Navigating Smart Order Routing in Fixed-income Trading with Linear Programming & Dijkstra's Algorithm

Cassel Robson[1] [robs7000@mylaurier.ca], Marko Misic[1] [misi2700@mylaurier.ca], Erika Capper[1] [capp9750@mylaurier.ca], Mila Cvetanovska[1] [cvet1400@mylaurier.ca], Majid Ghasemi[1] [ghas1560@mylaurier.ca], Fadi Alzhouri[2][fadi.alzhouri@concordia.ca], and Darish Ebrahami[1] [debrahimi@wlu.ca]

[1] Wilfrid Laurier University, Waterloo, Ontario, Canada

[2] Concordia University, Montreal, Quebec, Canada

Abstract. Over the past decade, electronic trading in financial markets has seen a rise in technological advancements. While foreign exchange (FX) and equities trading benefit from high demand and abundant data, the fixed-income asset class faces resistance due to credit traders' reluctance and liquidity constraints. The FX and equities traders leverage Smart Order Routing (SOR) algorithms for efficient order execution, a practice not widely adopted in fixed-income trading. This paper introduces the Fixed-income Order Route (FIOR) problem, addressing multi-venue order routing challenges in the trading of fixed-income securities. We propose a unique strategy that demonstrates pathfinding through a combination of optimization methodologies and evaluate the strategy in a simulated market environment.

Keywords: Fixed-income, Sorting Algorithms, Electronic Trading, Foreign Exchange, Securities, Bond, Pathfinding, Market Data, Order Execution, Liquidity, G-Spread, Dijkstra's Algorithm, Kruskal's Algorithm, Slippage, Smart Order Routing, Markov Decision Process, Reinforcement Learning, Exchange, Block Order

1 Introduction

The financial landscape has witnessed significant evolution propelled by rapid technological advancements in electronic trading. While the foreign exchange and equities markets have reaped the benefits, the fixed-income asset class has encountered resistance stemming from credit traders' reluctance and various constraints. Despite comparable market volumes, the fixed-income market continues to rely on traditional 'Request for Quote' (RFQ) methods for order execution, unlike its FX and equities counterparts that have embraced pathfinding for efficient trading strategies [1].

This research paper introduces a pivotal challenge in fixed-income trading named the Fixed-income Order Routing (FIOR) problem. The FIOR problem addresses the multi-venue order routing challenges specific to fixed-income securities, aiming to overcome the existing limitations associated with conventional execution methods. Our research proposes innovative strategies that leverage linear optimization – a method applicable to the solution of problems in which the objective function and the constraints appear as linear functions of the decision variables [10] – to navigate liquid markets and Djiktra's algorithm to navigate illiquid markets, minimizing the slippage cost incurred on block trades.

To evaluate the overall performance of the proposed solution to the FIOR problem, we implement it in a simulated environment that incorporates discrete market data. Comparative analysis with Kruskal's algorithm – an algorithm fundamentally used to find the minimum spanning tree of connected weighted graphs - serves to assess the effectiveness and efficiency of the proposed approach using Dijkstra's algorithm to navigate illiquid markets where slippage is incurred and demonstrates the impact of respecting routing constraints. By introducing this algorithm to address the challenges unique to fixed-income trading, this research contributes to building a foundation for smart order routing in this market segment. As the fixed-income landscape continues to evolve, our solution emerges as a different approach to order execution. This approach seeks to align fixed-income trading with the efficiency and sophistication witnessed in the FX and equities markets.

Throughout this paper, we will delve into related work in the field (Section II), present our system model (III) and problem description (IV), review our proposed approach (V), share a comprehensive performance evaluation (VI) and our final section will conclude the paper, summarizing the discoveries and discussing potential direction for future research

2 Related Works

2.1 FX Execution Algorithms and Market Functioning

This article emphasizes that Execution Algorithms (EA) play a crucial role in supporting price discovery and market functioning within an increasingly fragmented market [11]. While EAs are instrumental in navigating the intricacies of the FX market, particularly through Smart Order Routing (SOR), their widespread use contributes to ongoing changes in market structure and introduces new risks. SOR involves crucial decisions about which liquidity to access, in which pool, and at what time and price, serving as the interface between orders and the market. Recent advancements in SOR include considerations of factors such as quoted size, queue length, and venue efficacy, guiding the allocation of orders across trading venues. Machine learning techniques have been incorporated into EAs, enhancing their adaptability to real-time market conditions. Additionally, the impact of price differences on order routing is emphasized, with assumptions about lower price impact for internalized orders and larger impact for trades at external venues, verified through transaction cost analysis. This study underscores the evolving landscape of execution algorithms in response to market dynamics and the increasing role of technology in shaping financial trading strategies [11].

