Genetic Asset Management (GAM)

Michael Trequattrini[1], Giovanni Trombetta[2]
[1]University of Perugia,
[2]Gandalf Project
[1]michael.trequattrini@gmail.com
[2]giovanni.trombetta@gandalfproject.com

Abstract

This paper presents a comprehensive study of genetic algorithms (GAs) applied to portfolio management, focusing on both single-asset optimization and the innovative framework of Genetic Asset Management (GAM). GAs, known for their ability to evolve optimal solutions through iterative selection and mutation processes, are particularly suited to address the challenges of financial markets characterized by uncertainty and non-stationarity. We first outline their application in single-asset scenarios, highlighting the robustness achieved through the instability factor, which mitigates overfitting risks. The core contribution of this work lies in the development of GAM, a novel approach that extends the application of GAs to multi-asset portfolios. By evolving strategies across multiple assets simultaneously, GAM captures the interdependencies and diversification benefits inherent in portfolio management. Furthermore, we introduce a second methodological innovation that leverages the population-based nature of GAs. Instead of selecting a single optimal solution from the evolutionary process, we propose aggregating all solutions from the final generation into an equal-weighted portfolio. This population-based approach potentially enhances robustness, reduces overfitting risks, and provides greater adaptability to changing market conditions. This study highlights the potential of GAs as a powerful tool in quantitative finance, offering a resilient and adaptive methodology for portfolio management in complex and volatile markets.

Keywords: Quantitative Trading; Portfolio Management; Genetic Algorithms; Optimization

1 Introduction

In computational problem-solving, the demand for robust and efficient optimization techniques has intensified across disciplines, notably within engineering, economics, and finance. In the latter, optimization challenges such as portfolio construction, risk management, and trading strategy design involve high-dimensional, complex search spaces riddled with local optima, where traditional deterministic methods may often struggle to find global solutions or adapt to dynamic market conditions. Meta-heuristic algorithms—particularly genetic algorithms—have thus gained prominence for their adaptability in exploring these vast, non-linear landscapes. GAs employ principles of natural selection, enabling iterative refinement that is well-suited to the stochastic and multi-modal nature of financial markets. This paper investigates the application of genetic algorithms in quantitative finance, with a focus on developing trading strategies and quantitative investment rules. Initially, we review state-of-the-art methodologies in which GAs have been applied primarily to optimize trading strategies for individual assets, underscoring both their benefits and limitations. We then introduce an innovative extension: leveraging GAs to construct systematic, multi-asset quantitative investing rules that consider cross-asset dynamics. By expanding the scope of GA applications from single-asset strategies to diversified portfolio strategies, this research seeks to demonstrate the potential of genetic algorithms to enhance financial decision-making. Through empirical analysis, our objective is to reveal how GAs can navigate the complexities of asset selection and weighting across broad financial instrument baskets, offering valuable insight into their efficacy in creating robust, adaptive portfolios within financial markets. The remainder of this paper is structured as follows: Sections 2 and 3 introduce meta-heuristic algorithms, with a particular focus on genetic algorithms. Section 4 presents the framework for applying genetic algorithms to single-asset optimization. Section 5 extends this framework to enable its application across a basket of financial instruments. Finally, Section 6 provides a case study of Genetic Asset Management and analyzes its empirical results.

2 Introduction to meta-heuristic algorithms

In recent decades, numerous exact and approximate optimization algorithms have been developed to tackle complex problems that require efficient and reliable solutions. Exact algorithms, including those based on dynamic programming, backtracking, and branch-and-bound, have proven effective in many application contexts; however, they encounter significant limitations when applied to large-scale or highly nonlinear problems, especially in cases where the solution space grows exponentially as the problem's complexity increases. In such contexts, an exhaustive search of solutions becomes computationally impractical. Even traditional approximate methods, such as greedy algorithms, present limitations as they often rely on assumptions that are challenging to validate in real-world applications. To overcome these limitations, a class of algorithms known as meta-heuristics has emerged. Meta-heuristic algorithms encompass a set of computational techniques designed to solve complex optimization problems characterized by high levels of complexity, uncertainty, and nonlinearity. Unlike deterministic approaches, which follow a predefined sequence of steps and are prone to convergence at local optima, meta-heuristic algorithms leverage probabilistic mechanisms and diverse exploration strategies to identify solutions that approximate the global optimum. Furthermore, they operate without requiring extensive knowledge of the mathematical properties of the problem, adapting flexibly to various fitness landscapes. This ability to balance exploration and exploitation makes them highly versatile, particularly for high-dimensional problems or those with broad, multi-modal solution spaces. In the field of quantitative finance, meta-heuristics have gained particular relevance due to their ability to explore optimal configurations for complex problems, such as portfolio construction, asset allocation, and the design of trading strategies. These problems are often characterized by stringent constraints and conflicting objectives, such as balancing risk and return. For instance, genetic algorithms, inspired by principles of natural selection, are widely used to generate, evaluate, and iteratively optimize solutions, enabling the identification of optimal portfolio configurations or trading strategies even in dynamic and turbulent market contexts.

2.1 Classification of meta-heuristic algorithms

Meta-heuristic algorithms can be broadly classified according to various criteria, based on their structural and functional characteristics. These classifications include distinctions between nature-inspired and non-nature-inspired algorithms, population-based versus single-point search methods, dynamic versus static objective functions, single neighborhood versus various neighborhood structures, and memory-based versus memory-less methods.

- **Nature-Inspired vs Non-Nature-Inspired Algorithms**: Many meta-heuristic algorithms are inspired by natural phenomena or biological processes, such as evolutionary strategies, swarm behavior, or physical systems. Examples include Genetic Algorithms (GAs), inspired by natural selection, and Particle Swarm Optimization (PSO), based on the social dynamics observed in bird flocking. Alternatively, non-nature-inspired algorithms draw from diverse fields outside of natural sciences; for example, Simulated Annealing (SA) is derived from metallurgy processes, while Tabu Search leverages systematic memory structures to avoid revisiting previously explored solutions.

- **Population-Based vs Single-Point Search Algorithms**: Population-based algorithms operate with a set of candidate solutions evolving simultaneously, which enhances exploration by allowing multiple search paths within the solution space. Examples of population-based approaches include Ant Colony Optimization (ACO), Genetic Algorithms (GAs), and Gravitational Search Algorithm (GSA). Conversely, single-point search algorithms operate with a single candidate solution iteratively refined over time, as seen in Simulated Annealing and Tabu Search. Population-based methods are particularly advantageous in high-dimensional spaces where diversity of search paths is critical for avoiding local optima.

- **Dynamic vs Static Objective Functions**: Meta-heuristic algorithms can also be differentiated by their treatment of the objective function, which may be static or dynamic depending on the problem context. In cases where the optimization environment or objective function changes over time, such as in dynamic scheduling or real-time adaptive systems, dynamic objective functions are applied, and algorithms are designed to adjust their strategies dynamically. In contrast, algorithms with static objective functions operate under the assumption of a fixed optimization landscape, which is typical in scenarios where environmental conditions remain constant.

- **Single Neighborhood vs Various Neighborhood Structures**: Another distinction lies in the structure of the search neighborhood. Some algorithms operate with a single neighborhood structure, where candidate solutions are generated within a fixed proximity to the current solution. In contrast, algorithms with various neighborhood structures explore multiple or adaptive neighborhoods, allowing for a broader and more flexible search range. Variable Neighborhood Search (VNS), for example, strategically changes the neighborhood during the search process to escape local optima by diversifying the search space dynamically.

- **Memory Usage vs Memory-Less Methods**: Memory usage in meta-heuristic algorithms refers to the ability of certain methods to record and recall information about previously visited solutions to guide the search process. Memory-based algorithms, such as Tabu Search, use structured memory to avoid cycling and enhance search efficiency by systematically recording forbidden moves or states. Memory-less methods, such as Simulated Annealing and Particle Swarm Optimization, rely solely on the current state and probabilistic moves without retaining any history, which can simplify their implementation and reduce computational overhead in some contexts.

3 Introduction to Genetic Algorithms

Genetic Algorithms (GAs), as a subset of evolutionary algorithms, are adaptive heuristic search methods based on the principles of natural selection and genetics. Initially proposed by Holland (1975) and popularized by Goldberg (1989), GAs simulate the process of natural evolution, selecting and refining solutions to complex optimization problems. The iterative nature of GAs and their reliance on probabilistic operators enable them to effectively navigate large, non-linear, and multimodal solution spaces, where deterministic optimization methods often falter due to the complexity of the search landscape.

A Genetic Algorithm consists of a set of individuals, each representing a potential solution to the optimization problem. The algorithm operates on a population of such individuals, evolving it iteratively across successive generations. Each individual is composed of genes, which represent decision variables, and its effectiveness, or fitness, is evaluated through an objective function $F(X)$ where X denotes the solution vector representing the individual.

The population consists of N individuals, each of which occupies a point in the search space. This search space encompasses all possible values that the decision variables can take (N_{var}), with examples including $\mathbb{Z}^{N_{\text{var}}}$, $\mathbb{R}^{N_{\text{var}}}$, or $\{0,1\}^{N_{\text{var}}}$, among others. The search space is constrained by the domain of the objective function, ensuring that every individual corresponds to a feasible solution for the problem. Consequently, each individual's fitness is evaluated based on its objective function value, which depends on the specific problem being solved.

For an optimization problem with N_{var} variables, each individual X_j can be represented as a vector:

$$X_{j,i} = (x_{j,1}, x_{j,2}, \ldots, x_{j,N_{var}}) \quad \text{for } j = 1, \ldots, N \quad \text{and} \quad \text{for } i = 1, \ldots, N_{var}$$

where $x_{j,i}$ denotes the i-th gene or decision variable of the j-th individual.

The population matrix $\mathbf{X}$ can thus be expressed as:

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N_{var}} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N_{var}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,N_{var}} \end{bmatrix}$$

The fitness function $F(X_j)$ evaluates each individual's performance relative to the optimization objective, and it serves as a guiding metric for subsequent selection and reproduction phases.

The representation can vary depending on the problem requirements, primarily in binary or continuous forms:

- **Binary representation**: In binary encoding, each gene $x_{j,i}$ is a binary digit (0 or 1), making each individual a string of bits. The full population is then represented by an $N \times N_{\text{bits}}$ matrix, where N_{bits} denotes the length of the binary string for each individual. Each bit pattern corresponds to a unique solution, encoding variables either directly as integers or by

mapping bit strings to quantitative or qualitative values. This binary encoding is particularly advantageous for discrete and combinatorial problems due to its simplicity and high schemata density, maximizing the number of search patterns available per bit.

