Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Computers & Operations Research 31 (2004) 1033–1047

computers & operations research

www.elsevier.com/locate/dsw

Generating trading rules on the stock markets with genetic programming

Jean-Yves Potvin[a,c,*], Patrick Soriano[a,b], Maxime Vallée[b]

[a]Centre de Recherche sur les Transports, Université de Montréal, C.P. 6128, succ. Centre-ville, Montréal, Québec, Canada H3C 3J7

[b]Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-ville, Montréal, Québec, Canada H3C 3J7

[c]École des Hautes Études Commerciales, 3000 Chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7

Abstract

Technical analysis is aimed at devising trading rules capable of exploiting short-term fluctuations on the financial markets. Recent results indicate that this market timing approach may be a viable alternative to the buy-and-hold approach, where the assets are kept over a relatively long time period. In this paper, we propose genetic programming as a means to automatically generate such short-term trading rules on the stock markets. Rather than using a composite stock index for this purpose, the trading rules are adjusted to individual stocks. Computational results, based on historical pricing and transaction volume data, are reported for 14 Canadian companies listed on the Toronto stock exchange market.

Keywords: Trading rules; Stock markets; Technical analysis; Genetic programming

1. Introduction

In stock exchange markets, the "buy-and-hold" approach is a well-known strategy among traders. Basically, if a company and its activity sector look promising, the trader buys and keeps his assets over a relatively long time period. An alternative approach, known as market timing, is more dynamic and focuses on short-term fluctuations. Through technical analysis, trading rules are devised to generate appropriate buying and selling signals over short time periods. The purpose of this paper is to demonstrate that genetic programming, a recent development in the field of evolutionary algorithms, can be exploited to automatically generate such trading rules.

* Corresponding author.
E-mail address: jean-yues.potvin@umontreal.ca (J.-Y. Potvin).

0305-0548/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0305-0548(03)00063-7

There is a fairly large literature related to technical analysis in various financial domains. The first results in the 1960s and 1970s supported the "efficient market hypothesis", which states that there should not be any discernable and exploitable pattern in the data, as financial markets are efficient (Alexander [1], Fama and Blume [2], Fama [3], Jensen and Bennington [4]). Some recent results, however, seem to indicate otherwise. For example, Pruitt and White [5] developed the CRISMA trading system which showed positive returns over a 10-year period, based on transaction costs of 2%. Brock et al. [6], followed by Bessembinder and Chan [7], also demonstrated that simple trading rules could be profitable (but, without transaction costs). Other successful applications of technical analysis in the currency exchange market may be found in Sweeney [8], Levich and Thomas [9] and Osler and Chang [10].

Although interesting, these developments are based on a priori rules determined through technical analysis. The emergence of new technologies, in particular evolutionary algorithms, now allows a system to automatically generate and adapt trading rules to particular applications. Genetic algorithms (Holland [11]), for example, have already been applied to a number of financial applications (see, for example, Bauer [12]). For learning trading rules, however, the genetic programming (GP) approach of Koza [13] looks more promising, as it provides a flexible framework for adjusting the trading rules (which may be seen as 'programs') to the current environment. Although, the first attempts by Chen and Yeh [14] and Allen and Karjalainen [15] on the stock exchange markets did not show any excess returns with regard to the buy-and-hold approach, other recent applications of GP are more encouraging (Neely et al. [16], Neely and Weller [17], Marney et al. [18]), at least when the notion of risk is not considered (Marney et al. [19]).

Our goal here is to explore once more the application of GP on stock exchange markets, but to consider stocks offered by individual companies, rather than global market indices (e.g., Dow Jones, S&P 500), as it was done in previous studies. This approach looks more promising, given that each rule generated through GP will now be adjusted to an individual stock or activity sector. Also, the sale of stocks which the seller does not own will be allowed to take advantage of falling stock prices (i.e., by later purchasing the stocks at a lower price, a profit is made). This has never been done in the past because the implementation of simultaneous short sales in the case of a composite stock index would have been very difficult to realize. Another goal of this study is to identify markets where the trading rules generated with GP are particularly indicated.

The remainder of the paper is the following. In the next section, we first introduce the GP paradigm. Then, its adaptation to our application domain is presented in Section 3. Computational results obtained with the stocks of 14 Canadian companies operating in different activity sectors are reported in Section 4. The conclusion follows.

