of information such as fundamental and macro-economic data as well as past prices.

Over the years there has been a large literature on the effectiveness of various technical trading rules. The majority of this literature has found that such rules do not make money. For instance, Alexander (1961) tests a number of 'filter rules', which advise a trader to buy if the price rises by a fixed percentage (5%, say) and sell if the price declines by the same percentage. Although such rules appear to yield returns above the buy-and-hold strategy for the Dow Jones and Standard & Poor's stock indices, Alexander (1964) concludes that adjusted for transaction costs, the filter rules are not profitable. These conclusions are supported by the results of Fama and Blume (1966), who find no evidence of profitable filter rules for the 30 Dow Jones stocks. On the whole, the studies during the 1960s provide little evidence of profitable trading rules, and lead Fama (1970) to dismiss technical analysis as a futile undertaking. In a more recent study, Brock et al. (1992) consider the performance of various simple moving average rules in the absence of transaction costs. They find that these rules identify periods to be in the market (long the Dow Jones index) when returns are high and volatility is low and vice versa.

All of this previous literature uses ad hoc specifications of trading rules. This leads to a problem of data snooping whereby the ex post specification of rules can lead to biased test results. We show how genetic algorithms can be used to derive trading rules that are not ad hoc but are in a sense optimal. This approach avoids a potential bias caused by ex post selection, because the rules are chosen by a machine learning algorithm using price data available before the start of the test period.

We consider trading rules for the Standard & Poor's composite index (S&P 500). Our results confirm those of previous studies. They suggest that the markets for these stocks are efficient in the sense that it is not possible to make money after transaction costs using technical trading rules. Like Brock et al. (1992), we are able to identify periods to be in the market when returns are high and volatility is low and out when the reverse is true. A study of how the rules work indicates that many have trading patterns that are like those generated by simple moving average rules. This suggests they are exploiting well-known low-order serial correlation in market indices (see, e.g., Fisher, 1966; French and Roll, 1986; Lo and MacKinlay, 1990).

Section 2 describes genetic algorithms. Section 3 shows how the rules are found and evaluated, and addresses the robustness of the results. Section 4 contains concluding remarks.

2. Genetic algorithms

2.1. Introduction to evolutionary algorithms

Genetic algorithms constitute a class of search, adaptation, and optimization techniques based on the principles of natural evolution. Genetic algorithms were developed by Holland (1962, 1975). Other evolutionary algorithms include evolution strategies (Rechenberg, 1973; Schwefel, 1981), evolutionary programming (Fogel et al., 1966), classifier systems (Holland, 1976, 1980), and genetic programming Koza (1992). For an introduction to genetic algorithms, see Beasley et al. (1993).

An evolutionary algorithm maintains a population of solution candidates and evaluates the quality of each solution candidate according to a problem-specific fitness function, which defines the environment for the evolution. New solution candidates are created by selecting relatively fit members of the population and recombining them through various operators. Specific evolutionary algorithms differ in the representation of solutions, the selection mechanism, and the details of the recombination operators.

In a genetic algorithm, solution candidates are represented as character strings from a given (often binary) alphabet. In a particular problem, a mapping between these genetic structures and the original solution space has to be developed, and a fitness function has to be defined. The fitness function measures the quality of the solution corresponding to a genetic structure. In an optimization problem, the fitness function simply computes the value of the objective function. In other problems, fitness could be determined by a co-evolutionary environment consisting of other genetic structures. For instance, one could study the equilibrium properties of game-theoretic problems whereby a population of strategies evolves with the fitness of each strategy defined as the average payoff against the other members of the population.

A genetic algorithm starts with a population of randomly generated solution candidates.[2] The next generation is created by recombining promising candidates. The recombination involves two parents chosen at random from the population, with the selection probabilities biased in favor of the relatively fit candidates. The parents are recombined through a 'crossover' operator, which splits the two genetic structures apart at randomly chosen locations, and joins

[2] There are many variations of the basic genetic algorithm. The version described here employs continuous reproduction, as opposed to a more conventional approach where the whole generation is replaced by a new one. The idea of continuous reproduction was originally proposed by Holland (1975), and developed by Syswerda (1989) and Whitley (1989). Similarly to Whitley, we also use rank-based selection, instead of the more common method of computing the selection probabilities on the basis of scaled fitness values

a piece from each parent to create an 'offspring' (as a safeguard against the loss of genetic diversity, random 'mutations' are occasionally introduced into the offspring). The algorithm evaluates the fitness of the offspring and replaces one of the relatively unfit members of the population. New genetic structures are produced until the generation is completed. Successive generations are created in the same manner until a well-defined termination criterion is satisfied. The final population provides a collection of solution candidates, one or more of which can be applied to the original problem.

The theoretical foundation of genetic algorithms is laid out in Holland (1975). Maintaining a population of solution candidates makes the search process parallel, allowing an efficient exploration of the solution space. In addition to this explicit parallelism, genetic algorithms are implicitly parallel in that the evaluation of the fitness of a specific genetic structure yields information about the quality of a very large number of 'schemata', or building blocks. The so-called *schema theorem* shows that a genetic algorithm automatically allocates an exponentially increasing number of trials to the best observed schemata. This leads to a favorable tradeoff between exploitation of promising directions of the search space and exploration of less-frequented regions of the space (see also Vose, 1991). However, there is no general result guaranteeing the convergence of a genetic algorithm to the global optimum.

Evolutionary algorithms offer a number of advantages over more traditional optimization methods. They can be applied to problems with a nondifferentiable or discontinuous objective function, to which gradient-based methods such as Gauss–Newton would not be applicable. They are also useful when the objective function has several local optima. Because of the stochastic nature of the selection and recombination operators, evolutionary algorithms are less likely to converge to local maxima than hill-climbing or gradient-type methods. Evolutionary optimization methods can often be successfully applied when the size of the search space poses difficulties for other optimizations methods. In such problems, the parallel trial-and-error process is an efficient way to reduce the uncertainty about the search space.

Evolutionary algorithms do have limitations. Maintaining a population of genetic structures leads to an increase in execution time, due to the number of times the objective function must be evaluated. Since evolutionary algorithms embody very little problem-specific knowledge, they are unlikely to perform better and can be less efficient than special-purpose algorithms in well-understood domains. Despite the schema theorem, the lack of a theoretical convergence result is a weakness. In differentiable problems, an evolutionary algorithm may indeed converge to a point where the gradient is nonzero. This can happen if there is not enough genetic variation left in the population by the time the algorithm approaches the optimum. However, this last drawback can often be avoided by switching to hill-climbing when the genetic algorithm makes no further progress.

One should not view genetic algorithms or other members of the family as a replacement for other optimization methods. It is more productive to regard them as a complement, the usefulness of which depends on the nature of the problem. Even though evolutionary algorithms are not guaranteed to find the global optimum, they can find an acceptable solution relatively quickly in a wide range of problems.

Evolutionary algorithms have been applied to a large number of problems in engineering, computer science, cognitive science, economics, management science, and other fields (for references, see Goldberg, 1989; Booker et al., 1989). The number of practical applications has been rising steadily, especially since the late 1980s. Typical business applications involve production planning, job-shop scheduling, and other difficult combinatorial problems. Genetic algorithms have also been applied to theoretical questions in economic markets, to time series forecasting, and to econometric estimation (see, e.g., Marimon et al., 1990; Dorsey and Mayer, 1995). String-based genetic algorithms have been applied to finding market-timing strategies based on fundamental data for stock and bond markets by Bauer (1994).

2.2. *Genetic programming*

In traditional genetic algorithms, genetic structures are represented as character strings of fixed length. This representation is adequate for many problems, but it is restrictive when the size or the form of the solution cannot be assessed beforehand. Genetic programming, developed by Koza (1992), is an extension of genetic algorithms that partly alleviates the restrictions of the fixed-length representation of genetic structures. For the purposes of this paper, the main advantage of genetic programming is the ability to represent different trading rules in a natural way.

