Memetic　Algorithmsを用いたポートフォリオ最適化
Use of Memetic Algorithms for Portfolio Optimization

アランニャ　クラウス [1] *　　伊庭　斉志 [1]
Claus Aranha[1]　　Hitoshi Iba[1]

[1] 東京大学電気系工学学科
[1] University of Tokyo, Department of Electrical Engineering

Abstract: We present the use of Memetic Algorithms for the optimization of Financial Portfolios. Memetic Algorithms are hybrid algorithm where the evolution of individuals lead to the improvement of the portfolio structure, and local optimization rules contribute to the optimization of the weights of the financial assets. We compare this method with older GA-based methods for optimizing portfolios, and observe a noticeable improvement.

1 Introduction

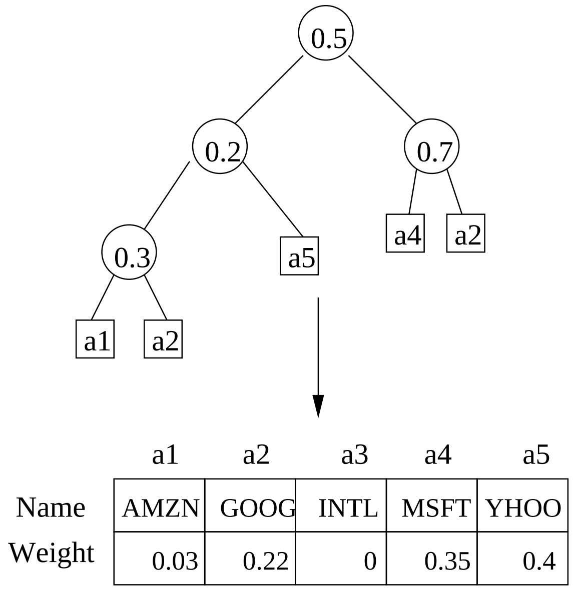
The Portfolio Optimization problem consists of dividing an amount of capital between multiple assets in order to maximize the return, and minimize the risk of the investment.

Investment Portfolios are used by financial institutions in the management of long term investments, like savings accounts, retirement funds, etc. When real life large data sets and constraints are added, though, this becomes a tough problem that cannot be solved by numerical methods.

Because of this, the use of computational heuristics like neural networks and evolutionary algorithms has been an active topic of research. In particular, Genetic Algorithms is one of the most popular approaches recently. This popularity is partly because it is very easy to represent a Portfolio as a real valued array, and use that array as the genome in the Genetic Algorithm.

However, this array representation has a limitation. It does not include information about the relationship between different assets in a portfolio. To address this issue, a Tree-based Genetic Algorithm was developed (*TGA*) [2]. It implements a binary tree representation of the portfolio, where the leaf nodes are the assets and the trunk nodes represent the relationship between these assets (See Figure 1). Early results showed that by using this representation, it was possible to reduce a portfolio produced by GA to its core components, thereby reducing its associated trading costs.

In this work we extend this Tree-based Genetic Algorithm by adding a local search step. We call this hybrid heuristic the Memetic Tree-based Genetic Algorithm (*MTGA*). In the MTGA, the Evolutionary step generates the tree structure, which will select the assets for the portfolio, and determine the hierarchical relationship between them. The local search optimizes the weights at each node recursively, starting from the nodes closest to the terminals towards the root.



	a1	a2	a3	a4	a5
Name	AMZN	GOOG	INTL	MSFT	YHOO
Weight	0.03	0.22	0	0.35	0.4

Figure 1: A tree genome and its corresponding portfolio. The values in the intermediate nodes indicate the weight of the left sub tree. The complement of that value is the weight of the right sub tree. The final weight of each asset (a_x) is given by the sum of the weights of all occurrences of that asset in the tree.

By using simulations with historical data from real-world markets, we observe that MTGA finds portfolios with higher risk-return values higher and when compared to the TGA. This method can also be used to reduce the trading cost between different scenarios.

*連絡先：東京大学電気系工学学科
E-mail: caranha@iba.t.u-tokyo.ac.jp

2 The Portfolio Problem

The resource allocation problem is a traditional optimization problem, which consists of distributing a limited "resource" to a number of "jobs", in order to satisfy one or more utility functions [4].