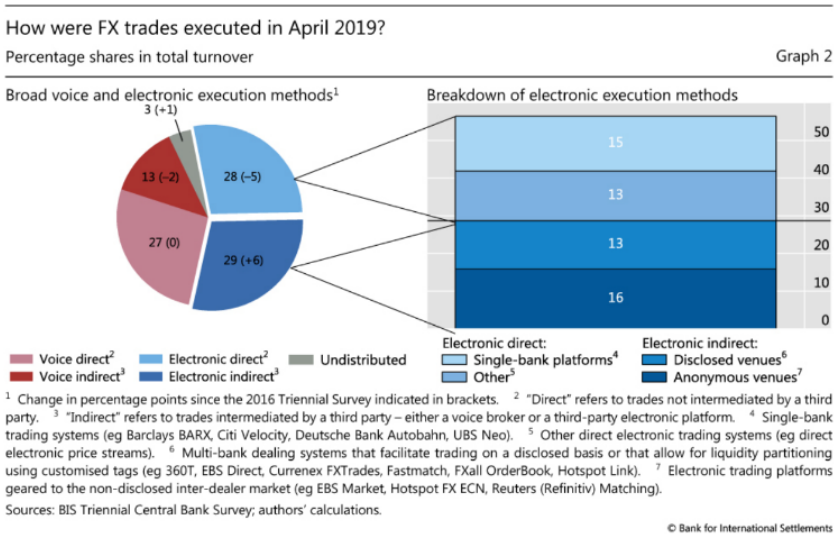


Fig. 1. "The main trend in FX trade execution has been increases 'electronification' – deeper penetration of the market by electronic and automated trading." – "Electronic trading enables automated and continuous trading, bringing together participants with diverse trading interests so that they can more seamlessly adjust and redistribute financial exposures". The survey shown in the above figure shows that the share of FX trading executed electronically reached up to 58% in 2019, however, the segment where electronification progressed the fastest was dealer-to-customer transactions [11].

2.2 Electronic Trading in Fixed-income Markets and Its Implications

This research paper outlines the drivers and implications of the rising use of electronic and automated trading in fixed-income markets. "Electronic and automated trading has become an increasingly important part of fixed-income markets in recent years. They have replaced voice trading as the new standard for many fixed-income asset classes." [12].

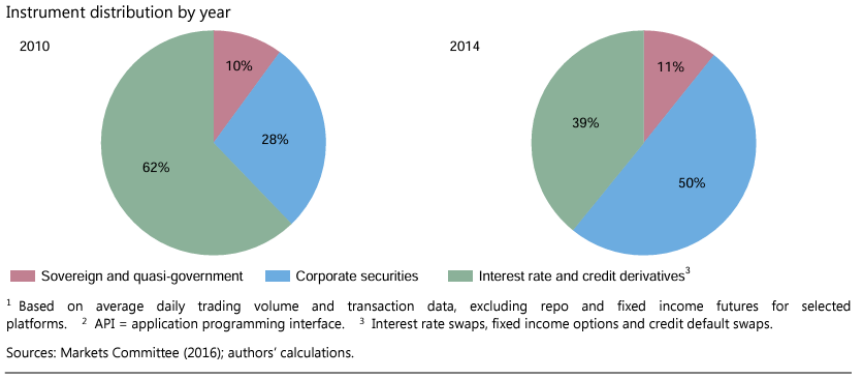


Fig. 2. One of the main drivers of the rise in electronic trading volumes has been a pickup in corporate bond trading. The figure above outlines the distribution of instruments that have seen an increase in trading volume. "Electronic trading via platforms may have helped overcome some of the liquidity challenges that have confronted credit markets by facilitating the matching of buyers and sellers and by reducing the reliance on individual dealers." [12]. It is with electronic trading (ET) that smart order routing (SOR) can be paired to improve order execution, and liquidity capture, and minimize slippage costs incurred, which is demonstrated in the later sections of this report.

2.2 Using Graph Algorithms & Reinforcement Learning to Solve Pathfinding Problems

This study emphasizes the use of different graph algorithms and how types of reinforcement learning can be used to solve complex path-finding problems optimally [4]. While conventional algorithms like Dijkstra's and A* are effective in deterministic and known environments with a single objective, they prove to be less effective in scenarios characterized by uncertainty, and conflicting objectives. To tackle such challenges, the study dives into Multi-Objective Reinforcement Learning (MORL), a paradigm that combines reinforcement learning with the capacity to handle stochastic environments and multiple conflicting objectives. The proposed approach, known as Multi-Objective Reinforcement Learning with Voting Q-Learning, leverages a Multi-Objective Markov Decision Process (MO-MDP) model, enabling the agent to learn a policy that simultaneously optimizes various objectives. The incorporation of voting methods from social choice theory within the Q-learning framework offers a promising strategy for efficient pathfinding in intricate and uncertain environments.

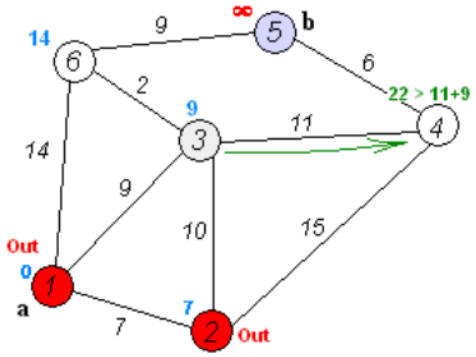


Fig. 3. Visualization of Dijkstra's algorithm. A simple and effective greedy algorithm for finding the shortest path between one node and all other nodes in a graph. It calculates the distance of each node one by one from a given source node, updating a list of shortest paths each time a longer path is undercut [4].