In genetic programming, solution candidates are represented as hierarchical compositions of functions. In these tree-like structures, the successors of each node provide the arguments for the function identified with the node. The terminal nodes (i.e., nodes with no successors) correspond to the input data. The entire tree is also interpreted as a function, which is evaluated recursively by simply evaluating the root node of the tree. The structure of the solution candidates is not specified a priori. Instead, a set of functions is defined as building blocks to be recombined by the genetic algorithm.

The function set is chosen in a way appropriate to the particular problem under study. Koza focuses much of his work on genetic structures that include only functions of a single type. However, genetic programming possesses no inherent limitations about the types of functions, as long as a so-called 'closure' property is satisfied. This property holds if all possible combinations of subtrees result in syntactically valid composite functions. Closure is needed to ensure that the recombination operator is well defined.

Genetic programming also maintains a population of genetic structures. The initial population consists of random trees. The root node of a tree is chosen at random among functions of the same type as the desired composite function. Each argument of that function is then selected among the functions of the appropriate type, proceeding recursively down the tree until a function with no arguments (a terminal node) is reached. The evolution takes place much as in the basic genetic algorithms, selecting relatively fit solution candidates to be recombined and replacing unfit individuals with the offspring.

In genetic programming, the crossover operator recombines two solution candidates by replacing a randomly selected subtree in the first parent with a subtree from the second parent. If different types of functions are used within a tree, the appropriate procedure involves choosing the crossover node at random within the first parent, and then choosing the crossover node within the second parent among the nodes of the *same type* as the crossover node in the first tree. Mutations are introduced by using a randomly generated tree in place of the second parent with a small probability.

Koza applies genetic programming to a diverse array of problems, ranging from symbolic integration to the evolution of ant colonies to the optimal control of a broom balanced on top of a moving cart. As one specific example illustrating the effectiveness of the algorithm, Koza (1992, Chapter 8) applies genetic programming to learning the correct truth table for the so-called six-multiplexer problem. In this problem, there are six binary inputs and one binary output. The correct logical mapping must specify the correct output for each of the 2^6 input combinations. Hence, the size of the search space is $2^{64} \approx 10^{19}$. Using genetic programming, no more than 160,000 individual solution candidates are necessary to find the correct solution with a 99% probability. By comparison, the best solution found in a random search over ten million truth tables produced the correct output for only 52 out of 64 possible input combinations.

3. Finding and evaluating trading rules

3.1. Ex ante optimal trading rules

Most of the previous trading rule studies have sought to test whether particular kinds of technical rules have forecasting ability. A machine learning approach to finding testable trading rules sheds light on a question that is subtly different. Instead of asking whether *specific* rules work, we seek to find out whether in a certain sense *optimal* rules can be used to forecast future returns. Allowing a genetic algorithm to compose the trading rules avoids much of the arbitrariness involved in choosing which rules to test. Were we to use a more traditional search method, we would have to decide the form of the trading rules before searching through the parameter space for the chosen class of rules. By

using a genetic algorithm, we can look for *both* the structure *and* the parameters of the rules at the same time.

The rules found by the genetic algorithm are optimal in a very specific sense. The algorithm finds rules that maximize one particular – though intuitive – measure of fitness. In addition, *some* structure must be imposed on the rule space. In this paper, the search space consists of logical combinations of simple rules that look at moving averages and maxima and minima of past prices. The choice of these rules as building blocks is supported by the analysis of Neftci (1991), who shows that many trading rules relying on specific patterns can be expressed in terms of local extrema of past prices. Moving average rules are used to model potential short-term or long-term trends. Of course, using building blocks similar to trading rules studied in the past poses the danger of data snooping. We try to mitigate this problem by restricting the building blocks to simple rules that have been used in practice for several decades.

3.2. Applying genetic programming to finding trading rules

This paper uses genetic programming to find technical trading rules for a composite stock index. The goal of the algorithm is to find decision rules that divide days into two distinct categories, either 'in' the market (earning the market rate of return) or 'out' of the market (earning the risk-free rate of return). Each genetic structure represents a particular technical trading rule. A trading rule is a function that returns either a 'buy' or a 'sell' signal for any given price history. The trading strategy specifies the position to be taken the following day, given the current position and the trading rule signal. The trading strategy is implemented as a simple automaton, which works as follows: If the current state is 'in' (i.e., holding the market) and the trading rule signals 'sell', switch to 'out' (move out of the market); if the current state is 'out' and the trading rule signals 'buy', switch back to 'in'. In the other two cases, the current state is preserved.

Building blocks for trading rules include simple functions of past price data, numerical and logical constants, and logical functions that allow the combination of low-level building blocks to create more complicated expressions. The root node of each genetic structure corresponds to a Boolean function. This restriction ensures that the trading strategy is well defined.

The function set includes two kinds of functions: real and Boolean. The real-valued functions include a function that computes a moving average of past prices (`average`) in a time window specified by a real-valued argument, rounded to an integer when the rule is evaluated. There are also functions that return the local extrema of prices (`maximum` and `minimum`) during a time window of a given length. Other real-valued functions include arithmetic operators ($+$, $-$, $\div$, $*$) and a function returning the absolute value of the difference between two real numbers (`norm`). Boolean functions include logical functions (`if-then-else`, `and`, `or`, `not`) and comparisons of two real numbers

($>$, $<$). In addition, there are Boolean constants (true, false) and real constants. The Boolean constants are initialized randomly to either of the two truth values, and the real constants are initialized to values drawn from the uniform distribution between zero and two when the initial population of genetic structures is created (and fixed thereafter). There is also a real-valued function (price) that returns the closing price of the current day. Finally, there is a function (lag) that causes its argument function to be applied to a price series lagged by a number of days specified by another argument.

The functions defined above can be used to implement many commonly used technical trading rules. For instance, Fig. 1 shows a 50-day moving average rule on the left and a simple 30-day trading range break rule on the right. The moving average rule asks whether the moving average of closing prices over the past 50 days is less than today's closing price and if so returns a buy (stay in) signal and a sell (stay out) signal otherwise. The 30-day trading range break rule is evaluated in the same way, asking whether today's closing price is above the maximum closing price during the last 30 days.

Fig. 2 depicts how the genetic algorithm combines simple rules to create more complicated ones (a crossover). The two trading rules shown in the top of Fig. 2 are used as the parents. In the first parent, the crossover node chosen at random corresponds to the Boolean subtree designated by the dotted line. In the second parent, the crossover node is selected at random among nodes of the Boolean type. Replacing the subtree in the first parent with the subtree of the second parent generates the trading rule shown in the bottom of Fig. 2. This new rule returns a buy signal if both the moving average rule and the trading range break rule are satisfied, and a sell signal otherwise. Proceeding in the same way, complex and subtle nonlinear rules can be created, although nothing precludes the discovery of quite elementary decision rules.

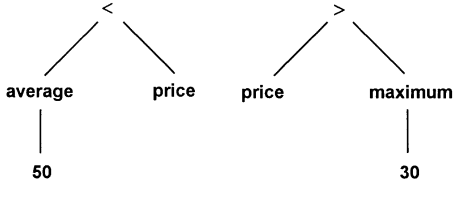

Fig. 1. Genetic structures corresponding to a 50-day moving average rule (left) and a 30-day trading range break rule (right). The moving average rule returns a 'buy' signal if the 50-day moving average of past closing prices is smaller than the closing price today, and a 'sell' signal otherwise. The trading range break rule returns a 'buy' signal if the price is greater than the maximum of the past 30 days' prices and a 'sell' signal otherwise.

