FIOR: Navigating the Fixed Income Order Route with Linear Programming

Cassel Robson, Mila Cvetanovska, Erika Capper, Marko Misic

CP486- Artificial Intelligence, Wilfrid Laurier University

Term Project – December 10th , 2023

***Abstract*- Over the past decade, electronic trading in financial markets has seen a rise in technological advancements. While FX and equities benefit from high demand, and abundant data, the fixed income asset class, faces resistance due to credit traders' reluctance and liquidity constraints. FX and equities traders leverage AI for efficient order execution, a practice not widely adopted in fixed income. Despite having a comparable market volume, fixed income execution relies on traditional RFQ methods. This paper introduces the Fixed Income Order Routing (FIOR) problem, addressing multi-venue order routing challenges in fixed income securities. We propose various strategies utilizing pathfinding through reinforcement learning. Evaluating FIOR's overall performance involves implementing it in a simulated environment with real-time market data, comparing outcomes with existing strategies. As fixed income dark pools expand amidst a high-interest-rate environment, FIOR offers a transformative approach to order execution in the fixed income market, aligning it with the efficiency seen in FX and equities.**

***Index Terms*- Fixed Income, Sorting Algorithms, Electronic Trading,**

I. INTRODUCTION

The financial landscape has witnessed significant evolution propelled by rapid technological advancements in electronic trading. While the foreign exchange (FX) 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 equity counterparts that have embraced Artificial Intelligence (AI) for efficient trading strategies [1].

This research paper introduces a pivotal challenge in fixed income trading known as the Fixed Income Order Route (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 pathfinding through reinforcement learning, a machine learning technique, known for its adaptability and ability to optimize decision-making processes [2].

To evaluate the overall performance of the proposed FIOR solution, we implement it in a simulated environment that incorporates market data. Comparative analysis with a combination of DFS and Kruskal's algorithm serves to assess the effectiveness and efficiency of the proposed approach. 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, the FIOR solution emerges as a revolutionary approach to order execution. This novel approach seeks to align fixed income trading with the efficiency and sophistication witnessed in the FX and equities markets.

Through this paper we will delve into related work in the field, present our system model and problem description, review our proposed approach, share a

comprehensive performance evaluation and our final section will conclude the paper, summarizing the discoveries and discussing potential direction for future research.

II. RELATED WORK

A. *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 [3]. While EAs are instrumental in navigating the intricacies of the Foreign Exchange (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.

B. 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.

C. *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 Actions(S), Transition model $P(s' \mid s, a)$ 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 [6]. Through this process, the agent refines its policy to make decisions that result in the most significant long-term reward.

III. SYSTEM MODEL AND PROBLEM DESCRIPTION

A.System Model

Consider the graph in Figure. 1 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 of trading in the pool which are adjusted on a rolling basis. Each edge u in set S is a calculated slippage cost using the formula $b/2 + 0.01\ sqrt(a/d)$ where b represents the median percentage of the bid-ask spread divided by the closing price every day over the last six weeks while d represents the median daily dollar volume of the security over the last 6 weeks, and a is the amount of money that will be bought in the specified security. The pools represented by k belong to $k \in P$ are shaded by the availability of the fixed income security where green signifies that the pool contains the specific security and blue signifies that it does not. It is considered that the pool the security is currently starting at can be traded within that

pool if it currently has the security. Each node has availability AV^z_k of security z and a predefined ask price represented by Ask^z_k for each individual z. Once order AV^z_k at Ask^z_k of a node is exhausted, the u value representing the slippage, the Ask^z_k , and the AV^z_k of the exchange between pools is recalibrated adjusting for the market effect of the order executed for every market order of 10 bonds. Slippage of a security only occurs once availability AV^z_k is exhausted as there exists no more available fixed income securities at the original ask price. Slippage occurs as market orders for the securities are placed and the orders are executed at a higher or lower execution price. The Ask^z_k changes after the

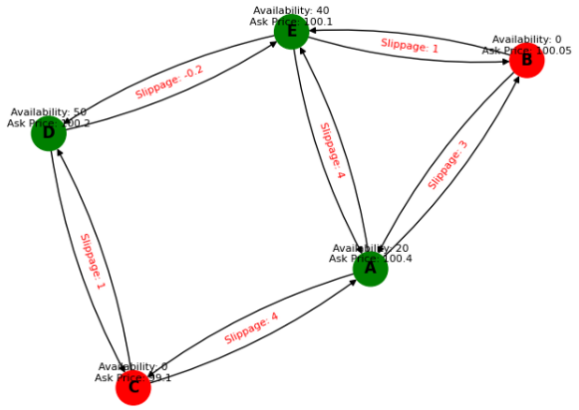

Figure 1. System model graph of FIOR approach

AV^z_k 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 the first exhaustion of AV^z_k , all u directed to k are applied on the Ask^z_k .

