PPC - Portfolio Optimization

Khaled Mili
Maxim Bocquillon
Aurélien Daudin
Samy Yacef
Matéo Lelong

March 2025

Contents

1 Introduction

Portfolio optimization is a core problem in quantitative finance, where the objective is to allocate capital across assets to maximize expected return while minimizing risk. The classical approach, introduced by **Markowitz**, models this as a mean-variance optimization problem. While elegant and tractable, the Markowitz model assumes normally distributed returns and may underestimate tail risk.

To address these limitations, more robust risk measures like **Conditional Value-at-Risk** (CVaR) have been introduced. CVaR captures the expected loss in the worst-case scenarios beyond a given confidence level, making it more suitable for real-world applications involving extreme market movements.

A variety of computational approaches have been proposed to maximize or minimze thos mathematicals models:

- **Genetic Algorithms (GA)**: Population-based metaheuristics inspired by natural selection. GA offers flexibility and robustness in navigating large, complex search spaces but may struggle with convergence guarantees.
- **Constraint Programming with CP-SAT**: A powerful paradigm from operations research, CP-SAT excels at modeling logical constraints and combinatorial structure. It is well-suited for problems with discrete decision variables, offering exact solutions for many classes of problems.
- **Mixed-Integer Nonlinear Programming (MINLP)**: This approach blends the modeling power of nonlinear programming with the ability to handle integer decision variables. MINLP is a natural fit for portfolio problems involving non-linear risk functions and allocation limits, but may suffer from scalability issues.

This paper presents a comparative study of these optimization techniques applied to the CVaR and Sharpe Ratio portfolio optimization problem. We formulate a common problem instance and explore the performance of each method in terms of solution quality, computational time, and scalability. We aim to highlight the trade-offs between expressiveness, optimality, and efficiency in modern portfolio optimization frameworks.

2 Modèles

2.1 Markowitz Model

Before entering the solution of this problem, we will define the problem.

The Sharpe ratio is defined as suit :

$$S(w) = \frac{w^T r - R_f}{\sqrt{w^T \Sigma w}} \tag{1}$$

où :

- w is the column vector of the portfolio weights..
- r is the column evector of the expected returns of the assets..
- Σ is the covariance matrix of the returns.
- R_f is the risk-free rate of return.

We seek to maximize the Sharpe ratio, which comes to solve the following optimization problem :

$$\max_w \quad \frac{w^T r - R_f}{\sqrt{w^T \Sigma w}} \tag{2}$$

sous les contraintes :

$$\sum_{i=1}^{n} w_i = 1 \tag{3}$$

$$0 \leq w_i \leq 0.4, \quad \forall i \tag{4}$$

We are dealing with a quadratic problem because the portfolio variance, $^2_p = w^T w$, is a quadratic function of the weights w. Moreover, the optimization of the Sharpe ratio involves a quadratic programming problem with linear constraints, which justifies the use of the SLSQP method for its resolution.

This optimization of the Sharpe ratio amounts to minimizing the standard deviation of the portfolio returns, which corresponds to the measure of risk in our case (according to the Markowitz model, volatility corresponds to risk), or to maximizing the difference between the risk-free rate of return and that of our portfolio.

2.2 CVaR Model

2.2.1 VaR

To correctly define the CVaR we need to explicit the VaR beforehand. **Value-at-Risk (VaR)** is a statistical measure that estimates the maximum potential loss of a portfolio over a given time horizon for a given confidence level. Formally,

for a confidence level $\alpha \in (0,1)$, the VaR of a portfolio is the smallest number l such that the probability that the loss L exceeds l is at most $1-\alpha$:

$$\text{VaR}_\alpha(L) = \inf l \in \mathbb{R} \mid \mathbb{P}(L \leq l) \geq \alpha \tag{5}$$

Let $\alpha \in (0,1)$ be the confidence level. The Value at Risk (VaR) at level α of a portfolio with loss distribution L is defined by:

$$\text{VaR}_\alpha(L) = \inf\{\ell \in \mathbb{R} : P(L \leq \ell) \geq \alpha\}$$

However, VaR has limitations: it does not provide information about the magnitude of losses beyond the VaR threshold. Therefore, Conditional Value-at-Risk (CVaR) was introduced.

