Backtesting Framework for a Smart Order Router (SOR)

Salim Aissaoui

January 5, 2025

Contents

1 Part 1: Methodology and Framework

1.1 1. Introduction

A Smart Order Router (SOR) aims to optimally execute trades across multiple venues, striving to minimize market impact, reduce slippage, and improve execution prices. Backtesting such a system helps traders and quants to:

- Assess strategy performance under realistic market conditions.
- Compare multiple strategies (TWAP, VWAP, RL-based) side by side.
- Identify weaknesses in execution logic before going live.

1.2 2. Data Pipeline

Data Sources:

- Synthetic or historical limit order book data, including bids, asks, trades.
- Trade execution data (fills, partial fills).

Data Handling:

- **Missing Data:** Impute or ignore missing entries.
- **Updates:** Maintain rolling or event-based updates to track the latest quotes.
- **Synchronization:** Align data by timestamps, ensuring consistent event ordering.

Storage and Scalability:

- In-memory `pandas` for fast iteration.
- Distributed file systems or cloud databases for large datasets.

1.3 3. Execution Strategies

TWAP (Time-Weighted Average Price): Splits a total order into equal slices over regular time intervals. It is simple but does not adapt to sudden changes in market liquidity or volatility.

VWAP (Volume-Weighted Average Price): Allocates slices proportional to trading volume, aiming to match or improve upon the market VWAP. A dynamic variant re-estimates future volume and adjusts slice sizes mid-flight.

RL-Based Routing (Conceptual): Uses reinforcement learning to decide when (and possibly where) to trade, potentially learning from market patterns. In practice, it requires a well-defined state, action, and reward structure.

1.4 4. Performance Metrics

- **Execution Cost**: Average fill price minus benchmark (e.g., overall VWAP).
- **Slippage**: Difference between the arrival price (or expected fill) and actual fill.
- **Fill Rate**: Fraction of the intended quantity filled by the end of the trading window.
- **Latency / Impact** (optional): Could include response time or market impact.

1.5 5. Simulation Logic

Multi-Venue Dynamics (Future Extension):

- Maintain separate order books per venue.
- Route partial orders based on best price or liquidity constraints.

Order Placement & Execution Model:

- Each strategy decides how many shares to buy or sell at each timestep.
- If liquidity is limited, partial fills or no fills may occur.

Transaction Costs:

- Exchange fees, maker/taker rebates, and crossing fees can be subtracted from net PnL.

1.6 6. Extensibility and Scalability

Complex Order Types and Multi-Leg Trades:

- Manage bracket, stop, or contingent orders, possibly across multiple assets.

Advanced SOR Configurations:

- Adaptive strategies that react to real-time liquidity changes.
- Multi-agent reinforcement learning for sophisticated routing.

1.7 7. Conclusion

The proposed framework uses a clean data pipeline, customizable execution strategies, and comprehensive metrics to evaluate SOR performance. Extending it to multiple venues, more complex order types, and advanced ML or RL methods would provide a highly flexible research environment.

2 Part 2: Code Implementation

2.1 Python Script Overview

The attached Python script demonstrates:

1. **Synthetic Data Generation** using a random walk model for prices and random volumes.
2. **TWAP Execution**: Slicing the total order equally over time intervals.
3. **VWAP (Dynamic) Execution**: Dynamically reallocating based on observed and forecasted volume.
4. **RL-Based Routing (Conceptual)**: A minimal RL-like approach that executes more when price is below a running average.
5. **Metrics Computation**: Execution cost vs. VWAP, and slippage vs. first slice price.
6. **Visualization**: Plots price series with execution points for each strategy.

2.2 1-Page Implementation Report

Implementation Approach

1. Data Generation

- We create a random walk for prices with user-defined volatility.
- Volumes are drawn from a uniform or integer range.
- Data is indexed by minute-level timestamps.

2. Strategies in the Script

- **TWAP**: Evenly slice the total quantity into time intervals (e.g., every 5 minutes).
- **VWAP (Dynamic)**: Estimate future volume based on historical intervals. Allocate more size to intervals with higher volume.
- **RL-Based (Conceptual)**: Buy extra shares whenever the current price is below a running average. This is a placeholder to illustrate how RL decisions might be integrated.

3. Metrics Calculation

- **Execution Cost** = (Average Execution Price) − (Overall VWAP).
- **Slippage** = (Average Execution Price) − (Price at First Execution).