2. Genetic programming

Genetic programming (Koza [13], Koza [20], Koza et al. [21]) is a recent development in the field of evolutionary algorithms which extends classical genetic algorithms by allowing the processing of non-linear structures. A genetic algorithm (Holland [11], Goldberg [22], Davis [23], Chambers [24], Michalewicz [25]) is a randomized search procedure working on a population of individuals or solutions encoded as linear bit strings. This population evolves over time through the application of operators which mimic those found in nature, namely, selection of the fittest, crossover and

$$\begin{array}{ccccccl}
1 & 0 & | & 1 & 0 & 0 & \text{(parent 1)} \\
\underline{\mathbf{0}} & \underline{\mathbf{0}} & | & \underline{\mathbf{1}} & \underline{\mathbf{1}} & \underline{\mathbf{1}} & \text{(parent 2)} \\
\hline
1 & 0 & & \underline{\mathbf{1}} & \underline{\mathbf{1}} & \underline{\mathbf{1}} & \text{(offspring 1)} \\
\underline{\mathbf{0}} & \underline{\mathbf{0}} & & 1 & 0 & 0 & \text{(offspring 2)}
\end{array}$$

Fig. 1. One-point crossover on two bit strings.

mutation. First, ”parent” chromosomes are selected in the population for reproduction. The selection process is probabilistic and biased towards the best individuals (to propagate good solution features to the next generation). Then, the crossover operator combines the characteristics of a pair of parent chromosomes to create two new offspring, in the hope that these offspring will be better fit than their parents. An example of one-point crossover is shown in Fig. 1, where the cross point is randomly chosen between the second and third bit. In this case, the end parts of the two parent chromosomes are exchanged to create the offspring.

Finally, the mutation operator is applied to each offspring. This operator processes the offspring position by position and flips the bit value from 0 to 1, or from 1 to 0, with a small probability at each position. Mutation is viewed as a “background” operator which slightly perturbs a small proportion of solutions. Its primary goal is to maintain or restore diversity in the population and to guarantee that every state of the search space is accessible from the current state. Through the selection/crossover/mutation process, it is expected that an initial population of randomly generated individuals will improve as parents are replaced by better offspring.

Genetic programming extends the above paradigm by allowing the evolution of programs encoded as tree structures. These programs are constructed from a predefined set of functions and terminals (which may be variables, like the state variables of a particular system, or constants, like integer 3 or boolean *False*). The evolution of programs within the genetic programming framework can be summarized as follows:

Step 1: *Initialization.* Create an initial random population of P programs and evaluate the fitness of each program by applying it on a set of *fitness cases* (examples). Set the current population to this initial population.
Step 2: *Selection*. Select P programs in the current population (with replacement). The selection process is probabilistically biased in favor of the best programs.
Step 3: *Modification.* Apply reproduction or crossover to the selected programs.
Step 4: *Evaluation*. Evaluate the fitness of each offspring in the new population.
Step 5: Set the current population to the new population of offspring.
Step 6: Repeat steps 2–6 for a predefined number of generations or until the system does not improve anymore.

The final result is the best program generated during the search. The main issues related with this algorithm are briefly discussed in the next subsections.

2.1. Encoding

The individual structures which evolve over time are computer programs represented as tree structures. In our context, these structures will encode trading rules. The size, shape and contents of

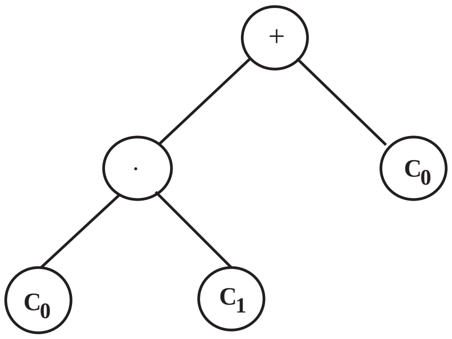

Fig. 2. A tree structure for a program.

the trees, which are recursively constructed from a predefined set of functions G and terminals T, dynamically change during the evolution process. Each particular function g_i in the function set takes a specified number of arguments and should be able to accept as argument the value returned by the other functions in the set. Fig. 2 illustrates a program with $G = \{+, ^*\}$ and $T = \{C_0,\ C_1\}$. Such a program computes $C_0 \times C_1 + C_0$, where C_0 and C_1 are numerical constants.

