Forecasting Financial Volatility Using Nested Monte Carlo Expression Discovery

Tristan Cazenave and Sana Ben Hamida
LAMSADE
Université Paris-Dauphine
Paris, France
Email: cazenave@lamsade.dauphine.fr sbenhami@u-paris10.fr

***Abstract*—We are interested in discovering expressions for financial prediction using Nested Monte Carlo Search and Genetic Programming. Both methods are applied to learn from financial time series to generate non linear functions for market volatility prediction. The input data, that is a series of daily prices of European S&P500 index, is filtered and sampled in order to improve the training process. Using some assessment metrics, the best generated models given by both approaches for each training sub sample, are evaluated and compared. Results show that Nested Monte Carlo is able to generate better forecasting models than Genetic Programming for the majority of learning samples.**

I. Introduction

Derivative markets provide market participants and policy-makers working in the financial markets with a rich source of information for gauging market sentiment. Option prices are especially useful for extracting such information. Due to their forward-looking nature, they efficiently encapsulate market perceptions. The price of an option therefore depends on the market opinion about the future volatility of the underlying asset upon which the option is written. The volatility that makes the theoretical option price equal to the observed option price is called the implied volatility. It may contain useful information about the market participant expectations regarding the future volatility of the underlying asset. It has been conjectured that the implied volatility is a good predictor of the future volatility of the underlying asset [1]. This effectively makes volatility forecasting a big challenge for many financial institutions around the word.

Traditional parametric methods have limited success in estimating and forecasting the implied volatility as they are dependent on restrictive assumptions and difficult to estimate. Several machine learning techniques have been recently used to overcome these difficulties. Genetic programming (GP) has been often applied to forecast financial time series and recently, it was successfully applied for the prediction of the implied volatility [2], [3].

This paper investigates the application of a new meta-heuristic for the implied volatility forecasting, that is the Nested Monte Carlo Search, and compares it to Genetic Programming.

Monte Carlo Tree Search (MCTS) is a general search algorithm that made a breakthrough in computer Go [4]. The standard MCTS algorithm for multi-player games is Upper Confidence bounds applied to Trees (UCT) [5]. A good MCTS algorithm for single player games is Nested Monte Carlo Search (NMCS) [6].

The principle of Monte Carlo Search algorithms is to play a lot of random games in order to evaluate a state. A random game is also called a playout. When using UCT the average of the playouts that start with a move is used to select the next move of the simulation. The key principle of UCT is to balance exploration and exploitation. Exploitation favors the moves with the greatest means while exploration favors the moves that have not been tried much.

The principle of NMCS is different from the principle of UCT. It still uses a lot of playouts in order to find good sequences of moves but instead of using the average of the playouts, it memorizes the best sequence of moves found so far and follows it. It uses nested levels of search and tries every possible moves once. Memorizing the best sequence of moves enables it to intensify the search, while trying all possible moves once enables it to diversify the search.

NMCS has been applied to puzzles such as SameGame and Morpion Solitaire and to optimization problems. It has found world records for difficult problems such as the Snake-In-The-Box problem [7] or the Weak Schur problem [8]. In order to find these records the random playout policy has been replaced with a more informed policy that still includes some randomness.

NMCS has also been applied to transportation problems such as the Bus Regulation problem [9] or the Traveling Salesman with Time Windows problem [10]. It has also been used for software engineering so as to generate structured test data with specific properties [11].

A related algorithm is Nested Rollout Policy Adaptation (NRPA) that learns a playout policy online using nested levels of learning [12]. NRPA has beaten human world records that were standings for more than thirty years at Morpion Solitaire. It has also been successfully applied to optimization problems such as the Traveling Salesman with Time Windows problem [13] and the Multiple Sequence Alignment problem [14].

The remainder of the paper is divided into six sections. Section 2 gives a brief introduction to the symbolic regression and its application to the prediction of the implied volatility in finance. Section 3 describes the design of the Nested Monte Carlo search. Section 4 reminds the basics of the Genetic Regression. Section 5 and 6 report the experimental settings

and the results.

II. Symbolic Regression for volatility forecasting

There are two approaches to generate volatility forecasts. One is to extract information about the variance of future returns from their history, the second is to elicit market expectations about the future volatility from observed option prices. The Black&Scholes(**BS**) [15] is the most commonly used model in the second category. It uses a backward induction technique to derive the implied volatility from a set of observable data for an option contract. An option contract is a derivative security that gives the holder the right to buy (call) or to sell (put) the underlying asset by a certain date for a certain price. The price in the contract is known as the exercise price or strike price. The date in the contract is known as the expiration date or maturity.

