

Evolutionary algorithms

The optimizing problem.

"I have the following problem:

I have a set of trading strategies. Let's say 5000.

Each of these strategies has N attributes. The number N is the same for each strategy.

Each of these strategies generates a profit value which I denote here with Profit=.

Here's an example:

N: Attribute1, Attribute2, Attribute3 => Profit

1: ProfitFactor=1.2, Stability=0.9, WinLossRatio=0.3 => Profit=5000 Euro

2: ProfitFactor=1.7, Stability=0.7, WinLossRatio=0.2 => Profit=-6000 Euro

3: ProfitFactor=1.3, Stability=0.2, WinLossRatio=4 => Profit=20 Euro

...

The Request:

From this set, I want to select a number of x strategies so that the total profit is maximized.

For example, if I select the 1st and 3rd strategies, I have a total profit of 5020 Euros.

The filtering process:

Consists of metric rules that filter the set.

For example, with the following metric rule, I would filter out the 2nd strategy and thus achieve a profit of 5020 Euros:

ProfitFactor > 1.2 AND ProfitFactor < 1.7

What I'm looking for:

I'm looking for software that can solve this task for me. The software should determine the appropriate metric rules for a given example set."

Question:

I asked ChatGPT I attached the solution from chatGPT. "I'm not sure if ChatGPT has solved the problem for me. Has anyone had experience with this type of optimization?"

I asked Chat-GPT

The Answer from Chat-GPT

```
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

class Strategy {
    double profitFactor;
    double stability;
    double winLossRatio;
    double profit;

    public Strategy(double profitFactor, double stability, double winLossRatio, double profit) {
        this.profitFactor = profitFactor;
        this.stability = stability;
        this.winLossRatio = winLossRatio;
        this.profit = profit;
    }
}

class Rule {
    double minProfitFactor;
    double maxProfitFactor;
    double minStability;
    double maxStability;
    double minWinLossRatio;
    double maxWinLossRatio;

    public Rule(double minProfitFactor, double maxProfitFactor, double minStability, double
maxStability,
        double minWinLossRatio, double maxWinLossRatio) {
        this.minProfitFactor = minProfitFactor;
        this.maxProfitFactor = maxProfitFactor;
        this.minStability = minStability;
        this.maxStability = maxStability;
        this.minWinLossRatio = minWinLossRatio;
        this.maxWinLossRatio = maxWinLossRatio;
    }

    public boolean matches(Strategy strategy) {
        return strategy.profitFactor >= minProfitFactor && strategy.profitFactor <= maxProfitFactor &&
        strategy.stability >= minStability && strategy.stability <= maxStability &&
        strategy.winLossRatio >= minWinLossRatio && strategy.winLossRatio <=
maxWinLossRatio;
    }
}

public class Main {
    public static void main(String[] args) {
        List<Strategy> strategies = new ArrayList<>();
        strategies.add(new Strategy(1.2, 0.9, 0.3, 5000));
        strategies.add(new Strategy(1.7, 0.7, 0.2, -6000));
        strategies.add(new Strategy(1.3, 0.2, 4, 20));

        int populationSize = 100;
        int generations = 50;
        List<Rule> population = new ArrayList<>();
```

```

// Generate initial population of rules
Random random = new Random();
for (int i = 0; i < populationSize; i++) {
    Rule rule = new Rule(random.nextDouble(), random.nextDouble(),
        random.nextDouble(), random.nextDouble(),
        random.nextDouble(), random.nextDouble());
    population.add(rule);
}

// Evolutionary loop
for (int gen = 0; gen < generations; gen++) {
    // Evaluate fitness
    double[] fitness = new double[populationSize];
    for (int i = 0; i < populationSize; i++) {
        Rule rule = population.get(i);
        double totalProfit = 0;
        for (Strategy strategy : strategies) {
            if (rule.matches(strategy)) {
                totalProfit += strategy.profit;
            }
        }
        fitness[i] = totalProfit;
    }

    // Select parents
    List<Rule> selectedPopulation = new ArrayList<>();
    for (int i = 0; i < populationSize; i++) {
        double totalFitness = 0;
        for (double fit : fitness) {
            totalFitness += fit;
        }
        double pick = random.nextDouble() * totalFitness;
        int index = 0;
        double sum = fitness[index];
        while (sum < pick) {
            index++;
            sum += fitness[index];
        }
        selectedPopulation.add(population.get(index));
    }

    // Crossover
    List<Rule> newPopulation = new ArrayList<>();
    for (int i = 0; i < populationSize; i++) {
        Rule parent1 = selectedPopulation.get(random.nextInt(selectedPopulation.size()));
        Rule parent2 = selectedPopulation.get(random.nextInt(selectedPopulation.size()));
        Rule child = new Rule(parent1.minProfitFactor, parent2.maxProfitFactor,
            parent1.minStability, parent2.maxStability,
            parent1.minWinLossRatio, parent2.maxWinLossRatio);
        newPopulation.add(child);
    }

    population = newPopulation;
}

// Find best rule
Rule bestRule = population.get(0);
double bestProfit = 0;
for (Rule rule : population) {
    double totalProfit = 0;
    for (Strategy strategy : strategies) {

```

```

        if (rule.matches(strategy)) {
            totalProfit += strategy.profit;
        }
    }
    if (totalProfit > bestProfit) {
        bestProfit = totalProfit;
        bestRule = rule;
    }
}

System.out.println("Best rule: " + bestRule.minProfitFactor + " <= ProfitFactor <= " +
bestRule.maxProfitFactor +
    ", " + bestRule.minStability + " <= Stability <= " + bestRule.maxStability +
    ", " + bestRule.minWinLossRatio + " <= WinLossRatio <= " + bestRule.maxWinLossRatio);
System.out.println("Best profit: " + bestProfit);
}
}

```