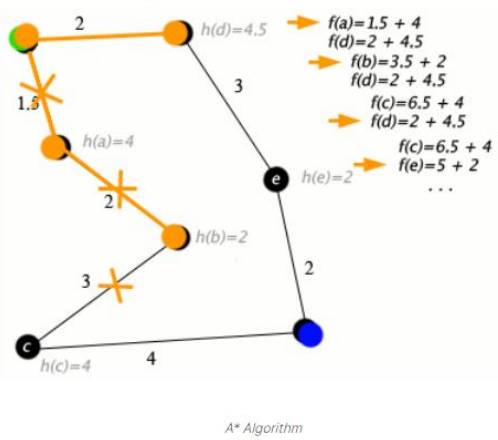


Fig. 4. Visualization of A*. Introduces a heuristic function that can be used to prioritize nodes, it is an informed algorithm. From a source node, it aims to create a path to a destination with the smallest possible cost by creating and minimizing a tree of paths between source and destination nodes until the terminal criteria are reached [4]

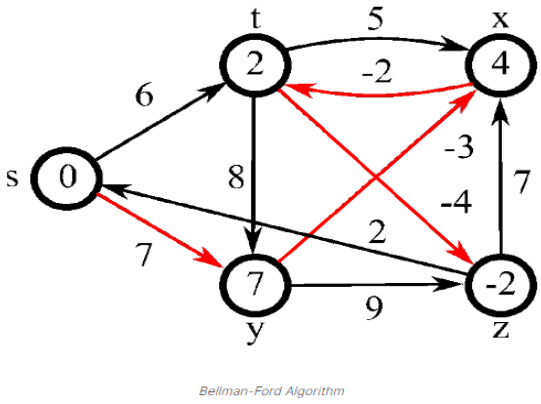


Fig. 5. Visualization of the Bellman-Ford Algorithm. It serves essentially the same purpose as Dijkstra's algorithm and A* and is often considered the best of the three approaches. It is versatile in the way that it can handle mixed signed weights [4].

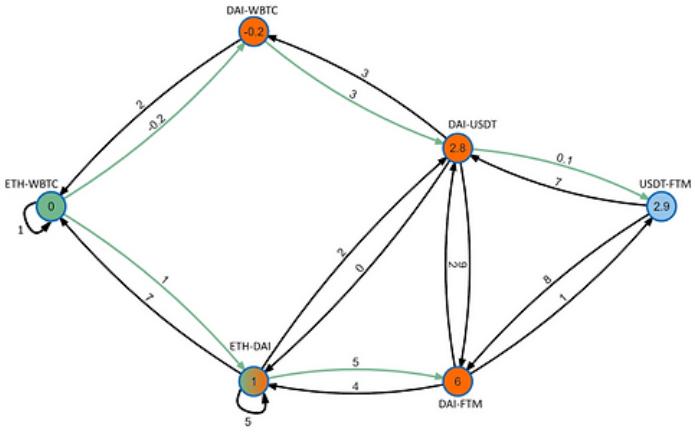


Fig. 6. Image from reference [4] that serves as a demonstrative example of how SOR can be treated as a pathfinding problem, to which our system model is heavily inspired.

2.3 The Markov Decision Process and Single Agent Reinforcement Learning

Markov decision processes (MDPs) offer a mathematical framework to model decision-making in scenarios where results are influenced by both random factors and the choices made by a decision-maker. [5] It has the following properties: Set of states *S*, set of actions *A*, Transition model *P(s' | s, a)*, and reward function *R(s, a, s')*. The MDP framework assumes the Markov property, which means that the future state depends only on the current state and action, not on the sequence of events leading up to that state. The goal in reinforcement learning is to find an optimal policy that maximizes the expected cumulative reward over time. The agent learns to associate higher rewards with actions that lead to more favorable states or outcomes and lower rewards with actions that lead to less desirable states [5]. Through this process, the agent refines its policy to make decisions that result in the most significant long-term reward. Applying this to our linear programming approach; assume we have a complete model of the MDP, including transitions and rewards. For any given state, we assume that the state's true value can be modeled by:

$$V^{\circ}(s) = r + \gamma \max_{a \epsilon A} \sum_{s' \epsilon S} P(s'|s, a) \cdot V^{\circ}(s') \qquad (1)$$

If we find the smallest value of *V(s)* that matches this requirement, that value makes exactly one of the constraints tight. With this, a "linear program can be formulated by writing a program like the one above for every state and minimize $\sum_{s \in S} V(s)$, subject to the union of all the constraints from all these subproblems we have reduced the problem of learning a value function to solve the linear program." [5][6].

3 System Model and Problem Description

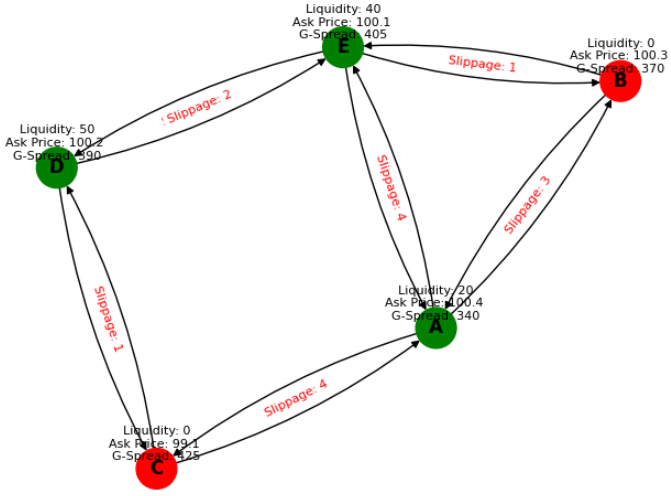


Fig. 7. Inspired by the model from [4], the above figure is a Matplotlib visualization of the system model representation. Circular nodes represent various exchanges in the market, these are referred to as "Pools of Liquidity". The exchanges in green represent pools that are currently offering liquidity for security *z*, while those in red are not, meaning they do not have security *z* available to sell to buyers. Connecting the exchanges are the slippage values that would be applied to market orders – orders that are not executed immediately at an asking price but wait for a seller to fulfill the order through a matching system – when liquidity is not readily available, making the difference between the price at order submission and the execution price increase with market volatility and time to execution