Whilst it is the most commonly used in financial markets, the BS pricing model has some well-known deficiencies. Indeed, it assumes that volatility is constant. However, it can be easily shown empirically that the implied volatility is not constant and changes with different option strike prices and expiry dates. Black [15] suggested that the non-stationary behavior of volatility could lead the BS model to over-price or under-price options...

To overcome the BS deficiencies, several parametric models have been introduced in financial engineering based on the realized volatility. They can be classified into three categories: models based on Past standard Deviations (such as the *Random walk* model, the *Simple Regression* method, ...), ARCH Class Conditional Volatility models (such as the well known GARCH model family [16], [17]) and the stochastic models (such as Monte Carlo models).

All these models are parametric models that need some assumptions. However, as mentioned in [18], the volatility forecasting approaches should be free of strong assumptions and more flexible than parametric methods.

Recently, to provide a new issue for implied volatility forecasting, several researchers have studied the application of supervised machine learning techniques. The aim is to offer explicit formulas which could compute directly the implied volatility expressed as a function of option prices and other observable variables.

The main objective of the supervised machine learning is to draw a mapping between input and output values in a training data set. The hope is that the resulting pattern (model) from the mapping process be able to correctly predict unknown examples of the same type.

In the case of learning regression models, the task is to discover a target function $f(X)$ that map a vector of real-valued inputs X to real valued target variable Y. For financial volatility forecasting, the input set X is a financial time series data ($X = X_{t1}, X_{t2}, ..., X_{tN}$), and Y is the BS implied volatility computed from the observed data. Each input X_{ti} is a vector of variables for one observed option at the date ti. The variables could be the strike price, the maturity, the call or put price, ...

III. Monte Carlo Expression Discovery

Monte Carlo sampling can be used to generate expressions. Expressions are seen as stacks of atoms with a maximum size. At each step of a sample (also called a playout), a random atom is chosen and added to the stack. In order to stop the playout a maximum number of atoms are allowed in the stack. When the number of atoms of the stack plus the number of open branches is equal to the maximum, only leaves are added to the stack so that when the stack is complete it contains less than the maximum number of atoms.

Nested levels of Monte Carlo search can be used in order to improve on random sampling [6]. The principle of NMCS is to use different levels of search. At level zero it does a playout. At greater levels it performs informed playouts. At each step of a playout, it generates the possible moves and for each move it does a level - 1 search. It keeps the move that is associated to the best level - 1 search and plays it. It continues to choose moves like this until the end of the informed playout.

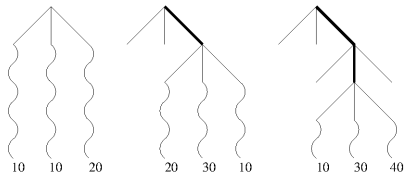

Fig. 1. At each step of the principal playout shown here with a bold line, a NMCS of level n performs a NMCS of level $n-1$ (shown with wavy lines) for each available move and selects the best one. At level 0, a simple pseudo-random playout is used.

The figure 1 explains how a playout of level 1 uses playouts to choose the moves to play.

Algorithm 1 Nested Monte-Carlo search

```
nested (position, level)
best playout ← {}
while not terminal (position) do
  if level = 1 then
    move ← argmax_m (sample (play (position, m)))
  else
    move ← argmax_m (nested (play (position, m), level − 1))
  end if
  if score of playout after move > score of the best playout then
    best playout ← playout after move
  end if
  position ← play (position, move of the best playout)
end while
return score (position)
```

The NMCS algorithm is given in algorithm 1. For a playout of level n it chooses the move that gives the best score when followed by a playout of level $n-1$. A search at level n calls a search at level $n-1$ to direct its search.

NMCS was applied with success to multiple expression discovery problems [19], [20]:

- The prime generating polynomial problem where the goal is to discover a polynomial that generates as

many different primes in a row as possible. Cartesian Genetic Programming has also been applied to the problem of finding polynomials that generate prime numbers [21].

- The finite algebra problem where the goal is to find a discriminator term for an algebra. The discriminator term is a formula that gives special results for the possible values of the variables of the algebra. Genetic Programming has also been applied to finding a discriminator term for finite algebras [22].

- The parity problem which is a classical problem for GP [23]. The goal is to find a formula that tells if the number of bits of the input is odd or even. The problem can be easy or difficult depending on the atoms used to discover expressions and the size of the input.

- The N-prisoners puzzle. N prisoners wear a hat and each prisoner can see the other's hats but not his hat. A hat is associated to either a zero or a one. All prisoners are asked to find the color of their hat. They can pass and give no answer. If all prisoners that have not passed have found the right number for their hats, all the prisoners are free, else they stay in prison. A basic strategy is to have one prisoner randomly choosing its answer. It wins 50 percent of the time. Much more elaborate strategies exist. Genetic Programming has also been applied to the N-prisoners puzzle [24].