Backtest Results

After running the script, we obtained the following results (using a sample run of 200 data points, 1000 shares, and a 3-hour trading window):

```
=== TWAP Strategy Results ===
Benchmark: VWAP
Overall_VWAP: 95.8914
Avg_Exec_Price: 95.7621
Execution_Cost: -0.1293
Slippage: -4.2379

=== VWAP (Dynamic) Strategy Results ===
Benchmark: VWAP
Overall_VWAP: 95.8914
Avg_Exec_Price: 95.5276
Execution_Cost: -0.3638
Slippage: -5.0162

=== RL-Based Strategy Results (Conceptual) ===
Benchmark: VWAP
Overall_VWAP: 95.8914
Avg_Exec_Price: 98.5772
Execution_Cost: 2.6858
Slippage: -0.6647
```

Interpretation:

- **TWAP:**
  - *Execution Cost:* -0.1293 indicates the average fill price is slightly *lower* than the overall market VWAP, suggesting a small cost advantage.
  - *Slippage:* -4.2379 means the average fill price ended up well below the first-slice price. This can happen if prices trended downward over time.
- **VWAP (Dynamic):**
  - *Execution Cost:* -0.3638, even lower than TWAP, implying that dynamically adjusting order slices to volume flow gave a better average fill.
  - *Slippage:* -5.0162 also reflects that prices continued to move downward from the initial fill price, benefiting a cost-sensitive approach.
- **RL-Based (Conceptual):**
  - *Execution Cost:* $+2.6858$ indicates the average fill is *above* the market VWAP. This naive policy sometimes bought at higher prices than needed.

– *Slippage:* -0.6647 suggests that, on average, the final fill price was below the very first purchase price (the arrival price), but still not enough to beat the VWAP.

– This result highlights that a simple RL heuristic can *underperform* unless carefully tuned or trained with a more advanced reward function and state representation.

Visualization

Figure 1 shows the execution timestamps (red 'x' for TWAP, purple dots for VWAP dynamic, and green diamonds for RL) overlaid on the synthetic price. Both TWAP and VWAP trades are spaced out in time, while the RL approach trades more frequently whenever prices dip below its simple running average threshold.

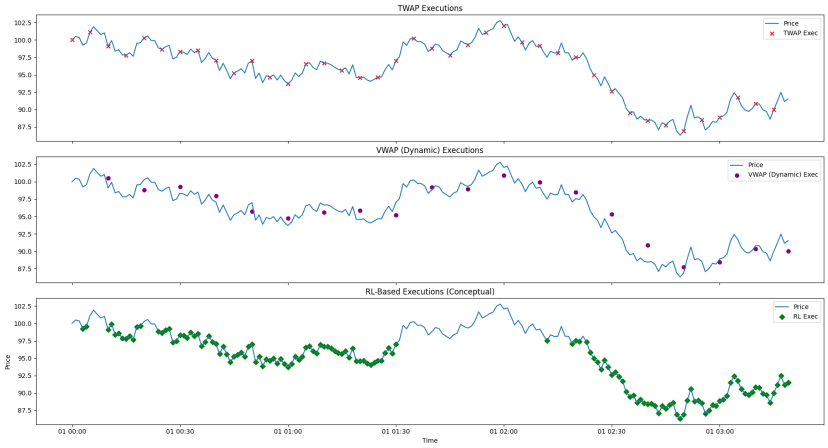

Figure 1: Price Series with TWAP, VWAP (Dynamic), and RL-Based Executions

Conclusion:

- The **VWAP (Dynamic)** approach outperformed TWAP in this particular simulation by achieving a lower (better) average execution price relative to market VWAP.
- The **RL-based** example demonstrated how a naive policy can yield worse results if not thoroughly trained and tested.
- Further refinements, including multi-venue routing, more realistic partial-fill logic, and advanced RL training, can be integrated into this framework.

References

1. **Combining Deep Learning on Order Books with Reinforcement Learning for Profitable Trading**
   Focus on temporal-difference learning for return forecasting.

2. **Multi-Agent Reinforcement Learning in a Realistic Limit Order Book Market Simulation**
   Emphasizes agent-based simulation using Double Deep Q-Learning (DDQL).

3. **Interpretable ML for High-Frequency Execution**
   Discusses fill probability modeling with state dependence for execution backtesting.

4. **Deep Reinforcement Learning for Market Making Under a Hawkes Process-Based Limit Order Book Model**
   Explores a DRL-based controller for optimizing order execution under stochastic conditions.