The Portfolio Optimization problem falls in this category. The limited resource is the capital available for investment, and the jobs are the varied assets in which this capital can be invested (for example, company stock or foreign currency. The utility functions in this problem are the Portfolio Estimated Return, to be maximized, and the Portfolio Risk, to be minimized.

The model for the Portfolio Optimization problem was formally proposed by Markowitz [6]. Markowitz's Portfolio Model could be solved by numerical methods, like Quadratic Programming [13].

However, when adding real world constraints to the problem (for example, large number of assets, restrictions to the values of weights, trading costs, etc), the search space becomes large and non-continuous, and unsolvable by numerical methods. This is what motivates the use of Search heuristics like Evolutionary Computation to solve Portfolio Optimization problems in real world conditions.

2.1 The Markowitz Model

A portfolio P as a set of N real valued weights $(w_0, w_1, ...w_N)$ which correspond to the N available assets in the market. These weights must obey two basic restrictions [13]: The total sum of the weights must be equal to one; and all weights must be positive.

The utility of a portfolio is measured by its *Estimated Return* and its *Risk*. It is calculated as:

$$R_P = \sum_{i=0}^{N} R_i w_i \tag{1}$$

Where N is the total number of assets, R_i is the given estimated return of each asset, and w_i is the weight of each asset in the portfolio.

The risk of an asset is given as the variance of its return over time (variability). The risk of the portfolio is defined as:

$$\sigma_p = \sum_{i=0}^{N} \sum_{j=0}^{N} \sigma_{ij} w_i w_j \tag{2}$$

Where $\sigma_{ij}, i \neq j$ is the covariance between i and j. While the risk is usually stated as the variance of the return of a given asset, there are other definitions of risk that have been used to bias the resulting portfolios towards certain kinds of investment strategies. For other risk metrics, see the works of Harish[10] and Shu[8].

These two utility measures can be used separately to determine the optimal portfolio, or they can be combined. The *Sharpe Ratio* measures the trade off ratio between risk and return for a portfolio, and is defined as follows:

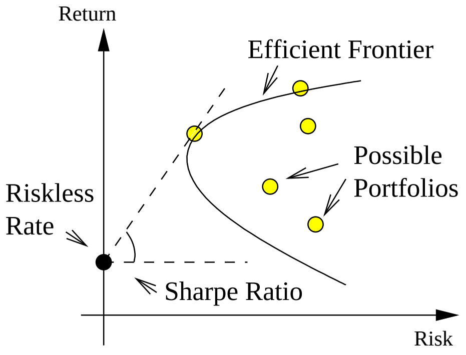


Figure 2: Risk-return projection of candidate portfolios. The search space is bounded by the Efficient Frontier. Sharpe ratio is the angle of the line between a portfolio and the risk-free rate.

$$Sr = \frac{R_P - R_{riskless}}{\sigma_p} \tag{3}$$

Where $R_{riskless}$ is the risk-free rate, an asset which has zero risk and a low return rate (for example, government bonds). The relationship between these three utility measures is illustrated in Figure 2.

2.2 Real World Constraints

The Markowitz Model, as described above, can be solved by optimization techniques such as Quadratic Programming [13]. However, when real world constraints are added, the problem becomes too complex for simple optimization techniques. Practical portfolios are composed from markets with hundreds to thousands of available assets, and the calculation of risk measures grows quickly in relation to the number of assets.

Also, real world applications have constraints related to the values of weights, and to trading. Weight constraints include maximum and minimum weights and lots (indivisible unit of a held asset). These constraints turn the search space non-convex, making the problem harder.

Trading constraints include minimum and maximum trading volume (how much of an asset you can buy at once) and trading cost (proportional to the amount of asset traded). These constraints take effect when multiple scenarios (time periods) are considered, and affect greatly the outcome of the optimization process. In our previous work [1], we have addressed the problem of how to reduce the difference between portfolios of consecutive scenarios to reduce trading cost. Our current proposal also addresses this concern, by removing from the candidate solutions assets that do not contribute to the final result, but increase the trading cost of the portfolio.