- The MAX problem is to find an expression that evaluates as great as possible with a limited expression size. The authorized atoms are 0.5, + and *. In [25] the limit was on the depth of the corresponding tree. For NMCS the limit was on the number of atoms of the generated expression, not on the depth of the tree.

- The target number problem is derived from a popular television game where the goal is to find a an expression that evaluates as close as possible to a target number given the four basic operations as atoms as well as a given set of predefined numbers.

A good property of NMCS for the generation of expressions is that it uses a maximum expression size. It avoids a problem of Genetic Programming called bloat which consists in generating bigger and bigger expressions that are very specific to the learning data and that hinder generalization of the learned expression. Another good property of NMCS is that it there is no parameter to tune except the maximum expression size. Even without tuning it has outperformed optimized Genetic Programming systems for the problems previously cited. It is still possible to tune NMCS for a problem by improving the playout policy. For most problems that were addressed with NMCS, big improvements came from using rules during the playouts so as to avoid bad moves. It could be possible to do this for Nested Monte Carlo expression discovery, however that would be a problem specific enhancement. In future work we will consider domain specific playout policies.

For a problem of height h, the NMCS algorithm of level l has a complexity of $O(|branchingFactor|^n h^{l+1})$ [26].

IV. Genetic Regression

Genetic Programming, that is an offshoot of Genetic Algorithms, is essentially applied to develop efficient computer programs. For the implied volatility forecasting, we used a variant of Genetic Programming called Genetic Regression that implement the Symbolic Regression. This is a numerical optimization tool to select a model which best fits a time series. With Genetic Regression, the desired program is a function that relates a set of inputs to one output. In our case, the output is the implied volatility.

Genetic Regression was one of the earliest applications of GP published by John Koza on 1992 [23], and has continued to be widely studied. The GP system takes as input a set of explanatory or lagged dependent variables (defined as terminals in GP), and a set of mathematical operators (defined as non-terminals in GP) and then randomly selects and combines independent variables and operators in search for a model specification that would satisfy some user-defined fitness function. A GP solution (individual) is a symbolic regression program able to generate predicted values of the dependent variable. The individual possessing the lowest predictive error is characterized as fittest and is kept in memory.

Several works have applied the Genetic Regression for financial times series mining. For example, for volatility forecasting, Chen and Yeh [27] have applied a recursive genetic programming (RGP) approach to estimate volatility historical returns of Nikkei 225 and S&P500 index. Zumbach and al. [28] and Neely and Weller [29] have applied GP to forecast foreign exchange rate volatility. Recently, Abdelmalek et al. [2] and Ben Hamida et al. [3] have extended the studies mentioned earlier by forecasting the implied volatility from the S&P500 index call options instead of historical volatility. For this work, we use the same Genetic Regression implementation as in [2], [3].

The steps necessary to implement the Genetic Regression are summarized in algorithm 2.

Algorithm 2 Abstract Genetic Regression Algorithm

```
Randomly create an initial population of models
while termination condition not satisfied do
    Evaluate the performance of each function according to the fitness criterion
    Select the best functions in the population using the selection algorithm
    Generate new individuals by crossover and mutation
end while
Report the best solution found
```

The starting GP population is generated using a ranked half and half method [23]. This method involves generating an equal number of trees using a pre-established maximum depth. Based on the fitness criterion, the selection of the individuals for reproduction is done with the tournament selection algorithm.

The main genetic operator involved in GP are:

- **Crossover:** It create an offspring function by recombining randomly chosen parts from two selected parent functions.
- **Mutation:** Different mutation operators are used. Point mutation operator consists of replacing a single node in a tree with another randomly-generated node of the same arity. Branch mutation operator randomly selects an internal node in the tree, and then it replaces the sub-tree rooted at that node with a new randomly-generated sub-tree. Expansion mutation operator randomly selects a terminal node in the tree, and then replaces it with a new randomly-generated sub-tree. Each mutation is applied with a rate given in the GP parameters (table II).

The method of replacing parents for the next generation is comma replacement strategy, which selects the best offspring to replace the parents. It assumes that offspring size is higher than parents' size. The stopping criterion is the maximum number of generations (table II).

To run a Genetic Regression system, four ingredients need to be specified: the terminal set, the function set, the fitness measure and the GP parameters. Definition and details of these ingredient are given in the section V.

V. Experiments

A. Learning Data

Data used to perform our empirical analysis are daily prices of European S&P500 index call options, from the CBOE for the sample period January 02, 2003 to August 29, 2003. S&P500 index options are among the most actively traded financial derivatives in the world.