Conditional Value-at-Risk (CVaR), also called **Expected Shortfall**, measures the expected loss given that the loss is beyond the VaR level. In other words, it gives the average of the worst α-percent losses. Formally, for a confidence level α, the CVaR is defined as:

$$\text{CVaR}_\alpha(L) = \mathbb{E}[L \mid L \geq \text{VaR}_\alpha(L)] \tag{6}$$

CVaR is a coherent risk measure, meaning it satisfies properties like subadditivity, convexity, monotonicity, and translation invariance. These properties make CVaR particularly suitable for optimization.

In portfolio optimization, we aim to minimize the CVaR of the portfolio loss, subject to constraints such as asset allocations.

The optimization problem can be formulated as follows using Rockafellar-Uryasev formula:

$$\min_{w,\zeta} \quad \zeta + \frac{1}{1-\alpha}\mathbb{E}[(L(w) - \zeta)^+] \tag{7}$$

Subject to:

$$\sum_{i=1}^{n} w_i = 1 \qquad\qquad 0 \leq w_i \leq 0.4, \quad \forall i \tag{8}$$

where:

- w is the vector of portfolio weights.
- ζ is an auxiliary variable approximating the VaR.
- $L(w)$ is the loss distribution of the portfolio.
- $(x)^+ = \max(x, 0)$ denotes the positive part of x.

This formulation can be transformed into a non-linear program by introducing auxiliary variables for each scenario, making the CVaR optimization tractable even for large problems.

3 Implementation

3.1 Genetic Algorithm

Genetic Algorithm is a class of evolutionary algorithm inspired by the process of natural selection. It is commonly used for optimization of any kind of measurable function using biological operators including: Initialization, Selection, Crossover, Mutation

In our implementation, we use genetic algorithms to solve the portfolio optimization problem, specifically finding the optimal weights for assets in a portfolio to maximize return, minimize risk, or optimize risk-adjusted returns (Sharpe ratio).

3.1.1 Components of Genetic Algorithms

- **Gene**: A single asset weight in our portfolio (e.g., the weight of Apple stock)
- **Chromosome/Individual**: A complete solution representing one potential portfolio (a vector of weights)
- **Population**: A collection of potential solutions (multiple different weight allocations). In our case, it is just the weights applied on each asset.
- **Fitness Function**: Evaluates how "good" a solution is. For us, Sharpe ratio, portfolio return ...
- **Generation**: One iteration of the genetic algorithm

3.1.2 Key Operators

- **Selection:**
  - **Best Selection**: Taking the best individual only at each generation
  - **Tournament Selection**: Randomly select subgroups and choose the best from each
  - **Rank Selection**: Selection based on the rank of individuals rather than absolute fitness values
- **Crossover:**
  - **Convex Crossover**: Genes are generated using a convex combination of parent's weights while abiding to the constraints: $x = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n$
  - **Arithmetic Crossover**: Special case of convex, taking weights from 2 parents
- **Mutation:**

– **Gaussian Mutation**: $x' = x + \mathcal{N}(0, \sigma)$
– **Directional Mutation**: $x' = x + \sigma_1 d + \sigma_2 \mathcal{N}(0, I)$

3.1.3 Genetic Algorithm Workflow

Genetic Algorithm for Portfolio Optimization [1] **Initialize** population with random solutions or specialized initialization termination condition not met **Evaluate** fitness of each individual **Select** parents based on fitness **Create** offspring through crossover **Apply** mutation to offspring **Replace** old population with new population **Check** for early stopping conditions **Return** best solution found

3.1.4 Implementation Details

We support several initialization strategies:

- Random uniform weights
- Equal weights
- Random with constraints (e.g., sum of weights = 1)

Our implementation includes visualization tools to analyze:

- Weight distribution across generations
- Fitness evolution over time
- Portfolio performance metrics

To improve efficiency, we implement early stopping when fitness improvement falls below a threshold:

```
if np.abs(current_best_fitness - best_fitness) < termination.epsilon:
    logger.warning(f"Early stopping at generation {ga_instance.generations_completed}")
    return "stop"
```

3.1.5 Conclusion

The genetic algorithm approach offers several advantages for portfolio optimization:

- Ability to handle non-convex objective functions
- Flexibility to incorporate various constraints
- Parallelizable computation
- Adaptable to different market conditions

3.2 CP-SAT

CP-SAT is a versatile portfolio solver, centered around a Lazy Clause Generation (LCG) based Constraint Programming Solver, although it encompasses a broader spectrum of technologies.

In its role as a portfolio solver, CP-SAT concurrently executes a multitude of diverse algorithms and strategies, each possessing unique strengths and weaknesses. These elements operate largely independently but engage in information exchange, sharing progress when better solutions emerge or tighter bounds become available.

