AI Trading Agent Using A* and Greedy Search

Yacine Kaizra

June 11, 2025

Abstract

This document describes the architecture and implementation of an AI trading agent that uses **A* search** and **Greedy Search** algorithms to simulate optimal investment decisions on historical stock data. The agent dynamically generates a state space of possible actions (buy/sell/hold) while respecting financial constraints (available funds, diversification limits). Heuristic scores guide the search toward maximizing portfolio value.

1 Overview

The trading agent operates on historical stock price data (e.g., 2023–2024) and simulates trades to maximize returns. It evaluates sequences of actions (buy/sell/hold) for a given set of stocks (e.g., `["AAPL", "MSFT", "TSLA"]`) and outputs the most profitable path based on heuristic-driven search algorithms.

2 Workflow Summary

1. **Data Retrieval**: Fetch historical prices using `yfinance` (daily close prices for the specified period).
2. **State Space Generation**: Construct a tree of valid actions (buy/sell/hold) for each stock.
3. **Heuristic Design**: Assign scores to actions based on price trends, portfolio impact, and risk.
4. **Search Algorithm**: Use A* or Greedy Search to find the optimal action sequence.
5. **Portfolio Update**: Track cash, owned shares, and portfolio value after each trade.

3 Technical Details

1. State Space Construction

The agent builds a state space where each node represents a portfolio state (cash, owned shares). Actions branch into:

- `BUY <stock>`: Requires sufficient funds.
- `SELL <stock>`: Requires ownership of the stock.
- `HOLD`: No change to the portfolio.

Invalid nodes are pruned:
- No purchase if funds are insufficient.
- No sale if stock is not owned.
- No purchase if stock allocation exceeds 20% of total portfolio value.

2. Heuristic Design

Each action receives a heuristic score $h \in [0.01, 1.0]$, normalized across stocks to ensure fairness. Scores are calculated

Normalization

Scores are scaled to $[0.01, 1.0]$ to prevent dominance by any single factor:

3. Search Algorithms

- **A* Search and Greedy Search**: Uses a max-heap priority queue to explore paths with the highest cumulative heuristic score.

4. Portfolio Management

The `Stocks` class tracks:
- Current cash balance.
- Owned shares per stock.
- Portfolio value (cash + market value of shares).

Constraints:
- Diversification limit: No single stock ¿ 20% of total portfolio value.
- Transaction validation (e.g., no selling unowned shares).

4 Code Structure

Listing 1: Example State Space Node

```
{
    'symbol': 'AAPL',
    'action': 'BUY',
    'value': 125.00,
    'heuristic': 0.85
}
```

5 Results

gives pretty good results between 115on the inital capita given

exampe results

```
Sold 1 share of JPM for $161.66. Updated funds: $607.11
Bought 1 share of PFE for $25.84. Remaining funds: $581.27
Bought 1 share of AMZN for $151.94. Remaining funds: $429.33
Bought 1 share of GOOGL for $138.86. Remaining funds: $290.47
Bought 1 share of TSLA for $248.48. Remaining funds: $41.99
Sold 1 share of PFE for $26.19. Updated funds: $68.18

Available funds: $68.18
Stock Portfolio:
- V: 3 shares
- NVDA: 4 shares
- AAPL: 5 shares
- PFE: 4 shares
- GOOGL: 5 shares
- AMZN: 1 shares
- TSLA: 1 shares
------------------------------
initial capite:  2000
Total Portfolio Value: 3193.858857154846
return on investment :  59.69294285774232 %
3193.858857154846
```

Figure 1: 59 percent investment return

6 Conclusion

This project demonstrates the application of classical AI search algorithms to financial decision-making. By combining heuristic scoring with real-world constraints, the agent balances exploration (A*) and exploitation (Greedy Search) to simulate profitable trading strategies.