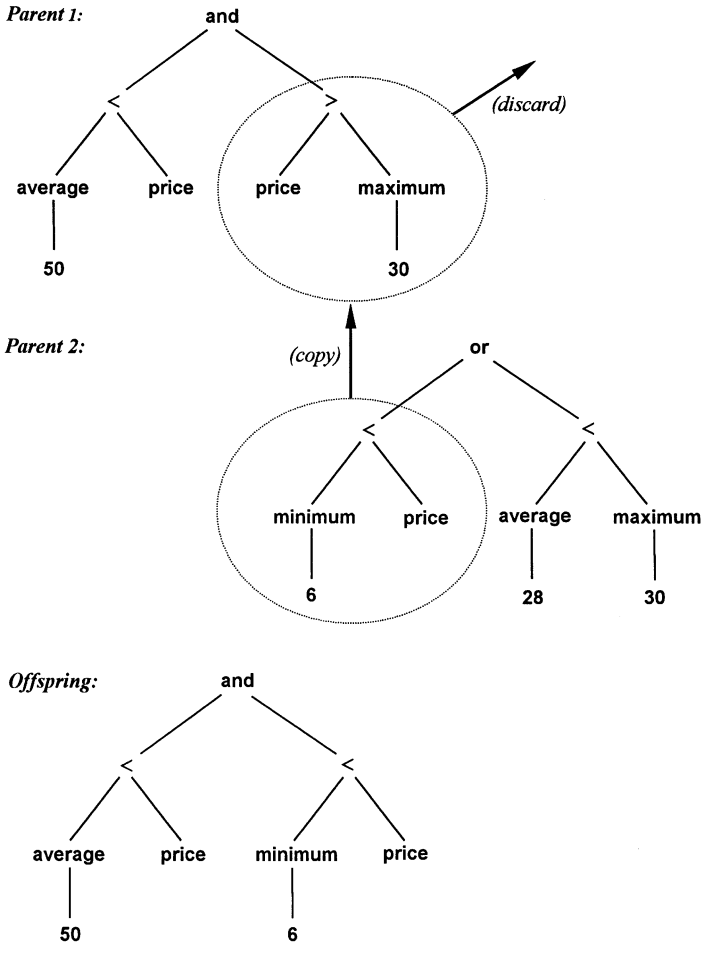

Fig. 2. An illustration of how the genetic algorithm creates a new trading rule by recombining two parent rules. This is known as a 'crossover', and it works as follows. A randomly chosen subtree (designated by the dotted line) in the first parent is discarded. The discarded subtree is replaced by a subtree of the same type (in this case Boolean), randomly chosen from the second parent. The resulting offspring rule is shown in the bottom of the figure.

3.3. Fitness measure

The fitness measure is based on the excess return over the buy-and-hold strategy. This is perhaps the most obvious choice. A more sophisticated fitness measure might take account of risk aversion by, for example, including a term

that penalizes for large daily losses or for large drawdowns of wealth, according to the risk attitude of a particular investor.

The fitness of a rule is computed as the excess return over the buy-and-hold strategy during a *training period*. To evaluate the fitness of a trading rule, it is applied to each trading day to divide the days into periods in (earning the market return) or out of the market (earning a risk-free return). The continuously compounded return is then computed, and the buy-and-hold return and the transaction costs are subtracted to determine the fitness of the rule. The simple return from a single trade (buy at date b_i, sell at s_i) is thus

$$\pi_i = \frac{P_{s_i}}{P_{b_i}} \times \frac{1-c}{1+c} - 1 = \exp\left[\sum_{t=b_i+1}^{s_i} r_t\right] \times \frac{1-c}{1+c} - 1$$

$$= \exp\left[\sum_{t=b_i+1}^{s_i} r_t + \log\frac{1-c}{1+c}\right] - 1, \tag{1}$$

where P_t is the closing price (or level of the composite stock index) on day t, $r_t = \log P_t - \log P_{t-1}$ is the daily continuously compounded return, and c denotes the one-way transaction cost (expressed as a fraction of the price). Let T be the number of trading days and let $r_f(t)$ denote the risk-free rate on day t. Define two indicator variables, $I_b(t)$ and $I_s(t)$, equal to one if a rule signals buy and sell, respectively, and zero otherwise with the indicator variables satisfying the relation $I_b(t) \times I_s(t) = 0\ \forall t$. Lastly, let n denote the number of trades, i.e., the number of buy signals followed by a subsequent sell signal (an open position in the last day is forcibly closed). Then, the continuously compounded return for a trading rule can be computed as

$$r = \sum_{t=1}^{T} r_t I_b(t) + \sum_{t=1}^{T} r_f(t) I_s(t) + n\log\frac{1-c}{1+c}, \tag{2}$$

and the total (simple) return is $\pi = e^r - 1$. The return for the buy-and-hold strategy (buy the first day, sell the last day) is

$$r_{bh} = \sum_{t=1}^{T} r_t + \log\frac{1-c}{1+c}, \tag{3}$$

and the excess return or fitness for a trading rule is given by

$$\Delta r = r - r_{bh}. \tag{4}$$

As short sales can only be made on an uptick, the implementation of simultaneous short sales for a composite stock index is difficult. Consequently, we do not consider short positions. Results by Sweeney (1988) suggest that large institutional investors can achieve one-way transaction costs in the range of

0.1–0.2% (at least after the mid-1970s), and floor traders can achieve considerably lower costs. However, realistic transaction costs including the market impact are likely to be higher for most investors. Initially, we use one-way transaction costs of 0.25%. We also investigate the robustness of the results with transaction costs of 0.1% or 0.5%.

One issue that needs to be addressed in the design of the genetic algorithm is the possibility of overfitting the training data. The task of inferring technical trading rules relies on the assumption that there are some underlying regularities in the data. However, there are going to be patterns arising from noise, and the trick is to find trading rules that generalize beyond the training sample. The problem is common to all methods of non-linear statistical inference, and several approaches have been proposed to avoid overfitting. These include reserving a part of the data as a validation set on which to test the predictions, increasing the amount of training data, penalizing for model complexity, and minimizing the amount of information needed to describe both the model *and* the data (for a discussion of overfitting, see Gershenfeld and Weigend, 1993). We reserve a *selection period* immediately following the training period for validation of the inferred trading rules. After each generation, we apply the fittest rule (based on the excess return in the training period) to the selection period. If the excess return in the selection period improves upon the best previously saved rule, the new rule is saved.

Table 1 summarizes the algorithm used to find trading rules.[3] To start with, an initial population of rules is created at random. The algorithm determines the fitness of each trading rule by applying it to the daily data for the S&P 500 index in the training period. It then creates a new generation of rules by recombining parts of relatively fit rules in the population. After each generation, the best rule in the population is applied to a selection period. If the rule leads to a higher excess return than the best rule so far, the new rule is saved. The evolution terminates when there is no improvement in the selection period for a predetermined number of generations, or when a maximum number of generations is reached. The best rule is then applied to the out-of-sample *test period* immediately following the selection period. If no rule better than the buy-and-hold strategy in the training period emerges in the maximum number of generations, the trial is abandoned.

We use a population size of 500. The size of the genetic structures is limited to 100 nodes and to a maximum of ten levels of nodes. Evolution continues for a maximum of 50 generations, or until there is no improvement for 25 generations. Each trial starts from a different random population.

[3] The computer code for finding trading rules for the S&P 500 index can be obtained using anonymous ftp from opim.wharton.upenn.edu (log in as 'anonymous' and type your e-mail address as password). The C code is located in the directory /pub/programs/risto/jfe99.

Table 1
One trial of the genetic algorithm used to find technical trading rules

Step 1
Create a random rule.
Compute the **fitness** of the rule as the excess return in the **training period** above the buy-and-hold strategy.
Do this 500 times (this is the intial **population**).

Step 2
Apply the fittest rule in the population to the **selection period** and compute the excess return.
Save this rule as the initial best rule.

Step 3
Pick two parent rules at random, using a probability distribution skewed towards the best rule.
Create a new rule by breaking the parents apart randomly and recombining the pieces (this is a **crossover**).
Compute the fitness of the rule as the excess return in the training period above the buy-and-hold strategy.
Replace one of the old rules by the new rule, using a probability distribution skewed towards the worst rule.
Do this 500 times (this is called one **generation**).

Step 4
Apply the fittest rule in the population to the selection period and compute the excess return.
If the excess return improves upon the previous best rule, save as the new best rule.
Stop if there is no improvement for 25 generations or after a total of 50 generations. Otherwise, go back to Step 3.