Figure 2 illustrates the evolution of the mean values of the S&P500 index implied volatility computed with the Black&Scholes formula [15].

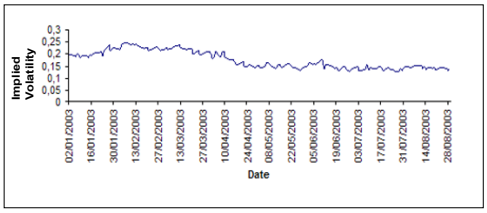

Fig. 2. Curve of the $B\&S$ Implied Volatility mean values of the S&P500 from January 02, 2003 to August 29, 2003

When using the symbolic regression technique, developers and users of learning meta-heuristics believe that a small sample size is usually sufficient, especially with GP. Thus, the full input sample is sorted by time series and divided chronologically into nine successive sub-samples $(S_1, S_2, \cdots, S_9)$ each containing 667 daily observations. These samples will be used simultaneously for training and test steps. Note that the training set is the data sub-sample on which the program operates to determine the relationship between input parameters and output, and the test set is the data sub-sample on which the resulting formula is tested.

If a sub-sample S_i is used as a training set, then the sub-sample of the the immediately following date S_{i+1} (S_1 if $i = 9$) is used as test set.

B. Settings

Both GP and NMCS used the same terminal and function sets to generate and evolve the tree based models. The terminal set includes input variables: the call option price divided by strike price C/K, the index price divided by strike price S/K and time to maturity τ (table I). The function set includes basic mathematical operators, such as arithmetic operators, and Black&Scholes components such as the normal cumulative distribution function ($Ncfd$) [15], which may be useful for implied volatility models (table I).

Function (node) set	
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$\%$	Protected Division
ln	Protected Natural Log
Exp	Exponential function
$Sqrt$	Protected Square Root
cos	Cosinus
sin	Sinus
$Ncfd$	Normal cumulative distribution
Terminal set	
C/K	Call Price/Strike Price
S/K	Index Price/Strike Price
τ	Time to Maturity

TABLE I. Terminal and f function for NMCS and GP trees.

1) NMCS parameter: There is only one NMCS parameter which is the maximum number of nodes in a tree representing an expression. We used a maximum of 40 nodes.

2) GP parameters: The implementation of genetic programming involves different parameters that determine its efficiency. A common approach in tuning GP is to undertake a series of trial to make parameter choices for the final GP runs. We used this approach to determine some parameters such as the crossover and mutation probabilities.

The final design of GP parameters used in this work is summarized in Table II.

Population size	100
Offspring size	200
Maximum number of generations	500
Maximum depth of new individual	6
Maximum depth of the tree	17
Tournament size	4
Crossover probability	0.6
Mutation :	
Branch mutation	0.2
Point mutation	0.1
Expansion mutation	0.1

TABLE II. GP parameters.

C. Evaluation metrics

Each formula given by NMCS or GP is evaluated to test whether it can accurately forecast the output value (*implied volatility*) for all entries in the training set. To assign a fitness measure to a given solution, we compute the Mean Squared Error (**MSE**) between the estimated volatility ($\hat{y_i}$) and the target volatility (y_i):

$$MSE = \frac{1}{N}\sum_{N}^{1}(y_i - \hat{y_i})^2$$

where N is the number of entries in the training data sample.

Beside the (**MSE**), we used another measure to evaluate the training accuracy and the generalization ability of the obtained models. This measure is the percentage of Poor Fitted Observations (**PFO**). It compute the proportion of input entries where the gap between the estimated and the target values is greater than a given level (l).

When using mining techniques in the field of automatic learning, two difficulties are often encountered: the over-fitting and the under-fitting. Over-fitting happens when the data is learned by rote and when the learned expression does not generalize past the learned data. Under fitting is encountered when the regression model does not fit correctly a part of the learning data, and the forecasting error of the corresponding cases is relatively high. The predictive ability of the model is then not satisfactory, mainly for finance prediction. So it is necessary to introduce an other metric able to detect these case more effectively than the mean square error. The PFO is then introduced in this purpose and is computed as follows.

$$PFO = (\sum_{i=1}^{N} z_i)/N; z_i = \begin{cases} 1 & if fe(\hat{y_i}) > l, \\ 0 & otherwise \end{cases} \quad (1)$$

$$fe(\hat{y_i}) = |\frac{\hat{y_i} - y_i}{y_i}|; \quad (2)$$

where N is the size of the input data and l is the maximum forecasting error level allowing for the prediction of one observation X_i.

The PFO value is used to estimate the percentage of variables in the input data that a given formula was unable to forecast without exceeding a fixed maximum error margin that could be accepted in the prediction. It is related to the closeness of the forecasted values against the target ones. The lower the values obtained by this metric, the better is the forecasting technique.