2.2. Fitness evaluation

The fitness of a program is evaluated by applying the program to a set of fitness cases. In our context, the excess return provided by a trading rule over the buy-and-hold approach is evaluated on historical data.

2.3. Initial population

Typically, the initial population is made of randomly generated programs or trees. Usually, an equal mix of trees is produced with the *Full* and *Grow* methods, which are defined as follows.

Full: This method creates trees such that the length of every path between a terminal and the root is *equal* to a predefined depth.
Grow: This method creates trees of variable shapes. That is, the length of a path between a terminal and the root is *at most* a predefined maximum depth.

2.4. Selection

Different methods have been reported in the literature for selecting the programs that either reproduce or are transformed by crossover. The basic idea is to bias the selection process towards the best programs through a so-called fitness-proportionate selection scheme, where the selection probability prb_i of program i is proportional to its fitness f_i. Namely:

$$prb_i = \frac{f_i}{\sum_{j=1}^{P} f_j}.$$

In this equation, the denominator sums the fitness values over all programs in the current population.

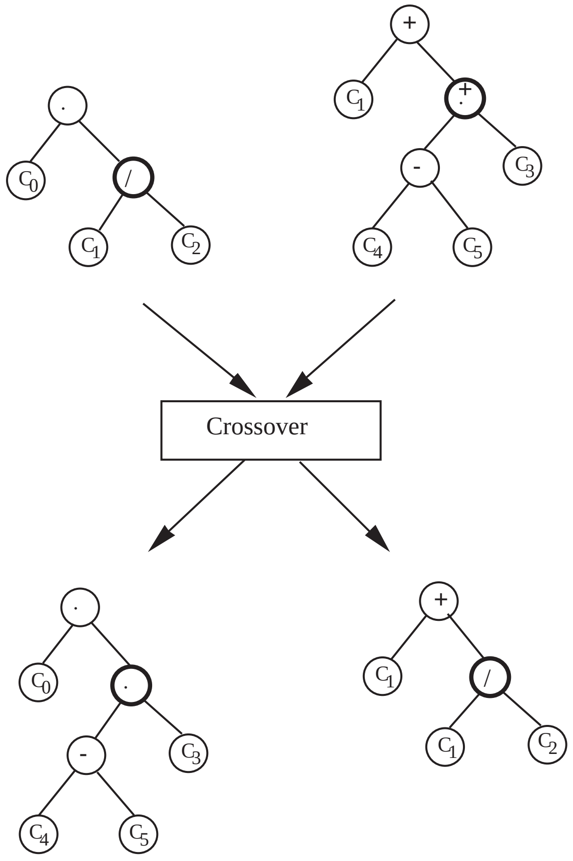

Fig. 3. Crossover in genetic programming.

2.5. *Reproduction and crossover*

A new generation of programs is obtained through the action of reproduction and crossover. Other secondary operators can also be applied to the programs, like mutation.

2.5.1. *Reproduction*

Reproduction simply makes a copy of the selected program in the new population, without any modification.

2.5.2. *Crossover*

Crossover applies to two selected programs. A crossover point, namely a function or terminal in the tree, is randomly selected within each parent. This crossover point defines a fragment consisting of the crossover point plus the entire subtree lying below the crossover point. Two new offspring programs are then produced by exchanging the two fragments, as depicted in Fig. 3. In this example, the mathematical operators / and $\times$ are selected in the first and second tree, respectively. Then, the subtrees are exchanged to create two new programs. Obviously, when both crossover points are terminals (i.e., leaves of their respective tree), the effect of crossover is simply to exchange the two terminals. Typically, the probability distribution for the selection of a crossover point favors the exchange of subtrees rather than terminals.

2.6. Mutation

Mutation is a unary operator aimed at restoring diversity in a population by applying random modifications to individual structures. Given that the trees manipulated by GP are more complex than the linear structures of GAs, the loss of diversity is not a primary concern here. In particular, applying crossover to two identical programs is unlikely to lead to offspring which are identical to their parents, as long as the cross points are different on both parents. The mutation operator is defined as follows in a GP. First, a random point is chosen in the tree. This point, as well as the subtree below it, are then removed and replaced by a new, randomly generated, subtree.