B. 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 AV^z_k of each pool k along with their Ask^z_k and the decision to execute the purchases of all available fixed income securities at pool k to incur the slippage u on Ask^z_k or to traverse to another pool for a better price for more securities. The problem's complexity comes into play when AV^z_k of k is executed in full and edge weights are applied to the pools Ask^z_k , and this value can be negative which would be beneficial or could be positive and increase the price of security z. Which 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 the first exhaustion of AV^z_k. The example in Fig. 1 shows an order that intends to purchase *100 MSFT* corporate bonds, beginning at pool A, the agent evaluates the liquid pools based on availability of MSFT corporate bonds at a predetermined ask price, 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 lowest ask at \$100.10/bond, all of $AV^z_E = 40$ is exhausted and remaining volume V_i becomes 80 bonds. Evaluating the lowest ask between *P(A)* and *P(D)* which both have open exchange with *P(E)*, the FIORP exchanges with *P(D)* with lowest ask at \$100.20/bond, purchasing 1 bond of $AV^z_D = 50$, recognizing that's optimal to exchange the minimum requirement of 1 bond with *P(D)* and immediately exchange back to *P(E)* from *P(D)* with –0.2% slippage, leaving 79 bonds in remaining order volume V_i. The FIORP exchanges a market order of 10 bonds (maximum) at \$100.10 * (1-0.002) = \$99.90/bond, $V_i = 69$. Thereafter, 1 more bond is exchange from *P(E)* to *P(D)* at the original Ask^z_D = \$100.20/bond, $V_i = 68$, and immediately exchanges back to *P(E)*, taking advantage of the negative slippage. This repeated exchange between *P(D)* and *P(E)* repeats until order volume V_i becomes 0 and total cost c_i = \$11922.64 for 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 nodes Ask^z_k and AV^z_k along with slippage u to find the most inexpensive c_i for V_i . Due to the complexity of the markets, Ask^z_k and AV^z_k are changing continuously throughout market hours and fixed income security z can be completely unavailable in pool k so the 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^z_k and AV^z_k along with slippages between the fill and execution of the order.

Problem Description I: 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 total cost of the order $c_i = \Sigma(Ask^z_k * AV^z_k) + (S * (V_i - AV^z_k))$.

IV. PROBLEM FORMULATION

The problem is mathematically formulated as a multi-inter-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:

$$(1)\ c_i = \text{Min} \sum_k (Ask^z_k \cdot AV^z_k) + (S * (V_i - AV^z_k))$$

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

TABLE 1

NOTATION USED IN PROBLEM FORMULATION

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
$Ask^z k$		Ask Price of Security z at Pool k
$AV^z k$		Availability 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
$ONPo_i$		Set of Optimal Neighboring Pools of Order o_i
OP_{SP}		Optimal Pool to exchange with from Starting Pool SP
Decision Variables		
$Ask^z_{<k, k'>}$	$\in \{0,1\}$	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}
AV^z_k	$\in \{0,1\}$	Indicate whether availability at $k > 0$
ESo_i	$\in \{0,1\}$	Indicate whether order oi has reached availability exhaustion state if $\sum(AV^z_k) = 0$ for all $k \in P$

Objective Function: Minimize c_i, the total cost of orders across pools k, by minimizing the sum of --> $Ask^z_k * AV^z_k + (S * (V_i - AV^z_k))$, optimizing the product of ask prices and security availability as well as the slippage on the remaining order volume once the exhaustion state is reached given by ESo_i.

A. Routing Constraints

i. Availability Routing Constraints

In this subsection, the path for each order is constructed from its source (starting pool) to its destination (optimal neighboring pools). Let $AV^z_k \in \{0, 1\}$ 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 $ONPo_i$ respectively. Now, $AV^z_k = 1$ if the destination of order o_i has availability of limit orders for security z greater than 0. Otherwise, $AV^z_k = 0$, and pool k cannot be considered optimal. The mathematical formulation for this constraint is written as follows:

$$(2)\ AV^z_k = 1,\ \forall o \in O,\ k \in N_{SP} \text{ and } k = ONPo_i$$

ii. Ask Price Routing Constraint

Following the availability routing constraints, a set of optimal neighboring pools is defined where "optimal" is defined by $AV^z_k = 1$ for each pool k in the set $ONPo_i$. Let pool k be the first pool in set $ONPo_i$, $Ask^z_{<k, k'>} = 1$ if the ask price Ask^z_k at pool k is less than ask price $Ask^z_{k'}$ at pool k' for each pool k in set $ONPo_i$. 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 ask price routing constraint. The mathematical formulation for this constraint is written as follows:

$$(3)\ \min_{k:<k, k'> \in ONPo} (\text{Ask}^z_{<k, k'>} = 1),\ \forall o \in O,\ <k, k'> \in ONPo_i$$

iii. Slippage Routing Constraint

Once exhaustion state is reached, given by $ESo_i = 1$, 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*:

(4) $\min_{k:<k,k'>,<k,k''> \in N_{SP=k}} (u^{z}_{<k,k'>,<k,k''>} = 1), \forall o \in O, k', k'' \in N_{SP=k}$

B. Exhaustion Implication Constraint

The exhaustion state given by $ESo_i = 1$, is reached once availability $AV^{z}_{k} = 0$ 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:

(5) $ESo_i = 1, \sum_{k=SPo}(AV^{z}_{k}) = 0, \forall o \in O, k \in P$

V. FIOR: THE PROPOSED APPROACH

Our implementation of smart order routing for fixed income securities is outlined in this section. The below algorithm uses a combination of linear programming [7] and optimization to identify the most affordable liquidity pools based on the ask prices offered for limit orders at each pool, the availability at the pool, and after exhaustion 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 "exhaustion state" in which the exhaustionState() 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 function work very similarly as they determine the optimal neighboring pool and exchange based on linear programming, considering both ask prices, availability, 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 availability 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 Programming, utilizing the 'minimize' function from the scipy.optimize 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, availability, 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.

VI. PERFORMANCE EVALUATION

To evaluate the performance of FIOR approach, we have to compare the DFS Kruskal solution versus the linear programming solution. The DFS Kruskal algorithm is a naturally 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] However, it is modified here to consider the ask price along with other variables that are within the system model and the second solution, a Markov Decision Process is used to solve the optimization problem by linear programming. The evaluation was done with pools ranging in size from 1-12, and we are comparing the # of pools vs the average slippage cost of the order and the # of pools vs the total cost of the order I. The results ran over several times on the same model used for both solutions and we found that it would produce the same result. The order I was for 120 MSFT bonds and the number of edges totalled

to 26 with slippage costs being all positive to avoid exploitations of negative slippages along with 12 pools with each pool producing a randomized set of connections using the rand() function. The results led us to several interpretations, correlations, and analysis outside of the figures shown.

A. Slippage Average vs. number of pools

Due to the order volume being 120 MSFT bonds, and the availability per pool ranging from 10-70 bonds with ask prices ranging from $96.50 - $104.00 per bond, the agents

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

```
1. Input: Graph G -- (P, S), starting node Sa.
2. Result: PoolsTraversed[], SlippageExchanges[],
   TotalCost, SlippageCost, Reward
3. Order <-- FIOR_LP(Graph G, State, Actions, OrderID,
   OrderLog, OrderAmount, StartingPool, TotalCost)
4. Visited[]
5. If TotalAvailability >= 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["Availability"].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
    = exhaustedState(PoolTraversed)
18. Return Visited[], Reward, SlippageExchanges[],
    PoolsTraversed[], TotalCost, SlippageCost
```

rarely exhausted pools availability as the order would be executed before the availability could be exhausted which is why with less pools, we noticed a higher slippage cost associated. If the order volume was to be 12,000,000 MSFT bonds, the slippage percentages would be much more magnified to being nearly 10% - 25% of the total order cost. With more pools, the slippage costs were in fact lower as there is more availability of the security which in return reduces slippage which is why slippage is zero after the size of pools reaches 4. (Figure 2) Analyzing where there is slippage associated, the DFS Kruskal solution automatically has a higher slippage cost as seen in the figure as it exhausts the availability immediately and the slippage costs would add up fast as it is simply going between two pools that have been both exhausted so the slippage cost within the order slowly compounded with slippage being an afterthought for when it is exhausted while linear programming monitors this during its traversals. However, even though the total cost ended up being higher, this aspect was completely based on the system model since there are less available decisions due to there being a limited route along with ask prices continuously compounding.