VI. Results and Discussion

For the comparative study, a set of eighteen generated models is constructed. It contains the NMCS and GP best solutions for each training sub-sample $(S_1, \cdots, S_9)$, selected according to the test MSE. Obtained models are noted $MN_1, \cdots MN_9$ for NMCS and $MG_1, \cdots, MG_9$ for GP as described in the following table.

Subsample	S_1	S_2	$\cdots$	S_9
NMCS model	MN_1	MN_2	$\cdots$	MN_9
GP model	MG_1	MG_2	$\cdots$	MG_9

To compare both forecasting and generalization ability of each model, the MSE according to the whole input data ("MSE Total") is computed for each best solution given by NMCS and GP using each sub-sample in the learning set. Results are shown in figure 3. It can be observed from this figure that the forecasting performance of all models is quite similar, except for the three first models. Indeed, training on the three first sub-samples S_1, S_2 and S_3 don't give solutions with high accuracy, and this is for both methods.

Otherwise, a comparison of the "MSE Total" of the two methods shows that NMCS is little better than GP, except for patterns obtained with the training set $S8$.

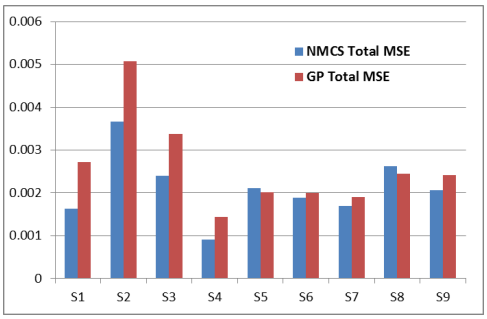

Fig. 3. Performance of the generated NMC and GP volatility models according to "MSE total"

Most of the models given by the two methods have a similar (and good) performance according to the MSE metric. It is therefore interesting to see if they present the same performance according to the PFO metric. Figure 4 and 5 illustrates these results for two maximum error levels:$l = 0.5$ and $l = 0.3$.

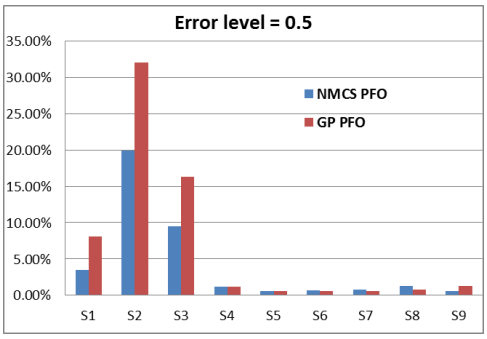

Fig. 4. Performance of the generated NMC and GP volatility models according to the PFO percentage for $l = 0.5$.

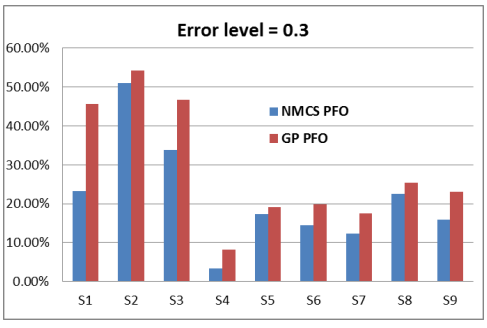

Fig. 5. Performance of the generated NMC and GP volatility models according to the PFO percentage for $l = 0.3$.

The figures 3, 4 and 5 show clearly that the NMCS outperform the GP for the volatility forecasting problem. Indeed,

all MSE Total and PFO percentage values of NMCS models are lower than those of GP models, except for the models corresponding to the sub-sample S_8.

It appears also, through the figures 4 and 5, that the MSE metric is not sufficient to compare in detail the generated models. Some solutions might have very low MSE, but they present a poor performance when a high precision for the prediction step is required, which is the case generally in finance. For example, the model MN1 presents high performance according to the MSE Total and it seems better than the model MN7. However, its PFO percentage for both levels $l = 0.5$ and $l = 0.3$ is much higher than that of MN7. In this case, for more accurate prediction, MN7 is preferred to MN1.

For a complete comparative study between NMCS and GP, the PFO percentage for all generated models are computed according to different maximum error levels (l) varying from 0.3 to 1. These values are illustrated in Figure 6. It is clear that the percentages of PFO according to the enlarged data sample are lower for the NMCS models than GP models for all values of l in the e interval test. We can already make a first conclusion that the learning capacity of NMCS from financial time series is better than that of GP.