3.4. Data

We use daily data for the S&P 500 index from 3 January 1928 to 29 December 1995.[4] For 1929–1991, the one-month risk-free rates corresponding to Treasury bills are from the Center for Research in Security Prices with Datastream providing data for 1992 to 1995. Since the S&P 500 series is clearly nonstationary (rising from 17 in early 1928 to 615 by the end of 1995), we transform the data by dividing each day's price by a 250-day moving average. The normalized prices have an order of magnitude around one. Since we use the first year of data for normalization, the usable data set covers 1929–1995. All test results are based on the unnormalized data.

[4] The authors are grateful to G. William Schwert for providing us with pre-1963 data

3.5. Results

To find trading rules for the S&P 500 index, we conduct 100 independent trials, saving one rule from each trial. We use a one-way transaction cost of 0.25% initially. To guard against potential data snooping in the choice of time periods, we use ten successive training periods and report the summarized results for each case. The five-year training and two-year selection periods start in 1929, 1934, …, 1974, with the out-of-sample test periods starting in 1936, 1941, …, 1981. For instance, the first trial uses years 1929–1933 as the training period, 1934–1935 as the selection period, and 1936–1995 as the test period. For each of the ten training periods, we carry out ten trials.

We find a total of 89 trading rules in the 100 trials (recall that no rules are saved from a trial if the excess returns are not positive in the selection period). As an overview of the results, Fig. 3 shows the average annual excess return and the average number of trades per year. The excess returns are predominantly negative. The trading frequency is relatively low, with an average of 3.8 trades per year. On average, the rules are long in the market for 57% of the days. The results are generally consistent for rules found from different training periods, except for those found in 1929–1935 and 1969–1975. The trading rules from these two periods stay mostly out of the market during the test period, which is perhaps not surprising given that both training periods involve substantial negative returns to stocks.

Panel A of Table 2 summarizes the statistical tests of the out-of-sample returns with the most reasonable level of transaction costs of 0.25%. The

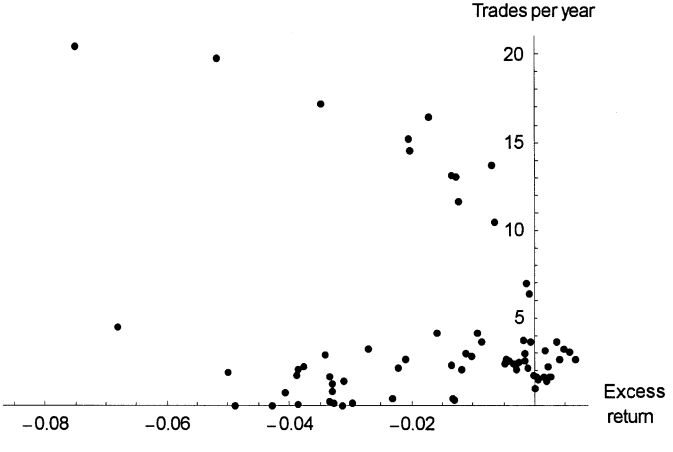

Fig. 3. The yearly excess return over a buy-and-hold strategy and the average number of trades per year for trading rules found by the genetic algorithm, using 0.25% one-way transaction costs. Each point is labeled with an indicator of the training period, with labels 1, 2, … corresponding to the training periods 1929–1933, 1934–1938, etc.

Table 2
A summary of the test results for trading rules found by the genetic algorithm for various levels of one-way transaction costs. Panel A considers the most reasonable case of 0.25%. Panels B and C consider 0.1% and 0.5% to show the sensitivity of the results. The first column shows the in-sample training and selection period, with a corresponding out-of-sample test period starting the following year and extending to the end of 1995. K is the number of trading rules found by the algorithm. *Excess* denotes the average yearly excess return in the out-of-sample period above the buy-and-hold strategy after transaction costs. K^+ is the number of rules with a positive excess return. Each trading rule divides the days into periods 'in' (long in the market) and 'out' of the market (earning a risk-free rate of return). Average daily returns during 'in' and 'out' periods are denoted by r_b and r_s, respectively, with the standard deviation of daily returns denoted by σ_b and σ_s and the number of days by N_b and N_s. T^* is the number of t-statistics for $r_b - r_s$ significant at the 5% level

In-sample	K	Excess	K^+	N_b	r_b	σ_b	N_s	r_s	σ_s	$r_b - r_s$	T^*
Panel A: Transaction costs of 0.25%											
1929–35	10	−0.0336	0	1614	−0.000068	0.001870	14202	+0.000220	0.008219	−0.000288	0
1934–40	10	−0.0105	0	9374	+0.000347	0.007652	4940	+0.000263	0.010297	+0.000085	8
1939–45	10	−0.0281	1	4746	+0.000371	0.010881	8081	+0.000582	0.008264	−0.000211	2
1944–50	10	−0.0022	3	9112	+0.000378	0.007665	2305	−0.000029	0.009777	+0.000406	6
1949–55	4	−0.0031	2	8872	+0.000295	0.008337	1236	−0.000051	0.008547	+0.000346	1
1954–60	8	+0.0014	7	8412	+0.000298	0.008506	436	−0.000857	0.008088	+0.001155	3
1959–65	10	−0.0107	0	6890	+0.000427	0.008592	700	−0.001473	0.012018	+0.001900	9
1964–70	7	−0.0386	2	4714	+0.000519	0.008514	1643	−0.000443	0.010932	+0.000962	5
1969–75	10	−0.0440	0	412	+0.003290	0.008771	4684	+0.000364	0.009180	+0.002926	2
1974–80	10	−0.0359	2	1145	+0.000183	0.003560	2687	+0.000320	0.010334	−0.000137	0
Average	8.9	−0.0205	1.7	5529	+0.000604	0.007435	4091	−0.000110	0.009566	+0.000714	3.6
Panel B: Transaction costs of 0.1%											
1929–35	10	−0.0372	0	13	−0.000181	0.002156	15804	+0.000244	0.009116	−0.000424	0
1934–40	10	−0.0031	5	9254	+0.000371	0.007576	5060	+0.000013	0.009707	+0.000359	10
1939–45	10	−0.0273	0	3690	+0.000400	0.010661	9137	+0.000225	0.007880	+0.000176	3
1944–50	10	+0.0110	7	9118	+0.000474	0.007555	2299	−0.000580	0.010395	+0.001054	9
1949–55	10	+0.0109	6	8251	+0.000464	0.008059	1857	−0.000430	0.009173	+0.000894	6
1954–60	8	+0.0042	8	8695	+0.000290	0.008623	153	−0.001043	0.008108	+0.001332	7

1959–65	10	+ 0.0358	10	6094	+ 0.000645	0.008151	1496	− 0.001380	0.011491	+ 0.002025	10
1964–70	10	− 0.0013	5	3439	+ 0.000816	0.008303	2918	− 0.000287	0.010288	+ 0.001103	10
1969–75	10	− 0.0511	0	2704	+ 0.000943	0.008355	2391	− 0.000275	0.010066	+ 0.001218	10
1974–80	10	− 0.0443	0	2303	+ 0.000841	0.008500	1529	− 0.000296	0.011128	+ 0.001137	10
Average	9.8	− 0.0102	4.1	5356	+ 0.000506	0.007794	4264	− 0.000381	0.009735	+ 0.000887	7.5
Panel C: Transaction costs of 0.5%											
1929–35	10	− 0.0294	0	3230	+ 0.000076	0.002363	12587	+ 0.000193	0.007320	− 0.000117	0
1934–40	10	− 0.0220	0	7848	+ 0.000258	0.007552	6466	+ 0.000122	0.009456	+ 0.000136	8
1939–45	10	− 0.0316	0	5276	+ 0.000399	0.010940	7550	+ 0.000738	0.007993	− 0.000338	0
1944–50	5	− 0.0161	0	8886	+ 0.000392	0.007464	2531	+ 0.000031	0.011322	+ 0.000360	3
1949–55	2	+ 0.0035	1	9489	+ 0.000294	0.008085	619	− 0.000282	0.012493	+ 0.000576	1
1954–60	6	− 0.0001	4	8738	+ 0.000285	0.008624	110	− 0.001168	0.007857	+ 0.001453	4
1959–65	10	− 0.0154	2	6881	+ 0.000254	0.008425	709	+ 0.000155	0.012507	+ 0.000099	1
1964–70	0										
1969–75	10	− 0.0512	0	193	+ 0.002583	0.010719	4902	+ 0.000375	0.009165	+ 0.002208	1
1974–80	10	− 0.0502	0	301	+ 0.000037	0.000950	3531	+ 0.000404	0.009697	− 0.000367	0
Average	7.3	− 0.0236	0.7	5649	+ 0.000509	0.007236	4334	+ 0.000063	0.009757	+ 0.000446	1.8