- **Continous representation**: In continuous encoding, each gene $x_{j,i}$ is represented as a floating-point number within a specified range. The population is represented by an $N \times N_{\text{var}}$ matrix, where each row corresponds to an individual, and each entry holds a continuous value for a variable. Continuous GAs often require less storage than binary GAs because each decision variable is represented by a single floating-point value rather than a sequence of bits. This representation is typically better suited for optimization problems with smooth solution spaces.

Given the discrete and combinatorial nature of the problem we address in this paper, the binary representation is particularly well-suited, allowing structured, granular exploration of the solution space that aligns closely with the problem requirements. The following sections will therefore emphasize the binary encoding approach.

3.1 Evolution process

Genetic Algorithms rely on an iterative process that begins with the generation of an initial population P_0, consisting of N feasible solutions. Each element of the population, defined as an individual, represents a potential solution to the optimization problem at hand. The quality of each individual is measured using the objective function, which returns a value called fitness. This value determines how effectively the individual solves the optimization problem. Following the principles of Darwinian theory, individuals with better fitness have a higher chance of surviving and reproducing, while those with lower fitness are eliminated from the population. Within each generation, a subset of M individuals, the fittest ones, is selected to form the core of the new population. These selected individuals are then modified through genetic operators, which have three main functions:

- To generate new individuals (offspring)
- To ensure diversity in the population
- To preserve advantageous traits inherited from previous generations (elitism)

The number of individuals each operator affects is determined by a fixed application rate, which, usually, remains constant throughout the evolutionary process. In this work, the traditional GA is considered. For this reason, only the three most useful operators (selection, crossover and mutation) are used in the algorithm. After the application of these operators, the new population is evaluated using the objective function and ranked based on fitness values. This cycle repeats until one of the stopping criteria is met, which could be achieving a satisfactory solution or completing a predefined number of iterations.

3.1.1 Selection of the Fittest Individuals

The aim of the selection procedure, is to select individuals of the existent population in order to reproduce more copies of individuals (offspring). This procedure is based on the survival probability that depends on the objective function value of the individual.

Once the fitness values are evaluated, the next step is to select a subset of M individuals, the fittest ones, who will form the core of the new population. The selection process can follow various approaches, but it is typically probabilistic and favors individuals with higher fitness. This relationship can be formally expressed as:

$$Pr(\text{select}(X_j)) \propto F(X_j)$$

Where $F(X_j)$ is the fitness value of individual X_j, and the selection probability increases with better fitness. The selected individuals form the subset $P_{elite} \subset P_0$, and will participate in crossover and mutation.

$$P_{elite} = \{X'_1, X'_2, \ldots, X'_M\}$$

where X_i' represents the individuals selected during the selection process. These individuals, marked with a prime symbol ($'$), have been chosen based on their fitness, ensuring that the most promising candidates are carried forward for further steps in the genetic algorithm.

3.1.2 Genetic Operators: Crossover and Mutation

The next step is to modify the selected individuals through the application of genetic operators.

Crossover: The crossover operator combines the genetic material of two selected parents X_i' and X_j' to produce two offspring. Mathematically, the crossover operation can be defined as a process that creates new individuals by combining segments of the parent chromosomes. For example, for a single-point crossover between two parents:

$$X_i' = (x_{i1}, x_{i2}, \ldots, x_{iN_{var}}) \quad \text{and} \quad X_j' = (x_{j1}, x_{j2}, \ldots, x_{jN_{var}}),$$

the offspring with k randomly selected crossover point are created as follows:

$$X_i'' = (x_{i1}, x_{i2}, \ldots, x_{ik}, x_{j(k+1)}, \ldots, x_{jN_{var}})$$
$$X_j'' = (x_{j1}, x_{j2}, \ldots, x_{jk}, x_{i(k+1)}, \ldots, x_{iN_{var}})$$

Mutation: The mutation operator introduces random changes in the offspring's chromosomes, ensuring genetic diversity and preventing premature convergence. If an individual

$$X_i' = (x_{i1}, x_{i2}, \ldots, x_{iN_{var}})$$

undergoes mutation at gene x_{ik}, it will be modified as follows:

$$X_i'' = (x_{i1}, x_{i2}, \ldots, x_{i(k-1)}, x_{ik}', x_{i(k+1)}, \ldots, x_{iN_{var}})$$

where x_{ik}' represents a new, randomly chosen value for gene x_{ik}. Both crossover and mutation are applied to the M selected individuals with predefined probabilities, $p_{\text{crossover}}$ and p_{mutation}, respectively.

3.1.3 Evolution of the population

The new population is formed by combining the M individuals (the best of the last population) with the offspring produced via crossover and mutation. The size of the new population is typically equal to the original population size, ensuring that there is no loss in the number of candidates. The new population P_{k+1} is evaluated using the objective function, and the cycle repeats for a predefined number of generations or until a stopping criterion is met. The new population can be represented by:

$$P_{k+1} = P_{elite_k} \cup \text{Crossover}(P_{elite_k}) \cup \text{Mutation}(P_{elite_k})$$

This iterative process continues until the stopping criterion is satisfied. Common stopping criteria include reaching a predefined number of generations G, achieving a satisfactory fitness value $F(X_i)$, or a time limit. To summarize, the codification of a basic genetic algorithm is represented by the follow algorithm:

Algorithm 1 Traditional Genetic Algorithm

Input: Population size N, number of elite individuals M, maximum generations G
Output: Best individual x^*
Initialize the population P_0 with N randomly generated individuals
Set generation counter $k \leftarrow 0$
while stopping criterion (e.g., $k \geq G$ or convergence) is not met **do**
  $P_{\text{elite}} \leftarrow$ Select the M best individuals from P_k ▷ Elitism selection
  $P_{\text{cross}} \leftarrow$ Apply crossover to individuals in P_{elite} ▷ Crossover operation
  $P_{\text{mut}} \leftarrow$ Apply mutation to individuals in P_{elite} ▷ Mutation operation
  $P_{k+1} \leftarrow$ Generate the new population as $\{P_{\text{elite}} \cup P_{\text{cross}} \cup P_{\text{mut}}\}$
  Update generation counter: $k \leftarrow k + 1$
end while
$x^* \leftarrow$ Best individual in P_G, where P_G is the final generation

4 Genetic Algorithms for Single Asset Trading Systems

Although the basic structure of the evolutionary framework remains unchanged, the peculiar nature of the application domain requires a specialized representation of candidate solutions, as well as parameterization that reflects the unique characteristics of financial markets.

4.1 Genetic pool

The genetic pool in a genetic algorithm applied to trading is a foundational element that dictates the diversity and potential efficacy of evolved solutions. Its creation involves careful selection and encoding of components derived from financial data and indicators, ensuring relevance to the specific dynamics of market behavior. The genetic pool serves as a repository of fundamental building blocks—such as price data, lagged observations, AI-generated indicators, and more—used to construct candidate trading rules.

The creation of the genetic pool begins with the identification of fundamental market variables. A basic example may include the four daily price levels—open, high, low, and close (OHLC)—along with their lagged values over L previous periods. For instance:

Genetic Pool Elements:

$$\{O_{(t-k)}, H_{(t-k)}, L_{(t-k)}, C_{(t-k)} \text{ for } k = 1, 2, \ldots, L\}.$$

To enhance the expressive power of the pool, it can be expanded by including:

- **Technical Indicators**, such as moving averages (MA), relative strength index (RSI), Bollinger Bands and/or any other more complex indicator coming from AI models.
- **Derived Metrics**, such as volatility, Sharpe Ratio ecc...
- **Custom Variables**, tailored to the specific characteristics of the trading strategy or asset class under consideration.

The encoding of these elements ensures compatibility with genetic operators, allowing their combination and manipulation to form valid trading rules.

The importance of the genetic pool

The genetic pool serves as the basis for generating initial candidate solutions and enables subsequent evolution. Its composition significantly impacts the algorithm's ability to discover robust and generalizable trading strategies. A well-constructed genetic pool must balance:

- **Diversity:** Ensuring the genetic pool includes a wide range of variables and transformations to represent a rich search space.
- **Relevance:** Focusing on elements that capture meaningful market dynamics to prevent wasted computational effort on irrelevant features.

A poorly constructed genetic pool, either too narrow or overly expansive, can impair the evolutionary process. Insufficient diversity may lead to premature convergence, while excessive complexity can introduce noise due to an excessive tailored solution to the idiosyncrasies of the training data, resulting in poor performance on unseen data. This issue arises due to the model's capacity to memorize spurious correlations or noise rather than capturing underlying market dynamics.

4.2 Representation of Solutions

The evolutionary process begins with the creation of an initial population composed of N solutions, where each solution represents an artificial trading agent. Each agent attempts to optimize various aspects of the trading rules, which include:

- **Entry signals:** define the optimal moment to open a position;
- **Exit signals:** determine when to close the position;
- **Position management:** specifies the size of open positions;
- **Money management parameters:** regulate capital management and risk management.

In the standard framework, the exit is typically fixed through a temporal closure after a predefined number of time bars. This approach isolates and more precisely evaluates the quality of the entry signal by eliminating potential interference from dynamic exit criteria, which could otherwise affect the system's overall performance independently of the entry quality. While the genetic algorithm can indeed be extended to incorporate all four aspects—entry, exit, position sizing, and risk management—such an extension must be undertaken with careful consideration to prevent overfitting and maintain the robustness of the optimization process.

4.3 Structure of the Decision Matrix

In a standard formulation of the problem, trading rules are represented using a two-dimensional structure that encodes the comparative relationships among the atomic elements of the genetic pool. Assuming the genetic pool is formed only from the lags of the four daily prices (OHLC), the structure of the matrix (illustrated in Figure 1) can be expressed as follows:

- **X-axis:** each column represent one single element of the genetic pool. For example, c_3 represents the closing price 3 days ago, while h_5 indicates the high from 5 days ago.
- **Y-axis:** mirrors the structure of the X-axis, creating multiple possible combinations considering a lookbacks of 5 periods.

