Portfolio Optimization Using Metaheuristic Algorithms

Artificial Intelligence Report

Carlos Sanches, up202107694
João Rebelo, up202107209
Gonçalo Sousa, up202207320

Problem Specification

Portfolio Optimization Tool

Objective: Develop a robust tool to optimize investment portfolios using metaheuristic algorithms.

Key Features:

- Determine optimal asset allocation to maximize returns while minimizing risk.
- Support multiple optimization algorithms (Hill Climbing, Simulated Annealing, Tabu Search, Genetic Algorithm).
- Allow customizable constraints (max/min weights, risk aversion, short selling).
- Provide interactive visualization of optimization results.

Expected Outcome: A user-friendly application that produces efficient portfolios based on historical data from Australian Stock Market (2000-2020).

Related Work

Literature Review

Modern Portfolio Theory (MPT):

- Markowitz, H. (1952). "Portfolio Selection." Journal of Finance, 7(1), 77-91.
- Introduced mean-variance optimization framework for portfolio construction.

Optimal Solution for Convex Optimization:

- Lesniewski, A. (2019). "Optimization Techniques in Finance". Baruch College
  - Convex optimization approaches

Metaheuristic Approaches:

- Chang, T.J., et al. (2000). "Heuristics for cardinality constrained portfolio optimisation." Computers & Operations Research, 27(13), 1271-1302.
- Cura, T. (2009). "Particle swarm optimization approach to portfolio optimization." Nonlinear Analysis: Real World Applications, 10(4), 2396-2406.

Dataset and Implementation:

- Australian Historical Stock Prices Dataset
- Implementation in Python using CVXPY and SciPy

Problem Formulation

Portfolio Optimization as an Optimization Problem
Solution Representation:

- Vector $w = (w_1, w_2, ..., w_n)$ representing portfolio weights for n assets.
- Constraint: $\sum_{i=1}^{n} w_i = 1$.

Objective Function:

$$U(w) = E[R_p] - \lambda \cdot \sigma_p^2$$

Where:

- $E[R_p] = \sum_{i=1}^{n} w_i \cdot \mu_i$ (expected portfolio return).
- $\sigma_p^2 = w^T \Sigma w$ (portfolio variance).
- λ is risk aversion parameter.

Maximize $U(w)$ or Sharpe ratio $\frac{E[R_p]}{\sigma_p}$.

Constraints:

- Weight bounds: $0 \leq w_i \leq max\_weight$ (without short selling).
- With short selling: $-1 \leq w_i \leq 1$.
- Minimum return: $E[R_p] \geq min\_return$.
- Maximum risk: $\sigma_p \leq max\_risk$.

Algorithms and Operators

Metaheuristic Approach

Hill Climbing:

- Neighborhood function: Random perturbation of weights with step size.
- Restart mechanism to avoid local optima.

Simulated Annealing:

- Temperature schedule: Initial_temp $\times$ Cooling_rate^{iteration}.
- Acceptance probability: $e^{\frac{\Delta U}{T}}$ for negative utility changes.

Tabu Search:

- Tabu list: Recent weight configurations to avoid cycling.
- Aspiration criterion: Accept tabu moves if they improve best solution.

Genetic Algorithm:

- Crossover: Weighted average of parent solutions.
- Mutation: Random perturbation with mutation rate.
- Selection: Tournament selection with elite.

Advanced Optimization Algorithms

Particle Swarm Optimization (PSO):

- **Approach:** Particles explore space, remembering personal best and following swarm's best
- **Purpose:** Mimics social behavior of bird flocking/fish schooling
- **Key Feature:** Collective intelligence emerges from simple particle rules

Differential Evolution (DE):

- **Approach:** Creates new solutions by combining existing ones
- **Purpose:** Efficient global optimization through mutation and recombination
- **Operators:**
  - Mutation: $v = x_a + F(x_b - x_c)$ ($F = 0.8$)
  - Crossover: Binomial (CR=0.7)
  - Greedy selection

Exact Quadratic Programming:

- **Approach:** Mathematical solution using convex optimization
- **Purpose:** Finds an optimal solutions that we used for benchmarking across the entire project
- **Methods:**
  - Sharpe ratio maximization (reward/risk)
  - Minimum variance optimization (lowest risk)
  - Utility maximization (balanced trade-off)
- **Key Feature:** Finds true optimum (unlike heuristic methods)

Hill Climbing

Hill Climbing is a local search algorithm that iteratively moves to better solutions in the neighborhood of the current solution. For portfolio optimization, it explores different weight allocations by making small adjustments and accepting improvements.

Background:

- **Step Size:** Controls the magnitude of weight transfers between assets.
- **Random Restarts:** Helps avoid getting trapped in local optima by initiating searches from different starting points.
- **Weight Constraints:** Ensures diversification by limiting maximum allocation to any single asset.

