Tarbiat Modares University

Metaheuristic Algorithms — Fall 2025

# Phase 2 — Rule Discovery with Metaheuristics

December 5, 2025

## 1 Overview

In Phase 2 you will design and optimize a *rule-based* trading strategy for Ethereum using the engineered features created in Phase 1 (see the Phase 1 specification for the feature interface and examples).

This phase is technically the "final" phase of the project. However, to motivate iterative improvement, the evaluation is done in **two stages**:

- an **intermediate submission** at the end of Azar (first deadline);
- a **final submission** at the end of Dey (second deadline).

At each stage, teams submit their code plus the currently *best rule set* they have discovered. These rule sets will be evaluated on common datasets and the results will be published on a leaderboard inside the LMS (see Section 5).

You are given a historical dataset of ETH/USDT with 5-minute OHLCV candles and a set of feature columns. Your task is to:

- define a family of explicit **IF–THEN trading rules**;
- use at least one **metaheuristic algorithm** (e.g., Genetic Algorithm, Particle Swarm Optimization, Differential Evolution, Simulated Annealing) to automatically discover a good set of rules;
- backtest the discovered rule set on historical data with a capital-based simulation and evaluate it with a specified fitness function;
- deliver your final rule set in a *standard JSON format* so that all groups can be evaluated and compared on the same test data.

Let $x_t$ be the vector of features at time $t$. In this phase we restrict strategies to an ordered list of rules of the form

$$\text{IF conditions}(x_t) \quad \Rightarrow \quad \text{action at time } t$$

where each action is a decision to open or manage a long/short position with given position size, stop-loss, and take-profit levels. A metaheuristic search algorithm explores different rule lists and aims to maximize a backtesting-based fitness score.

## 2 Data and Backtesting

### 2.1 Datasets and time periods

You are provided with two datasets of ETH/USDT at 5-minute resolution (dataset link):

- **Train set**: 1 March 2025 – 31 August 2025.

- **Public test set**: 1 September 2025 – 30 November 2025.

Each row corresponds to one 5-minute bar and includes:

- timestamp (5-minute steps, chronological);
- open, high, low, close, volume;
- one column per feature (each corresponding to a `FEATURE_CODE` from Phase 1, e.g. `mom_rsi_14`, `trend_sma_20`, ...).

In addition, the instructors will hold out an extra **private test set** that is not released to students. This private dataset is used for the *final* ranking on the private leaderboard (see Section 5).

## 2.2 Data cleaning and preprocessing

The dataset is *not* cleaned. Each group is responsible for deciding how to treat the raw data before using it for rule discovery. You must explicitly document all decisions in your report.

Typical issues to address:

- Handling missing values (NaNs) and indicator warm-up periods.
- Checking for and handling duplicate or irregular timestamps.
- Deciding whether to normalize or standardize features (e.g., z-score, min–max).

## 2.3 Rule-based backtesting semantics

All groups must follow a common interpretation of how a rule-based strategy operates in backtesting. This ensures that different solutions are comparable.

We assume a single-asset account denominated in USDT with a fixed initial capital $C_0$ (in the evaluation script we use $C_0 = 1000$). At any time the strategy is either flat (no position) or in a single long/short position. When a rule fires while flat, a fraction of the current equity is allocated to that trade; there is no leverage and no overlapping positions.

### Rule structure

You will work with an ordered list of rules:

- Each rule has the form:

$$\text{IF } C_1(x_t) \wedge C_2(x_t) \wedge \cdots \wedge C_K(x_t) \text{ THEN action}$$

- Conditions are simple threshold tests on single features, e.g.
  - `mom_rsi_14` $< 30.0$
  - `band_bb_percB_20_2` $> 0.8$

  using operators "$<$" or "$>$" and numeric thresholds.
- The action part specifies:
  - side: `BUY` (long) or `SELL` (short);
  - take-profit level (TP) as a fraction of entry price (e.g., 0.04 for +4%);
  - stop-loss level (SL) as a fraction of entry price (e.g., 0.02 for $-2\%$);
  - position size as a fraction of current capital (e.g., 0.10 for 10% of equity, typically restricted to a range such as $[0.05, 0.50]$).