>	c	c1	c2	c3	c4	c5	o	o1	o2	o3	o4	o5	l	l1	l2	l3	l4	l5	h	h1	h2	h3	h4	h5
c		0	1	2	3	4	5	6	7	8	9	10		11	12	13	14	15		16	17	18	19	20
c1			21	22	23	24	25	26	27	28	29	30	31		32	33	34	35	36		37	38	39	40
c2				41	42	43	44	45	46	47	48	49	50	51		52	53	54	55	56		57	58	59
c3					60	61	62	63	64	65	66	67	68	69	70		71	72	73	74	75		76	77
c4						78	79	80	81	82	83	84	85	86	87	88		89	90	91	92	93		94
c5							95	96	97	98	99	100	101	102	103	104	105		106	107	108	109	110	
o								111	112	113	114	115		116	117	118	119	120		121	122	123	124	125
o1									126	127	128	129	130		131	132	133	134	135		136	137	138	139
o2										140	141	142	143	144		145	146	147	148	149		150	151	152
o3											153	154	155	156	157		158	159	160	161	162		163	164
o4												165	166	167	168	169		170	171	172	173	174		175
o5													176	177	178	179	180		181	182	183	184	185	
l														186	187	188	189	190		191	192	193	194	195
l1															196	197	198	199	200		201	202	203	204
l2																205	206	207	208	209		210	211	212
l3																	213	214	215	216	217		218	219
l4																		220	221	222	223	224		225
l5																			226	227	228	229	230	
h																				231	232	233	234	235
h1																					236	237	238	239
h2																						240	241	242
h3																							243	244
h4																								245
h5																								

Figure 1: Decision matrix with OHLC prices

4.4 Genetic Encoding of Trading Rules

The genome of each trading agent consists of a single chromosome, composed of a predefined number of genes, where each gene encodes a specific trading rule. The encoding of each rule follows a Boolean logic, which can be represented in binary format, allowing the application of genetic operators. These rules correspond to specific cells within the matrix representation introduced earlier (Figure 1), where the comparative relationships between elements of the genetic pool are defined. An example of a possible encoding:

- **Rule 1:** $c > c_2$, where today's closing price (c) is greater than the closing price from two days ago (c_2);
- **Rule 16:** $c > h_1$, where today's closing price is greater than yesterday's high (h_1);
- **Rule 159:** $o_3 > l_5$, where the open from 3 days ago (o_3) is greater than the low from 5 days ago (l_5).

Within the first population, each trading agent randomly selects a fixed number of rules (genes), which are combined into a chromosome. A market entry occurs when all selected rules are simultaneously satisfied. The objective function for optimization is thus determined by the combination of activated rules, without the use of logical OR operators, making each chromosome a sequence of rules that activate sequentially. Alternatively, it is possible to introduce a variant with multiple chromosomes, where each chromosome is combined using logical OR operators.

It is crucial to construct the decision matrix and the potential combinations of the elements that will form the genes in a logically coherent and contextually appropriate manner. This matrix serves as a foundational component in ensuring that the trading rules are both logically consistent and meaningful within the financial market context. Specifically, the comparison between elements should be made only between quantities that are inherently comparable. For example, it is valid to compare the maximum price (H_t) with the minimum price (L_t), or a specific market metric, such as the Sharpe ratio (SR_t), with an absolute threshold (θ). Mathematically, these relationships can be expressed as:

$$SR_t > \theta, \quad \text{or} \quad H_t > L_t,$$

where SR_t represents the Sharpe ratio at time t, and θ is a fixed threshold, or H_t and L_t represent the maximum and minimum prices at time t, respectively. Conversely, it is imperative to avoid comparisons between quantities that are not logically related or cannot be meaningfully evaluated together. For instance, comparing the minimum price (L_t) with trading volume (V_t) is nonsensical, as these are distinct dimensions and do not inherently share a relationship that would yield a valid trading rule. Such comparisons may result in illogical or erroneous conditions that undermine the integrity of the decision-making process.

Formally, the set of trading rules must be constructed such that each rule is a valid logical expression over consistent market quantities. This ensures that the system's rules are interpretable, reliable, and applicable within the context of the financial markets. By adhering to these principles of logical coherence and contextual relevance, the trading system's robustness and the interpretability of the evolved solutions are significantly enhanced.

4.5 Evolutionary Process

The evolutionary process begins with the creation of N candidate solutions, where each solution is represented by a DNA composed of a fixed number of randomly selected genes. This initial population serves as the starting point for the genetic algorithm. Following this initialization, the selection phase identifies the most promising individuals, as explained in the previous section, and the population is reconstructed for the next generation through evolutionary operations.

The fitness function is used to evaluate the performance of individual agents in the population. This function can be any metric or combination of metrics that reflects the desired characteristics of the trading system. For instance, one might opt for a single performance measure such as net profit or Sharpe ratio, or a more complex, custom-built function that combines multiple metrics. An example of such a combination is:

$$F(X) = \alpha \cdot \text{Sharpe Ratio} + \beta \cdot \text{Profit Factor}$$

where α, β are weighting coefficients that determine the relative importance of each parameter in the overall evaluation.

The selection phase retains a fraction M of the population, corresponding to the individuals with the best performance according to the fitness function. The choice of the fitness function is one of the most critical aspects of the genetic algorithm, as it directly shapes the evolutionary process. A well-defined fitness function guides the algorithm toward solutions that meet specific objectives, such as maximizing returns, minimizing risk, or optimizing a balance between the two.

Conversely, an inadequately chosen fitness function may lead to suboptimal outcomes or overfitting. For example, overemphasizing short-term profit without accounting for risk could result in a trading system that performs well on historical data but fails under real-world conditions. Therefore, the fitness function must be aligned with the overall goals of the optimization process to ensure the robustness and generalizability of the resulting trading strategy. When creating a custom fitness function, it must be carefully defined, avoiding the introduction of excessive parameters that could be subject to arbitrary or incorrect choices. Such complexity in the metric definition may lead to overfitting. Alternatively, different methodologies allow for genetic evolution with multiple fitness functions, which can help mitigate these risks.

Once the selection phase has reduced the population to the top-performing individuals (a fraction M), evolutionary operators—crossover and mutation—are applied to regenerate the population to its original size. These operators introduce diversity and innovation into the population, preventing premature convergence and fostering the discovery of better solutions.

Given the binary representation of the solutions, the application of these evolutionary operations is computationally straightforward. At the end of the evolutionary process, the best-performing individual from the last generation is selected. The trading rule encoded in this individual's DNA is then adopted for live trading, embodying the culmination of the algorithm's optimization efforts.

4.6 Validation and robustness

The approach described so far relies on analyzing the entire available historical series. However, financial markets are characterized by their ever-changing and unpredictable nature. For this reason, it is essential to incorporate a validation layer into the development process, whether using standard methodologies or genetic approaches.

Validation represents a crucial element—arguably the most important—in the development pipeline. To understand its fundamental role, we can imagine the price as the sum of two components: the signal (the trend) and the noise (pseudo-random fluctuations arising from the book's configuration at a given moment). By nature, noise is random and, as a result, impossible to predict accurately due to its stochastic variability.

A robust validation system is essential to ensure that strategies are designed to identify the signal without being influenced by the noise. Various methodologies are available to address different needs, and one of the most intuitive consists of dividing the data into two parts: one for strategy development (in-sample, IS) and the other for testing (out-of-sample, OoS).

There are various approaches to splitting the data, as illustrated in Figure 2:

- Approach A: This approach allows for a single backtest, making the process simpler but potentially risky if one does not have full mastery of the tools. The risk of overfitting the model to the noise in the series is high, especially for beginners in systematic trading. It is easy to produce seemingly solid backtest curves that are, in reality, based on specific configurations unlikely to occur in the future.

- Approach B: This is the classic approach, where trading rules are determined in the IS period and evaluated in the subsequent OoS period. Typically, an IS-OoS ratio of 80-20 or 70-30 is used. This enables testing the system on data that was never used during the training phase. If the metrics remain consistent between IS and OoS, the likelihood—though not the certainty—of the strategy performing well over time increases. The advantage is that this method provides a test to filter out ineffective strategies before applying them in a real-world context. However, the system is created using older historical data, increasing the risk that market changes may compromise its validity.

- Approach C: This approach reverses the temporal order between IS and OoS, training the model on the most recent data and validating it on older data. It allows leveraging current data for training but makes it more challenging to test the system's effectiveness under present market conditions. This method also requires careful consideration to strike an optimal balance.

- Approach D: Duplicates the OoS period by placing the IS in the middle. This approach combines the benefits of approaches B and C but reduces the length of the training period. In markets with limited historical data, such as electronically traded metals since 2007, this configuration may not be reliable.

- Approach E: Effective validation depends not only on the temporal duration of the In-Sample and Out-of-Sample periods but also on the market conditions captured during these phases. For example, an IS focused on a prolonged, high-volatility bearish trend might inadequately represent low-volatility or sideways markets, making the strategy vulnerable to changing scenarios. For this reason, it is essential to include a variety of market conditions in both IS and OoS. This need is central to the GSA (Gandalf Segmented Architecture), developed by the Gandalf Project R&D team and first introduced in 2017 during the IFTA annual conference, "Sailing to the Future", in Milan. The GSA segments the entire historical dataset with variable granularity, creating symmetric or asymmetric sequences of IS and OoS periods. More frequent segmentation enables more homogeneous aggregated periods, significantly improving the robustness and reliability of validation. This methodology allows the development and testing of strategies across a broader range of market conditions, greatly enhancing their resilience. However, a common drawback of both GSA and other traditional approaches is the arbitrariness in selecting the size of IS and OoS segments, which can affect validation results. This limitation can be largely overcome with a Monte Carlo approach, as discussed in [1].

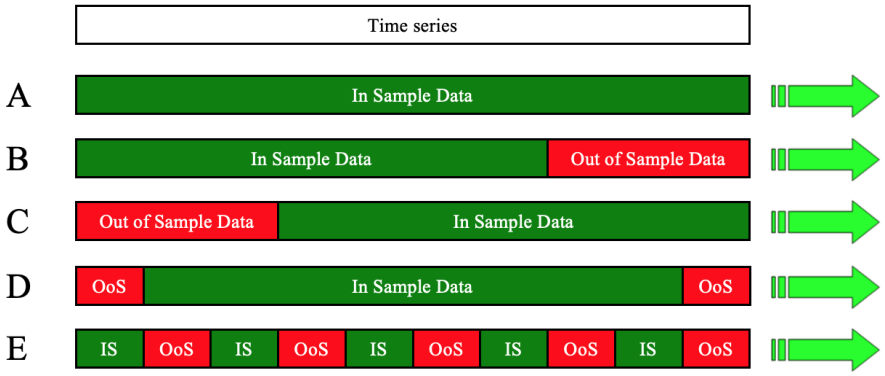


Figure 2: Different approaches of validation

In the case of the genetic process, this work uses the GSA architecture (approach E) as an initial foundation and then integrating it, during the genetic asset management phase, into a dynamic walk-forward system. This combined approach ensures further robustness by continuously adapting to evolving market conditions.

4.7 Instability Factor

To simulate the GSA during the genetic optimization process, we introduced what we call the instability factor $\pi \in [0, 1]$, where π represents the probability of skipping each trade. In other words, whenever an individual, based on its genetic trading rule, is supposed to execute a trade at time t, the instability factor determines the likelihood of actually recording that trade during the backtest. For any trade proposed by an individual's trading rule at generation g, the trade is executed with a probability $1 - \pi$.

