Monte Carlo Tree Search for Trading and Hedging

Edoardo Vittori
Politecnico di Milano
Intesa Sanpaolo
edoardo.vittori@polimi.it

Amarildo Likmeta
Universitá di Bologna
Politecnico di Milano
amarildo.likmeta2@unibo.it

Marcello Restelli
Politecnico di Milano
marcello.restelli@polimi.it

ABSTRACT

Monte Carlo Tree Search (MCTS) has had very exciting results in the field of two-player games. In this paper, we analyze the behavior of these algorithms in the financial field, in trading where, to the best of our knowledge, it has never been applied before and in option hedging. In particular, using MCTS algorithms capable of handling stochastic states and continuous actions, we setup a practical framework testing it on real data both in the trading and hedging case.

CCS CONCEPTS

• **Computing methodologies** → *Online learning settings*; Markov decision processes; *Reinforcement learning*; **Planning under uncertainty**.

ACM Reference Format:
Edoardo Vittori, Amarildo Likmeta, and Marcello Restelli. 2021. Monte Carlo Tree Search for Trading and Hedging. In *ICAIF '21: ACM International Conference on AI in Finance, November 3–5, 2021, New York, NY*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3490354.3494402

1 INTRODUCTION

Trading and hedging are two of the most important paradigms in the financial world. Trading refers to buying and selling financial assets to make a profit. Most of the existing trading algorithms are expert systems, where experienced traders with computer scientists write hard coded rules to exploit arbitrages or implement ideas from the trader's experience. Options market makers are also heavily reliant on algorithms and their objective is to make a profit by maximizing the number of trades and not by speculating. A market maker will often price many of the options of an asset and in order to properly manage all these options, she will be aided by an automatic software to hedge the *delta risk*, so that she can focus on other risks. In general, the software will hedge the delta risk of each new trade and every few hours will hedge the delta risk of the entire portfolio so to keep it delta neutral. This creates transaction costs which can be quite relevant, depending on the liquidity of the underlying instrument. There are two main methods of optimizing costs: either by optimizing the execution (and blindly following the Black & Scholes [5] delta hedge), by optimizing the hedging policy, or a combination of the two. In this work we will focus on optimizing the hedging policy.

Both of the problems described before can be modeled as sequential decision-making problems, since in both cases the goal is to maximize the cumulative sum of rewards over a temporal period. These types of problems can be solved with two main approaches in a machine learning context, Reinforcement Learning (RL) and online planning. RL algorithms solve these problems by interacting with the environment, or in the case of Batch RL by optimizing a policy in a fixed dataset of demonstrations from the environment. This means that, in the continuously changing world of financial markets, RL algorithms would suffer from the non-stationarity of the price processes, requiring continuous updates of the policy. On the other hand, online planning algorithms, require a 1-step model of the environment and use it in each step to construct a search-tree to evaluate the different available actions. This makes them more robust to non-stationary environments, since when the environment changes, it suffices to update the generative models used during the search, which in general is less expensive than updating the policy maintained by RL algorithms. For these reasons, we focus on OP as we believe it offers greater flexibility when dealing with the continuously changing market environment, giving for example the possibility to easily handle changes in the volatility, in the bid ask spread and in the market impact. The MCTS algorithm proposed in this paper is a variant of Upper Confidence Tree (UCT) [15]. This algorithm has been designed for application in finite sequential decision models: the state transition model is deterministic and the action space is discrete and finite. On the other hand, both of our scenarios of interest have a continuous state space with stochastic state transitions *i.e.* given a state-action pair, there is a high uncertainty about the possible next state. Hedging also has the added complexity that the decision space is continuous.