average excess returns after transaction costs are negative for nine out of the ten test periods. In most of the periods, there are only a few rules with positive excess returns. Hence, the results are consistent with market efficiency. It is interesting to note, however, that in seven out of the ten test periods, the average daily return during 'in' periods is higher than the unconditional return, while the return during 'out' periods is lower. The difference between 'in' and 'out' returns is statistically significant for 36 rules out of the total of 89 (the t-statistic for the difference is

$$t = (r_b - r_s) \Big/ \left(s \sqrt{\frac{1}{N_b} + \frac{1}{N_s}} \right),$$

where s^2 is the sample variance). Also, the standard deviation during 'in' days is smaller than during 'out' days for most training periods. The volatility of the annual trading rule returns is 10.0%, averaged over the trading rules and over the different out-of-sample periods. The sample standard deviation of the annual returns for the S&P 500 index during the same period is 14.7%. It is also interesting that constructing a 'portfolio' of trading rules appears to diversify risk in much the same way as adding companies to a stock portfolio. For a composite trading strategy in which an equal amount of capital is allocated to each of the rules from a particular trading period, the volatility drops to 8.7%, averaged over the out-of-sample periods.

We investigate the impact of trading costs on the results by carrying out two more sets of 100 trials, using transaction costs of 0.1% and 0.5% but holding the other parameters fixed. Panel B of Table 2 summarizes the results for low transaction costs (0.1%) and Panel C for high transaction costs (0.5%). With low transaction costs, the trading frequency is high, with an average of 18 trades per year. Average excess returns are negative for six out of the ten test periods, again consistent with market efficiency. The difference between daily returns during 'in' and 'out' periods is positive for nine periods out of ten, and statistically significant for almost all the rules. With high transaction costs, the trading frequency drops to an average of 1.4 trades per year. The excess returns are negative for nine periods out of ten. The difference between daily returns during 'in' and 'out' periods is positive for six periods out of ten, but statistically significant in only a few cases. The pattern of reduced volatility during days long in the market remains similar for different levels of transaction costs. The proportion of days long in the market is around 56–57% regardless of the level of transaction costs. These results indicate that while trading rules found using unrealistically low transaction costs lead to higher returns and better forecasting ability, the excess returns above the buy-and-hold strategy generally remain negative. High transaction costs lead to diminished forecasting ability.

Overall, the out-of-sample results show that the rules found by the genetic algorithm do not earn consistent excess returns after transaction costs. It is

interesting that in most of the training periods, the rules appear to have some forecasting ability in the sense that the average daily return during days in the market is higher compared to days out of the market, and the volatility is generally lower during the days the rules are long in the market. Even though the rules do not lead to higher absolute returns than a buy-and-hold strategy, the reduced volatility might still make them attractive to some investors on a risk-adjusted basis. Finally, there appears to be a diversification benefit in using a number of trading rules in a portfolio, with the position to be taken in each period determined as a weighted average of the trading rule signals.

The finding that the trading rules tend to be long in low-volatility periods is somewhat surprising, given that the fitness of the rules does not depend on the volatility of the returns. However, it is possible to make the case that the observed pattern is what one might expect based on economic theory. In most equilibrium models, expected excess returns are positively related to risk as measured by the expected volatility (Merton, 1980). French et al. (1987) find that ex post stock market returns are negatively related to the unexpected change in volatility, supporting the theoretical risk-return relation. For instance, when the volatility turns out to be higher than expected, investors revise their volatility forecasts upwards, requiring higher expected returns in the future, or lower stock prices and hence lower realized returns at the present. Conversely, when the volatility is lower than expected, investors revise their volatility forecasts downwards, requiring lower expected returns and higher stock prices and realized returns now. Applied to the present study, high ex post volatility when the rules are out of the market would be consistent with the theory if the high-volatility periods represent unexpected increases in risk. The trading rule positions will change when the rules pick up subsequent higher realized returns. Similarly, if the low volatility when the rules are long in the market were unexpected, the trading rule signals would change when the downwards-revised expected returns start to be realized.

3.6. Characterizing the trading rules

There are a number of interesting questions regarding the nature of the rules found by the genetic algorithm. This section describes what the rules look like and analyzes what kind of price behavior triggers the trading rule signals. We analyze one rule for each level of transaction costs and for each training period.

The structure and complexity of the rules found by the algorithm vary across different trials. The size of the rules varies from nine to 94 nodes, with a depth between five and ten levels. While many of the rules appear to be quite complicated at first sight, there is often a lot of redundant material in the form of duplicated subtrees. If some of these subtrees are never visited when the rules are evaluated, seemingly complex tree structures can be effectively similar to much

simpler ones. Consequently, measuring the complexity of the trading rules cannot in general be done by inspection of the tree structures.

One measure of the complexity of the trading rules can be obtained by comparing the behavior of the rules to a class of simple technical trading rules. The simple rules are of the form illustrated in Fig. 1. These rules compare one indicator to another and take a position in (out of) the market if the value of the first indicator is greater than (less than or equal to) the value of the second one. Each indicator is chosen from the set {max, min, average}, and the length of the time window for an indicator is chosen from the set {1, 2, 3, 4, 5, 10, 15, 20, 40, 60, 80, 100, 150, 200, 250}. In addition, we consider rules that compare the normalized price to a constant in the range from 0.9 to 1.1 with increments of 0.01. For each rule, we retain the combination of indicators that leads to the highest correlation between the daily positions, after evaluating all the combinations during the appropriate out-of-sample period.

With 0.25% transaction costs, most of the rules can be matched quite well with a rule that compares the normalized price to a constant. These rules are effectively similar to a 250-day moving average rule for unnormalized prices. In two periods, there is no close match among the simple rules. With 0.1% transaction costs, about half the rules are similar to a rule that compares the normalized price to a constant, while the remaining rules resemble either 10-to-40-day moving average rules or a trading range break rule comparing today's price to a three-day minimum. With 0.5% transaction costs, rules from the early training periods are similar to a rule comparing the normalized price to a constant, while the rules from the late training periods match none of the simple rules considered.

As an example of what the trading rules look like, let us first consider the best rule from the training period 1964–1968, learned using transaction costs of 0.25%. The tree structure shown in Fig. 4 can be reduced to the expression $\mathrm{price} \times \mathrm{minimum(price)} - 0.0688 \times \mathrm{price} > 0.8943$. This is roughly equivalent to a rule that takes a position in the market when normalized prices are above 0.89 and is out of the market otherwise. If we were to use unnormalized prices, an equivalent rule would take a long position when the price rises within 2% of the 250-day moving average or higher.