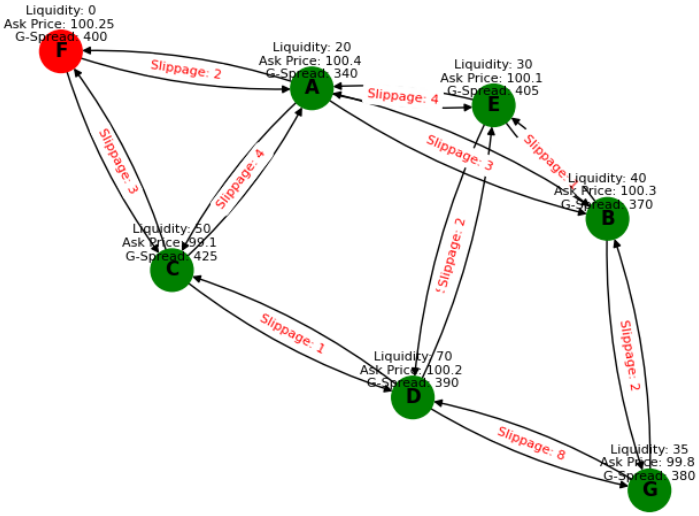


Fig. 8. Matplotlib visualization of an increased number of pools (exchanges).

3.1 System Model

Consider the model in Figure 7, which is a directed graph G = (P, S) where P is a set of liquidity pools (nodes) and S is a set of weighted edges representing slippage percentages applied to order prices – slippage is the difference between the original order price and the price at which an order was executed, slippage will typically be higher for securities with higher relative volatility. For simplicity, our model has fixed sample slippage values u in set S between each pool. In theory, slippage values can be calculated by:

$$u = \frac{b}{2} + 0.01\left(\sqrt{\frac{a}{d}}\right) \qquad (2)$$

where b represents the median percentage of the bid-ask spread divided by the closing price every day over a rolling period, d represents the median daily dollar volume of the security over the same rolling period, and a is the total value of the order in dollars. The pools represented by $k \in P$ are shaded by the liquidity of security where green signifies that the pool is offering liquidity for the security and red signifies that it is not. Each node has liquidity LQ_k^z of security z and a predefined G-Spread represented by $GSpr_k^z$– the calculated difference between security z's YTM and its benchmarked government securities' YTM, in bps, along with the spread corresponding price Ask_k^z for the security, inversely related to the width of the spread - the larger the spread, the lower the price. Once order o_i at Ask_k^z and liquidity LQ_k^z of a node is exhausted, the u value representing the slippage, the Ask_k^z, and the LQ_k^z of the exchange between pools is recalibrated adjusting for the market effect of the order executed for every block trade executed – order of 10 or more securities in this model. Slippage of a security only occurs once liquidity LQ_k^z is exhausted at all available trading venues. Slippage occurs as market orders are placed and sit in the order book, waiting for a liquidity provider, executing at a higher price than the original Ask_k^z. The Ask_k^z changes after the LQ_k^z collected based on a variety of metrics such as trading volume, liquidity constraints, volatility and other variables included within the dynamics of the market. However, after illiquidity of LQ_k^z is reached, all u directed to k are applied on the Ask_k^z.

3.2 Problem Definition

FIOR is designed to determine the best route for each order i from starting pool k_i so that the cost represented by c_i is minimized for the order volume V_i by considering the LQ_k^z of each pool k along with their $GSpr_k^z$ & Ask_k^z, making the decision to execute the purchases of all available fixed-income securities at pool k to incur the slippage u on Ask_k^z or to traverse to another pool for a better price for more securities. The problem's complexity comes into play when LQ_k^z of k is executed in full and edge weights are applied to the ask price at pool k (Ask_k^z), and this value can be negative which would be beneficial or could be positive and increase the price of security z. This means it could be most valuable to continue routing the order to that specific pool or it could be best to avoid that pool after exhaustion of LQ_k^z. The example in Fig. 1. shows an order that intends to purchase 120 *Microsoft (NASDAQ: MSFT)* corporate bonds, beginning at pool *A*, the agent (router) evaluates the exchanges (pools of liquidity) based on the liquidity of MSFT corporate bonds, G-spread of the security, and a correlated ask price quoted by market makers at said exchanges (pools), the only pool with open exchange with *P(A)*, and supply of MSFT corporate bonds is *P(E)*. Identifying that the *P(E)* has the largest spread and lowest ask at \$100.10/bond, all $LQ_E^z = 40$ is exhausted and the remaining volume V_i becomes 80 bonds. Evaluating the lowest ask between *P(A)* and *P(D)* which both have an open exchange with *P(E)*, the router (agent) exchanges with *P(D)* with the next largest spread and lowest ask at \$100.20/bond, exhausting $LQ_D^z = 50$, leaving 30 bonds in remaining order volume V_i. Identifying that *P(A)* is the only remaining pool offering liquidity ($LQ_D^z = 20$) at an ask of 100.40/bond. Once the block trade is executed at *P(A)*, 10 bonds remain in the order. The router identifies that slippage between *P(A)* and *P(B)* is minimal and executes the remaining 10 bonds between the exchanges, incurring a slippage cost of \$20.06, bringing the total cost c_i = \$12045.06 for a complete order of 120 MSFT corporate bonds. As seen in the example, the order i is split into several blocks of orders at each pool and the router evaluates the neighbour node's Ask_k^z and LQ_k^z along with slippage u to find the most inexpensive c_i for V_i. Due to the complexity of the markets, Ask_k^z and LQ_k^z are changing continuously throughout market hours and fixed-income security z can be completely unavailable in pool k so the agent (router) must be able to identify this when it happens so it can change routes to different pools. When a new order is placed into the market at pool k_i, the same problem occurs again, and the router must divide the order based on each pool k Ask_k^z and LQ_k^z along with slippages between the fill and execution of the order.