3. Generating trading rules with GP

Based on the description provided in the previous section, the adaptation of GP for our application is now presented. The GP will generate trading rules encoded as programs or tree structures. These rules are boolean functions which return either a buy (True) or a sell signal (False).

3.1. Encoding

The set of functions G is made of classical arithmetic, boolean and relational operators, plus the boolean function if–then–else and a number of real functions. The set of terminals T correspond to numerical and boolean constants, plus constants and variables related to stock prices and volumes. They are described in the following.

Functions:

Arithmetic operators:	$+, -, /, \times$;
Boolean operators:	and, or, not;
Relational operators:	$<, >$;
Boolean functions:	if–then–else.
Real functions	

In the functions presented below, variable s stands either for constant *Price* or *Volume*.

$norm(r_1, r_2)$:	absolute value of the difference between two real numbers;
$avg(s, n)$:	average of price or volume over the past n days;
$max(s, n)$:	maximum value of price or volume over the past n days;
$min(s, n)$:	minimum value of price or volume over the past n days;
$lag(s, n)$:	price or volume is lagged by n days;
$volatility(n)$:	variance in daily returns over the past n days;
$RSI(n)$:	relative strength index;
$ROC(n)$:	rate of change.

The last two functions, which are often used in technical analysis, are defined as

$$ROC(n) = \left(\frac{\text{closing price of current day}}{\text{closing price } n \text{ days ago}} - 1\right) \times 100,$$

and

$$RSI(n) = 100 - \left(\frac{100}{1 + RS(n)}\right), \quad \text{where } RS(n) = \frac{\sum_{i \in D^+(n)} r_i}{-\sum_{i \in D^-(n)} r_i}$$

and where $D^+(n)$ is the set of days (over the past n days) with rising stock prices, $D^-(n)$ is the set of days (over the past n days) with falling stock prices and r_i is the return of day i, which is positive when the stock price is rising and negative otherwise.

Terminals:

Constants

Real:	chosen in the interval [0,250], where 250 is the approximate number of working days in a year;
Boolean:	True, False;
Others:	Price, Volume.

Real variables

p:	stock price of the current day;
v:	transaction volume of the current day.

Clearly, the functions and terminals chosen to construct the trading rules violate the closure assumption of GP. That is, a given function cannot necessarily accept as argument the value returned by another function in set G, given that both boolean and real functions are found in that set. This particularity imposes restrictions on the structure of the trading rules. Consequently, real functions are always found in the lower part of the tree, boolean functions and operators in the upper part (including the root) and relational operators make the transition between the two. A relational operator may also be found at the root of the tree.

A typical tree representing a trading rule is illustrated in Fig. 4. In this example, the rule sends a buy (True) signal if the average stock price over the past 50 days is greater than the current price or the current transaction volume is less than 20. A sell (False) signal is sent otherwise. This rule is applied as follows. If, at the end of the current day, the position on the market is "open" and a buy signal is sent by the rule, the stock is bought at the opening of the market the next day and the position switches to "close". Conversely, if the current position is "close" and a sell signal is sent, the stock is sold and the position switches to "open". Otherwise, the current position does not change.

3.2. Fitness evaluation

The raw fitness of a trading rule is its excess return over the buy-and-hold approach, as evaluated on historical data. For an investment period extending from day 0 to day t, the return of the buy-and-hold approach is simply the price of day t minus the price of day 0. In the case of the

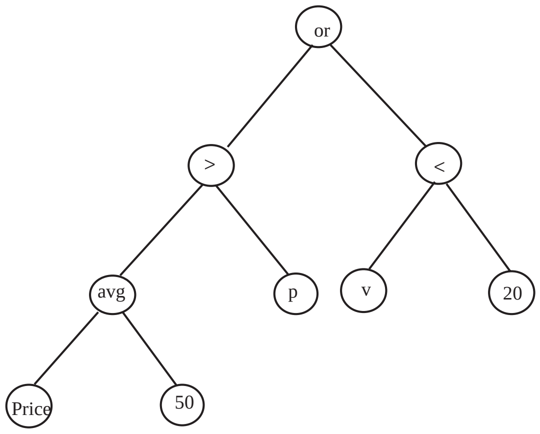

Fig. 4. Example of a trading rule.

trading rules generated by GP, the gains or losses (if any) are evaluated on a daily basis and summed up. The latter minus the former correspond to the excess return of the trading rule.