Fig. 5a illustrates the price behavior during the trades signaled by this rule. Each line represents the cumulative market return (without transaction costs) for a trade in a long position. When the rule returns a 'buy' signal this becomes day 0 and the line represents the price pattern that is realized until the rule returns a 'sell' signal, at which point the line ends. The lines for each trade are superimposed on top of each other. Fig. 5a shows that once the rule takes a long position, it can last up to several years. Fig. 5b shows the cumulative market returns after rule 1 switches out of the market. Again each line represents a particular trade. When the rule generates a sell signal this becomes day 0 and

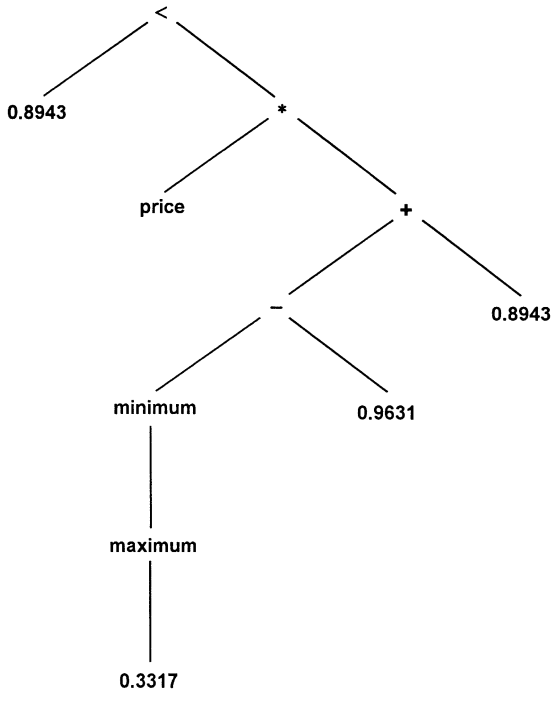

Fig. 4. An example of the trading rules found by the genetic algorithm. The tree structure can be reduced to the expression price × minimum (price) − 0.0688 × price > 0.893. This is roughly equivalent to a rule that takes a position in the market when normalized prices are above 0.89, and out of the market otherwise.

the line shows the price path until the rule generates a buy signal. Positions out of the market are usually covered well within a year.

As a more complicated example, consider the best rule from the same training period (1964–1968) but learned using lower transaction costs (0.1%). After simplification of the structure shown in Fig. 6, the rule is found to be equivalent to the expression lag(price,/1) > average(price/‖price − 1.1727‖). This rule compares a moving average of past prices to the price lagged by one day from the current date, with a nonlinear subexpression (price/‖price − 1.1727‖) determining the time window of the moving average. During bear markets, the rule looks for very short-term trends, with the moving average time window varying between three and four days. In bull markets, the length of the time window increases to 16 days at a normalized price level of 1.1, and rises rapidly thereafter to soon use all the available price history. Loosely speaking, this rule corresponds to a hypothesis that stock prices are in general going up but are prone to corrections after prolonged bull

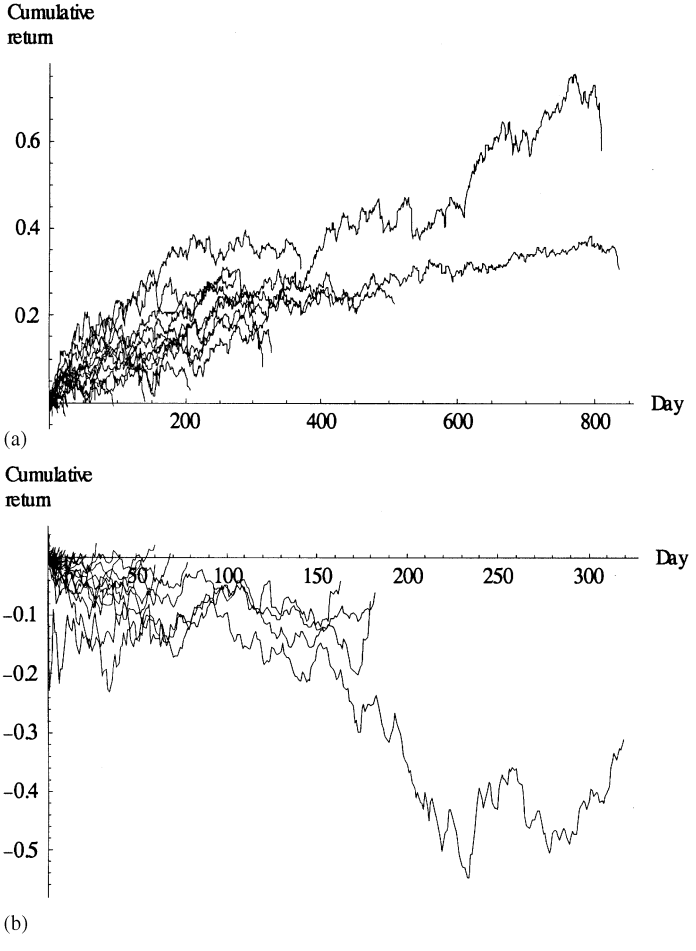

Fig. 5. a. The cumulative market return during each trade going *long in the market* for a rule found in 1964–1970 using 0.25% transaction costs. Each line starts when the rule returns a 'buy' signal and ends when the rule returns a 'sell' signal. The graphs for all the trades in the test period of 1971–1995 are superimposed on top of each other. b. The cumulative market return during each trade *out of the market* for a rule found in 1964–70 using 0.25% transaction costs. Each line starts when the rule returns a 'sell' signal and ends when the rule returns a 'buy' signal. The graphs for all the trades in the test period of 1971–1995 are superimposed on top of each other.

markets. This rule is quick to take a long position in bear markets, but is increasingly hesitant to do so when prices are high.

Fig. 7a and b show the price behavior during the trades signaled by this rule. The trading frequency is much higher than for the medium-transaction-cost rule from the same period. Fig. 7a shows that trades in a long position typically last

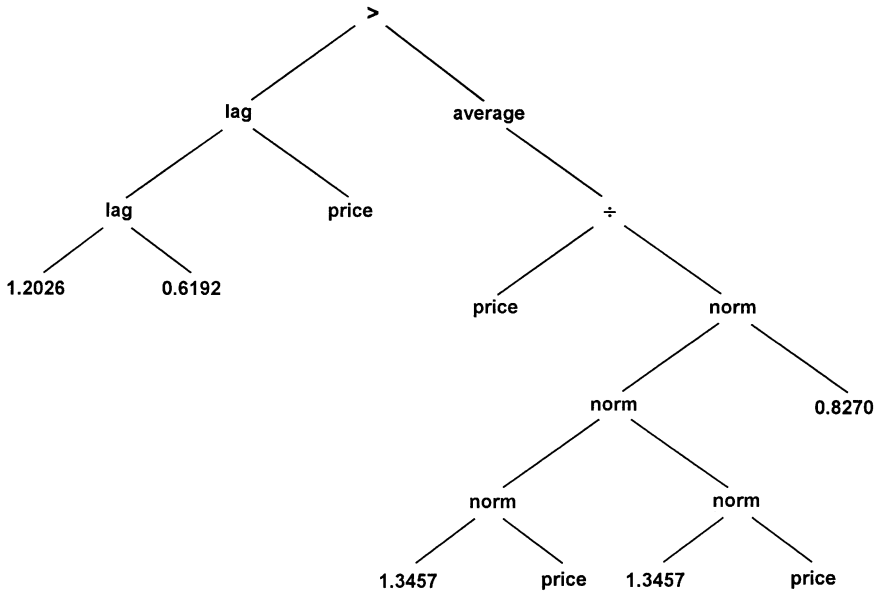

Fig. 6. An example of the trading rules found by the genetic algorithm. The tree structure is equivalent to the expression $\mathtt{lag(price,1)} > \mathtt{average(price/\|price - 1.1727\|)}$. This rule compares a moving average of past prices to the price lagged by one day from the current date, with a nonlinear subexpression $(\mathtt{price/\|price - 1.1727\|})$ determining the time window of the moving average.

for one to four weeks. Without transaction costs, most of the trades are profitable. The rule switches out of the market after a large negative return, consistent with its trend-following nature. Fig. 7b shows that after a sell signal small negative returns tend to persist for a couple of weeks. The rule switches back into the index after a large positive return.

