Strategy Backtesting Framework for Smart Order Routing

Part 1: Methodology and Framework

An SOR is an algorithm aimed at streamlining the process of placing orders for traders. By analyzing order books across multiple markets and comparing factors such as price, liquidity, and volume, an SOR quickly determines the best place to execute an order, even going as far as splitting large orders to take advantage of better prices that appear in lower volumes across the different venues. Without an effective SOR, traders are left at a significant disadvantage since they are unable to keep up with the vast amount of information and millisecond response time needed to make the most optimal trade that beats the market. For this reason, proper backtesting is a crucial step in developing an SOR, ensuring that the trader's technology can be relied on with confidence.

To test an SOR algorithm effectively, we first need a dataset that we can use to create a realistic market simulation, allowing us to test the SOR without the risk associated with live trading. High-frequency limit order books from multiple venues are a great basis for our simulated environment since they contain actual random events that occurred during trading. However, it is also important to consider the limitations of this data. We cannot assume that these order books are complete due to the high possibility of logging and latency errors as well as the erratic behavior of order placement and cancellation that can cause data gaps and desynchronization, especially when considering orders across multiple venues. To address these gaps, the LOB data must be processed, ensuring timestamp alignment, using interpolation to fill in minor gaps, and normalizing the data from the different venues. Another limitation of the data is that it only covers events that occurred during a fixed period, meaning any rare scenarios, a short squeeze for example, that didn’t occur during the timeframe would not be tested for in our simulation. Therefore, supplementing our historical LOB data with synthetic data created using Hawkes Processes and Monte Carlo methods can serve to further fill in the gaps mentioned previously, as well as add more plausible scenarios to the simulated market. This hybrid strategy allows for a comprehensive testing environment, while also ensuring scalability as more data can be added to the simulation data set as it becomes available.

Using this robust dataset, we can now create a realistic market simulation that enables us to test an SOR against numerous iterations of potential market scenarios. One of the biggest challenges that our simulation must address is the creation of multi-venue routing decisions to test the decision logic of the SOR. One approach to the issue is to create virtual venues, each with its unique characteristics by varying factors such as liquidity, volatility, and latency. Then using stochastic processes, we can create synthetic LOBs with these distinct characteristics creating opportunities for the SOR to utilize its decisioning algorithm to determine the best execution. Another key aspect that the backtester must include is the placement of complex order types. This can be done using an algorithm that converts a portion of the synthetic orders in each venue to special order types, including but not limited to iceberg orders, stop-loss orders, and spread orders. A final element to make the simulated market more realistic would be to implement a fill probability model based on the factors affecting each venue to simulate events such as partial fills, order cancellations, and dynamic order modifications. With these logistics detailed, we can now consider how orders in the market will be filled and executed. Using an iterative approach, where market transactions occur at each timestamp would allow us to cleanly simulate order placements, order executions, and routing decisions while also measuring the key metrics, such as slippage and execution cost, associated with each of these actions. This simulation logic will create a testing environment that encompasses a wide range of market scenarios while also being simple enough to easily identify unexpected behaviors and areas of improvement for an SOR.

Under these simulation conditions, we now understand how certain execution strategies would perform in the market. The simplest execution strategy is to simply place a market order for the desired volume of the asset. This strategy is easy to implement and guarantees execution, making it a good choice for time-critical situations that require immediate execution. However, being able to immediately execute large orders means taking whatever open trades are available, which can lead to high execution costs and slippage, especially during periods of high volatility or low volume. To improve performance during these less favorable market conditions, we can consider Time-Weighted-Average-Price and Volume-Weighted-Average-Price execution strategies that break down large orders into smaller orders to obtain better trades. A TWAP execution strategy takes advantage of time, evenly breaking up a larger order and executing it in smaller chunks over a set time interval. This strategy allows volume to build in the market as the order is executed, reducing execution costs and slippage from a generic market order execution strategy. Though this strategy performs better in poor market conditions, since orders are evenly broken up over time, it is unable to capitalize on periods of high liquidity where placing a generic market order would result in immediate execution with minimal costs. A VWAP strategy can prove useful here, by distributing order volume based on the expected volume within a time interval. This expected volume is based on historical VWAP and is used by the algorithm to increase trades in periods of high expected volume and reduce trades during periods of low expected volume. A VWAP strategy overcomes the limitations of a TWAP strategy, however, it is prone to failure during periods of high volatility or when the market deviates from historical VWAP movements, making it difficult for the algorithm to predict volume. We can avoid having to predict market volume by using a Percentage of Volume execution strategy instead. With this strategy, we monitor the real-time volume in a time interval and then execute only a portion of the observed trading volume. In this manner, we can keep up with high volatility and unpredictable markets. However, this strategy also has its drawbacks since real-time monitoring can pose a challenge during extremely volatile periods, where waiting to observe the current market can result in missing opportunities or increasing execution costs due to latency. All these execution strategies have their strengths and weaknesses, and no strategy is strictly better than the other making backtesters a useful tool when deciding which of these strategies is better on average in a random market environment.

Deciding between these strategies requires having metrics that we can use to make objective comparisons about performance. There are three main metrics that we can look at to evaluate the performance of an execution strategy: execution cost, slippage, and fill rate. The first metric, execution cost, measures the price of execution for an order minus a benchmark price such as VWAP. This metric tells us how good an SOR is at obtaining the best prices in the market. For buy orders, a negative execution cost indicates that the SOR has found favorable trades for execution while a positive execution cost indicates an unfavorable execution. The opposite is true for sell orders. The second metric, slippage, measures the expected execution price minus the actual execution price. This metric is useful for identifying how well an SOR performs in a dynamic market. For buy orders, a positive slippage indicates that the SOR can place orders and capitalize on opportunities in the market, while a negative slippage means that the algorithm suffers from latency or poor prediction resulting in missed opportunities and decreased profits. Finally, fill rates are a measure of the ratio of executed volume to order volume. A high ratio means that the SOR can fill most of the order volume being placed, while a low ratio indicates that the SOR struggles to place orders that get filled in the market. Typically, fill rate is a trade-off of execution cost, which means that to achieve a higher fill rate, we must take less favorable orders with higher execution costs, and to reduce execution costs, we must be willing to risk a lower fill rate. Using these three metrics we can accurately determine where the SOR excels and where the SOR can be improved, allowing us to robust SOR capable of succeeding in all market scenarios.

To extend the framework for multi-leg strategies and advanced SOR configurations, we can make our simulation modular. Similar to how in our simulation logic, we have an algorithm that converts regular market orders in each of the venues into special order types, we can add more multi-leg orders to our market by including them in this algorithm. Having this modular structure we can easily adjust what types of orders we want in our simulation without having to change the core logic of the simulation itself. We can also scale our backtester to be able to test more advanced SOR configurations by implementing a configuration system for the backtester which would allow us to change properties of our simulated environment. Factors that might be changed could include the number of venues we want to simulate, the dynamic of the properties of each venue, and the volatility and occurrence of rare events in our market. By changing these factors we would be able to test a wide range of SOR configurations using the same market simulation. As for extending the simulation to different asset classes, we can build our simulation logic using abstract classes for asset information and behavior, which would allow us to easily add asset-specific features at the time of simulation for SOR to trade. Overall, the key to ensuring that our backtester is scalable and extendable is ensuring that we follow a modular framework that can easily be built upon or simplified to suit our specific simulation needs.