While this may initially appear as an inefficient approach due to potential redundancy, it proves highly effective in practice. The rationale behind this lies in the inherent challenge of predicting which algorithm is best suited to solve a given problem (No Free Lunch Theorem). Thus, the pragmatic strategy involves running various approaches in parallel, with the hope that one will effectively address the problem at hand. Note that you can also specify which algorithms should be used if you already know which strategies are promising or futile.

In contrast, branch- and cut-based mixed integer programming (MIP) solvers like Gurobi implement a more efficient partitioning of the search space to reduce redundancy. However, they specialize in a particular strategy, which may not always be the optimal choice, although it frequently proves to be.

CP-SAT employs branch-and-cut techniques, including linear relaxations and cutting planes, as part of its toolkit. Models that can be efficiently addressed by an MIP solver are typically a good match for CP-SAT as well. Nevertheless, CP-SAT's central focus is the implementation of Lazy Clause Generation, harnessing SAT-solvers rather than relying primarily on linear relaxations. As a result, CP-SAT may exhibit somewhat reduced performance when faced with MIP problems compared to dedicated MIP solvers. However, it gains a distinct advantage when dealing with problems laden with intricate logical constraints.

3.2.1 Lazy Clause Generation

The concept behind Lazy Clause Generation involves the (incremental) transformation of the problem into a SAT formula, subsequently employing a SAT solver to seek a solution (or prove bounds by infeasibility). To mitigate the impracticality of a straightforward conversion, Lazy Clause Generation leverages an abundance of lazy variables and clauses.

Notably, the Cook-Levin Theorem attests that any problem within the realm of NP can be translated into an SAT formula. Optimization, in theory, could be achieved through a simple binary search. However, while theoretically sound, this approach lacks efficiency. CP-SAT employs a more refined encoding scheme to tackle optimization problems more effectively.

3.2.2 What Happens in CP-SAT During `solver.solve(model)`?

Model Loading and Verification

- The model is read from its protobuf representation.
- The model is verified for correctness.

Preprocessing (multiple iterations, controlled by `max_presolve_iterations`)

- **Presolve (domain reduction)**:
  - Simplifies variable domains to reduce problem size.
  - Relevant resources include videos on SAT/MaxSAT preprocessing and MIP presolving papers.
- **Expansion of higher-level constraints**:
  - Transforms user-friendly constraints into lower-level ones that CP-SAT can propagate efficiently.
  - Similar to the FlatZinc flattening process.
- **Detection of equivalent variables and affine relations**.
- **Substitution with canonical representations**.
- **Variable probing**.

Loading and Relaxation

- The preprocessed model is loaded into the solver.
- Linear relaxations are created.

Solution Search

- The solver searches for solutions until bounds converge or stopping criteria (e.g., time limit) are met.
- Several subsolvers, each with different strategies, are run in parallel:
  - Linearized models
  - Aggressive restarts
  - Focus on upper or lower bounds

First Solution Search and Local Search

- "First solution searchers" attempt to find an initial feasible solution.
- If found, local search heuristics such as Large Neighborhood Search (LNS) are applied using a round robin approach:
  - Copy the model and select a previous solution.
  - Remove a subset of variables and explore their neighborhood.
  - Fix other variables and presolve the model.
  - Solve the simplified model quickly and check for better solutions.

Solution Transformation

- Final solution is transformed back to the original model format.

3.2.3 Use of Linear Programming Techniques

As already mentioned, CP-SAT utilizes the (dual) simplex algorithm and linear relaxations. These are implemented as low-priority propagators and can be executed at every node in the search tree.

- Only a few pivot iterations are performed to save time.
- Deeper search tree and warm-starts allow for near-optimal relaxations.
- At root level, cutting planes (e.g., Gomory cuts) improve the relaxation.
- Used to detect infeasibility, improve bounds, and guide branching decisions.
- RINS (Relaxation Induced Neighborhood Search), a powerful heuristic, uses LP techniques.

3.2.4 Limitations of CP-SAT

While CP-SAT is a powerful solver, it does have certain limitations compared to other approaches:

- May not match the speed of dedicated SAT-solvers on SAT problems.
- May not outperform MIP-solvers on classical MIP problems.
- No support for continuous variables; workarounds are often inefficient.
- Lacks support for lazy constraints or iterative model building.
- Does not support interior point methods (e.g., Barrier), unlike Gurobi.
- May struggle with specific constraints like modulo constraints.