This instability factor is applied to each individual in every generation, leading to the random exclusion of certain trades. This mechanism is conceptually similar to dropout in neural networks, where random nodes are deactivated during training to improve generalization.

In this context, the instability factor serves as both a computational shortcut for implementing the E approach saw before (Figure 2), and a mechanism to introduce randomness into the backtesting process. This added randomness challenges the optimization algorithm to adapt under uncertain conditions, making the process more robust and significantly reducing the risk of overfitting. Furthermore, if the instability factor π is not too small (recall that it represents the probability of skipping a trade), it expands the exploration phase, helping the algorithm escape lo-

cal optima and avoid suboptimal solutions caused by premature convergence. Indeed the instability factor π directly affects the balance between exploration and exploitation:

- Low π (close to 0): Emphasizes exploitation, with most trades executed, allowing the genetic algorithm to fine-tune solutions.
- Moderate π: Enhances exploration by introducing variability, helping the algorithm escape local optima.
- High π (close to 1): Too many skipped trades may hinder convergence, requiring careful tuning of π to avoid undermining the optimization process.

For example, Figure 3 illustrates the fitness evolution curve during a genetic process for developing a trading strategy, comparing results with and without the instability factor.

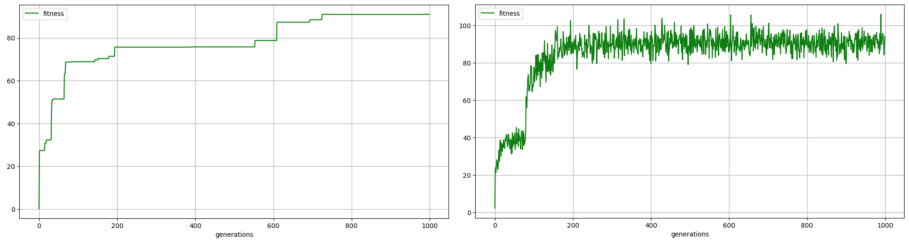


Figure 3: Fitness function evolution with and without instability factor

The curve on the left, obtained without applying the instability factor, exhibits a monotonically non-decreasing trend. This behavior risks pushing the algorithm toward excessive optimization, increasing the likelihood of overfitting. In contrast, the curve on the right, generated by applying a 5% instability factor, shows a different pattern. During the initial generations, the curve grows noticeably, indicating that the individuals are improving their understanding of the market through the available genetic pool. However, as the genetic process progresses, the improvement slows down and eventually becomes negligible. This behavior prevents the algorithm from over-optimizing, instead promoting more robust and generalizable strategies. This also suggests that we might not necessarily need to wait for the full 1000 generations, but could terminate the process much earlier, when further iterations no longer lead to meaningful improvements.

Therefore, the benefits of the instability factor can be summarized as follows:

- **Reduced Overfitting**: By probabilistically skipping trades, the genetic process is less likely to overfit to historical data, as solutions must remain effective under uncertainty.
- **Enhanced Exploration**: Skipping trades allows the algorithm to explore a broader solution space, reducing the likelihood of converging prematurely to suboptimal solutions.
- **Computational Efficiency**: The instability factor simplifies the implementation of GSA principles by introducing randomness directly into the backtesting process without requiring explicit segmentation.

In conclusion, the instability factor π is a crucial innovation that balances robustness and adaptability in genetic optimization, ensuring that strategies remain resilient across varying market conditions.

4.8 Choice of the best individuals

In the framework outlined, in the case of a single asset, at the conclusion of the genetic evolution process, which takes place during the in-sample phase, a single solution will be selected. This solution will then be tested on the out-of-sample data and, subsequently, implemented in real trading, at least until the decision is made to restart the genetic evolution process in order to identify a new trading rule.

This methodology results in the identified genetic strategy being static over time, with its respective advantages and drawbacks. The primary advantage lies in the reduced computational demand and the increased ease of implementing the process. However, this approach does not account for the dynamic nature of the markets. In the subsequent section, we will explore how this issue is addressed by introducing a dynamic trading rule, derived through dynamic genetic evolution.

5 Theory and Architecture of Genetic Asset Management

This section differs from the previous one as it focuses not on applying a genetic rule to design a trading or investment system for a single asset, but rather on a typical portfolio management challenge. Specifically, the issue addressed concerns the decision-making process of a portfolio manager who, having a basket of securities at their disposal, must develop a model to determine which securities to keep in the portfolio and which to exclude. The innovation proposed in this work lies in the use of genetic algorithms to develop a diversified portfolio of assets, thereby introducing an original approach to the field of quantitative investment management.

5.1 GAM: Design and Implementation

As the term itself suggests, Genetic Asset Management refers to an allocation approach on a basket of assets using a genetic rule. The main objective of this methodology is to enable the system to identify a rule that optimizes a fitness function. Once defined, this rule is applied to each asset in the basket, with the task of deciding whether or not to keep each security in the portfolio for the following period. To better understand how this approach works, let's consider two time snapshots (Figure 4): one at period 1, and one at time period 2. In the first period, the selected assets in the portfolio are numbers 5, 8, 12, 14, 23, and 26. In the subsequent period, new assets are activated, including numbers 4, 6, 10, 15, 20, and 25, which may partially overlap with assets already included in the previous period.

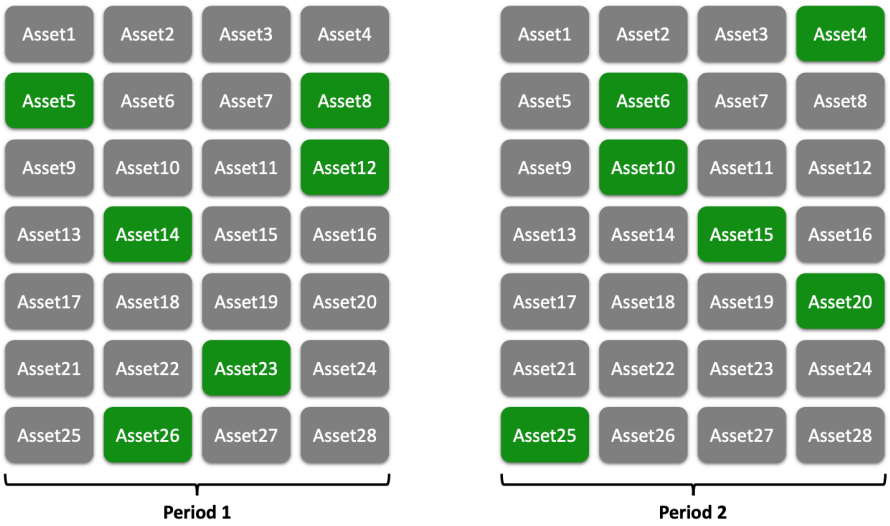


Figure 4: Example of two-period snapshots showing the assets selected in periods 1 and 2.

5.2 Rebalancing Methods

Before examining how to manage portfolio rebalancing, it is useful to introduce a fundamental distinction that will guide the rest of the discussion:

- **Trigger**: the time point at which the portfolio rebalancing occurs.
- **Period**: the time interval during which the securities identified by the previous trigger are held in the portfolio with a buy-and-hold strategy.

"On-off" with continuous triggers

In this scenario, the concepts of period and trigger are considered equivalent. Suppose, for instance, that the period is daily. With a daily trigger, the system updates the portfolio allocation each day. For example, in the first period (day 1, trigger 1), the portfolio may include assets 1, 2, and 5. On day 2 (trigger 2), the allocation is updated to include assets 2, 3, and 5, and so on.

In summary, each period represents a change in the portfolio allocation. The goal is to develop the optimal allocation from one trigger to the next, with the triggers and periods being directly aligned.

"On-off" with periodic triggers

In the on-off approach with periodic triggers, the relationship between trigger and period is no longer one-to-one. Here, each trigger spans n periods. For instance, in the example illustrated (Figure 5), three periods are considered, but the concept can be generalized. To clarify, in a weekly rotation scenario, the portfolio allocation remains unchanged throughout the entire trading week, from Monday to Friday (each day is a period). During this time, the securities in the portfolio are held, and at the end of the week, typically on Friday, the portfolio's performance is assessed, and a new allocation is determined. This updated allocation is then applied at the start of the following week, specifically on Monday, initiating a new cycle.

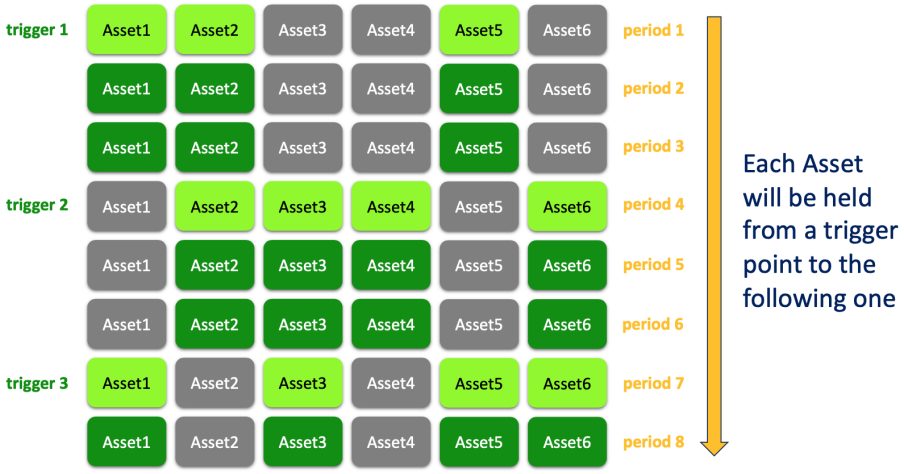


Figure 5: On-off approach with period triggers

Following the picture, the buy-sell rule generated by the genetic algorithm defines an allocation on assets 1, 2, and 5. With this second on-off approach with periodic triggers, what happens after the first trigger, up to the next one, is that the assets defined in the first trigger are maintained in the portfolio with a buy-and-hold strategy. Therefore, during period 2 and period 3, the same assets defined in period 1 are kept. When trigger 2 arrives, the allocation changes, and the portfolio is now composed of assets 2, 3, 4, and 6. From that moment, for period 5 and period 6, the assets defined by trigger 2 will continue to be held. This process repeats when trigger 3 arrives, which will change the allocation again in period 7 and maintain it in subsequent periods, as in period 8 and period 9, with assets 1, 3, 5, and 6.