3.3. Initial population

A mix of trees produced by the Full and Grow methods is used. Due to the particular structure of our trees, the recursive construction process (from the root to the leaves) must observe the following guidelines.

(1) The root of the tree is selected among the boolean functions and operators.
(2) Once the root has been selected, its descendants may be selected among boolean constants, boolean functions, boolean or relational operators.
(3) When a relational operator is selected, its descendants must be selected among real functions or terminals.

In the case of the Full method, the trees must be fully developed to a predefined depth, hence terminals can only be selected when that depth is reached. In the case of the Grow method, the trees need not be fully developed and terminals can be selected at any level (although terminals must be chosen when the predefined depth is reached).

3.4. Selection

In this work, a rank-based method is used (Baker [26], Whitley [27]). First, all programs in the current population are sorted from best (rank 1) to worst (rank P) based on their "raw" fitness value. A new fitness value f_i is then associated with the program of rank i as follows:

$$f_i = Max - \left[(Max - Min)\frac{i-1}{P-1} \right].$$

Hence, the best program gets fitness *Max*, the worst program gets fitness *Min* and the fitness values of the remaining programs are equally spaced between *Min* and *Max*. In our experiments, *Max* and *Min* were set to 1.8 and 0.2, respectively. For example, when the population is composed of $P = 5$ programs, these programs are evaluated from best to worst as follows: $f_1 = 1.8$, $f_2 = 1.4$, $f_3 = 1.0$, $f_4 = 0.6$ and $f_5 = 0.2$. Stochastic universal sampling (Baker [28]), a variant of fitness-proportionate selection where all parents are selected at once in a single trial, is then invoked on the transformed fitness values.

The ranking method basically modifies the raw fitness values before stochastic universal sampling is called upon. Otherwise, a "super-program" with high fitness would typically be overselected (leading to premature convergence on a possibly suboptimal solution). Conversely, a random selection process would occur when all programs have similar fitness values. Through ranking, the gap between very close fitness values is enlarged, thus alleviating the random selection behavior, while the impact of a super-program is greatly reduced by bounding its fitness to the *Max* value.

3.5. Reproduction and crossover

The classical reproduction and crossover operators presented in Section 2 were used in this study. Due to the particular layered structure of our trees (see Section 3.1), a special care was given to the crossover operator. Once a cross point is chosen on the first parent, the subset of compatible cross points is identified on the other parent and only this subset is considered for crossover. If there is no compatible cross point, two other parents are selected.

4. Computational results

In this section, we present results obtained with our GP on stocks of Canadian companies. First, the data at our disposal are described in Section 4.1. Then, Section 4.2 presents the parameter settings for the GP. Section 4.3 reports numerical results obtained with each stock.

4.1. Data

For the study, Canadian companies were chosen from the TSE 300 index which spans 14 different activity sectors. One company was selected in each sector, but only among those which were active on the market before 1990. The companies used for the study are listed in Table 1. The historical data included the stock price and the transaction volume for each working day between June 30, 1992 and June 30, 2000, for a total of 2003 days.

We considered both a long training period and a short one. In each case, an initial period of 250 days was put aside to start the process (given that some of the primitive functions used by the trading rules, such as *avg*, may require data from the last 250 days in the worst case, see Section 3.1). The trading rules obtained on the training period were then evaluated on previously unseen data associated with a testing period. The exact time periods used for the experiments are reported below.