Experiment	Avg. Time (s)	Output					Iterations	Step Size	Restarts
		Avg. Return (%)	Avg. Risk (%)	Sharpe Ratio	Best Return (%)	Worst Return (%)			
Exp 1	0.0657	46.3037 ± 2.4938	24.7531 ± 2.0541	1.8777	52.9971	41.5329	1000	0.05	5
Exp 2	0.0331	35.1951 ± 1.9266	22.9209 ± 1.1987	1.5376	39.5087	31.7818	500	0.05	5
Exp 3	0.1275	47.1868 ± 2.5050	24.5850 ± 1.8387	1.9250	53.0040	42.7707	1000	0.05	10
Exp 4	0.0636	50.2769 ± 3.2067	26.0077 ± 2.4074	1.9420	58.6264	43.2897	1000	0.10	5
Exp 5	0.0677	39.7177 ± 1.9974	23.2045 ± 0.8690	1.7132	46.7238	35.4784	1000	0.02	5

Insights:

- Experiment number 4 had the best return and Sharpe ratio, showing larger step size (0.10) helps explore better solutions.
- The third performed well with more restarts, aiding in escaping local optima.
- (fewer iterations) had the weakest performance, stressing the need for deeper search.
- Smaller step size in Exp 5 limited exploration, lowering returns.

Simulated Annealing

Simulated Annealing (SA) is inspired by the annealing process in metallurgy, where a material is heated and slowly cooled to reduce defects. In optimization, SA probabilistically accepts worse solutions early in the search to escape local optima, gradually reducing this acceptance rate as the "temperature" cools.

Background:

- **Temperature Schedule:** Controls the probability of accepting worse solutions. Higher initial temperatures allow more exploration, while cooling focuses keeping the existing portfolio.
- **Minimum Temperature:** The stopping threshold. Too high risks premature convergence; too low wastes computation.
- **Cooling Rate:** Determines how quickly the temperature decreases. Slower cooling often leads to better solutions but increases computation time.

Experiment	Avg. Time (s)	Output					Iterations	Initial Temp.	Cooling Rate	Min Temperature
		Avg. Return (%)	Avg. Risk (%)	Sharpe Ratio	Best Return (%)	Worst Return (%)				
Exp 1	2.592	51.17 ± 5.43	26.88 ± 3.33	1.91 ± 0.19	63.09	40.20	500	100	0.95	0.01
Exp 2	0.373	32.66 ± 7.35	24.78 ± 3.67	1.31 ± 0.23	49.05	18.91	500	100	0.70	0.01
Exp 3	1.552	24.36 ± 2.75	24.15 ± 2.46	1.01 ± 0.10	32.47	19.11	500	100	0.95	0.10
Exp 4	3.121	52.75 ± 5.47	27.34 ± 4.26	1.95 ± 0.25	63.39	40.90	500	1000	0.95	0.01
Exp 5	5.261	63.50 ± 2.30	31.20 ± 2.67	2.04 ± 0.16	68.91	58.33	500	1000	0.97	0.01
Exp 6	5.462	54.25 ± 5.26	26.93 ± 3.42	2.03 ± 0.20	64.51	541.19	1000	100	0.95	0.01

Insights:

- Higher initial temperatures enable better solutions but increase runtime.
- Cooling rate controls quality: Slower cooling yields superior results (higher returns, better Sharpe ratio) at the cost of computation time.
- Minimum temperature is crucial: Setting it too high leads to premature convergence and poor solutions.
- Iterations improve robustness: More iterations enhance worst-case performance while increasing runtime.

Tabu Search

The fundamental concept behind Tabu Search is to restrict or prohibit previously visited moves or states in search areas that don't offer a better solution.
We can say Hill climbing combined with short-term memory give us Tabu Search.

Background:

- **Tabu Tenure:** The number that limits how long a move stays on the tabu list to prevent revisiting.
- **Aspiration Criteria:** This strategy lets the algorithm escape poor solutions by allowing a move on the tabu list if it improves the current best solution.
- **Frequency Memory:** The move that was chosen frequently without finding a solution is less likely to be chosen again.

Experiment	Avg. Exec. Time (s)	Output					Iterations	Tabu Tenure	Aspiration Critera	Frequency Memory
		Avg yr. Return (%)	Avg yr. Risk (%)	Avg yr. Sharpe Ratio	Best yr. Return (%)	Worst yr. Return (%)				
Exp 1	0.706	67.94 ± 2.24	30.60 ± 2.83	2.23 ± 0.15	73.70	63.71	500	20	Yes	Yes
Exp 2	0.59	67.38 ± 2.29	29.52 ± 2.38	2.29 ± 0.13	72.15	62.87	500	20	Yes	No
Exp 3	0.61	67.30 ± 1.98	30 ± 2.73	2.25 ± 0.15	71.82	61.91	500	20	No	No
Exp 4	0.76	67.57 ± 2.43	30.17 ± 3.05	2.25 ± 0.17	73.29	63.62	500	50	Yes	Yes
Exp 5	0.59	67.56 ± 2.23	30.32 ± 3.04	2.24 ± 0.17	70.92	60.52	500	50	Yes	No
Exp 6	1.65	73.71 ± 1.80	33.07 ± 2.75	2.24 ± 0.14	72.37	69.96	1000	50	Yes	Yes
Exp 7	1.65	73.71 ± 1.80	33.07 ± 2.75	2.24 ± 0.14	72.37	69.96	1000	50	Yes	Yes

Insights:

- In the first 3 experiments we can conclude that Frequency Memory increases execution time, without having a significant impact on the results, within this dataset. This can happen if the dataset does not have many similar bad solutions that need to be avoided.
- In this experiments, we can conclude that this change in the table tenure size does not have much of an impact within this data set, which can happen due to the similarity of the values 20 and 50, or search space does not have many local optimal.
- The better results were when we increased the iterations, which makes sense since it will have more probability to be closer to the global optimal value, due to the algorithm structure it self.