Problem Description: Given a graph G which consists of P(I) nodes connected through S slippages. The problem is, for each order volume V_i enters the markets, to find the optimal route of pools through exchanges of securities that minimizes the total cost of the order.

4 Problem Formulation

The problem is mathematically formulated as a linear programming problem. All notations used are listed in Table 1. Let c_i be the cost of the order o_i within set O. The objective of the optimization model is to minimize the cost for all orders within set O using an agent router. The formula for this minimal cost for o_i is:

$$c_i = Min\ (\sum_{k \epsilon P}(Ask_k^z \cdot LQ_k^z) + (S \cdot (V_i - LQ_k^z))) \qquad (3)$$

Objective Function: Minimize c_i, the total cost of orders across pool k, by minimizing the product of ask prices and security liquidity as well as the slippage on the remaining order volume once the illiquid state is reached given by ILS_{o_i}.

There are strict constraints that must be taken into consideration when formulating this problem which will be explored in the upcoming sections.

Table 1. Problem Formulation Notation Reference Table

Parameters	
P	Set of Pools (nodes)
S	Set of Slippages (edges)
O	Set of Orders
o	Individual Order
i	Order Identifier
u	Slippage Cost Percentage
k	Individual Pool
Z	Fixed-income Security Type
Ask_k^z	Ask Price of Security z at Pool k
LQ_k^z	Liquidity of Security z at Pool k
Vi	Order Volume of Order i
SPo_i	Starting Pool of Order o_i
N_{SP}	Set of Neighboring Pools of Starting Pool SP
ONP_{o_i}	Set of Optimal Neighboring Pools of Order o_i
OP_{SP}	Optimal Pool to exchange with from Starting Pool SP
$GSpr_k^z$	G-Spread of Security z at Pool k

Decision Variables		
$GSpr^{z}{}_{<k,\,k'>}$	$\in \{0,1\}$	Indicate whether ask price at k' is greater than k, to determine whether to traverse to k
$u^z_{<k,k'>,<k,k''>}$	$\in \{0,1\}$	Indicate whether slippage $u^{z}{}_{<k,\,k'>}$ between <k, k'> is greater than slippage $u^{z}{}_{<k,\,k''>}$ between <k, k''>, to determine whether to exchange with k' {1} or k'' {0}
LQ_k^z	$\in \{0,1\}$	Indicate whether liquidity at $k > 0$
ILS_{o_i}	$\in \{0,1\}$	Indicate whether order o_i has reached the liquidity illiquid state if $\sum(LQ_k^z) = 0$ *for all* $k \in P$

4.1 Routing Constraints

Liquidity Routing Constraints. In this subsection, the path for each order is constructed from its source (starting pool) to its destination (optimal neighboring pools). Let $LQ_k^z \in \{0,1\}$ indicate whether pool k has security type z available for limit orders. Let the source and destination of order o_i be SPo_i and ONP_{o_i} respectively. Now, $LQ_k^z = 1$ if the destination of order o_i has liquidity of limit orders for security z greater than zero. Otherwise, $LQ_k^z = 0$, and pool k cannot be considered optimal. The mathematical formulation for this constraint is written as follows:

$$LQ_k^z = 1, \forall o \in O, k \in N_{SP} \; and \; k = \; ONP_{o_i} \qquad (4)$$

G-Spread Routing Constraint. Following the liquidity routing constraints, a set of optimal neighboring pools is defined where "optimal" is defined by $LQ_k^z = 1$ for each pool k in the set ONP_{o_i}. Let pool k be the first pool in set ONP_{o_i}, $GSpr^{z}{}_{<k,\,k'>} = 1$ if the G-Spread $GSpr_k^z$ at pool k is more than G-Spread $GSpr_{k'}^z$ at pool k' for each pool k in set ONP_{o_i}. The optimal pool given by OP_{SP} represents the optimal pool to exchange with the starting pool, and it is the result of operations that perform within the G-Spread routing constraint. The mathematical formulation for this constraint is written as follows:

$$min_{k:<k,k'> \in ONP_{o_i}} \left(GSpr^z_{<k,k'>} = 1\right), \forall o \in O, < k, k' > \in \; ONP_{o_i} \qquad (5)$$

Slippage Routing Constraint. Once the illiquid state is reached, given by $ILS_{o_i} = 1$, the path for the remaining order volume V_i is constructed from its source, represented by current pool k, and its destination, represented by a decision between pools k' and k''. Let N_{SP} be a set of all direct neighbors where SP (starting pool) = k. Using the decision variable $u^z_{<k,k'>,<k,k''>}$, the mathematical formulation for this constraint is written as follows to minimize slippage u induced by order o_i of security z:

$$min_{k:<k,k'>,<k,k''>\in N_{SP=k}}\left(u^z_{<k,k'>,<k,k''>} = 1\right), \forall o \in O, k', k'' \in N_{SP=k} \quad (6)$$

Illiquidity Implication Constraint. The illiquid state given by $ILS_{o_i} = 1$, is reached once liquidity $LQ^z_k = 0$ for all pools $k \in P$. Implying that there remains no more limit order offering at any pool $k \in P$ of security z. This can be represented by the following mathematical formulation:

$$ES_{o_i} = 1, \sum_{k=SP_{o_i}} (LQ^z_k) = 0, \forall o \in O, k \in P \quad (7)$$

5 Proposed Approach

Algorithm 1: FIOR: Finding the best neighboring pool based on ask prices, liquidity, and slippages for order o given graph G, state {}, and action{}.