3.2.5 Application to our problem

Overview

- **Objective:** Optimize a portfolio by maximizing a surrogate Sharpe ratio
- **Portfolio Metrics:**
  - Expected Return: Weighted sum of asset returns
  - Risk: Weighted sum of asset volatilities (standard deviations)
- **Key Idea:** Trade off between excess return (after risk-free rate) and risk penalty using integer weights.

Data Acquisition and Preprocessing

- **Data Retrieval:** Historical adjusted-close prices for stocks are fetched.
- **Returns Calculation:**

$$r_{i,t} = \frac{P_{i,t} - P_{i,t-1}}{P_{i,t-1}}$$

  where $P_{i,t}$ is the price of asset i at time t.

- **Statistics:**
  - Expected Return per asset: Mean of daily returns.
  - Risk per asset: Standard deviation of daily returns.
- **Scaling:**
  - A scaling factor (e.g. 10^4) converts floating-point values to integers.
  - This is essential since the CP-SAT solver handles integer coefficients.

CP-SAT Model Fundamentals

- **What is CP-SAT?** CP-SAT is a constraint programming solver that utilizes SAT-based techniques and branch-and-bound search to solve combinatorial problems.
- **Advantages for Portfolio Optimization:**
  - Efficient handling of discrete (integer) decision variables.
  - Flexibility to model complex constraints (e.g., bounds, sum-to-100 conditions).
- **Why it fits our problem:**
  - Portfolio weights are naturally expressed as integer percentages.
  - Linear constraints (e.g., total weight equals 100%) are directly implementable.

Model Formulation and Objective Function

- **Decision Variables:** Each asset i is assigned a weight $w_i \in \mathbb{Z}$ bounded by:

$$w_i \in [\text{lower bound}, \text{upper bound}]$$

- **Portfolio Constraint:**

$$\sum_i w_i = 100$$

- **Scaled Coefficients:**

  - Expected returns are scaled: $\text{scaled\_return}_i = \text{expected return}_i \times 10^4$
  - Risk coefficients are scaled similarly: $\text{scaled\_risk}_i = \text{std}_i \times 10^4$

- **Objective Function:** Formulated as a linear combination capturing the trade-off:

  $$\text{Objective} = \text{risk\_rate\_int} \cdot (\text{portfolio excess return}) - \text{non\_risk\_rate\_int} \cdot (\text{portfolio risk})$$

  where:

  $$\text{portfolio excess return} = \sum_i w_i \cdot \text{scaled\_return}_i - (\text{risk-free rate scaled}) \times 100.$$

Implementation Details

- **Trading-off Risk and Return:** The risk rate is normalized with respect to the maximum risk from the dataset. Two parameters are computed:

  $$\text{risk\_rate\_int} \quad \text{and} \quad \text{non\_risk\_rate\_int} = 10^4 - \text{risk\_rate\_int}$$

  ensuring proper weighting between the return and risk components.

- **Solver Parameters:**
  - A time limit (e.g., 10 seconds) is set.
  - Verbose logging is enabled to monitor the search progress.

- **Recovering the Solution:** After solving, the model converts integer weights back to percentage values (dividing by 100) to report the optimal portfolio.

Conclusion

- The CP-SAT approach efficiently handles the discrete and combinatorial nature of portfolio optimization.
- By scaling continuous financial data to integers and implementing a clear linear trade-off, the model strikes a balance between maximizing excess returns and minimizing risk.
- The framework is flexible for incorporating additional constraints or asset parameters, making it a robust option for financial optimization tasks.

3.2.6 Conclusion

CP-SAT offre une alternative compétitive aux solveurs MIP traditionnels pour les problèmes de portefeuille, notamment grâce à sa gestion native des contraintes discrètes.

3.3 MINLP

Mixed-Integer Nonlinear Programming (MINLP) problems involve the optimization of a nonlinear objective function subject to nonlinear constraints, where some decision variables are constrained to be integers.

In portfolio optimization, MINLP allows for modeling more realistic scenarios such as cardinality constraints (limiting the number of assets selected) and linking binary decisions (selecting an asset) with continuous decisions (allocating weight to an asset).

The MINLP structure includes:

- **Continuous variables**: Represent portfolio weights.
- **Integer (binary) variables**: Indicate whether an asset is selected.
- **Nonlinear constraints**: Capture relationships like weight-to-selection linkage and CVaR computation.

