Framework Document
Strategy Backtesting Framework for Smart Order Routing (SOR)

Tengrui Kong(Ray)

January 7, 2025

Version: 1.1
updated: 2025-1-6
Strategy Backtesting Framework for Smart Order Routing (SOR)
: Part 1: Methodology and Framework.
https://github.com/Riokong/Backtesting_Project

1 Instructions

This report is intended to outline my proposed back-testing framework for Smart Order Routing (SOR).

2 Data Pipeline

2.1 How to Source, Process, and handle market/order book data

Explanation: The Data Pipeline forms the backbone of the Strategy Backtesting Framework for Smart Order Routing (SOR), ensuring seamless sourcing, processing, and handling of market data. For budget concern, I prefer Polygon.io as the data provider. Using Polygon.io, the pipeline begins by sourcing high-quality market data, including Level 1 (best bid/ask), Level 2 (full order book), and trade data. Historical data can be fetched using RESTful APIs for backtesting, while real-time data is streamed via WebSocket for live simulation.

2.2 Data processing

Explanation: The data is normalized into consistent formats, such as pandas DataFrames, and enriched with timestamps to facilitate synchronization. Additionally, any missing data is addressed through interpolation or fallback mechanisms, ensuring a clean and reliable dataset. Latency simulation is incorporated to replicate real-world conditions, where order book updates and trade executions are processed sequentially with artificial delays.

Once the data are acquired, it undergoes transformation to meet the framework requirements. For Level 1 data, the pipeline approximates a Limit Order Book (LOB) by estimating multiple bid/ask levels based on historical trends or predefined spreads. For Level 2 data, the pipeline directly integrates the full-depth order book, reducing preprocessing overhead. Synchronization is key, aligning trade data with order book snapshots using precise timestamps. The processed data is then stored in an efficient database, such as SQLite, for quick querying and scalability.

3 Execution Strategies

3.1 Description of Time-Weighted Average Price straregy

Explanation: **Time-Weighted Average Price (TWAP)** is an execution strategy that divides a large order into smaller, equally sized trades executed at regular intervals over a specified time period. The goal of TWAP is to minimize market impact by avoiding large trades that could disrupt the market and achieving an average execution price close to the market's time-weighted average during the trading period. TWAP is simple to implement and does not rely on external factors such as volume or liquidity, making it a preferred choice in illiquid markets or when trading without real-time volume data.

Example

TWAP performs well in stable or low-volatility markets where price fluctuations are minimal. However, during high-volatility conditions, TWAP may result in suboptimal execution, as it does not adapt to price movements or volume changes. For example, if prices move significantly during the execution window, the trades executed at fixed intervals could lead to a worse average price compared to a more dynamic strategy. Additionally, in highly liquid markets, TWAP may struggle to blend in with the natural flow of trades, increasing the risk of detection and potential adverse price movements by other market participants.

3.2 Description of Volume-Weighted Average Price strategy

Explanation: **Volume-Weighted Average Price (VWAP)** is a strategy that executes trades proportionally to the traded volume of the market over a specified time period. By aligning trade execution with the market's natural volume profile, VWAP aims to minimize market impact and slippage. VWAP is particularly effective in liquid markets with stable trading volumes, as it ensures that trades are executed more aggressively when the market is active and conservatively during periods of low activity.

Example

VWAP is more adaptive to market conditions as it aligns with trading volume. It performs well in liquid markets where trading activity is high and volume patterns are predictable. However, in illiquid markets or during irregular volume spikes, VWAP can face challenges. For instance, if a sudden surge in trading volume occurs, the strategy may execute a large portion of the order at potentially unfavorable prices, deviating from the intended volume-weighted target. Similarly, during periods of low activity, VWAP's conservative execution may result in incomplete order execution within the desired timeframe.