Contribution. To the best of our knowledge, this is the first work to apply MCTS to the trading problem and in a realistic option hedging setting. We also innovate on the MCTS algorithms: we use an *open-loop* variant of UCT [16] to deal with the stochastic transition model, combined with a progressive widening procedure in the case of continuous action spaces, as is the hedging scenario. Furthermore, we propose a novel backup procedure for the MCTS algorithms, which uses Q-Learning (QL) [23] Temporal Difference (TD) [20] updates to address the high variance of the returns observed in the nodes of the search-tree. For the trading scenario, we propose a novel generative model to employ during planning, which uses past observations of the asset of interest to generate possible future realizations of the market to be used during planning in order to search for the optimal trading strategy. Finally, we perform an evaluation of the proposed algorithm and generative

Equal contribution of the first two authors.

ICAIF '21, November 3–5, 2021, New York, NY

ACM ISBN 978-1-4503-9148-1/21/11…$15.00
https://doi.org/10.1145/3490354.3494402

models on real financial data. To our knowledge, it is the first paper that tests a MCTS option hedging approach on real market data.

Paper Structure. The paper is structured as follows. In Section 2 we will comment on the most relevant existing approaches to trading and option hedging. In Section 3 we define MCTS and explain the Progressive Widening (PW) and Open Loop UCT extensions that allow MCTS to work in problems with stochastic state transitions and continuous actions. The main part of the paper is then divided into two Sections, in Section 5 we describe the trading paradigm and our approach with the experimental results. Section 6 is structured analogously to the previous one, but focusing on the option hedging paradigm. Finally, we present the concluding comments.

2 RELATED WORKS

Although, to our knowledge, this is the first paper that considers the use of MCTS in a trading framework, the use of machine learning and especially RL has become popular in recent years [11]. [18] uses recurrent RL and is considered as the first work in this direction. Following this first paper, other approaches were tested e.g. genetic algorithms [25] or adaptive RL [9]. The latter, as well as [10; 12] concentrate on FX. Then there are approaches that consider Q-learning [3; 14]. Finally, in [4] they use a novel risk-averse policy search approach applied to FX and S&P data. Recently, the use of Deep RL for trading has been "commoditized" thanks to an open source library [17], but it does not contain MCTS-like algorithms.

The issue of option hedging has been analyzed by various authors [6; 13; 22]. There are two main drawbacks of all these approaches compared to ours that make them "less compatible" with real data: first, they generally consider a constant volatility, which is unrealistic as market data is non-stationary; second, they require a relevant training set that necessarily needs to be simulated as thousands of realizations of an option with the same characteristics are not available, this means that the data are necessarily simulated thus losing the advantage of using model free approaches. Finally, [21] uses MCTS in a simplified hedging scenario.

3 PRELIMINARIES

A discrete-time Markov Decision Process [MDP, 20] is defined as a 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the (possibly infinite) action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathcal{S})$ is the transition model, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\gamma \in [0, 1)$ is the discount factor. The behavior of an agent is defined by means of a Markovian policy $\pi : \mathcal{S} \to \mathscr{P}(\mathcal{A})$. When the environment is in state $s \in \mathcal{S}$, the agent performs action $a \sim \pi(\cdot|s)$ and the environment transitions to the next state $s' \sim \mathcal{P}(\cdot|s, A)$ providing the agent with the reward $r = \mathcal{R}(s, A)$. The (policy-dependent) state value function is defined as $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_t \sim \pi(s_t) \right]$. We also define the state-action value function as $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a, a_t \sim \pi(s_t) \right]$, where the first action is fixed. The goal of the agent is to find the policy π^* maximizing the value function in all states: $\pi^*(s) = \arg\max_\pi V^\pi(s), \quad \forall s \in \mathcal{S}$.

3.1 Monte Carlo Tree Search

The MCTS family of algorithms combines tree-search algorithms with Monte Carlo sampling to iteratively build a search tree of possible future scenarios, which is in turn used to build an estimator of the optimal value of each action in the current state of the environment. These algorithms are characterized by 4 phases:

(1) **Selection:** Starting from the root of the planning tree, a *tree policy* is iteratively applied until an unexpanded (a node with unvisited children) node is reached.
(2) **Expansion:** One or more the successors of the reached node are added to the tree. A common best practice is to add only the first newly visited node.
(3) **Simulation:** A Monte Carlo simulation (*rollout*) is started from the expanded node to provide an initial estimate of the nodes' value.
(4) **Backpropagation:** The values of the states visited during the tree traversal and the simulation are backpropagated up the tree until the root, updating the relevant statistics.