Table 1
Canadian companies in the study

Activity sector	Company	Symbol
Mining	Alcan Aluminium Ltd	*AL*
Precious metals	Barrick Gold Corporation	*ABX*
Oil and gas	Cdn. Occidental Petroleum Ltd	*CXY*
Forestry	Abitibi-Consolidated Inc	*A*
Consumer products	Molson Inc	*MOL*
Industrial products	Bombardier Inc	*BBD*
Real estate	Cambridge Shopping Centers	*CBG*
Transportation and env.	Trimac Corporation	*TMA*
Pipelines	TransCanada Pipelines Ltd	*TRP*
Utilities	Canadian Utilities Ltd	*CU*
Communications	The Seagram Company Ltd	*VO*
Merchandising	Sears Canada Inc	*SCC*
Financial services	Royal Bank of Canada	*RY*
Diversified (conglomerate)	Canadian Pacific Ltd	*CP*

Short training period:

Initial (for training):	June 25, 1997 to June 22, 1998 (250 days),
Training:	June 22, 1998 to June 25, 1999 (256 days),
Initial (for testing):	July 07, 1998 to June 25, 1999 (250 days),
Testing:	June 28, 1999 to June 30, 2000 (256 days).

Long training period:

Initial (for training):	June 30, 1992 to July 02, 1993 (250 days),
Training:	July 02, 1993 to June 25, 1999 (1498 days),
Initial (for testing):	July 07, 1998 to June 25, 1999 (250 days),
Testing:	June 28, 1999 to June 30, 2000 (256 days).

4.2. Parameter settings

Preliminary experiments were performed to determine the best parameter settings for the GP. Based on these experiments, the parameter values shown in Table 2 were finally selected.

Tests with populations of size 100, 250 and 500 were performed. In all cases, the best results were achieved with the largest populations. However, the computation times also increased from an average of 5 min for $P = 100$ to more than 1 h for $P = 500$. Thus, populations with more than 500 individuals were not considered. The number of generations was fixed to 50, as no significant improvement was observed beyond that point. In fact, most of the improvement was achieved before generation 40 (see Table 5 in Section 4.3). Apart from the maximum number of generations, the GP was also stopped after 15 consecutive generations without any improvement to the best solution. Different values were also tried for the other parameters shown in Table 2, but the best solutions were obtained with the values suggested in Koza [13].

Table 2
Parameter settings

Population size	500
Number of generations	50
Initialization method	Equal mix of Full and Grow
Selection method	Ranking + SUS
Reproduction rate	0.35
Crossover rate	0.60
Mutation rate	0.05
Probability of choosing an internal point (crossover)	0.90
Probability of choosing a terminal (crossover)	0.10
Maximum depth of programs in initial population	6
Maximum depth of programs in following generations	17

Table 3
Numerical results for the short training period

Symbol	Training	Testing
A	144.16%	15.29%
ABX	202.16%	38.12%
AL	135.95%	15.79%
BBD	123.04%	−34.92%
CBG	174.01%	−75.09%
CP	120.70%	0.86%
CU	56.69%	7.88%
CXY	183.40%	−102.16%
MOL	99.23%	17.38%
RY	125.68%	3.25%
SCC	110.80%	−21.84%
TMA	211.93%	42.27%
TRP	85.78%	36.04%
VO	146.78%	−10.83%
Average	131.17%	−4.85%
Number of positive returns	14	9

4.3. Numerical results

Computational tests were run on a 300 MHz Pentium II PC. The results obtained with GP on the 14 stocks for the short and long training periods are shown in Tables 3 and 4. For each stock, the numbers are the average of 10 different runs.

As expected, the results obtained on the training period are much better than those obtained on the testing period. In fact, the overall average return on the testing period is negative for both the short and long training periods, which indicates that the trading rules do not provide any improvement over the buy-and-hold approach. The positive returns on the training data for the short period (131.17%)

Table 4
Numerical results for the long training period

Symbol	Training	Testing
A	31.00%	18.55%
ABX	34.27%	7.69%
AL	21.99%	19.83%
BBD	18.30%	−79.46%
CBG	33.15%	−35.58%
CP	19.67%	12.11%
CU	9.77%	6.45%
CXY	30.39%	−44.85%
MOL	19.36%	13.74%
RY	19.59%	8.13%
SCC	16.53%	−7.33%
TMA	30.57%	54.80%
TRP	16.57%	21.09%
VO	19.88%	−45.41%
Average	22.93%	−3.59%
Number of positive returns	14	9

are much higher than those obtained for the long period (22.93%), although the results are very similar when the trading rules are evaluated on the testing period.

Due to the smaller amount of data in the case of the short training period, the GP may have developed trading rules too tightly fitted to those data. An intermediary control period was thus introduced to stop the training when a new best rule on the training period degraded the excess returns by more than 25% on the control period or when a degradation was observed for three consecutive iterations. Although this additional control mechanism allowed the GP to stop earlier in many cases, no significant improvement was observed on the testing period.