4 Performance Metrics

Evaluating the performance of execution strategies such as TWAP and VWAP requires a set of well-defined metrics that provide information on their effectiveness and efficiency. Here are the commonly used key metrics:

4.1 Execution Cost

Explanation: Execution cost measures the difference between the final price at which an order is executed and a benchmark price, providing an assessment of the strategy's efficiency. Common benchmarks include the market price at the time the order was placed, the mid-price, or TWAP/VWAP prices during the execution window. Execution cost can be broken down into:

- Explicit Costs: These include commissions, fees, and taxes incurred during the execution process.

- Implicit Costs: These represent the price impact of executing the order, including market impact and opportunity cost.

Execution cost is often expressed in basis points (bps) relative to the benchmark price and is a critical metric for determining the overall cost-effectiveness of a strategy.

4.2 Slippage

Explanation: Slippage quantifies the difference between the intended execution price (e.g., the price at the time the order was placed) and the actual price at which the trade is executed. It is influenced by market conditions, order size, and strategy efficiency.

Example

Slippage can be categorized into:

- Positive Slippage: When the execution price is better than the intended price.
- Negative Slippage: When the execution price is worse than the intended price.

Minimizing slippage is a key objective for execution strategies, as it directly impacts profitability. High slippage often indicates poor strategy design or unfavorable market conditions.

4.3 Fill Rates

Explanation: Fill rate measures the proportion of the order that was successfully executed relative to the total order size. It provides insights into the strategy's ability to execute the desired quantity within the specified time frame.

Example

The fill rate is calculated as:

$$\text{Fill Rate} = \frac{\text{Executed Order Size}}{\text{Total Order Size}} \times 100 \tag{1}$$

A high fill rate indicates that the strategy effectively achieved its execution target, while a low fill rate may signal issues such as insufficient liquidity, poor timing, or overly conservative execution parameters.

4.4 Simulation Logic

Explanation: Simulating multi-venue routing decisions involves modeling the behavior of multiple trading venues, including their liquidity profiles, latency characteristics, transaction costs, and specific market rules. The simulation begins by splitting large parent orders into smaller child orders based on strategies like TWAP or VWAP. These child orders are routed to venues using algorithms that prioritize key factors such as best execution price, available liquidity depth, and overall cost efficiency including exchange fees and rebates. Once routed, the simulation models how orders interact with each venue's

order book, whether as liquidity-taking market orders or liquidity-providing limit orders.

Multi-leg trades, such as arbitrage or pairs trading strategies, are carefully coordinated across venues to ensure proper execution synchronization and risk management. The framework also supports advanced order types like iceberg orders for large size concealment, pegged orders for dynamic price adjustment, and conditional orders for more complex trading scenarios.

5 Extensibility and Scability

Explanation: To scale the framework for multi-leg strategies and advanced SOR configurations, implement a robust order management system (OMS) to coordinate dependent legs, handle partial fills, and manage execution risks. Parallelize order processing across venues and use **machine learning** to optimize routing decisions in real-time. Incorporating **high-performance messaging systems like Apache Kafka** ensures low-latency communication and seamless data synchronization, enabling scalability for complex strategies.

Example

How the system can be extended to different asset classes:

Extending the framework to different asset classes requires modular design with asset-specific modules for order book modeling, price normalization, and execution rules. APIs or market feeds for various instruments can be integrated while maintaining a shared core logic for routing. Flexible data schemas and abstraction layers accommodate unique trading conventions, ensuring the framework remains adaptable and supports multi-asset SOR strategies effectively.

6 Future Work

Explanation: Future enhancements will integrate advanced methodologies to expand the framework's capabilities. A hybrid forecasting pipeline combining PatchTST and TimeGAN will augment synthetic data for improved time series prediction. Sentiment features extracted via BERT will be fused with numerical data processed using NumPy, Pandas, and scikit-learn to enhance analytics. Backtesting frameworks like PyAlgoTrade and Backtrader will be employed alongside stochastic processes to evaluate forecasting strategies.

Additionally, a real-time API developed with Flask and Docker, and deployed on AWS SageMaker, will enable scalable applications for portfolio optimization, risk management, and transaction cost analysis. These efforts aim to create a robust, real-time financial strategy platform combining advanced analytics and scalable deployment..