1. **Input:** Graph G(P, S), starting node
2. **Result:** PoolsTraversed[], SlippageExchanges[], TotalCost, SlippageCost, Reward
3. Order ← FIOR(Graph G, State, Actions, OrderID, OrderLog, OrderAmount, StartingPool, TotalCost)
4. Visited[]
5. **If** TotalLiquidity ≥ orderAmount then
While orderAmount > 0 Then
TotalCost, PoolTraversed, OrderTicket, OrderAmount, BondsBought ← Transition(NextNode, Visited)
8. PreviousPool = StartingPool
9. StartingPool = PoolTraversed
10. NextNode ← OptimalNodeLP(G, State, StartingPool, PreviousPool, BondsBought)
11. **Else:**
12. **While** Sum(State["Liquidity"].Values()) > 0 Then
13. TotalCost, PoolTraversed, OrderTicket, OrderAmount, BondsBought ← Transition(NextNode, Visited)
14. PreviousPool = StartingPool
15. StartingPool = PoolTraversed
16. NextNode ← OptimalNodeLP(G, State, StartingPool, PreviousPool, BondsBought)
17. SlippageExchanges, TotalCost, SlippageCost, Reward = illiquidState(PoolTraversed)
18. **Return** Visited[], Reward, SlippageExchanges[], PoolsTraversed[], TotalCost, SlippageCost

Our implementation of smart order routing for fixed-income securities is outlined in this section. The above algorithm uses a combination of linear programming and optimization to identify the most affordable liquidity pools based on the ask prices offered for limit orders at each pool, the liquidity at the pool, and after illiquidity is reached, the slippage values between each exchange that ultimately increase the total cost of the order. The program works in stages based on the concept of order execution. Each pool has a predetermined number of available bonds and once those bonds have all been bought, if the order still has not been completely fulfilled, the program enters the "illiquid state" in which the *illiquidState()* function is called. This function works in a very similar way to the FIOR_LP function, except instead of evaluating pools based on ask prices, we're evaluating exchanges between pools based on the slippage values between them, with the goal of minimizing the slippage.

The *OptimalNodeLP* and the *OptimalSlippageLP* functions work very similarly as they determine the optimal neighboring pool and exchange based on linear programming, considering both ask prices, liquidity, and slippage values.

The initial guess for the optimization, denoted as x0, serves as the starting point and is initialized as a vector of zeros. Coefficients (c) are linked to the objective function, representing either ask prices or slippages for available neighbors. Variable bounds, denoted as 'bounds', are defined to ensure that liquidity remains greater than zero, and slippages are greater than or equal to zero. The objective function, calculated as the dot product of ask prices or slippages and coefficients, is encapsulated within the *'objective_function.'*. Constraints are imposed to ensure orders can be executed at each node at most once for pools and that only one neighbor can be selected for slippages.

The subsequent phase employs linear optimization, utilizing the 'minimize' function from the SciPy optimization method (Python library). The overarching goal is to minimize the dot product of ask prices and coefficients, as well as slippage, while adhering to specified constraints. Result extraction involves determining the index of the optimal neighbor based on ask prices or slippage values obtained from the linear programming results. The final output includes either the optimal neighbor alone or both the optimal neighbor and the corresponding slippage.

Therefore, the selection criteria of the linear programming approach of FIOR are based on three factors; ask prices, liquidity, and slippage values. Lastly, a summary of the order is returned that outlines the total cost, the pools visited, the reward summary, the slippage cost, and the slippage exchanges.

6 Performance Evaluation

To evaluate the performance of our approach to the FIOR problem, we test Dijkstra's algorithm implementation against Kruskal's algorithm implementation to see how they compare within illiquid markets. Kruskal's algorithm is a traditional approach to solving graph-like problems that involve weighted edges and directed graphs where the minimal cost is found through the edge weights [8]. We choose to benchmark the linear programming model against Kruskal's algorithm to check the spread between a model that must conform to the constraints of a routing problem, to an algorithm that does not consider the reachability of destination nodes from a source node. The benchmark was done across a range of order sizes and market access (# of pools), comparing slippage cost as a percentage of the total order cost as the order size, or the number of pools increases. For consistency, for each variation of the 'pools' test, the order volume V_i stayed consistent at 250 corporate bonds, therefore the impact of varying liquidity, and number of venues (pools) become clear, revealing a non-linear pattern, whereas the market liquidity for the 'order size' test stayed consistent at 210 corporate bonds.

6.1 Slippage as a Percentage of Total Cost Versus Number of Pools

Referring to Figure 9, it is evident that as the number of pools increases, the impact of slippage on the cost of order execution settles as the market has liquidity to offer. As expected, using Kruskal's algorithm to identify the path between pools with the lowest slippage figure – regardless of what pool the agent currently rests, and what pools are accessible – results in a lower slippage figure for every iteration. However, the implementation of Dijkstra's algorithm follows the competing curve quite closely, effectively minimizing the slippage incurred, while respecting the constraints of the FIOR problem.