B. Number of Pools vs Total Cost

The results of the performance evaluation are highly dependent on the specific implementation of the test values. With a lower number of pools, the results highlight the main differences in the way that the algorithms work, as the number of pools increases, the operations the algorithms perform are executed more frequently, producing more consistent, and less volatile results as both algorithms have the same end goal. It is evident that as the number of pools increases, the linear programming approach more consistently minimizes the total cost for the order, whereas the DFS Kruskal's approach is a little more volatile in its results. (Figure 3) This demonstrates the consistency of the linear programming approach and how it shows to be a more reliable solution. After the pool size is greater than 4, the total cost of the order begins to decrease as a result of

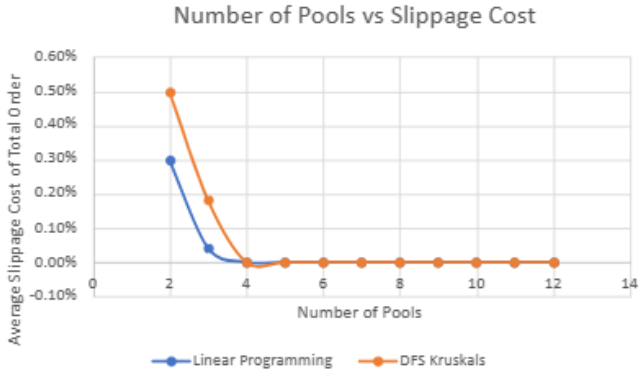

Fig. 2. Performance evaluation of different methods by Number of Pools with respect to Average Slippage Cost of Total Order

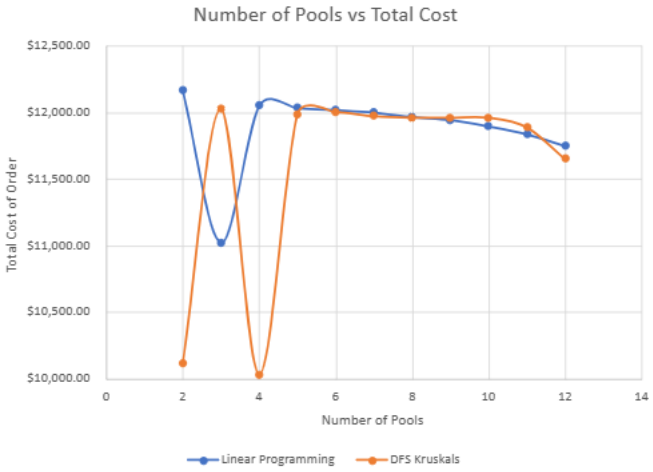

Fig. 3. Performance evaluation of different methods by Number of pools with respects to Total Cost of order

more possible routes which in turn allows for smaller ask prices to be available. The DFS Kruskal solution falls behind slowly from the linear programming approach in this state as it may traverse to pools that it is not required to traverse to and does not have the decision making of the linear programming approach where it can create new paths each time rather than sticking to the same ones which is what DFS Kruskal's did for systems of size 8, 9,and 10. Again with the constraint of the order being small which was done for simplicity purposes, the total cost differences between the two solutions were very small spreads in the range of $10-$100 but as said before, the increase in order volume, the spreads would vastly grow with slippage costs further pushing the cost as more pools become exhausted and slippages becoming active.

C. Further Results and Interpretations

One result that has been mentioned previously but not recorded, is the total order cost vs the total order volume, and the slippage cost of the total order cost vs the order volume. With the order growing higher in volume, the pools availability of the security Z will be exhausted, and slippages therefore will be applied to ask prices in later traversals. This will increase the amount of slippage within the total cost and the total cost in general. DFS Kruskal will be most affected by this with worsening performance as it would traverse through more pools and exhausting more unnecessary pools availability that may apply high slippages while linear programming solution would monitor these aspects in each traversal and search for a more optimal route for the several order blocks that it would split the order into. Another metric that it would affect is computational time as the DFS Kruskal would traverse through more nodes and has for even the simulation that we have conducted and recorded, it would take exponentially longer when the # of pools exist in large instances and therefore would result in the order taking longer to execute. In a live market scenario, this would be specifically disastrous as ask prices, availability and slippages can change in a matter of seconds within each pool and if the decisions are made too slowly, it can miss several opportunities for cheaper alternatives while possibly resulting in more expensive order blocks.

VII. 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. [1] Through a comprehensive evaluation of FIOR's performance in a simulated environment with real-time market data, this research has provided valuable insights into its effectiveness when compared to existing strategies such as DFS Kruskal's.

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 innovate 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 towards 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/