In this work, we focus on Upper Confidence Tree (UCT) [15]. UCT applies as a tree policy the well-known Multi Armed Bandit (MAB) algorithm, Upper Confidence Bounds (UCB1) [1]. At iteration n, UCB1 chooses the action that maximizes a high probability upper bound of the value of the actions according to:

$$a_n = \arg\max_{i=1..K} \overline{X}_{i,T_i(n-1)} + C\sqrt{\frac{2\log n}{T_i(n-1)}}, \tag{1}$$

where K is the number of actions, C is a constant that regulates the exploration-exploitation tradeoff, $T_i(n-1)$ is the number of times action i has been played up to time $n-1$ and $\overline{X}_{i,T_i(n-1)}$ is the average payoff observed from arm i.

Progressive Widening. When the action-space is continuous, we cannot apply UCB in the nodes of the search tree since the number of actions is infinite, and as a consequence also the true search-tree. One of the main techniques to deal with infinite action spaces is Progressive Widening (PW) [7]. With PW, the actions are explored progressively as the node visitation counts increase, the idea being that the nodes visited the most are more promising. Specifically, when a node is visited for the n-th time, if the number of children of a node $|C_n(\mathcal{N})|$ is larger than n^α, where $0 < \alpha < 1$, a new action is explored by sampling from the action space, otherwise, one of the previous selected actions is explored. In the second case, it is common to chose the action previously visited according to UCB1. Importantly, PW only specifies when to add new nodes to the search tree, but not which new actions to explore. Usually, in literature, new actions are sampled uniformly in the action space, but the empirical performance of the algorithms strongly depends on the action sampling distribution employed.

Open Loop Planning. The open-loop planning approach is used in problems with continuous state spaces and stochastic transition models since the true search tree is infinite. In this setting, the problem considered is to find the optimal sequence of actions to be employed at the root state of the tree, without considering the states visited during the search, transforming the infinite search tree of the original problem into a finite tree with branch factor equal to the number of actions. Formally, given

a starting state $s \in \mathcal{S}$ and a sequence of actions of length m: $\tau = (a_1, \ldots, a_m)$, $a_i \in \mathcal{A}, \quad i = 1, \ldots, m$, we define the open-loop value of the sequence τ starting from state s as the discounted sum of the rewards collected executing the complete sequence τ starting from state s: $V_{OL}(s, \tau) = \mathbb{E}\left[\sum_{t=1}^{m} \gamma^t r_t \middle| s_0 = s, a_t \in \tau\right]$. The sequence length m can be infinite when $\gamma < 1$. The optimal open-loop value function is the maximizer over the possible sequences $\tau \in \mathcal{A}^m$: $V^*_{OL}(s) = \max_{\tau \in \mathcal{A}^m} V_{OL}(s, \tau)$. Similarly, we define the optimal open-loop state-action value function for each state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$ as the maximizer of the open-loop value over all the possible sequences of actions starting with a, $\tau_a \in \left\{(a, \tau) \middle| \tau = a_1, \ldots a_{m-1}\right\}$: $Q^*_{OL}(s, a) = \max_{\tau_a} V_{OL}(s, \tau_a)$.

4 OUR APPROACH: OPEN LOOP Q-LEARNING UCT

In this section, we present the planning algorithm used in this work. To tackle the continuous state space with stochastic transitions of both problems, we resort to an open-loop approach that looks for the optimal sequence of actions to apply to the environment. Even though the planning phase is done in an open-loop fashion, only the first action identified by the planning procedure is applied, since in our online framework, planning is interleaved with acting in the environment. The alternative to an open-loop setting is the application of PW to the state space, but this comes with higher memory costs as well as a higher planning budget needed to represent the (approximate) full search tree.