3.7. Robustness of the results

The out-of-sample test results indicate that the trading rules do not earn consistent excess returns after transaction costs. Nevertheless, they appear to have some ability to forecast daily returns. In this section, we consider potential explanations for this. It is well known that there is low-order serial correlation in market indexes (see, e.g., Fisher, 1966; French and Roll, 1986; Lo and MacKinlay, 1990). We start by considering the effect of incorporating a lag of one day in the implementation of trades. We then discuss the biases introduced by the exclusion of dividends in our data set. We also address the effect of the 1987 stock market crash on the results.

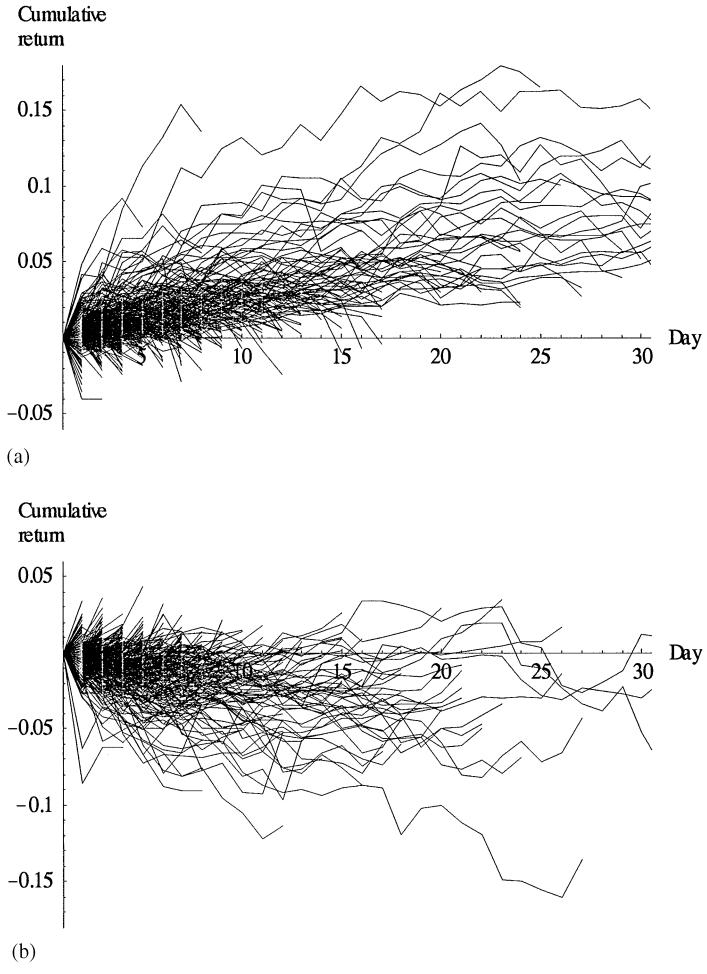

Fig. 7. a. The cumulative market return during each trade going *long in the market* for a rule found in 1964–1970 using 0.1% transaction costs. Each line starts when the rule returns a 'buy' signal and ends when the rule returns a 'sell' signal. The graphs for all the trades in the test period of 1971–1995 are superimposed on top of each other. b. The cumulative market return during each trade *out of the market* for a rule found in 1964–1970 using 0.1% transaction costs. Each line starts when the rule returns a 'sell' signal and ends when the rule returns a 'buy' signal. The graphs for all the trades in the test period of 1971–1995 are superimposed on top of each other.

To test whether the genetic algorithm exploits low-order serial correlation in returns, the ex post best rules are retested with a one-day lag before the trading signals are implemented. Panel A of Table 3 shows the test results without a lag for 0.25% transaction costs. Panel B of Table 3 shows the results with a one-day

Table 3
Test results for the ex post best trading rules found by the genetic algorithm, using one-way transaction costs of 0.25%. Panel A shows the results with trading signals computed and implemented using the same day's closing price. Panel B shows the results with trading signals delayed by one day. In each panel, the first column shows the in-sample training and selection period, with a corresponding out-of-sample test period starting the following year and extending to the end of 1995. *Excess* denotes the average yearly excess return in the out-of-sample period above the buy-and-hold strategy after transaction costs. Each trading rule divides the days into periods 'in' (long in the market) and 'out' of the market (earning a risk-free rate of return). Average daily returns during 'in' and 'out' periods are denoted by r_b and r_s, respectively, with the standard deviation of daily returns denoted by σ_b and σ_s and the number of days by N_b and N_s. T-statistics are given in parentheses

In-sample	Excess	N_b	r_b	(t)	σ	N_s	r_s	(t)	σ	$r_b - r_s$	(t)
Panel A: *Test results using signals computed and implemented using the closing price*											
1934–40	−0.0030	9952	+0.000388	(+0.966)	0.007390	4362	+0.000047	(−1.663)	0.009888	+0.000340	(2.277)
1939–45	−0.0188	6244	+0.000430	(+1.180)	0.008606	6583	+0.000134	(−1.139)	0.008061	+0.000296	(2.009)
1944–50	+0.0048	9085	+0.000410	(+0.966)	0.007805	2332	−0.000136	(−2.328)	0.009660	+0.000547	(2.864)
1949–55	+0.0107	8566	+0.000411	(+1.243)	0.008012	1542	−0.000595	(−3.710)	0.010310	+0.001007	(4.326)
1954–60	+0.0023	6236	+0.000393	(+0.889)	0.007654	2612	−0.000036	(−1.576)	0.010563	+0.000429	(2.137)
1959–65	−0.0178	6558	+0.000531	(+1.865)	0.008597	1032	−0.001539	(−6.023)	0.010763	+0.002070	(6.906)
1964–70	+0.0008	4913	+0.000387	(+1.865)	0.007751	1444	−0.000002	(−1.113)	0.013195	+0.000389	(1.402)
1974–80	+0.0052	2906	+0.000527	(+0.558)	0.007903	926	−0.000021	(−1.176)	0.013745	+0.000548	(1.505)
Average	−0.0020	6808	+0.000435		0.007965	2604	−0.000269		0.010773	+0.000703	
Panel B: *Test results using signals delayed by one day*											
1934–40	−0.0069	9951	+0.000366	(+0.764)	0.007433	4363	+0.000097	(−1.314)	0.009816	+0.000269	(1.801)
1939–45	−0.0177	6243	+0.000439	(+1.250)	0.008673	6584	+0.000126	(−1.206)	0.007992	+0.000313	(2.127)
1944–50	+0.0010	9084	+0.000391	(+0.804)	0.007778	2333	−0.000064	(−1.938)	0.009749	+0.000455	(2.385)
1949–55	+0.0158	8565	+0.000288	(+0.242)	0.007964	1543	+0.000092	(−0.722)	0.010553	+0.000196	(0.842)
1954–60	+0.0040	6235	+0.000403	(+0.957)	0.007720	2613	−0.000059	(−1.697)	0.010445	+0.000462	(2.300)
1959–65	−0.0807	6557	+0.000243	(−0.043)	0.008574	1033	+0.000291	(+0.138)	0.011046	−0.000047	(−0.158)
1964–70	−0.0121	4912	+0.000322	(+0.131)	0.008420	1445	+0.000221	(−0.289)	0.011721	+0.000101	(0.364)
1974–80	−0.0172	2905	+0.000412	(+0.072)	0.009004	927	+0.000341	(−0.151)	0.011431	+0.000070	(0.193)
Average	−0.0184	6808	+0.000358		0.008196	2604	+0.000131		0.010344	+0.000227	

lag. Using delayed signals, the average difference in returns during 'in' and 'out' days drops from seven to two basis points. The pattern of reduced volatility during days long in the market is not affected by the delay. In the case of 0.1% transaction costs, a one-day delay removes all the difference between returns during 'in' and 'out' days. In the case of 0.5% transaction costs, there is little difference between the returns with or without a one-day lag. These results, together with simple characterizations of the rule obtained in the previous section, indicate that with low or medium transaction costs, most of the forecasting ability can be explained by low-order positive serial correlation in stock index returns. It is interesting that the trading rules produced by the genetic algorithm apparently exploit a well-known feature of the data.