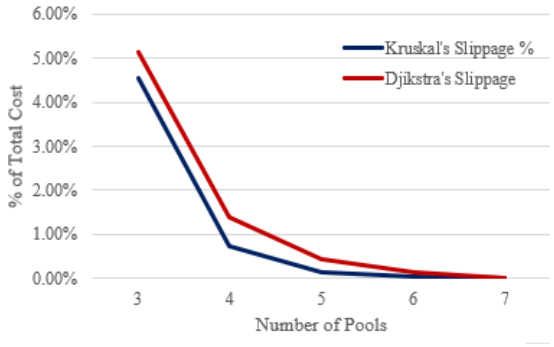


Fig. 9. Slippage cost % of total order cost as the number of pools increases, increasing market liquidity for the requested security.

6.2 Number of Pools Versus Total Order Cost

Demonstrating the inverse relationship between the number of pools and total cost of the order, it is evident that as the market becomes more liquid, the cost of the order lowers and the slippage incurred substantially decreases. Correlated to the test demonstrated in Fig. 7, the total cost of the order increases as slippage costs increases, demonstrating a linear relationship. As expected, Kruskal's algorithm results in a lower cost as it does not consider the reachability of destination nodes from starting nodes, but Dijkstra's follows quite closely while respecting direction, and reachability of the model, and conforms to the slippage routing constraint mentioned in Section 4.

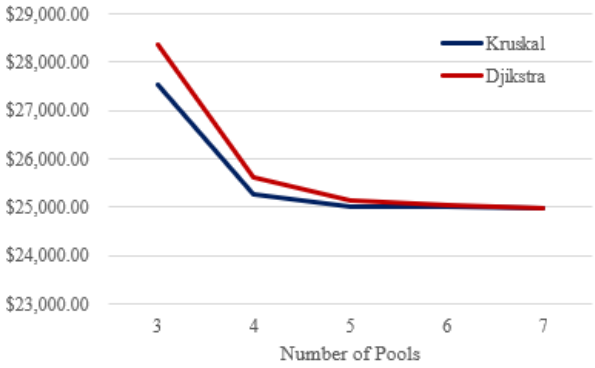


Fig. 10. Total order cost as the number of pools increases, and with it, market liquidity for the security.

6.3 Relationship between Cost c_i and Order Volume V_i

With order size growing in volume, security z quickly become illiquid, and slippage takes a quicker effect on the cost of the larger orders. Not only does a larger order increase the slippage applied to order, but the ratio of slippage cost to total order cost increases all else equal. See Figures 11-14 below.

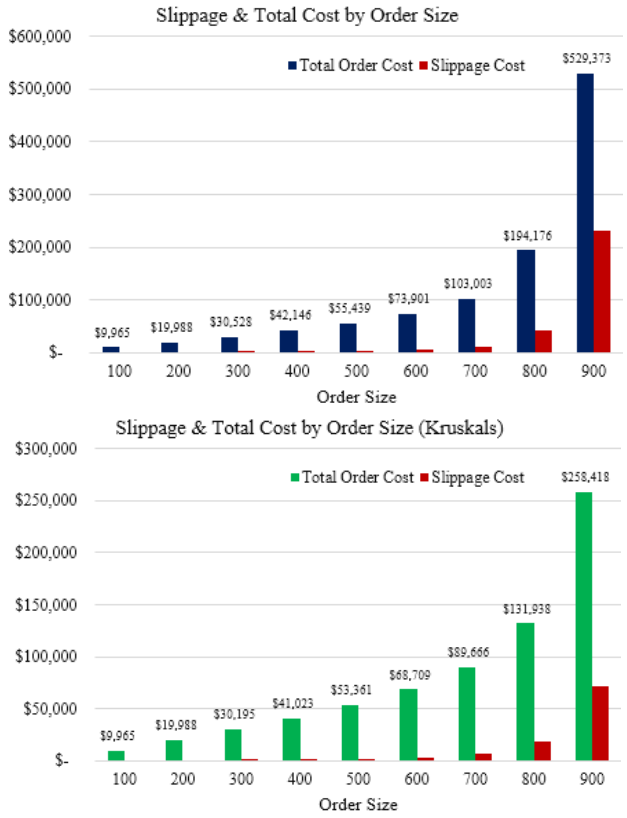
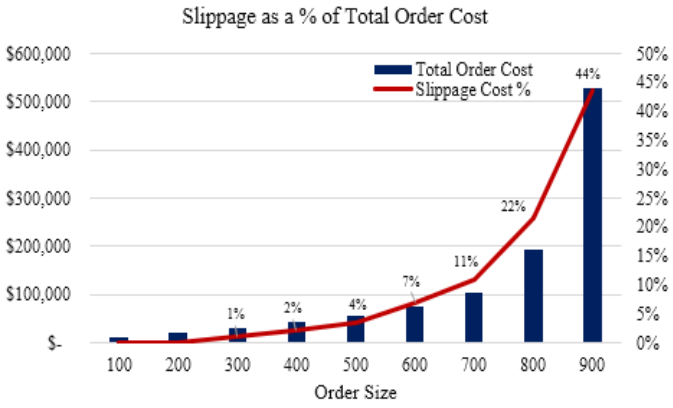


Fig. 11 (Dijkstra) - 12 (Kruskal). This chart demonstrates the non-linear relationship between order volume and total order cost. As order size increases, and orders are executed, liquidity becomes scarce and the price of the security moves away from the buyer, which is referred to as slippage (shown in red). Our implementation of Dijkstra's algorithm, used to navigate the illiquid markets (state where slippage is applied in our model), while incurring a higher cost due to its abiding by slippage routing constraint, whereas Kruskal's algorithm does not, representing the lowest possible cost. Our performance is evaluated on the spread between the results of the two methods.