In essence, in this approach, each allocation is maintained in the portfolio until the next trigger, which establishes a new allocation. Unlike the previous case, where period and trigger were synonymous, in this model, the two quantities do not coincide. The period may be daily, but the trigger could be weekly, for example. This allows for slower rotation compared to the previous model, where each period corresponded to a single allocation. It is worth noting that, for the rest of the paper, the on-off approach with periodic triggers will be adopted, as it is considered more suitable for the type of analysis and portfolio rebalancing to be developed.

Generation of the Genetic Pool and Gene Classes

The next step involves generating gene classes from the historical price data. The goal at this stage is to identify macro-level gene classes that effectively capture the distinctive characteristics of the assets in the portfolio. Various metrics derived from historical price data are considered as potential genes within the genetic pool.

For instance, one possible gene class could be based on the cumulative percentage change over a specific time window. This metric captures the total movement in the asset's price over a defined period. Another gene class might represent the average percentage change during the same window, providing a smoother indication of the asset's general trend. In addition to these, other gene classes can be defined to capture different aspects of asset behavior, such as volatility, profitability, or risk-adjusted performance. These measures are useful for quantifying and comparing the performance of assets over time, and they provide essential inputs for the genetic algorithm to evolve and optimize portfolio strategies.

The genetic pool, therefore, is composed of a diverse set of gene classes, each representing a distinct characteristic or metric derived from asset price data. The combination of these classes

allows the genetic algorithm to explore a broad range of potential solutions, ultimately enabling the model to select the most effective strategy for portfolio construction and dynamic market adaptation.

The Rule Matrix and Gene Combination

As we did earlier with the single-asset case, we need to construct a rule matrix from which potential genetic solutions can draw. Since the logical mechanism is already familiar, Figure 6 provides a concise example. Suppose the building blocks in the genetic pool consist of the Sharpe ratio calculated over 20-day and 50-day periods, each with a lag ranging from 1 to 5 days.

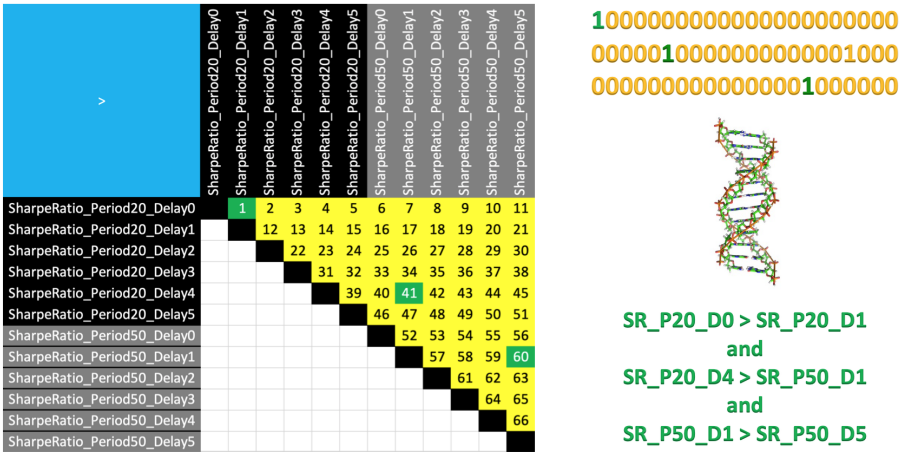


Figure 6: Example of decision matrix with Sharpe Ratio as atomic element

Examples of rules could include:

- Rule 1: Compares the Sharpe ratio calculated over 20 periods (today) with the one calculated over the same 20 periods (20) from the previous day (yesterday).
- Rule 41: Compares the Sharpe ratio calculated over 20 periods with a 4-day delay, relative to the Sharpe ratio calculated over 50 periods, with a 1-day delay.
- Rule 60: Compares the Sharpe ratio calculated over 50 periods (with a 1-day delay) with the Sharpe ratio calculated over the same 50 periods, but 5 days ago.

When all three genes (rules) are activated, the asset is included in the portfolio. This combination of rules generates a single chromosome, composed of three genes, each representing an elementary rule. Each gene contains a comparison operation based on a specific indicator calculated over a defined time period, with a defined delay. In this case, the genetic pool is limited exclusively to the Sharpe ratio, but the rules can be extended to other indicators and combinations of genes depending on the model's optimization objectives.

The rules can be combined in many ways, generating a wide range of solutions, each of which could be an individual in the evolutionary process. As in biological evolution models, the genetic algorithm will select the most performant combinations of these rules, generating new combinations of genes and trying to optimize the objective function, which in our case can be the improvement of the portfolio in terms of return, risk, and other performance indicators.

In this way, the rule matrix becomes a crucial tool for representing the possible combinations of logical operations, which, when applied to historical data, lead to the generation of optimal solutions for trading and portfolio management.

Construction of Rules for the Genetic Algorithm

More practically, defining the rules from which the genetic algorithm can draw requires adopting a specific approach for managing a basket of assets—a context significantly more complex than the single-asset case discussed in the previous chapter. The construction of rules in this scenario requires meticulous selection and design, so that each rule can operate effectively on a broad set of assets with different characteristics.

The first step involves generating a set of operational rules, each describing a logical condition applicable to one or more assets. The rules are based on a variety of indicators, such as percentage changes, moving averages, and other performance metrics. These tools are used to identify specific market conditions that suggest an opportunity to take trading actions, such as buying or selling assets. For example, a rule might involve a rolling sum of the percentage changes of the closing prices, considering only positive changes. If the cumulative change over the last 5 days is positive, the asset is considered eligible for inclusion in the portfolio. Another rule might compare two averages of percentage changes calculated over different time windows, activating when the short-term average exceeds the long-term one, suggesting a potential bullish trend. An additional rule could involve comparing a specific metric, such as the Sharpe ratio, calculated over different periods or the same period but with different delays, in order to identify favorable trading conditions.

These are just a few examples of the possible rules that can be generated for managing a diversified portfolio. Figure 7 shows a selection of rules that can be implemented.

Gene	Formula
Rule_1	$\sum_{n-5}^{n} \text{Close\%} > 0$
Rule_2	$\sum_{n-5}^{n} \text{Close\%} > \sum_{n-10}^{n-1} \text{Close\%}$
Rule_3	$\frac{\sum_{n-19}^{n} \text{Close\%}}{20} > 0$
Rule_4	$\frac{\sum_{n-9}^{n} \text{Close\%}}{10} > \frac{\sum_{n-5}^{n-2} \text{Close\%}}{4}$

Gene	Formula
Rule_5	$SharpeRatio(Delta\%) > 1$
Rule_6	$\text{Kestner}Ratio(Delta\%) > 0.4$
Rule_7	$ProfitFactor(Delta\%) > 1$
Rule_8	$ProfitFactor_P20_D0 > ProfitFactor_P50_D5$

Figure 7: Example of gene classes for building the decision matrix

The set of generated rules can be extremely varied, with combinations that allow exploring different operational modes and optimizing the decision-making process. However, it is essential that the designer ensures that these rules are logically consistent within the context of a basket of assets, which includes diversified financial instruments and requires a cohesive and well-structured approach. The construction of the rules is, therefore, not just a mathematical exercise but requires a deep understanding of the overall portfolio behavior. The designer must ensure that the selected rules are capable of identifying complementary trading opportunities so that the overall model responds to market dynamics in a balanced and robust way. Only with careful design, taking into account the specific characteristics of the asset basket, can the rules generate effective and resilient trading strategies.

Recap and Code Structure: Genetic Asset Management Architecture

To understand how to structure the code and generate the Genetic Asset Management Architecture, we begin by recall the logic applied to a single asset, called Asset A (e.g., Apple). This architecture can be easily extended to a basket of assets, but we will start with the basic representation. Figure 8 will help guide you through the explanation that follows:

DateTime	Asset A
Day_1	$Price_1$
Day_2	$Price_2$
Day_3	$Price_3$
...	...
Day_{n-2}	$Price_{n-2}$
Day_{n-1}	$Price_{n-1}$
Day_n	$Price_n$

DateTime	$Rule_1$	$Rule_2$	$Rule_3$	$Rule_4$	$Rule_5$
Day_1	True	False	True	False	False
Day_2	False	False	False	False	True
Day_3	False	False	True	False	False
...	...	...	...	...	...
Day_{n-2}	True	True	False	False	False
Day_{n-1}	False	True	True	False	True
Day_n	False	False	True	True	False

DateTime	dna_1	dna_2	...	dna_{n-1}	dna_n
Day_1	False	False	...	True	False
Day_2	False	False	...	False	True
Day_3	False	True	...	False	True
...	...	...	...	...	...
Day_{n-2}	True	True	...	True	False
Day_{n-1}	True	True	...	True	False
Day_n	False	False	...	False	True

Population	formula
dna_1	Rule 1 and Rule 3
dna_2	Rule 1 and Rule 3
...	...
dna_{n-1}	Rule 2 and Rule 3 and Rule 4
dna_n	Rule 1 and Rule 3 and Rule 5

Figure 8: Single-asset genetic evolution framework

Defining Rules for a Single Asset

Looking at the Figure 8, on the left, we have the initial data: Date/Time and the prices related to Asset A. From this data, a series of operational rules (referred to as *Rules*) are defined (in the illustrative example, from rule 1 to rule 5). Each rule generates a boolean column representing the result of a logical condition applied to the asset. For example:

- Rule 1: True, False, False, True... up to day n.
- Rule 2: Another boolean chain reflecting a different condition, and so on for all the rules.

This set of rules constitutes our genetic pool, a kind of logical "bucket" from which the genetic algorithm will draw.

Generating the DNA for the Asset

From the rules matrix, we build a population composed of n DNA (e.g., DNA1, DNA2, ..., DNAn). Each DNA represents a logical combination of selected rules (initially chosen randomly). In our example:

- DNA1 might be: Rule 1 AND Rule 3 $\rightarrow$ A boolean column derived from the logical combination of the two rules.
- DNA2 might combine other rules, e.g., Rule 2 AND Rule 5, and so on.

This structure is represented in a new dataframe, where each column is a DNA and each row corresponds to a timestamp.

Extending to a Basket of Assets

Up to this point, we have worked with a single asset. Now we extend the process to a basket. Let's imagine we have three assets: Asset A, Asset C, and Asset D. For each asset there will be:

- **Rules Matrix**: We generate asset-specific rules, e.g., Rule 1 might have a different impact on Asset A compared to Asset C due to the unique characteristics of each asset.
- **DNA Matrix**: We combine the rules for each asset, creating specific DNA. It is important to note that the same DNA (e.g., DNA1) might have a different boolean configuration for different assets, as it depends on the logic applied to the individual asset's rules.

Overall Representation

The final result is a matrix of DNA for each asset, which contains:

- **Time Index (Date/Time)**
- **DNA Columns**: Each column represents a DNA, derived from logical combinations of the rules specific to the asset.