The optimization capabilities of the GP are illustrated in Table 5 on each stock. In this table, we show the best and average trading rules in the initial population, as well as the best and average trading rules in the final population. We also indicate the generation number where the best trading rule was found. These results correspond to those observed with the long training period. From the start to the end of the GP, the best rule has improved on average from 11.22% to 22.93%, while the average rule in the population has improved from −8.64% to 16.19%.

The results in Tables 3 and 4, although disappointing at first glance, hide some interesting findings. First, nine stocks out of 14 show positive returns on the testing period. Also, a few stocks with negative returns, like CBG and CXY, have a significant impact on the overall results. In Table 3, for example, a positive average return of 9.1% is observed on the 12 stocks other than CBG and CBX. In these two cases, important discrepancies in the transaction volumes and prices over the training and testing periods are observed (i.e., relatively stable market during the training period versus a rising market during the testing period), which may explain the poor results.

By collecting the results obtained over all runs on the testing period, the graph of Fig. 5 was obtained. It shows the excess returns generated by the various trading rules over the buy-and-hold,

Table 5
Evolution of the trading rules over the long training period

Symbol	Generation	Initial population		Final population	
		Best (%)	Average (%)	Best (%)	Average (%)
A	37	15.08	−3.53	27.48	21.89
ABX	42	20.76	3.28	31.76	27.13
AL	36	8.53	−10.32	20.03	14.93
BBD	36	3.50	−37.89	14.80	8.53
CBG	37	24.73	10.86	31.90	28.38
CP	34	7.34	−8.80	17.36	12.13
CU	37	2.57	−10.21	8.72	5.16
CXY	37	18.02	−7.09	29.23	24.28
MOL	38	9.73	−2.44	17.51	13.61
RY	33	8.70	−14.91	17.85	12.44
SCC	40	4.90	−24.99	14.50	9.66
TMA	37	18.69	2.26	29.17	24.09
TRP	39	7.41	−3.73	14.62	11.45
VO	40	7.09	−13.42	17.78	12.94
Average	37	11.22	−8.64	22.93	16.19

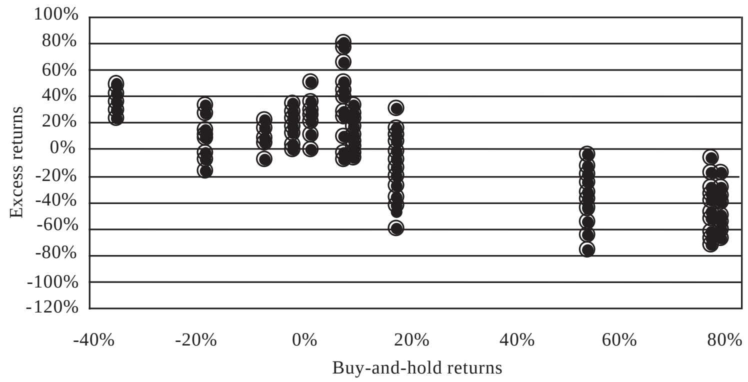

Fig. 5. Excess returns versus buy-and-hold returns on the testing period.

as compared with the returns of the buy-and-hold alone. The most interesting feature in this graph is the performance of the trading rules in situations where the buy-and-hold approach generates small positive returns close to 0% or negative returns (i.e., when the market falls or is relatively stable). In these cases, the excess returns generated by the trading rules are generally positive. This observation is interesting as it indicates an appropriate context or "timing" for triggering the application of technical trading rules.

It is worth noting that the raw trading rules produced through GP are rather intricate, but can often be greatly simplified (e.g., boolean expressions which can be reduced to True or False). The rule illustrated in Fig. 4, for example, was one of the rules produced through GP during our simulations. However, in its raw form, the decision tree contained more than 30 nodes distributed over eight hierarchical levels.

5. Conclusion

In this paper, an application of the GP paradigm for automatically generating trading rules on the stock markets was described. The results show that the trading rules generated by GP are generally beneficial when the market falls or when it is stable. On the other hand, these rules do not match the buy-and-hold approach when the market is rising.