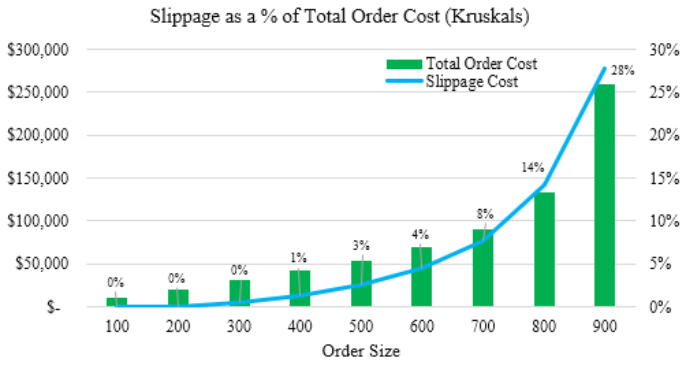


Fig. 13 (Dijkstra) – 14 (Kruskal). The line plots in the above figures demonstrate the non-linear relationship between slippage as a percentage of total order cost and increases in order size. While still not the majority of the total order cost, the spread between slippage cost and total order cost quickly narrows as the higher order sizes eat up market liquidity, and incurs significant slippage, regardless of the algorithm used, both of which provide optimal routes given their respective constraints or lack there of.

7 Conclusion

This study has addressed the intricate dynamics of electronic trading in financial markets, with a particular focus on the challenges faced by the fixed-income asset class. Through a comprehensive evaluation of the proposed approach's performance in a simulated environment with discrete market data, this research has provided valuable insights into its effectiveness when benchmarking against Kruskal's algorithm for navigating illiquid markets, which, in theory, finds the minimal slippage cost, but does not respect the constraints of the routing problem in the same manner that Dijkstra's is used to generate minimum weighted paths between starting destination nodes.

Acknowledging the inherent complexity of the fixed-income trading landscape, characterized by numerous variables that contribute to its intricacies, this research serves as a foundational step. By introducing the FIOR problem and proposing innovative strategies, our aim is to act as a catalyst for further exploration and development in this field. The potential avenues for future research are expansive, encompassing the refinement of FIOR strategies, exploration of additional variables, and adapting to evolving market conditions. This study is a starting point, charting a trajectory toward a more sophisticated and efficient future for fixed-income trading through technological innovation and data-driven strategies.

References

1. S.-H. Chen, M. Kaboudan, and Y.-R. Du, "Algorithmic Trading in Practice," *The Oxford Handbook of Computational Economics and Finance*, no. 10, Art. no. 9780199844371, 2018, doi: 10.1093/oxfordhb/9780199844371.013.12.
2. A. G. Barto and R. S. Sutton, "Reinforcement learning in artificial intelligence," in *Advances in psychology*, 1997, pp. 358–386. doi: 10.1016/s0166-4115(97)80105-7.
3. B. Tozer, T. A. Mazzuchi, and S. Sarkani, "Many-objective stochastic path finding using reinforcement learning," *Expert Systems With Applications*, vol. 72, pp. 371–382, Apr. 2017, doi: 10.1016/j.eswa.2016.10.045.
4. "Pathfinding and route optimization | Deeplink Network," *Deeplink Network*. https://www.deeplink.network/pathfinding-and-route-optimizations
5. X. Wang, Z. Yang, G. Chen, and Y. Li, "A reinforcement learning Method of solving Markov decision processes: an adaptive exploration model based on temporal

difference error," *Electronics*, vol. 12, no. 19, p. 4176, Oct. 2023, doi: 10.3390/electronics12194176.
6. "How can we use linear programming to solve an MDP?," *Artificial Intelligence Stack Exchange*. https://ai.stackexchange.com/questions/11246/how-can-we-use-linear-programming-to-solve-an-mdp
7. W. Van Heeswijk PhD, "Using linear programming to boost your reinforcement learning algorithms," *Medium*, Sep. 18, 2022. [Online]. Available: https://towardsdatascience.com/using-linear-programming-to-boost-your-reinforcement-learning-algorithms-994977665902
8. H. Li, "Research and improvement of Kruskal algorithm," 2017. https://www.semanticscholar.org/paper/Research-and-Improvement-of-Kruskal-Algorithm-Li-Xia/6204351a3614cc6acada2867f10175748e44aa2d?p2df
9. J87as, "What is Smart Order Routing?," *CenterPoint Securities*, Apr. 22, 2022. https://centerpointsecurities.com/smart-order-routing/
10. Libretexts. (2023, March 11). *8.2: Linear Optimization*. Engineering LibreTexts. https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/08%3A_Optimization/8.02%3A_Linear_Optimization#:~:text=Linear%20optimization%20takes%20into%20account,the%20most%20profitable%20production%20plan.&text=Linear%20optimization%20has%20been%20used,manufacturing%20plants%20to%20various%20warehouses.
11. Schrimpf, A., & Sushko, V. (2019, December 8). *FX trade execution: Complex and highly fragmented*. The Bank for International Settlements. https://www.bis.org/publ/qtrpdf/r_qt1912g.htm
12. Bech, M., Illes, A., Lewrick, U., & Schrimpf, A. (2016, March). *Electronic trading in fixed-income markets and its implications*. bis.org. https://www.bis.org/publ/qtrpdf/r_qt1603h.pdf