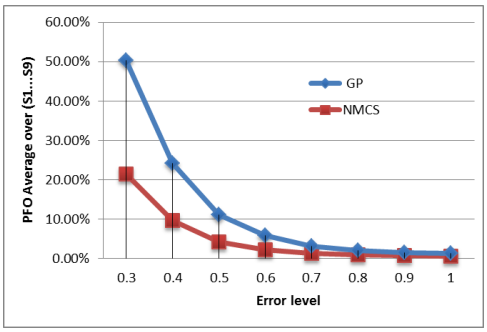

Fig. 6. Evolution of the total PFO percentage for $l =$ varying from 0.3 to 1.

For a more detailed comparative study, only four models are selected: the two NMCS best models and the two GP best models according to the MSE Total and the PFO percentage. The selected models are given in table III with their corresponding "MSE Total".

Model	MN_4	MN_7	MG_4	MG_7
MSE Total	0.000905	0.001702	0.001444	0.0019

TABLE III. "MSE TOTAL" OF THE FOUR BEST MODELS GIVEN BY BOTH METHODS NMCS AND GP.

The performance of the four best models is compared against three criteria:

1) PFE percentage computed according to three different maximum error levels (0.3, 0.4 and 0.5);
2) Shape of the forecasting error of all observations (options) in the input data;
3) Number of PFO in each sub-samples for the three levels 0.3, 0.4 and 0.5.

The figure 7 shows PFO percentage for the first comparison criterion. Since the selected models are the best given by the two methods, their prediction accuracy are quite similar, specially MN4 and MG4.

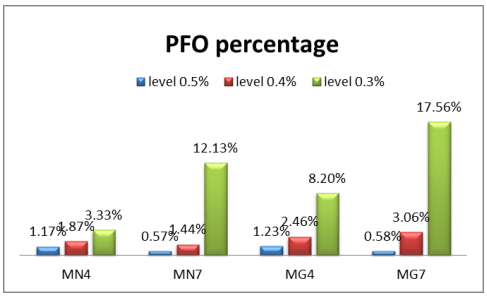

Fig. 7. Variation of the PFO percentage of the four best generated NMC and GP volatility models for a maximum error level equal to 0.3, 0.4 and 0.5.

For the second comparison criterion, we have represented in figure 8 the pattern of the forecasting errors of the retained models. Each point represents the squared error in implied volatility units for one option (observation) in the data set. More monotonous the squared error pattern is, more accurate the corresponding model is. It is therefore clear that the dispersion of the squared errors given by the models MN7 and MG7 presents more inhomogeneity than that of the models MN4 and MG4. They have some difficulties in forecasting options which observed date is far from the training options' dates. This observation could be confirmed by the figures 9, 10 and 11.

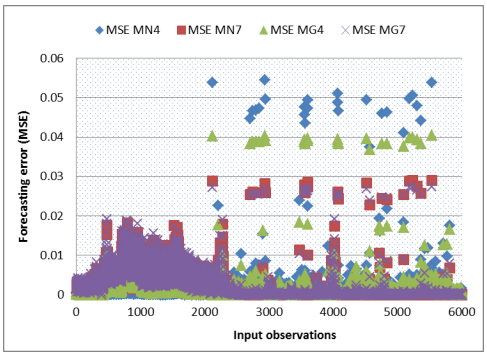

Fig. 8. Pattern of the forecasting errors of the models MN_4, MN_7, MG_4 and MG_7. Each point represents the squared error in implied volatility units for one option (observation) in the data set.

The detailed comparison of the best models showed that the model MN4 given by NMCS is the more accurate for the volatility forecasting. The MG4 could also be considered as an accurate model, but with a little less performance than the MN4.

The formula of the best model MG4 is:

$$MG4 = \exp\left[(ln(\phi(\frac{C}{K})) \times \sqrt{\tau - 2 \times \frac{C}{K} + \frac{S}{K}}) - cos(\frac{C}{K})\right] \quad (3)$$

A detailed examination of the formula for MG4 shows that the implied volatility is function of all the inputs used, namely

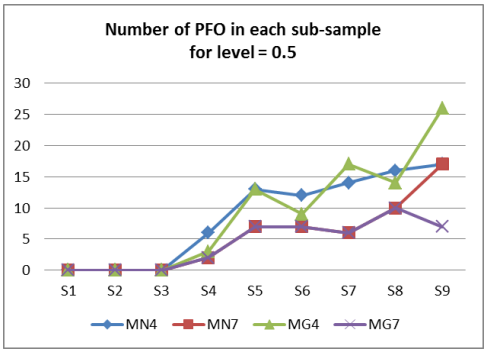

Fig. 9. Variation of the PFO values of the four best generated NMC and GP volatility models computed for each sub-sample with $l = 0.5$.