Acknowledgements

Financial support was provided by the Canadian Natural Sciences and Engineering Research Council (NSERC) and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR). This support is gratefully acknowledged.

References

[1] Alexander SS, Price movements in speculative markets: trends or random walks. In: P.H. Cootner (Ed.) The random character of stock market prices, vol. 2. Cambridge, MA: MIT Press, 1964. p. 338–72.
[2] Fama EF, Blume ME. Filter rules and stock market trading. Journal of Business 1966;39:226–41.
[3] Fama EF. Efficient capital markets: a review of theory and empirical work. Journal of Finance 1970;25:383–417.
[4] Jensen M, Bennington G. Random walks and technical theories: some additional evidences. Journal of Finance 1970;25:469–82.
[5] Pruitt SW, White RE. The CRISMA trading system: who says technical analysis can't beat the market? Journal of Portfolio Management 1988;14:55–8.
[6] Brock W, Lakonishok J, LeBaron B. Simple technical trading rules and the stochastic properties of stock returns. Journal of Finance 1992;47:1731–64.
[7] Bessembinder H, Chan K. The profitability of technical trading rules in the Asian stock markets. Pacific Basin Finance Journal 1995;7:257–84.
[8] Sweeney RJ. Beating the foreign exchange market. Journal of Finance 1986;14:163–82.
[9] Levich R, Thomas L. The significance of technical trading rule profits in the foreign exchange market: a bootstrap approach. Journal of International Money and Finance 1993;12:451–74.
[10] Osler CL, Chang PHK, Head and shoulders: not just a flaky pattern. Staff Paper No. 4. Federal Reserve Bank of New York, 1995.
[11] Holland JH. Adaptation in natural and artificial systems. Ann Arbor, MI: The University of Michigan Press, 1975.
[12] Bauer RJ. Genetic algorithms and investment. New York: Wiley, 1994.
[13] Koza JR. Genetic programming: on the programming of computers by means of natural selection. Cambridge, MA: MIT Press, 1992.
[14] Chen S-H, Yeh C-H. Toward a computable approach to the efficient market hypothesis: an application of genetic programming. Journal of Economic Dynamics & Control 1996;21:1043–63.
[15] Allen F, Karjalainen R. Using genetic algorithms to find technical trading rules. Journal of Financial Economics 1999;51:245–71.

[16] Neely C, Weller P, Dittmar R. Is technical analysis in the foreign exchange market profitable? a genetic programming approach. Journal of Financial and Quantitative Analysis 1997;32:405–26.
[17] Neely C, Weller P. Technical trading rules in the European monetary system. Journal of International Money and Finance 1999;18:429–58.
[18] Marney JP, Tarbert H, Fyfe C. Technical trading versus market efficiency—a genetic programming approach. Computing in Economics and Finance, Society for Computational Economics, Barcelona, Spain, July 2000 (paper #169).
[19] Marney JP, Fyfe C, Tarbert H, Miller D. Risk adjusted returns to technical trading rules: a genetic programming approach. Computing in Economics and Finance, Society for Computational Economics, Yale University, USA, June 2001 (paper #147).
[20] Koza JR. Genetic programming II: automatic discovery of reusable programs. Cambridge, MA: MIT Press, 1994.
[21] Koza JR, Bennett III FH, Andre D, Keane MA. Genetic programming III: Darwinian invention and problem solving. San Francisco, CA: Morgan Kaufmann, 1999.
[22] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley, 1989.
[23] Davis L. Handbook of genetic algorithms. New-York, NY: Van Nostrand Reinhold, 1991.
[24] Chambers L, editor. Practical handbook of genetic algorithms: applications. Boca Raton, FL: CRC Press, 1995.
[25] Michalewicz Z. Genetic algorithms + data structures = Evolution Programs., 3rd ed. Berlin: Springer, 1996.
[26] Baker JE. Adaptive selection methods for genetic algorithms. In: Proceedings of the First International Conference on Genetic Algorithms and their Applications. Hillsdale, NJ: Lawrence Erlbaum, 1985. p. 101–111.
[27] Whitley D. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: Proceedings of the Third International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann, 1981. p. 116–121.
[28] Baker JE, Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, NJ: Lawrence Erlbaum, 1987. p. 14–21.