3 Related Research

Two important questions in the Portfolio Optimization problem are how to select the assets and the weights. The simplest answer is to use a single array with one real value for the weight of each asset [3, 5].

A more elaborated strategy to select the assets which will participate of the portfolio is to use two arrays: a binary array, which indicates whether an asset is part of the portfolio or not, and the real valued array to calculate the weights of the assets [1, 9].

A somewhat different way to assemble the portfolio is to use GP to evaluate each asset. The GP can be used to calculate the suggested weight of each asset from technical indicators [11], or to generate a ranking of assets, which will be used to select the assets to add to the portfolio [12].

4 Memetic Tree-based Genetic Algorithm

The basic idea of the MTGA is to establish a hierarchical set of relationships between the assets that belong to the portfolio, and use those relationships to improve the exploitation abilities of the Genetic Algorithm.

The tree structure leads to this exploitation by dividing-and-conquering the portfolio in two different ways: It allows the evaluation of the fitness of individual trees, which leads to the crossover based on these fitness values. It also allow the local search step to optimize many 2-variable nodes, instead of one giant portfolio with hundreds of variables at once.

Some of the ideas here, like the local optimization and guided crossover, are inspired by Inductive Genetic Programming [7], a Genetic Programming algorithm used for system identification.

However, unlike iGP, the MTGA is still a GA and not a GP. The main difference here is that while the GP walks the trees from the inputs in the terminals and perform various operations in them to obtain an output in the root node, the tree-based GA goes the inverse way, starting with all the resource in the root node, and dividing it as it progresses down through the tree. The output is the list of weights obtained from the terminal nodes together.

4.1 Tree Representation

Each solution in the Genetic Algorithm is represented as a binary tree. Each non-terminal node holds the weight between its two sub trees. This weight is a real value, w, between 0 and 1, which indicate the weight of its left sub tree (the choice of left over right is arbitrary). The right sub tree of has weight $1 - w$. Each terminal node holds the index of an asset in the market. It is possible to have more than one terminal pointing to the same asset in the same tree. Figure 1 shows this representation.

To extract the portfolio from this representation, we calculate the weight of each terminal node by multiplying the weights of all nodes that need to be visited to reach that terminal, starting from the root of the tree. After all terminal nodes are visited, the weights of those terminals that point to the same asset are added together. The assets which are not mentioned in the tree are assigned a weight of 0.

There are some characteristics of this structure which are important to consider when implementing an Evolutionary Algorithm based on it:

First Every sub tree in an individual can be treated as if it were a normal tree. This is because the root node's structure is identical to that of any intermediate node. This allows each sub tree to have its own individual fitness, calculated in the same way as the fitness of the main tree. This is used in the specialized genetic operators.

Second A portfolio extracted from this representation is always normalized. This is because the weight on each node is limited to the 0..1 interval, and the weight of each terminal is the multiplication of the node weights. Because of the first characteristic, this also applies to sub trees.

Third The maximum number of assets in a portfolio represented by a tree is limited by the depth of the tree. As each terminal corresponds to one asset, a tree with depth d may hold at most 2^{d-1} assets. Because of incomplete trees and terminals with repeated assets, usually the actual number of assets in a tree is much smaller than this.

4.2 Evolutionary Operators

The tree representation for an individual's genetic material in the MTGA requires the redesign of the basic evolutionary operators (crossover and mutation), but it also allow the development of new operators that use the unique characteristics of the tree representation.

The *mutation* operator works by cutting off the tree at a point, and replacing the cut-off sub tree with a randomly generated sub tree. In this work, the cut-off point is selected by first randomly choosing the target depth (with a linear probability), then following a random path from the root node until the desired depth is achieved. This selection method favors cut-off points near the leaves, which results in less aggressive mutations (see Figure 3).

The *crossover* operator works by exchanging sub trees between two individuals. One crossover point is chosen for each tree, and the sub trees that start from that point on are swapped between the two trees.

If the crossover point is chosen at random, the operator is called *Simple Crossover*. Like in the mutation operator, in the simple crossover a depth is chosen with linear probability, and a path is randomly followed from the tree until the target depth.