This structure, which appears as a series of "intermediate matrices," is simply an organized system of dataframes, one for each asset and can be described as in Figure 9. The next phase involves consolidating these matrices and defining operational choices, i.e., which assets to "turn on" or "turn off" based on the DNA configurations.

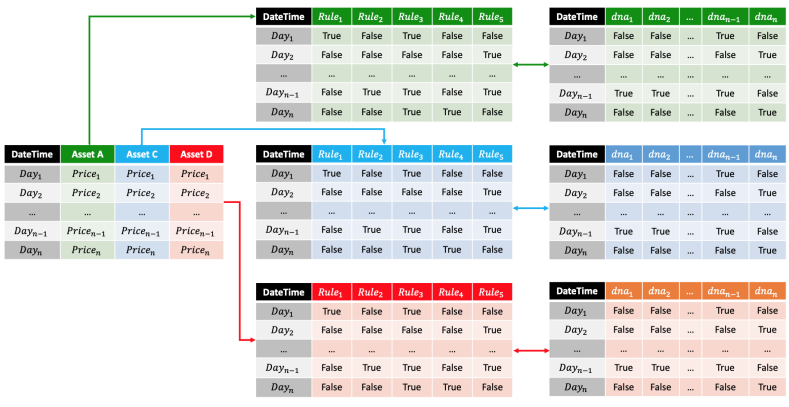


Figure 9: Genetic Asset Management framework

5.3 Validation Architecture

For the development and evaluation of the Genetic Asset Management framework, we adopted a walk-forward rolling validation approach, designed to create a dynamic genetic rule capable of adapting to evolving market conditions. This method, illustrated in Figure 10, builds upon a classic in-sample and out-of-sample framework while introducing the possibility of generating time-varying genetic rules.

The validation process begins by selecting an initial in-sample period, during which the genetic algorithm identifies the optimal solution. In this context, the solution represents an investment rule that determines, at the moment of a trigger, which assets to include in the portfolio and which to exclude. The identified rule is then applied to the subsequent out-of-sample period, spanning from the current trigger to the next. Following this, the in-sample window is shifted forward by a duration equivalent to the out-of-sample period, and the process is repeated iteratively.

This rolling methodology simulates a forward-looking scenario, avoiding the pitfalls of standard backtesting, which often lacks the dynamic recalibration necessary for real-world market applications. To further enhance robustness, we introduced the instability factor within the walk-forward analysis. This factor, detailed in Section 4.7, ensures that the genetic algorithm avoids overfitting by penalizing overly optimized solutions during the in-sample period.

By combining the walk-forward rolling approach with the instability factor, the validation process provides a realistic simulation of market dynamics while fostering generalizable and adaptive investment strategies.

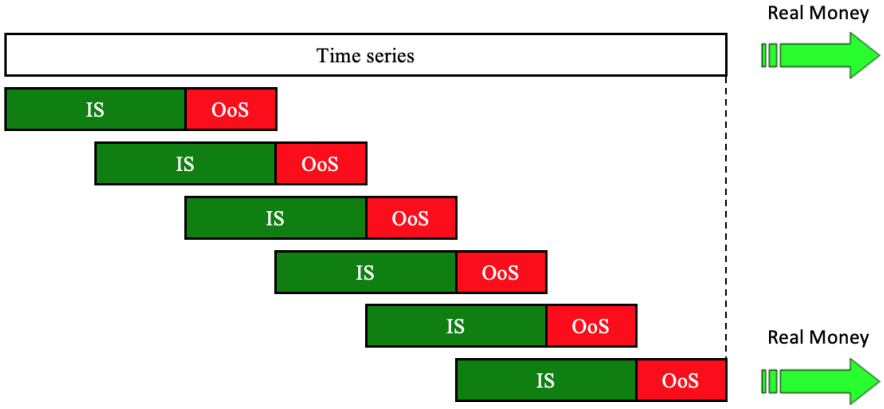


Figure 10: Rolling Walk Forward

6 Case Study

This section provides a comprehensive description of the framework used in this case study, followed by the presentation of the simulation results, where the performance of the algorithm is compared to the benchmark to assess its effectiveness and potential for real-world applicability.

6.1 Case Study: Framework

In this section, we introduce the key parameters selected for the simulation, which are crucial in shaping the algorithm's behavior and influencing the optimization results.

Basket of Assets

The portfolio is built from the 100 largest market capitalization stocks in the S&P500 index. This selection is based on current market data and does not account for survivorship bias. To maintain the theoretical integrity of the study, an equal-weighted portfolio of these 100 stocks is used as the benchmark for comparison. This ensures that the results are unaffected by survivorship bias within the scope of this research. Therefore, this approach allows for demonstrating the effectiveness of the proposed model in identifying an active management strategy that outperforms the respective benchmark. For real-world applications, however, we recommend using a dataset that has been adjusted for survivorship bias.

Allocation Strategy

The portfolio follows a Static Equal-Weighted Allocation strategy, meaning that each active asset receives an equal share of the total capital, represented by $\frac{1}{n}$, where n is the number of assets in the basket. Inactive assets, on the other hand, receive zero allocation. This strategy simplifies the decision-making process while ensuring that each asset's influence on the portfolio is equal, regardless of its individual volatility or market cap.

With this type of allocation, however, the entire capital is not always fully invested, unlike a traditional buy-and-hold strategy. While this results in a lower overall return, it addresses the need to construct a portfolio that mitigates risk. As we will see later, periods where the invested capital is at its minimum often coincide with times of stress in the stock market. This highlights how the genetic system intrinsically adopts a risk-on/risk-off approach, effectively providing a form of risk hedging. Of course, if the objective is to maintain a 100% invested portfolio at all times, the allocation method can be adjusted accordingly, while the proposed framework remains unchanged.

Rebalance

The strategy operates on a weekly trigger, meaning that portfolio decisions and evaluations are updated once a week. This regular adjustment interval allows the portfolio to remain responsive to ongoing market changes, improving its adaptability to fluctuating conditions.

Training Mode

The genetic algorithm is trained using a walk forward rolling window approach, as described in Section 5.3. This method ensures that the model remains adaptive to changing market conditions, using the most up-to-date information while mitigating the risks of overfitting to historical data.

- **In-Sample (IS) Period**: A 10-year historical window is used to train the genetic algorithm in each iteration.
- **Out-of-Sample (OoS) Period**: The trained genetic rules are then applied to a subsequent 13-week period to evaluate the model's performance and robustness.

For example, the first training window spans from January 1, 2000, to January 1, 2010. During this period, the genetic algorithm identifies the optimal asset allocation rules. These rules are then applied to a 13-week out-of-sample period from January 2, 2010, to April 2, 2010. Each week within the OoS period, the algorithm dynamically selects the assets to include or exclude from the portfolio based on the current market data. After 13 weeks, the in-sample window shifts forward by 13 weeks, maintaining a 10-year rolling window, and the process is repeated. This ensures the strategy remains adaptable to changing market dynamics.

Fitness Function

The Kestner ratio is chosen as the fitness function for this study. This ratio is designed to assess strategies based on their ability to achieve high returns relative to risk, taking into account both the magnitude and frequency of drawdowns. In doing so, it emphasizes risk-adjusted returns, prioritizing strategies that maintain consistent performance with minimal downside risk. Naturally, the choice of the fitness function guides the genetic evolution towards specific families of solutions, aligning the portfolio's behavior with the desired risk-return profile.

Population and Generations

- **Population Size**: 100 individuals
- **Number of Generations**: 10

The product between individuals and generations is an arbitrary choice, depending on several factors. The most important of these is the composition of the genetic pool. Not so much its overall size – which in the case study presented here consists of 161,600 rules – but rather the variety of the atomic elements that make it up. In the present case, despite the large number of rules, they are derived from a few basic elements, such as prices and volumes. As a result, the generated rules differ from each other, but share the same fundamental nature. In such a scenario, it is preferable to limit the extent of genetic evolution, as excessive optimization would focus on a narrow set of ideas. In contrast, by employing a more diverse genetic pool, for example, incorporating alternative data

such as macroeconomic indicators, sentiment, or AI-based information, it is possible to expand the product of population size and generations (in the standard case: 1,000 individuals for 100 generations).

Evolution parameters

- **DNA SIZE = 3**: Each evaluated solution consists of 3 rules combined using a logical AND operation.
- **BEST DNA RATIO = 0.3**: At each generation, the top 30% of the population, representing the best-performing solutions, are selected for further evolution.
- **CROSS DNA RATIO = 0.3**: 30% of the new population is formed through crossover between the best-performing individuals. The remaining individuals are generated through mutation.
- **MUTATION PROB = 0.10**: There is a 10% chance of mutation for each trading rule (gene) in the top 30% of the population.

Instability Factor

An instability factor of 5% is included to discourage overly optimized solutions that may overfit the in-sample data. This factor improves the robustness of the model by preventing the genetic algorithm from exploiting transient patterns or noise that are specific to the training data, as discussed in Section 4.7.

Transaction Costs

Transaction costs are modeled at 0.1% per trade (0.001 per side), reflecting realistic trading conditions. These costs are accounted for to ensure that the results accurately reflect the net performance of the portfolio, incorporating the practical challenges and frictions inherent in real-world trading environments.

6.2 Case Study: Analysis of Evolutionary Process

In this section, we present an analysis of the evolutionary process, which, as previously described, is embedded within an enhanced walk-forward rolling framework incorporating the instability factor. To illustrate the evolutionary process, we include a selection of snapshots extracted during the genetic evolution.

For instance, Figure 11 compares the first and fifth genetic generations of Walk Forward Cycle 32, corresponding to the training period from 2007-09-22 to 2017-09-22, and the validation period, which ends on 2017-12-22. The red dashed line marks the boundary between the 10-year in-sample period, during which the genetic evolution was conducted with the integration of the instability factor, and the 13-week out-of-sample period used for validation. During this period, the portfolio was rebalanced weekly, with assets bought or sold based on the genetic rule. Afterward, a new trading rule was identified in the subsequent walk-forward cycle. It is evident that the genetic evolution swiftly identified rules that contributed to the enhancement of the fitness function, specifically the Kesten ratio. The increased regularity of the curve offers a clear visual indication of the trading strategy's improved performance across generations.