### Execution logic

At each time $t$:

1. Evaluate the rules in order: Rule 1, then Rule 2, and so on.

2. The first rule whose *all* conditions are satisfied by $x_t$ is said to "fire".
3. If the strategy is currently *flat* and a rule fires:
   - compute the capital allocated to this trade as
     $$\text{capital\_alloc} = \text{size} \times \text{equity}_t;$$
   - open a new position with the side (long/short), TP/SL, and position size defined by that rule;
   - record the entry price, the rule index responsible for the trade, and the allocated capital.
4. If the strategy already has an open position:
   - no new positions are opened, even if other rules fire;
   - the existing position is managed according to TP/SL based on the original rule.
5. If no rule fires:
   - if flat: do nothing;
   - if in a trade: stay in the trade unless an exit condition (below) is met.

**Exit logic**

For a trade opened at time $t_0$ with entry price $P_{\text{entry}}$ and side $s \in \{+1, -1\}$ (long/short), the unrealized return at time $t$ is

$$r_t = s\left(\frac{P_t}{P_{\text{entry}}} - 1\right),$$

where $P_t$ is the current price (e.g., close price).

The position is closed when any of the following happens:

- **Take-profit hit:** $r_t \geq \text{TP}$.
- **Stop-loss hit:** $r_t \leq -\text{SL}$.

If capital_alloc is the capital allocated to this trade when it was opened, then when the trade is closed at time $t$ with unrealized return $r_t$, the profit/loss in account currency is

$$\text{pnl} = r_t \times \text{capital\_alloc},$$

and the equity is updated as

$$\text{equity}_{t^+} = \text{equity}_{t^-} + \text{pnl}.$$

Each completed trade thus contributes to the equity path. You may still record per-trade returns for analysis and plotting, but the main fitness is based on final equity.

## 2.4 Fitness function

The core fitness that will be used for ranking is the **final equity** on the training period.

We run the backtest starting from a fixed initial capital $C_0$ (e.g. $C_0 = 1000$). Let $\text{equity}_0 = C_0$ and let $\text{equity}_T$ be the final capital after processing all bars with the rule-based strategy and position sizing. The fitness is

$$J = \text{equity}_T.$$

Special case: if a candidate produces no trades on the train set, assign a slightly penalized fitness (e.g. $J = C_0 - 1$) so that such "do nothing" solutions are worse than any strategy that trades at least once.

You may optionally add regularization penalties (e.g. for too few trades, too many rules, or excessive complexity), but you must clearly describe your final fitness formula in the report. For example, you could define

$$J_{\text{final}} = \text{equity}_T \; - \; \alpha \cdot \text{Complexity} \; - \; \beta \cdot \text{Risk},$$

with well-documented definitions of each term.

## 3 Project requirements

Each team must build a complete rule discovery pipeline that satisfies the following requirements:

1. **Data preparation**
   - Load the provided train and public test datasets.
   - Perform appropriate cleaning and preprocessing.
2. **Rule representation**
   - Represent candidate solutions as an ordered list of rules.
   - Each rule must be a conjunction of simple threshold conditions on individual features, and an action (side, TP, SL, position size).
   - Internally, you may encode rules in any convenient way (e.g. real-valued genes mapped to thresholds), but you *must* be able to export them to the common JSON format described in Section 6.
3. **Metaheuristic optimization**
   - Choose at least one metaheuristic algorithm from the course (e.g. GA, PSO, DE, SA).
   - Define how candidate rule lists are initialized, modified, and selected.
   - Use the common fitness function based on final equity on the train data (starting from a fixed initial capital), possibly with documented extensions.
4. **Backtesting**
   - Implement the standard rule-list backtest described in Section 2, with no look-ahead bias.
   - Use only information available up to time $t$ when making decisions at $t$.
   - Use a capital-based simulation with the same starting capital and position sizing rules you use for fitness evaluation.
5. **Generalization and evaluation**
   - Perform a sensible evaluation on the train period.
   - Avoid directly tuning hyperparameters on the public test set.
   - Report the performance of your final chosen rule set on both train and public test data (e.g. final equity, number of trades, drawdown).
6. **Documentation and reproducibility**
   - Provide readable and commented code.
   - Use fixed random seeds where possible so that results are approximately reproducible.