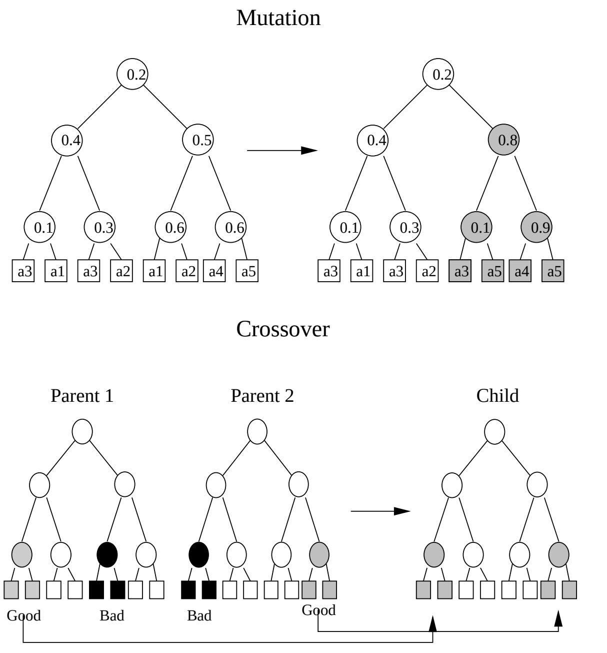


Figure 3: Crossover (BWS) and Mutation operators for the tree representation.

A second crossover operator used in this work is the *Best-Worst Sub tree* crossover (BWS). In this operator, the sub tree with the highest fitness from the first parent is exchanged with the sub tree with worst fitness in the second parent. This operator usually improves the fitness of the individual receiving the better sub tree [2]. This means that the BWS can be used to emphasize a policy of exploitation in the search (See Figure 3).

4.3 Local Search

The local search operator executes a simple hill climbing optimization on each node of an individual. It starts on the deepest non-terminal nodes, and then works its way back towards the root.

For each visited node, the return and risk value for the left and right children are obtained, and used to calculate the utility function as if the node was a two-asset portfolio. The pseudo-code for the hill climbing function can be seen in Figure 4.3.

Where the parameter *meme_speed* is the value by which the weight changes every iteration, *meme_accel* must be < 1.0, and is the value by which *meme_speed* changes every time the weight cross the optima point. And *meme_tresh* is the minimum value of *meme_speed* which signalizes the end of the search. The search also ends if the weight reaches 1.0 or 0.0 (when the optima is not in the weight range 1..0).

The main use of this operator is to improve the weights of the individuals during the evolutionary run. A second utility of this operator, though, is to rebalance the portfolio between scenarios. When we use a portfolio through a long period of time (multiple

```
while (meme_speed > meme_tresh AND 0 < weight < 1)
do
   old_fitness = fitness;
   weight = weight + meme_speed;

   if (weight > 1)
      weight = 1;
   if (weight < 0)
      weight = 0;

   calculate_fitness(weight);

   if (fitness < old_fitness)
      meme_speed = meme_speed * meme_accel * -1;
done
```

Figure 4: Algorithm for local search

scenarios), it is necessary to adjust the portfolio to changes in the market [1].

By repeating the use of local-search operator after a change of market scenario, we can adapt the weights between the assets to the new conditions of the market. Because the the assets in the portfolio does not change, we achieve lower costs than if we started the evolutionary system anew for each scenario.

4.4 Computational Cost

Using Guided Crossover and Local Search (Memetic Step) improve the exploitation capabilities of the Genetic Search in exchange for an increased computational cost.

The basic cost to calculate the fitness value of a portfolio is $O(mn^2)$, where n is the number of assets, and m is the length of the historical data.

Guided Crossover requires that each node in the tree has its own fitness value. This increases the computational cost by one order of magnitude, to $O(mn^3)$. The local search is also applied to each node, and its total computational cost is $O(ns\log_a \frac{t}{s})$, where s, a and t are the *memespeed*, *memeaccel* and *memetresh* parameters, respectively. In practice, the cost of the memetic step is much smaller than the evaluation step.

However, this computational burden can be greatly reduced by an appropriate implementation of the tree structure. Since the fitness evaluation and the memetic step occur in a bottom-up fashion, the number of nodes which need to have their fitness calculated is actually small.