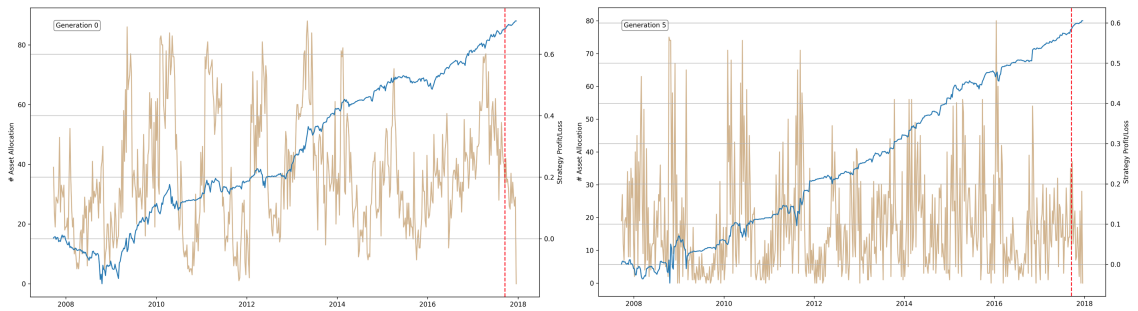


Figure 11: Generations within same WFA cycle

In the pictures, alongside the strategy's $P\&L$, we also report the number of assets held in the portfolio at each rebalancing. This additional information is crucial for understanding the interplay between the genetic rule and the portfolio's structural evolution. Specifically, it provides insights into how the rule dynamically adjusts the portfolio composition, influencing turnover, diversification, and exposure levels. By examining these dynamics in conjunction with the $P\&L$, we can better evaluate the trade-off between performance and the stability of the strategy across different market conditions.

In Figure 12, we present the best individual from the final generation of the last walk-forward cycle (Walk Forward Cycle 60), corresponding to the training period from 2014-09-13 to 2024-09-13 and validation up to 2024-11-10. Although intermediate stages are omitted for brevity, it is evident that the genetic evolution process progressively identified better solutions for both the IS and OoS phases.

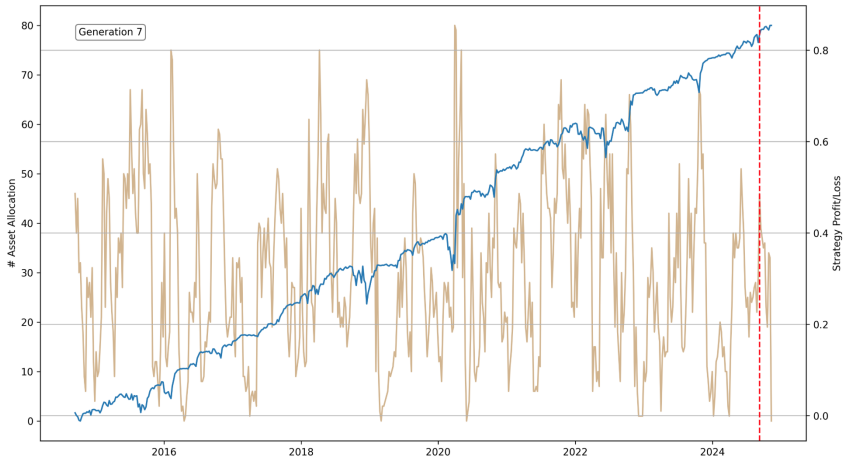


Figure 12: Single-asset genetic evolution

Having obtained all the 13-week out-of-sample segments produced by the genetic evolution process and the walk-forward framework, we can seamlessly combine them to construct the overall OoS equity curve, starting from the first period (2010-04-02) onward. Figure 13 displays the equity curve (without the reinvestment of profits), which represents the concatenation of the best genetic rule identified for each OoS period.
Two key observations emerge:

1. In terms of performance, the genetic portfolio does not outperform the benchmark, a result that is not unexpected given that the evolution was guided by an alternative fitness function. Moreover, the benchmark itself exhibits exceptionally high performance, which further accentuates this outcome. It is important to highlight that the lower returns of the genetic portfolio can also be attributed to the fact that the capital is rarely fully invested. This decision is made automatically by the genetic rule, which adjusts the portfolio's exposure with the goal of regularizing returns. By doing so, the algorithm aims to reduce volatility and smooth performance, potentially at the cost of maximizing short-term gains when compared to the benchmark.

2. Despite the active management of this portfolio being constrained to the basket of assets provided as input—representing the benchmark—it still demonstrates the ability to effectively mitigate market crises during periods such as 2011, 2018, 2020, and 2022. It is important to note that the equity curve we are analyzing is entirely out-of-sample, meaning it reflects the actual results that would have been achieved by trading with new data at each step. Therefore, we are not viewing the equity curve that was subject to optimization, but rather a performance track that showcases the robustness of the strategy in real-world, forward-looking scenarios.

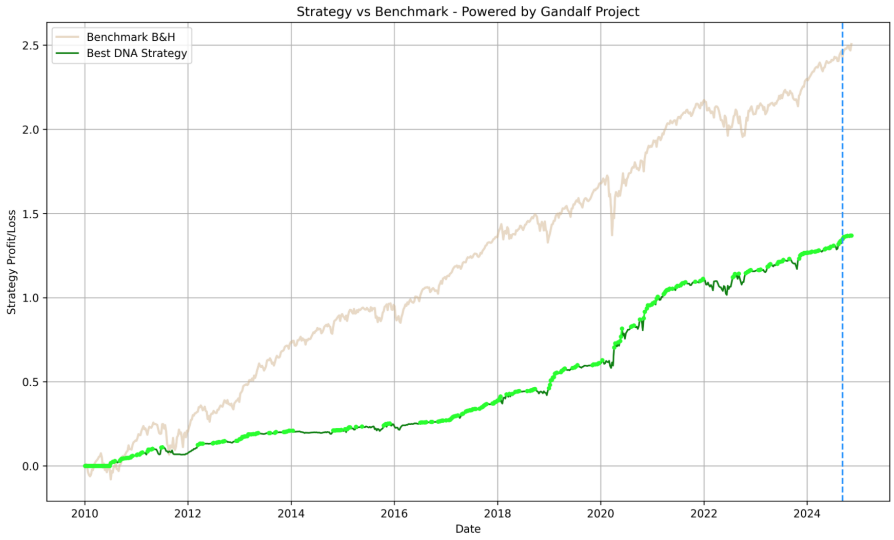


Figure 13: Best DNA strategy vs Benchmark

In Figure 14, the comparison of the equity lines with reinvested profits is presented, matching the volatility of the benchmark (since the benchmark's volatility is much higher than that of the portfolio). As shown in the figure, despite the reduced capital usage, the genetic active strategy outperforms the benchmark. This comparison with adjusted volatility provides a fairer evaluation, as it isolates the strategic decisions from the inherent differences in risk levels between the portfolio and the benchmark. By aligning the volatility, we are able to focus purely on the strategy's ability to generate superior returns adjusted for risk. If the benchmark has higher volatility than the portfolio, a direct comparison of returns might favor the benchmark simply because it carries higher risk. Therefore, adjusting the volatility ensures that both the portfolio and the benchmark are exposed to similar levels of risk, allowing for a more accurate comparison of their performance. This method highlights whether the strategy is able to add value not just in terms of absolute returns, but also in its ability to manage risk, demonstrating if it can produce a more robust performance without taking on excessive risk.

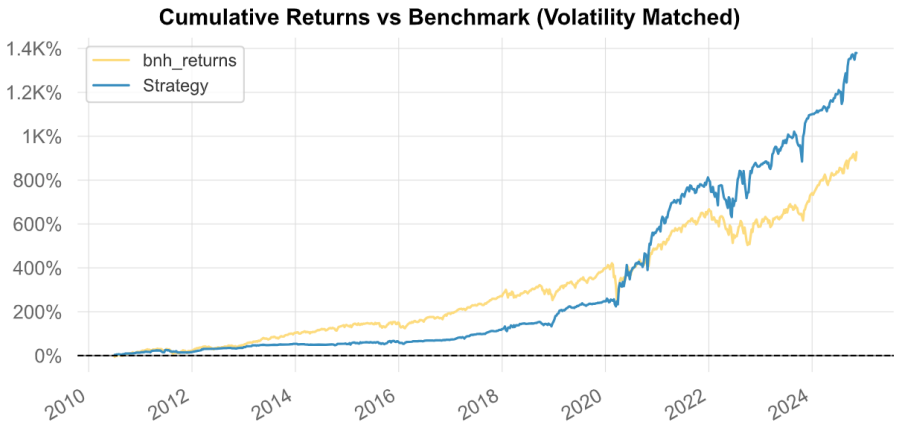


Figure 14: Best individual strategy vs Benchmark (Vol matched)

Figure 15 (each vertical bar denotes an individual 13-week out-of-sample segment) shows the strategy's equity line alongside the number of assets held in the portfolio at each moment. As previously mentioned, this information is essential for thoroughly understanding the impact of the genetic rule on the portfolio structure. Specifically, tracking the number of assets held allows us to analyze the degree of diversification, the stability of positions, and the overall turnover.

These aspects are critical for evaluating the strategy's effectiveness beyond mere performance ($P\&L$), as an excessively concentrated portfolio or one subject to frequent changes could result in higher transaction costs or a less balanced risk exposure. Conversely, a stable and diversified portfolio may indicate greater robustness of the genetic rule in balancing profitability and risk.

Furthermore, jointly monitoring the equity line and the number of assets provides a more com-

prehensive view of the strategy's dynamics allowing for the identification of potential correlations between changes in portfolio composition and variations in strategy performance, offering valuable insights for future optimizations.

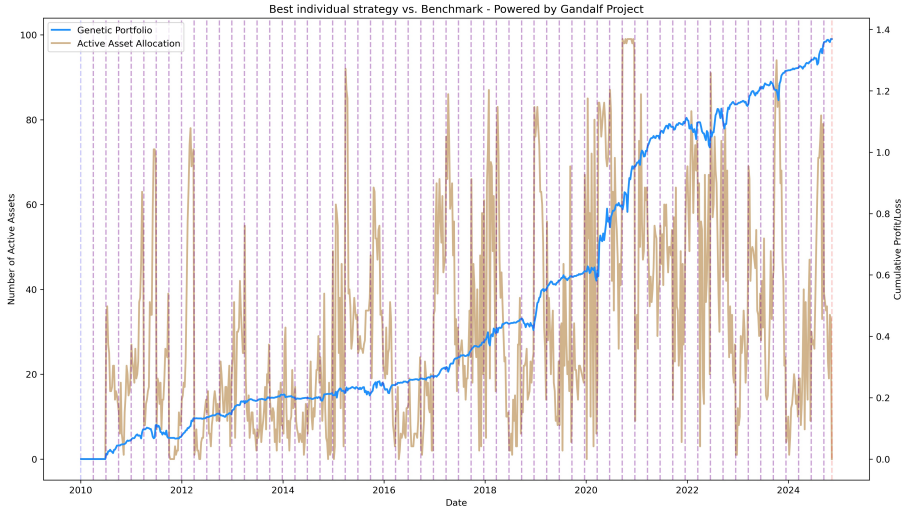


Figure 15: Best DNA strategy and number of selected assets

In Figure 16, we present the heatmap of the returns of the benchmark (left) alongside that of the genetically optimized portfolio (right). This visualization provides an intuitive yet powerful way to assess the effectiveness of the genetic algorithm in stabilizing portfolio returns. The benchmark heatmap reveals areas of heightened volatility, often characterized by abrupt shifts and clustering of high-stress periods, which are markedly subdued in the genetic portfolio's heatmap. This suggests that the evolutionary process effectively redistributes risk and smoothens return patterns, demonstrating its robustness across various market conditions.