Genetic Algorithm Optimization

Genetic Algorithms (GA) mimic biological evolution to solve optimization problems through selection, crossover and mutation. For portfolio optimization, GA evolves a population of candidate solutions over generations.

Background:

- **Population Size:** Number of candidate solutions (typically 50-100)
- **Crossover (0.8 rate):** Combines parent solutions via weighted average
- **Mutation (0.1 rate):** Introduces small random perturbations
- **Elite Count:** Preserves top 2 solutions each generation

Exp	Time (s)	Performance Metrics					Pop. Size	Gens	Elite Count
		Avg yr. Return (%)	Avg yr. Risk (%)	Avg yr. Sharpe Ratio	Best yr. Return (%)	Worst yr. Return (%)			
GA1	0.070	54.82 ± 5.28	27.44 ± 4.09	2.01 ± 0.18	64.38	43.37	50	100	2
GA2	0.141	54.74 ± 3.30	29.40 ± 3.26	1.87 ± 0.18	60.47	46.07	50	200	2
GA3	0.137	42.53 ± 3.39	26.99 ± 2.88	1.58 ± 0.16	49.07	31.04	100	100	2
GA4	0.284	59.24 ± 2.29	31.27 ± 2.9	1.90 ± 0.14	65.02	54.08	100	200	2

Insights:

- Larger populations lead to worse solutions in this dataset, possibly due to increased complexity, which causes slower convergence.
- Larger generations lead to worse solutions, potentially because more generations increase the risk of premature convergence, where the algorithm becomes stuck in the local optimal rather than exploring the rest of solution space.
- Elite Count helps preserve good solutions but may reduce diversity

Where did we find the Global Optimal Solution?

Purpose of the Exact Solver

- Implements convex optimization methods to guarantee global optimal solutions
- Utilizes quadratic programming through CVXOPT library
- Serves as the benchmark baseline for evaluating metaheuristic approaches
- Quadratic Programming (QP) explained:
  - Optimization technique that minimizes or maximizes curved (quadratic) expressions
  - Ideal for portfolio problems where risk is measured by variance
  - Works with linear constraints like "weights must sum to 1"
  - Finds the mathematically provable best solution when the problem is well-formed

How We Find the Optimal Solution

- Formulates portfolio allocation as a quadratic programming problem and solves using interior-point methods to guarantee global convergence
- Interior-Point Methods explained:
  - Starting from a point inside the feasible region (hence "interior")
  - Creating a path through the inside of the feasible region toward the optimal solution
  - Never touching the boundaries until the final solution is reached
  - Using a barrier function that prevents the algorithm from crossing constraint boundaries
  - Gradually reducing the influence of the barrier as the algorithm progresses toward the optimal solution

Implementation Details

Development Environment: Python 3

- GUI: Tkinter framework with custom scrollable components
- Visualization: Matplotlib for interactive plots and PDF exports

Key Data Structures & Libraries:

- pandas.DataFrame for historical price data and returns calculation
- NumPy arrays for efficient matrix operations
- CVXOPT for exact optimization methods

Implemented Features:

- Six metaheuristic algorithms with customizable parameters
- Multiple interactive visualization modes (portfolio allocation, efficient frontier, etc.)
- Comprehensive testing/benchmarking framework for algorithm comparison
- Export functionality (CSV and PDF reports)
- Advanced constraints management (risk aversion, position limits, short selling)

Future Roadmap of Improvements:

- Alternative Data Integration by incorporation of sentiment analysis from news articles, social media, or other csv files and datasets
- Optimization of algorithm performance through parallel processing techniques
- Break down portfolio risk by sector or geography
- Allow users to create personalized reports with selected metrics

Conclusion

Key Findings

- Metaheuristic algorithms provide effective approximate solutions to the complex portfolio optimization problem.
- Parameter tuning significantly impacts algorithm performance and efficiency.
- Different algorithms excel in different scenarios, offering flexibility for various investor needs.

Practical Applications:

- Our tool enables investors to balance risk-return objectives through user customizable parameters.
- The visualization components help identify optimal asset allocations and understand risk-return trade-offs.
- The framework accommodates both conservative and aggressive investment strategies.

Future Improvements Roadmap:

- Integration of alternative data sources (sentiment analysis, macroeconomic indicators)
- Incorporation of transaction costs and tax considerations into the optimization framework