## 4 Rule representation and metaheuristic search

### 4.1 Encoding IF–THEN rules

Although the final rule set is expressed in human-readable form, a metaheuristic algorithm needs a numerical encoding. A convenient approach is:

- Fix a maximum number of rules $N_{\max}$ (e.g. 5) and a maximum number of conditions per rule $K_{\max}$ (e.g. 5).
- For each potential condition, store:
  - a feature index (or feature ID);
  - an operator bit ($<$, $>$);
  - a threshold parameter (e.g. a value in $[0, 1]$ mapped to a quantile or numeric range of that feature);
  - an "active/inactive" flag (to allow a rule to have fewer than $K_{\max}$ active conditions).
- For each rule, store:
  - the rule activation flag;

  - the action side (`BUY`/`SELL`);
  - TP and SL values, encoded as bounded real numbers;
  - a size gene encoding the position size fraction (later mapped to a range such as $[5\%, 50\%]$ of capital).

A complete candidate solution (chromosome/particle) is then a concatenation of the parameters for all potential rules and conditions. During decoding you:

1. read the genes and construct up to $N_{\max}$ rules;
2. skip inactive conditions;
3. discard rules that have no active conditions;
4. keep rules in order.

### 4.2 Metaheuristic algorithm choices

You are free to implement your preferred metaheuristic. Examples:

- **Genetic Algorithm (GA):** population of chromosomes (rule lists), tournament or roulette selection, crossover that swaps subsets of rules between parents, mutation that perturbs thresholds, toggles conditions, or changes TP/SL/size.
- **Particle Swarm Optimization (PSO):** real-valued encoding of all rule parameters, velocity updates to move particles in parameter space towards high-fitness solutions.
- **Differential Evolution (DE):** vector differences between candidate solutions to generate new ones.
- **Simulated Annealing (SA):** single-solution stochastic search with temperature-controlled acceptance of worse moves.

Regardless of the algorithm, you must clearly specify:

- initialization procedure;
- variation operators (what a "neighbor" or "mutation" means in rule space);
- selection/survival mechanism;
- stopping criteria (max generations, convergence, time budget).

### 4.3 Fitness extensions and regularization (optional)

To reduce overfitting and encourage simpler, more robust rules, you may extend the core fitness $J$ (final equity) with penalty terms. Example:

$$J_{\text{final}} = \text{equity}_T \; - \; \alpha \cdot \text{Complexity} \; - \; \beta \cdot \text{Risk},$$

where:

- Complexity could measure number of rules, total number of conditions, or number of distinct features used.
- Risk could be expressed via maximum drawdown, return volatility, or a penalty for too few trades.

More advanced teams may consider multi-objective metaheuristics that jointly optimize return and risk, but this is optional.

## 5 Timeline, submissions, and leaderboards

To align with the course schedule and to encourage iterative improvement, Phase 2 is evaluated using **two submissions** and **two leaderboards**.

### Intermediate submission (end of Azar)

Around the end of Azar (exact calendar date will be announced on the LMS), each team must:

- submit their current code (Jupyter notebook + any helper modules);
- submit a JSON file with their *current best rule set* (according to their own training evaluation);
- submit a short intermediate report (can be more concise than the final report), summarizing:
  - the current rule representation and metaheuristic setup;
  - preliminary experimental results on train and public test sets.

The instructors will evaluate each submitted rule set using a **public evaluation script** on:

- the common train set (for consistency checks), and
- the public test set (open dataset provided to all teams).

The resulting scores (e.g. final equity on the public test set) will be displayed on a **public leaderboard** in the LMS by around the middle of Dey. This leaderboard reflects the evaluation of the intermediate (end-of-Azar) submission.

### Final submission (end of Dey)

Around the end of Dey (exact calendar date on LMS), each team must submit:

- updated and cleaned code (Jupyter notebook + any helper modules);
- a final JSON rule file with their *best* rule set at the end of the project;
- the full final report (see Section 6).

The instructors will then re-evaluate the final rule sets using:

- the same public test set as in the intermediate stage;
- an additional **private test set** that is not provided to students.

The scores on the private test set will be published on a **private leaderboard** (visible in the LMS) around the middle of Bahman and will form the basis for the final grading of Phase 2.

*Important:* While you can look at the public leaderboard to see how your strategy ranks on the public test set, you should avoid overfitting to this leaderboard. The private leaderboard, based on unseen data, is what ultimately matters for robustness.