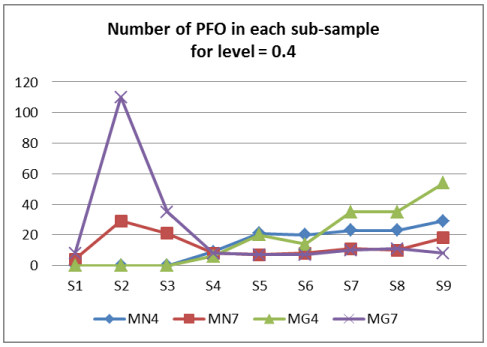

Fig. 10. Variation of the PFO values of the four best generated NMC and GP volatility models computed for each sub-sample with $l = 0.4$.

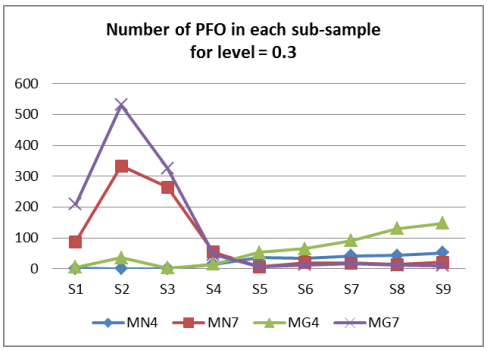

Fig. 11. Variation of the PFO values of the four best generated NMC and GP volatility models computed for each sub-sample with $l = 0.3$.

the option price divided by strike price ($\frac{C}{K}$ for calls and $\frac{C}{K}$ for puts), the index price divided by strike price ($\frac{C}{K}$)and time to maturity (τ). The implied volatility generated for calls and puts cannot be negative since they are computed using the square root and the normal cumulative distribution functions as the root nodes.

A. Discussion

Both GP and NMCS were able to give accurate predictions of the implied volatility. However, the "MSE Total" plot suggest that NMCS seems to produce slightly less error than GP. Indeed, the total MSE is higher for GP models than for NMCS models, excepted for MG_8. According to the total MSE, the model built using NMCS method outperforms the model built using GP.

However, to select a model for prediction, the analysis should be related to efficiency, generalization, stability, and prediction ability of all models. Thus, models are compared also according to the PFO values. This comparison shows that the PFO scores are better for NMCS than for GP.

Another interesting aspect between GP and NMCS regression is related to the trained model generalization capabilities. As we have stated before, the purpose of a forecasting model is to capture important features of real phenomena in order to perform a good prediction about its future behavior. In this sense, models are compared according to the second evaluation metric, that is the PFO. According to this criteria, NMCS could generalize with higher efficacy than GP should we use this measure during the learning phase.

Otherwise, regardless the good quality of the models obtained by the two methods, the use of NMCS for the implied volatility forecasting was simpler and faster than the use of Genetic Programming. Indeed, the latter requires a phase of parameter adjustment. This step is not needed with NMCS. One must note that a choice of unsuitable values for GP parameters could have a negative effect on the quality of the results. This risk does not exist with NMCS.

VII. Conclusion

In this paper Nested Monte Carlo Search is applied to the volatility forecasting problem and compared to Genetic Programming.

Results presented show that NMCS excels in the volatility forecasting and outperforms Genetic Programming for this task. It could generate efficient models with high prediction ability. However, model evaluation can also use measures other than the Mean Squared Error. Using the Poor Fitted Observations measure on expressions learned with the Mean Squared Error fitness function, NMCS gives better results than GP.

The ability of NMCS to avoid over-fitting by limiting the size of a learnt expression is a key factor for improving its generalization ability over GP. Moreover it is a simple and parameter free method that gives good results.

In future work we will apply NMCS to other problems in finance and economy and improve it so as to cope with large amounts of data. We will also investigate the use of playout policies for expression discovery in order to improve even more the effectiveness of NMCS. Furthermore, the method will be compared, besides the GP, with the classic Black&Scholes method.

References

[1] C. J. Corrado and T. W. Miller, Jr., "The forecast quality of cboe implied volatility indexes," *Journal of Futures Markets*, vol. 25, no. 4, pp. 339–373, 2005. [Online]. Available: http://dx.doi.org/10.1002/fut.20148

[2] W. Abdelmalek, S. Ben Hamida, and F. Abid, "Selecting the best forecasting-implied volatility model using genetic programming," *Journal of Applied Mathematics and Decision Sciences*, vol. 2009, 2009.

[3] S. B. Hamida, W. Abdelmalek, and F. Abid, "Applying dynamic training-subset selection methods using genetic programming for forecasting implied volatility," *Computational Intelligence*, 2014. [Online]. Available: http://dx.doi.org/10.1111/coin.12057

[4] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, ser. Lecture Notes in Computer Science, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer, 2006, pp. 72–83.