Formally, we consider a planning phase with a limited search-depth D. The open-loop search tree of depth D is denoted by $\mathcal{T}_D$. We denote by $\mathcal{N}_{d,i}$ the i-th node at depth $d \geq 0$ for $i \in \mathbb{N}$ and $d = 1, \ldots, D$. In the open-loop case, each node in the tree is uniquely identified by the sequence of actions representing the path from the root to the node. $\mathcal{N}_{0,0}$ contains a single state, $s_0 \in \mathcal{S}$, from which we want to perform planning. Nodes $\mathcal{N}_{d,i}$, with $d > 0$, at deeper levels of the tree, represent the distribution of states given the sequence of actions from the root of the tree to $\mathcal{N}_{d,i}$. Specifically, given a node $\mathcal{N}_{d,i}$ and the sequence of actions that identifies it $\tau_{d,i} = (a_{1,i}, \ldots, a_{d,i})$, the possible states observed in the node $\mathcal{N}_{d,i}$ represent the state distributions induced by executing $\tau_{d,i}$ starting from state s_0.

We start by defining the open-loop value of a node $\mathcal{N}_{d,i}$ as:

$$\mathcal{V}\left(\mathcal{N}_{d,i}\right) = \mathop{\mathbb{E}}_{s \sim \mathcal{P}(\cdot|s_0, \tau_{d,i})}\left[V^*_{OL}(s)\right], \tag{2}$$

and the value of an action $a \in \mathcal{A}$ in node $\mathcal{N}_{d,i}$ as:

$$\begin{aligned} Q\left(\mathcal{N}_{d,i}, a\right), &= \mathop{\mathbb{E}}_{s \sim \mathcal{P}(\cdot|s_0, \tau_{d,i})}\left[Q^*_{OL}(s, a)\right] \\ &= \mathop{\mathbb{E}}_{s \sim \mathcal{P}(\cdot|s_0, \tau_{d,i})}\left[r(s, a)\right] + \gamma \mathcal{V}\left(\mathcal{N}_{d+1,j}\right), \end{aligned} \tag{3}$$

where $\tau_{d+1,j} = \left(\tau_{d,i}|a\right)$ is the sequence of action derived from concatenating $\tau_{d,i}$ with a and $\mathcal{N}_{d+1,j}$ is the child of node $\mathcal{N}_{d,i}$ corresponding to action a. The goal of our proposed planner is to estimate the optimal open-loop action values at the root node by applying a UCT-like selection policy that selects, in each node, the action that maximizes the upper bound of the Q values according to Equation (1). In the case of continuous actions, we employ a PW strategy as described in Section 3.1.

Algorithm 1 Q-Learning Open Loop Planning

```
procedure OLSearch(s_0)
    Create root node N_{0,0} from state s_0
    while within computational budget do
        N_{d,i}, s ← TreePolicy(N_{0,0})
        V(N_{d,i}) ← Rollout(N_{d,i}, s)
        Backup(N_{d,i})
    end while
    return BestChild(N_{0,0})
end procedure
procedure TreePolicy(N)
    while N not terminal do
        if N not fully expanded then
            return Expand(N)
        else
            N ← BestChild(N, C_p)
        end if
    end while
    return N
end procedure
procedure Backup(N, V)
    C'(N) denotes explored children nodes of N
    N' ← parent of N
    N.n ← N.n + 1
    while N' is not null do
        if N is leaf then
            Δ ← V
        else
            Δ ← max_{a'∈C'(N)} Q(N, a')
        end if
        Q(N', a) ← Q(N', a) + α(N'.r + γΔ − Q(N', a))
        N'.n ← N'.n + 1
        N ← N'
        N' ← parent of N
    end while
end procedure
```