## 6 Deliverables

Each team must submit the following items to the LMS at both deadlines, with the expectation that the final submission is a polished and improved version of the intermediate one.

### 6.1 Source code (Jupyter notebook)

Submit all code required to:

- load and preprocess the provided datasets;
- represent and decode rule lists;
- run the metaheuristic search;
- backtest a rule set using the common semantics;
- export the final chosen rule set.

The code should be organized and documented. Include a small `README` file.

### 6.2 Final rule set file

Each team must submit a single JSON file containing the discovered rule list:

- file name: `rules_<teamname>.json`
- place it in a folder called `rules/` in your project

The JSON file must follow this exact schema:

```
{
  "rules": [
    {
      "conditions": [
        { "feature": "FEATURE_NAME", "op": "<", "threshold": 0.0 },
        { "feature": "FEATURE_NAME", "op": ">", "threshold": 0.0 }
      ],
      "action": {
        "side": "BUY", // or "SELL"
        "tp": 0.04, // take profit (fractional, e.g. 0.04 = 4%)
        "sl": 0.02, // stop loss (fractional)
        "size": 0.10 // position size (fraction of capital, e.g. 0.10 = 10%)
      }
    }
  ]
}
```

**Rules and formatting requirements:**

- The top-level JSON object must contain a key `"rules"` whose value is a list.
- Each rule in the list must contain:
  - `"conditions"`: an array of condition objects;
  - `"action"`: an object describing the trading action.
- Each condition object must have exactly:
  - `"feature"`: a dataset column name (string);
  - `"op"`: either `"<"` or `">"`;
  - `"threshold"`: a numeric value.
- The action object must include:
  - `"side"`: `"BUY"` or `"SELL"`;
  - `"tp"`: take-profit as a fractional number (e.g., `0.04`);
  - `"sl"`: stop-loss as a fractional number (e.g., `0.02`);
  - `"size"`: position size as a fractional number (e.g., `0.10` for 10% of current capital). In the evaluation script we typically clamp this to a range such as $[0.05, 0.50]$.
- Students may include up to five rules. Rules are interpreted in order.
- No additional keys, formatting, or comments should be included.

This JSON file will be parsed automatically and evaluated on the public and private test sets using a unified backtesting script.

### 6.3 Report

Submit a PDF report named `report_<teamname>.pdf`, including figures and tables. You will submit:

- a shorter **intermediate report** for the end-of-Azar deadline (can be a concise version focusing on current progress);
- a full **final report** for the end-of-Dey deadline.

A suggested structure for the final report:

1. **Introduction**
   - Problem description: rule discovery with metaheuristics for ETH trading.
   - High-level summary of your chosen approach.
2. **Data and preprocessing**
   - Description of train and public test periods.
   - All cleaning steps (missing values, outliers, scaling).
   - Any additional features created and why.
3. **Rule representation**
   - Internal encoding of rules and rule lists.
   - Constraints on number of rules and conditions.
   - How encoded parameters are mapped to thresholds, TP/SL, position size, etc.
4. **Metaheuristic algorithm**
   - Choice of metaheuristic(s) and justification.
   - Initialization, variation operators, selection, and stopping criteria.
   - Hyperparameters such as population size, number of generations, mutation rate, etc.
5. **Fitness and evaluation**
   - Exact fitness function used (equations), including how final equity is computed.
   - Treatment of rule sets that produce no trades.
   - Any additional metrics tracked (number of trades, drawdown, volatility).
6. **Experiments and results**
   - Training dynamics (e.g. best fitness vs. iteration).
   - Performance of the final rule set on train and public test data (e.g. final equity, number of trades).
   - Summary tables and plots (equity curve, distribution of trade returns, etc.).
   - Comparison of design variants if you tried multiple approaches.
7. **Discussion and conclusions**
   - Interpretation of the discovered rules (are they intuitive?).
   - Overfitting and robustness issues observed.
   - Lessons learned and ideas for future improvements.
8. **Appendix (optional)**
   - Full final rule set (same content as the submitted rule file).
   - Extra plots or analysis.

**Final note.** This phase focuses on interpretable rule discovery: you should end up with a small, human-readable set of trading rules discovered by a metaheuristic algorithm. Be creative in your design, but justify your choices, and use the two submission stages to iteratively improve both performance and robustness.

**Good luck!**