Beyond the reduction in volatility, the heatmap comparison also underscores differences in the temporal and spatial concentration of risk. The genetic portfolio tends to exhibit a more balanced risk distribution over time, likely driven by the algorithm's adaptive capacity to adjust allocations dynamically. This adaptability is crucial in real-world applications, where market regimes can shift unpredictably. Overall, the heatmaps provide compelling evidence of the advantages offered by the genetic solution, not only in terms of volatility reduction but also in achieving a more stable and resilient portfolio structure.



Figure 16: Heatmaps benchmark (sx) and Genetic portfolio (dx)

From a quantitative perspective, the results are summarized in Table 1, which highlights key performance metrics, taking into account the reinvestment of profits. The table highlights that, despite the benchmark displaying a higher CAGR (motivations discussed earlier), the GAM model outperforms the benchmark in several key aspects related to risk management and performance stability. Notably, the skew value for the benchmark is negative (-0.42), indicating a distribution of returns that is biased toward lower values, while the GAM model shows a positive skew (2.1), suggesting that the returns are more evenly distributed or tend toward higher values. This has favorable implications for the long-term stability of the model's performance.

Another crucial aspect is the Maximum Drawdown, which measures the largest peak-to-trough loss during the observation period. The benchmark exhibits a significantly higher drawdown (-31.53%) compared to the GAM model, which shows a much more contained drawdown (-9.53%). This reflects the GAM model's superior ability to limit losses during market turbulence, underscoring a more effective risk management approach.

The Recovery Factor, which indicates the speed at which a portfolio recovers losses, is also higher for the GAM model (14.36) compared to the benchmark (8.03). A higher recovery factor implies a quicker recovery from drawdowns, signaling that the GAM model is more efficient at bouncing back from significant losses.

Last but not least, the GAM model shows a beta of almost 0, indicating that it is uncorrelated with the benchmark. This suggests that the GAM model's performance is independent of market fluctuations, and it does not follow the same trends as the benchmark. The genetic portfolio model's zero beta highlights its potential to offer diversification benefits by not being directly affected by market movements. This characteristic can be particularly valuable during periods of high market volatility, as the GAM model is less likely to experience large swings due to broader market trends, further enhancing its role as a stable, risk-managed investment.

In summary, while the benchmark may generate higher returns in the long run, the GAM model distinguishes itself through a considerably stronger risk management profile. With a positive skew, a reduced drawdown and volatility, and a higher recovery factor, the GAM model presents a more attractive option for investors seeking not only stable performance but also stronger protection against fluctuations and capital losses.

Table I: Performance Metrics Comparison

Metric	**Benchmark**	**GAM**
CAGR	**11.85%**	6.57%
Volatility (ann.)	36.38%	**17.23%**
Skew	-0.42	**2.41**
Sharpe Ratio	2.13	**2.44**
Sortino Ratio	3.5	**5.51**
Maximum Drawdown	-31.53%	**-9.53%**
Avg. Drawdown	-3.05%	**-1.17%**
Daily Value-at-Risk	-3.43%	**-1.6%**
Recovery Factor	8.03	**14.36**
Serenity Index	10.85	**31.57**
Beta	–	-0.07

7 Population based GAM

In this section, we propose an alternative and innovative approach that fully leverages the population-based nature of genetic algorithms. In previous versions, whether in the single-asset case or within the context of genetic asset management, the process involved selecting a single solution, specifically the best individual identified at the end of the genetic evolution. This choice, made possible by mitigating the risk of overfitting through the use of the instability factor, ensured a certain methodological robustness. However, while effective, this approach does not fully exploit the adaptive and diversifying potential inherent in genetic algorithms.

A distinctive property of genetic algorithms is their ability to generate and evolve a population of solutions, unlike single-point search algorithms such as simulated annealing. Based on this observation, we questioned whether it would be possible to leverage the entire population of solutions generated in the final generation of the evolutionary process, rather than limiting ourselves to selecting a single individual. Such an approach is particularly advantageous in contexts characterized by high uncertainty and non-stationarity, such as financial markets. In this domain, adopting a population of solutions could offer the following benefits:

- Reduce the risk of selecting a solution overfitted to historical data.
- Eliminate the implicit assumption, present in traditional approaches, that the best solution from the past will continue to be optimal in the near future.

The implementation of this conceptual modification is extremely straightforward and does not require any structural changes to the previously described framework. The primary difference

lies in how the returns for each out-of-sample period are aggregated: instead of relying on a single solution, these are calculated as an equal-weighted portfolio composed of all the solutions generated in the last generation of the previous in-sample period. This represents the standard version of the approach, yet it opens the door to numerous opportunities for optimization and customization to further enhance the model's performance.

In Figure 17, we present the equity line resulting from the aggregation of all out-of-sample periods, with returns calculated as described earlier. This visual representation allows for the evaluation of the performance of the proposed approach, highlighting its effectiveness in leveraging the full range of solutions identified during the genetic evolution process.

In this case study, the results of the population-based strategy are slightly inferior because, by incorporating all solutions at each rebalancing, the portfolio inevitably includes rules that were not part of the best-performing strategies (as seen in the standard version). As a result, this introduces a "dilution" effect on the optimal solution. However, this approach facilitates greater diversification at the strategy level, which could offer significant advantages, particularly with a much larger genetic pool than the one used in this example. While this diversification may yield better results in such a scenario, it is crucial to test this hypothesis. The effectiveness of this approach could vary depending on both the size of the genetic pool and the basket of assets utilized, suggesting that the benefits of diversification may not be absolute and could be influenced by various factors.

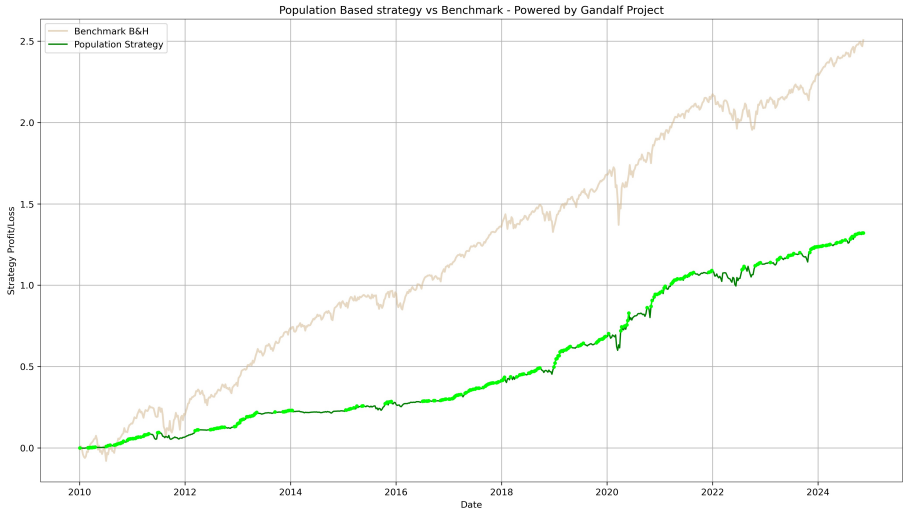


Figure 17: GAM population based framework

Ultimately, the approach outlined here represents a significant advancement in the methodology of genetic algorithms applied to portfolio management. Leveraging the entire population of solutions at the end of the evolutionary process offers the potential to fully capitalize on the population-based nature of genetic algorithms, with the promise of greater resilience and adaptability to the identified solutions. While these benefits remain potential, they highlight a paradigm shift that, coupled with the simplicity of implementation within the existing framework, suggests how small conceptual adjustments could lead to meaningful improvements in robustness and operational efficiency. In addition to the potential for reducing the risk of overfitting and eliminating potentially problematic assumptions about future market behavior, this approach lays the groundwork for further developments. The possibility of integrating advanced techniques, such as dynamic weighting or combination models based on ensemble learning, holds the potential to open new horizons for optimizing and customizing the method, making it even more adaptable to complex and volatile contexts.

8 Conclusion

In this paper, we explored the application of genetic algorithms in portfolio management, with a particular focus on their adaptation to single-asset scenarios and, more importantly, in the context of Genetic Asset Management. The initial exploration of GAs in single-asset management highlighted their capacity to evolve optimized solutions, but it was within the GAM framework that the true potential of these algorithms was realized. The innovative aspect of GAM, as presented here,

lies in its ability to handle multiple assets and strategies simultaneously, enabling the identification of optimal solutions across a diverse set of market conditions.

Additionally, we introduced a potential enhancement by leveraging the population-based nature of genetic algorithms, where all solutions from the final generation of the in-sample period are aggregated into an equal-weighted portfolio. While this population-based approach offers promising advantages in terms of diversification and adaptability, its benefits are not guaranteed and depend heavily on the size and quality of the genetic pool, as well as the specific assets under consideration. Though it holds potential for reducing overfitting and enhancing resilience in non-stationary environments, further empirical testing is needed to validate its effectiveness across different contexts and datasets.

Ultimately, this research advances the application of genetic algorithms in financial portfolio management by demonstrating the flexibility and power of the GAM framework. By building on the inherent strengths of GAs and focusing on robust, adaptable strategies, this approach offers a meaningful contribution to quantitative portfolio management. While the population-based innovation introduces intriguing possibilities, the GAM base remains a solid, reliable foundation for navigating the complexities of financial markets.

References

[1] M. Trequattrini. "Validation of a trading system: Monte Carlo analysis applied to the IS & OOS architecture", *IFTA Magazine*, [2024].

[2] G. Trombetta. *Strategia di trading con Python: Modelli di analisi quantitativa al servizio dell'investitore*, Hoepli, 2020.

[3] E. Wirsansky. *Hands-On Genetic Algorithms with Python: Apply genetic algorithms to solve real-world AI and machine learning problems*, Packt Publishing, 2024.

[4] M. De Prado. *Advances in Financial Machine Learning*, John Wiley Sons Inc, 2018.

[5] Chun-Wei Tsai and Ming-Chao Chiang. *Handbook of Metaheuristic Algorithms*, Elsevier Science, 2023.