3.3.1 Introduction of IPOPT Solver

IPOPT (Interior Point OPTimizer) is an open-source software package for large-scale nonlinear optimization. It is designed to solve continuous optimization problems with smooth nonlinear functions in the objective and constraints.

In the given implementation, IPOPT is used within the GEKKO optimization suite to solve the continuous relaxations of the MINLP problem.

3.3.2 Application to Our Problem

3.3.3 General Purpose

The code implements a portfolio optimization model that minimizes Conditional Value at Risk (CVaR) while controlling the number of assets selected. CVaR is a risk measure that captures expected losses beyond a specified confidence level, offering a coherent and robust alternative to Value at Risk (VaR).

3.3.4 Constraints

The model includes the following constraints:

- **Weight bounds**: Each stock has specified minimum and maximum allocation limits.
- **Linking constraints**: If an asset is not selected (Step 1: KKT Conditions
- **Cardinality constraint**: The number of selected assets cannot exceed a given maximum.

- **Portfolio weight constraint**: The sum of portfolio weights must equal 1.
- **CVaR constraints**: Ensure that excess losses are correctly modeled above the VaR threshold.

3.3.5 Objective Function

The objective function minimizes the CVaR at a given confidence level, defined as:

$$\text{CVaR} = \text{VaR threshold} + \frac{1}{(1-\alpha)N} \sum_{i=1}^{N} \xi_i \tag{9}$$

where α is the confidence level (e.g., 95%) and ξ_i are the excess losses beyond the VaR threshold.

3.3.6 Steps of MINLP model

Step 1: KKT conditions

The Karush-Kuhn-Tucker (KKT) conditions are necessary conditions for optimality in constrained optimization problems. They state that there exist multipliers λ (for equality constraints) and μ (for inequality constraints) such that:

- **Stationarity:**

$$\nabla f(x^*) + \sum_i \lambda_i \nabla g_i(x^*) + \sum_j \mu_j \nabla h_j(x^*) = 0$$

- **Primal feasibility:**

$$g_i(x^*) = 0, \quad h_j(x^*) \leq 0$$

- **Dual feasibility:**

$$\mu_j \geq 0$$

- **Complementary slackness:**

$$\mu_j h_j(x^*) = 0$$

These conditions define a system that the solution must satisfy.

Step 2: Newton's Method Applied to KKT

Now, we treat the KKT conditions as a large system of nonlinear equations:

$$F(x, \lambda, \mu) = 0$$

Newton's method is used to solve this system. The approach consists of the following steps:

- **Linearize** the system around the current point (x_k, λ_k, μ_k).
- **Solve for the step** $(\Delta x, \Delta\lambda, \Delta\mu)$ by solving the linearized system:

$$\nabla F(x_k, \lambda_k, \mu_k) \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = -F(x_k, \lambda_k, \mu_k)$$

This linear system is known as the KKT matrix system.

Step 3: What is the KKT Matrix?

The system derived from the KKT conditions can be written as:

$$\begin{bmatrix} \nabla^2 L(x_k, \lambda_k, \mu_k) & \nabla g(x_k)^T & \nabla h(x_k)^T \\ \nabla g(x_k) & 0 & 0 \\ \nabla h(x_k) & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta\mu \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) + \nabla g(x_k)^T \lambda_k + \nabla h(x_k)^T \mu_k \\ g(x_k) \\ h(x_k) \end{bmatrix}$$

Where:

$$\nabla^2 L(x_k, \lambda_k, \mu_k)$$

is the Hessian of the Lagrangian $L(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x)$.

Step 4: Search Direction

Once the system for $(\Delta x, \Delta\lambda, \Delta\mu)$ is solved, the Δx vector tells us the direction in which to move the variables x, and how large the step should be.

This Δx is the Newton step for the constrained optimization problem. The optimizer then updates the current point:

$$x_{k+1} = x_k + \alpha \Delta x$$

Where α is a step size determined by a line search or trust region method to ensure that the objective improves and the solution remains feasible.

3.3.7 Limitations of MINLP

- **Scalability**: MINLP problems are computationally intensive. The addition of integer variables drastically increases the search space, making large-scale problems challenging.
- **Solver Sensitivity**: The quality of the solutions depends heavily on the solver used. IPOPT does not handle integer variables natively; BONMIN or other dedicated MINLP solvers are preferred.
- **Local Minima**: Nonlinearities can lead to convergence to local minima, especially without good initial guesses.
- **Complexity in Constraint Design**: Designing tight and feasible constraints is critical. Poorly defined constraints can make the problem infeasible or extremely difficult to solve.