During tree generation all nodes have to be evaluated. During mutation, the new nodes, and the parent nodes of the new tree only need to be re-evaluated. In the worst case, this means $n/2$ nodes for a depth 2 mutation, but this value decreases sharply for deeper mutations. For crossover, only the parent nodes of the cut-off points need to be re-evaluated, which in the worst case, means that the evaluation cost for an individual generated by crossover is $O(mn^2 \log n)$.

5 Experiments

To test the validity of our system, we ran 24 simulations based on historical data. For each simulation, we compared the results of the MTGA, the MTGA without guided crossover, the TGA, the TGA without guided crossover, and three array based systems - a binary array GA, a real array GA and a mixed array GA.

All results displayed in this section, unless otherwise noted, are the average results of 10 experiment runs with different random seeds.

5.1 Data set

The experiments described here were performed on the NASDAQ100 data set, composed of 100 securities from technology related companies. We selected 12 one month periods as scenarios, for this data set and, for each period, used the moving average of the returns of the previous year as a measure of Expected return.

5.2 Parameters

We used 300 generations and 200 individuals per generation. The crossover rate was 0.8, and the mutation rate 0.03. The tree depth was 8 (128 terminals in a full tree). The riskless asset's return was set as 0.003.

For the MTGA system, we used a 0.6 chance of executing the local search step for each individual. The chance of executing the guided crossover was 0.6 per crossover. The sensitivity of the system for these parameters is not explored in this work.

The parameter for the local optimization step are: 0.1 for meme_speed, 0.333 for meme_delta, and 0.003 for meme_tresh. Other than meme_tresh, which changes the precision of the search, changing these values does not seem to affect the quality of the local search.

5.3 Fitness Evaluation Results

The results from the experiment are presented on Table 1. The value of the expected return seems to be high, independently of the method. This is because optimizing the return only requires the selection of the asset with the highest return in the period.

However, to achieve a high Sharpe ratio, a variety of assets must be selected, and we can observe how the MTGA is able to achieve a higher Sharpe ratio than any of the previous methods.

By observing the fitness behavior of each system, we can understand why the MTGA has a higher performance than previous system. Figure 5 shows the fitness improvement of the MTGA, the TGA, the mixed array based GA, and the real valued array based GA.

It can be seen that the MTGA has large jumps in its fitness during the process. These jumps represent mutations which are successfully optimized by the local search step. Usually such mutations would more often than not result in a lower fitness, but the use of local search after each crossover increases the number of possibly useful mutations, improving the search.

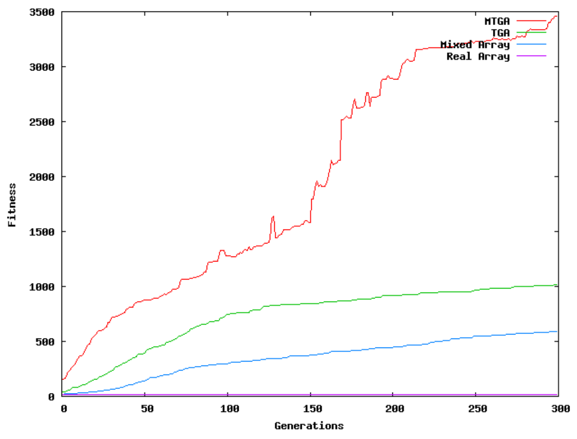


Figure 5: Fitness progression for different systems

In the experiments we noticed that some times the MTGA would produce super positive outliers. As an example, while the average fitness for the NASDAQ data set was between 3000 and 5000, outliers with fitness up to 50.000 were observed. Observing the fitness progress of the outliers and normal runs of the MTGA we see that the cause of this difference are mutation jumps like those shown in figure 5, but of a higher magnitude.

This result indicates that the problem's fitness surface of the portfolio problem is much more bumpy than was perceived, and may require a more aggressive exploration policy for search systems. For instance, higher mutation settings might improve the general performance of MTGA.

6 Discussion

We have expanded the tree-based Genetic Algorithm by adding a local search step, which maximizes the relative weights of each intermediate nodes.