The computation of the S&P 500 index ignores the payment of dividends on the component stocks. Ignoring dividends affects the results in two ways. Any seasonality can distort the results, if trading rules happen to pick periods to be out of the market when a disproportionate number of stocks go ex-dividend. There is, in fact, a rather pronounced yearly pattern in dividend payments, which tend to be concentrated in February, May, August, and November (Luskin, 1987, pp. 140,141). The clustering of dividends is unlikely to affect the results, because the calendar dates are not part of the information set of the genetic algorithm. However, ignoring the dividend yield leads to underestimation of the buy-and-hold return. The trading rule returns are underestimated, too, but to a lesser extent. Comparing the results for a subperiod using a value-weighted index of NYSE and AMEX stocks with and without dividends shows that the main effect of the dividends is to lower the annual returns as expected, while the pattern in daily returns and volatility is not affected (the details are available upon request).

Since the trading rules use daily data, the extreme market volatility around the stock market crash in October 1987 crash could have a potentially large impact on the results. We explore these effects by testing the rules on a data set that excludes the crash. Since the level of the S&P 500 index at the end of 1986 is roughly the same as at the start of 1988, a simple way to construct such a data set is to exclude all of 1987. For the case of 0.25% transaction costs, the average difference in daily returns during 'in' and 'out' days is generally within one basis point of the results shown in Table 2. The rules from the training period of 1969–1975 lose their forecasting ability when the crash period is excluded from the analysis. When 1987 is excluded, for most rules the volatility goes down proportionally during both 'in' and 'out' periods. Averaged over the different rules and over the out-of-sample periods, the sample standard deviation of daily returns during 'in' days decreases from 0.007435 (from Table 2, Panel A) to 0.006308. The standard deviation during 'out' days decreases from 0.009566 to 0.008804. The pattern of reduced volatility during days long in the market is still evident. With low transaction costs, the results are very similar to those in Table 2, Panel B. For high transaction costs, the results are robust with respect

to the crash in all the training periods except 1969–1975, when any forecasting ability is lost (the details are available upon request).

4. Concluding remarks

We use a genetic algorithm to learn technical trading rules rather than having them exogenously specified as in previous studies. Out of sample, the rules do not earn excess returns over a simple buy-and-hold strategy after transaction costs. The rules take long positions when returns are positive and daily volatility is low, and stay out of the market when returns are negative and volatility is high. The results concerning daily returns are sensitive to transaction costs but are robust to the impact of the 1987 stock market crash. The pattern in volatility is robust to different levels of transaction costs, to the impact of dividends, and to the 1987 crash. Introducing a one-day delay to trading signals removes most of the forecasting ability, indicating that the rules exploit positive low-order serial correlation in stock index returns.

The results raise the important question of why there appears to be a systematic relation between trading rule signals and volatility. One hypothesis is that the trading rule results can be explained by investors' reactions to changes in the expected volatility. In most equilibrium models, expected excess returns are positively related to risk as measured by the expected volatility of the market (Merton, 1980). When volatility is lower (higher) than expected, expected returns decrease (increase), and the corresponding upward (downward) revision of prices is picked up by the trading rules. Given the negative excess returns after trading costs and the lack of ability to forecast returns with delayed signals, markets process the information about the expected volatility in an efficient way.

This paper studies a base case in which a genetic algorithm is applied to a broad stock index, and finds little evidence of economically significant technical trading rules. Rules learned by evolutionary algorithms could also be tested on liquid markets with low transaction costs, including financial futures, commodities, and foreign exchange markets. One could also develop the methodology further. The genetic algorithm we use is a relatively simple one, and the parameters are not necessarily optimal. More importantly, the current algorithm uses very limited information for its inputs. It would be interesting to apply a similar technique to learn fundamental trading rules by changing the building blocks to include the desired fundamental variables.

References

Alexander, S.S., 1961. Price movements in speculative markets: trends or random walks. Industrial Management Review 2, 7–26.

Alexander, S.S., 1964. Price movements in speculative markets: trends or random walks. In: Cootner, P. (Ed.), The Random Character of Stock Market Prices, vol. 2, MIT Press, Cambridge, pp. 338–372.

Bauer, R.J. Jr., 1994. Genetic Algorithms and Investment Strategies. Wiley, New York.

Beasley, D., Bull, D.R., Martin, R.R., 1993. An overview of genetic algorithms: part I, fundamentals. University Computing 15, 58–69.

Booker, L.B., Goldberg, D.E., Holland, J.H., 1989. Classifier systems and genetic algorithms. Artificial Intelligence 40, 235–282.

Brock, W., Lakonishok, J., LeBaron, B., 1992. Simple technical trading rules and the stochastic properties of stock returns. Journal of Finance 47, 1731–1764.

Dorsey, R.E., Mayer, W.J., 1995. Genetic algorithms for estimation problems with multiple optima, nondifferentiability, and other irregular features. Journal of Business and Economic Statistics 13, 53–66.

Fama, E.F., 1970. Efficient capital markets: a review of theory and empirical work. Journal of Finance 25, 383–417.

Fama, E.F., Blume, M.E., 1966. Filter rules and stock market trading. Security prices: a supplement. Journal of Business 39, 226–241.

Fisher, L., 1966. Some new stock-market indexes. Journal of Business 39, 191–225.

Fogel, L.J., Owens, A.J., Walsh, M.J., 1966. Artificial Intelligence Through Simulated Evolution. Wiley, New York.

French, K.R., Roll, R., 1986. Stock return variances: the arrival of information and the reaction of traders. Journal of Financial Economics 17, 5–26.

French, K.R., Schwert, G.W., Stambaugh, R.F., 1987. Expected stock returns and volatility. Journal of Financial Economics 19, 3–29.

Gershenfeld, N.A., Weigend, A.S., 1993. The future of time series: learning and understanding. In: Weigend, A.S., Gershenfeld, N.A. (Eds.), Time Series Prediction: Forecasting the Future and Understanding the Past, Santa Fe Institute Studies in the Sciences of Complexity, Proceedings vol. XV. Addison–Wesley, Reading, pp. 1–70.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison–Wesley, Reading.

Holland, J.H., 1962. Outline for a logical theory of adaptive systems. Journal of the Association for Computing Machinery 3, 297–314.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Holland, J.H., 1976. Adaptation. In: Rosen, R., Snell, F.M. (Eds.), Progress in Theoretical Biology, vol. 4, Academic Press, New York, pp. 263–293.

Holland, J.H., 1980. Adaptive algorithms for discovering and using general patterns in growing knowledge-bases. International Journal of Policy Analysis and Information Systems 4, 217–240.

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge.

Lo, A.W., MacKinlay, A.C., 1990. An econometric analysis of nonsynchronous trading. Journal of Econometrics 45, 181–211.

Luskin, D.L., 1987. Index Options and Futures. Wiley, New York.

Marimon, R., McGrattan, E., Sargent, T.J., 1990. Money as a medium of exchange in an economy with artificially intelligent agents. Journal of Economic Dynamics and Control 14, 329–373.

Merton, R.C., 1980. On estimating the expected return on the market: an exploratory investigation. Journal of Financial Economics 8, 323–361.

Neftci, S.N., 1991. Naive trading rules in financial markets and Wiener-Kolmogorov prediction theory: a study of 'technical analysis'. Journal of Business 64, 540–571.

Rechenberg, I., 1973. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog Verlag, Stuttgart.

Schwefel, H.-P., 1981. Numerical Optimization of Computer Models. Wiley, New York.
Sweeney, R.J., 1988. Some new filter rule tests: methods and results. Journal of Financial and Quantitative Analysis 23, 285–300.
Syswerda, G., 1989. Uniform crossover in genetic algorithms. In: Schaffer, D.J. (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, pp. 2–9.
Vose, M.D., 1991. Generalizing the notion of a schema in genetic algorithms. Artificial Intelligence 50, 385–396.
Whitley, D., 1989. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: Schaffer, D.J. (Ed.), Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, pp. 116–121.