[5] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *17th European Conference on Machine Learning (ECML'06)*, ser. LNCS, vol. 4212. Springer, 2006, pp. 282–293.

[6] T. Cazenave, "Nested Monte-Carlo Search," in *IJCAI*, C. Boutilier, Ed., 2009, pp. 456–461.

[7] D. Kinny, "A new approach to the snake-in-the-box problem," in *ECAI 2012*, 2012, pp. 462–467.

[8] S. Eliahou, C. Fonlupt, J. Fromentin, V. Marion-Poty, D. Robilliard, and F. Teytaud, "Investigating monte-carlo methods on the weak schur problem," in *Evolutionary Computation in Combinatorial Optimization - 13th European Conference, EvoCOP 2013, Vienna, Austria, April 3-5, 2013. Proceedings*, 2013, pp. 191–201.

[9] T. Cazenave, F. Balbo, and S. Pinson, "Monte-Carlo bus regulation," in *ITSC*, St. Louis, 2009, pp. 340–345.

[10] A. Rimmel, F. Teytaud, and T. Cazenave, "Optimization of the nested monte-carlo algorithm on the traveling salesman problem with time windows," in *Applications of Evolutionary Computation - EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II*, 2011, pp. 501–510.

[11] S. M. Poulding and R. Feldt, "Generating structured test data with specific properties using nested monte-carlo search," in *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*, 2014, pp. 1279–1286.

[12] C. D. Rosin, "Nested rollout policy adaptation for Monte Carlo tree search," in *IJCAI*, 2011, pp. 649–654.

[13] S. Edelkamp, M. Gath, T. Cazenave, and F. Teytaud, "Algorithm and knowledge engineering for the TSPTW problem," in *2013 IEEE Symposium on Computational Intelligence in Scheduling, CISched 2013, Singapore, Singapore, April 16-19, 2013*, 2013, pp. 44–51.

[14] S. Edelkamp and Z. Tang, "Monte-carlo tree search for the multiple sequence alignment problem," in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[15] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.

[16] R. Engle, "Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation," *Econometrica*, vol. 50, no. 4, pp. 987–1007, 1982. [Online]. Available: http://EconPapers.repec.org/RePEc:ecm:emetrp:v:50:y:1982:i:4:p:987-1007

[17] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986. [Online]. Available: http://EconPapers.repec.org/RePEc:eee:econom:v:31:y:1986:i:3:p:307-327

[18] I. Ma, T. Wong, and T. Sanker, "An engineering approach to forecast volatility of financial indices," *International Journal of Computational Intelligence*, vol. vol. 3, no. 1, pp. 23–35, 2006.

[19] T. Cazenave, "Nested monte-carlo expression discovery," in *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, 2010, pp. 1057–1058.

[20] ——, "Monte-carlo expression discovery," *International Journal on Artificial Intelligence Tools*, vol. 22, no. 1, 2013.

[21] J. A. Walker and J. F. Miller, "Predicting prime numbers using cartesian genetic programming," in *Genetic Programming*, ser. Volume 4445 of LNCS. Springer, 2007, pp. 205–216.

[22] L. Spector, D. M. Clark, I. Lindsay, B. Barr, and J. Klein, "Genetic programming for finite algebras," in *Genetic And Evolutionary Computation Conference*. ACM New York, NY, USA, 2008, pp. 1291–1298.

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

[24] E. Burke, S. Gustafson, and G. Kendall, "A puzzle to challenge genetic programming," in *Genetic Programming*, ser. Volume 2278 of LNCS. Springer, 2002, pp. 136–147.

[25] W. B. Langdon and R. Poli, "An analysis of the max problem in genetic programming," in *Genetic Programming*, J. R. K. et al., Ed. Morgan Kaufmann, 1997, pp. 222–230.

[26] J. Méhat and T. Cazenave, "Combining UCT and Nested Monte Carlo Search for single-player general game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 271–277, 2010.

[27] S.-H. Chen and C.-H. Yeh, "Using genetic programming to model volatility in financial time series: the cases of nikkei 225 and s&p 500," in *Genetic programming 1997: Proceedings of the second annual conference*. Citeseer, 1997.

[28] G. Zumbach, O. V. Pictet, and O. Masutti, "Genetic programming with syntactic restrictions applied to financial volatility forecasting," in *Computational Methods in Decision-Making, Economics and Finance*. Springer, 2002, pp. 557–581.

[29] C. J. Neely and P. A. Weller, "Predicting exchange rate volatility: Genetic programming vs. garch and risk metrics," *Federal Reserve Bank of St. Louis Working Paper Series*, no. 2001-009, 2001.