Simulation experiments show that the MTGA has higher performance than previous methods, by increasing the number of positive mutations. It was also observed that the method produces many outliers, which indicates that the problem has a very bumpy fitness landscape.

Current and future investigations include the use of the local search step to reduce trading cost between scenarios, the investigation of “introns” generated by the MTGA, and the sensitivity of the method to mutation parameters.

References

[1] Claus Aranha and Hitoshi Iba. Modelling cost into a genetic algorithm-based portfolio optimization system by seeding and objective shar-

Table 1: Fitness Evaluation Results

	01/2006		04/2006		07/2006		10/2006	
System	Return	Sharpe	Return	Sharpe	Return	Sharpe	Return	Sharpe
MTGA	0.0210	1141.1462	0.0212	3455.6691	0.0193	3596.0163	0.0165	3215.5809
Guided Tree	0.0237	911.6547	0.0272	2102.0846	0.0109	2304.7839	0.0123	2841.1883
Simple Tree	0.0218	875.9712	0.0308	1012.7830	0.0152	2234.6500	0.0166	1936.6126
Mixed Array	0.0296	205.9883	0.0290	588.5863	0.0232	803.2149	0.0158	208.0087
Real Array	0.0169	9.4539	0.0250	20.2493	0.0183	16.7285	0.0131	7.6266
Index	0.0078	2.7457	0.0189	10.3174	0.0109	5.9217	0.0049	1.1830

ing. In *Proc. of the Conference on Evolutionary Computation*, pp. 196–203, 2007.

[2] Claus Aranha and Hitoshi Iba. A tree-based ga representation for the portfolio optimization problem. In *GECCO - Genetic and Evolutionary Computation Conference*. ACM Press, July 2008.

[3] Ronald Hochreiter. An evolutionary computation approach to scenario-based risk-return portfolio optimization for general risk measures. In M. Giacobini et al., editor, *EvoWorkshops 2007*, No. 4448 in LNCS, pp. 199–207. Springer-Verlag, 2007.

[4] Toshihide Ibaraki and Naoki Katoh. *Resource Allocation Problems - Algorithmic Approaches*. The MIT Press, 1988.

[5] Piotr Lipinski, Katarzyna Winczura, and Joanna Wojcik. Building risk-optimal portfolio using evolutionary strategies. In M. Giacobini et al., editor, *EvoWorkshops 2007*, No. 4448 in LNCS, pp. 208–217. Springer-Verlag, 2007.

[6] H. Markowitz. *Mean-Variance analysis in Portfolio Choice and Capital Market*. Basil Blackwell, New York, 1987.

[7] Nikolay Y. Nikolaev and Hitoshi Iba. Regularization approach to inductive genetic programming. *IEEE Transactions on evolutionary computation*, Vol. 5, No. 4, pp. 359–375, August 2001.

[8] Shu ping Chen, Chong Li, Sheng hong Li, and Xiong wei Wu. Portfolio optimization with transaction costs. *Acta Mathematicae Applicatae Sinica*, Vol. 18, No. 2, pp. 231–248, 2002.

[9] Felix Streichert, Holger Ulmer, and Andreas Zell. Evolutionary algorithms and the cardinality constrained portfolio optimization problem. In D. Ahr, R. Fahrion, M. Oswald, and G. Reinelt, editors, *Operations Research Proceedings*. Springer, September 2003.

[10] Harish Subramanian, Subramanian Ramamoorthy, Peter Stone, and Benjamin J. Kuipers. Designing safe, profitable automated stock trading agents using evolutionary algorithms. In *GECCO 2006 - Genetic and Evolutionary Computation Conference*, pp. 1777–1784, Seattle, Washington, July 2006. ACM Press.

[11] James Cunha Werner and Terence C. Fogarti. Genetic control applied to asset managements. In J.A. Foster et Al., editor, *EuroGP*, LNCS, pp. 192–201, 2002.

[12] Wei Yan and Christopher D. Clack. Evolving robust gp solutions for hedge fund stock selection in emerging markets. In *GECCO 2007 - Genetic and Evolutionary Computation Conference*, London, England, July 2007. ACM Press.

[13] Yuh-Dauh-Lyu. *Financial Engineering and Computation*. Cambridge Press, 2002.