Our second change of the base UCT algorithm is the backup operator employed in the backpropagation phase. During the tree expansion, exploitation and exploration is interleaved thanks to the selection rule of UCB, meaning that the values observed in each node come from very different policies. Also, in the simulation phase, a suboptimal rollout policy is employed to give an initial evaluation of each node. This rollout policy is clearly suboptimal (if we had an optimal policy for the rollout we would not need to perform planning) adding further noisy samples being backed-up. Both of these factors make the backup values observed extremely noisy, which is a further problem in our financial settings. For these reasons, instead of the plain Monte Carlo updates, that average the return values observed in each node in the tree, we employ a Temporal Difference update, based on the Q-Learning update rule [23], as follows:

$$\begin{aligned} Q_t\left(\mathcal{N}_{d,i}, a\right) = (1 - \alpha_t) Q_t\left(\mathcal{N}_{d,i}, a\right) \\ + \alpha_t \left(r_t + \gamma \max_{a'} Q_t\left(\mathcal{N}_{d+1,j}, a'\right)\right), \end{aligned} \tag{4}$$

where r_t is the reward observed in the current search pass at node $\mathcal{N}_{d,i}$ and α_t is the learning rate employed, which constitutes an added hyperparameter of our planner.

We present the pseudocode of our planner in Algorithm 1. This planner is devised to be employed in each decision interval with a given planning budget, specified in environment transition samples from the model. At each search iteration, we perform the selection phase, plain UCB, or UCB interleaved with PW in the hedging case, until a leaf of the tree is reached. We then perform a rollout from the leaf. The specific rollout policies employed in both scenarios are described in the following sections. The rollout gives us an initial estimate of the node value. Next, we recursively employ QL updates up the tree, updating the node and action values and counts. This means that the initial noisy back-up value given by the rollout, even though it is stored in the leaf node, might not make its way up to the root, since at each node we employ the max operator to define the target value, as shown in the Backup procedure. If all the children of a node have not been explored yet, for the Q-learning update of Equation (4), we apply the max operator only to the visited nodes, disregarding the unexplored actions. Finally, the BestChild procedure has not been described since it depends on the action space of the specific problem. In the trading environment, the selection is based on UCB, whereas in the hedging scenario, since the action space is continuous, we apply PW together with UCB.

Optimality Remark. It is worth noting that, while in general the open-loop setting comes with a loss of performance compared to the closed-loop setting, this is not an issue in the specific tasks of trading and hedging. This is due to the fact, that the market is not influenced by our actions. In the next sections, we will describe the state-space of both these tasks. The state space is composed of features relative to the market (the price history) and features related to the agent (the agent's current position). While the market features follow stochastic transitions they are not influenced by the agent's actions. The only features influenced by the agent are its own positions regarding the underlying assets which follow deterministic transitions. Trading and hedging, in our setting, form a special case of Factored MDPs [8], where the state transitions consist in two independent clusters of features. This means that the agent can effectively react to the differences in the features that it can control, also in the open-loop setting.

5 TRADING

In this section, we concentrate on the trading framework. Specifically, we analyse trading in the Foreign Exchange (FX) market, the largest type of financial market in the worldin terms of daily volumes. For our experiments, we look at the most liquid FX currency pair which is the EURUSD.

5.1 Problem Formulation

We model the trading scenario as a continuous state, discrete action MDP, as follows:

State. The state contains a window of the last M observed prices. In our experiments, the decisions are taken every minute, so we use a window of 60 prices *i.e.* an hour of observations. This window is used to incorporate in the state information about the trends of the market, as including only the current price would make the state non-Markovian. Since we consider trading in a finite horizon of length H, we include in the state also the current timestep $t \in [0, H]$ and the portfolio position $x_{t-1} \in \{-1, 0, 1\}$ of the previous timestep.

Action. We consider a discrete action space where the action at time t, a_t is the portfolio position the agent will hold, so -1 indicates keeping a short EURUSD position: selling 1 EUR and buying the equivalent amount of USD, 0 indicates not holding any exposure, +1 indicates buying 1EUR and selling the equivalent amount of USD. Each action has the same size of unitary amount.

Reward. Given the current portfolio position x_t, the action taken a_t, and the prices, the reward is defined as:

$$r_t = \underbrace{a_t \cdot (p_t - p_{t-1})}_{\text{market variation}} - \underbrace{\frac{bid - ask}{2} \cdot |a_t - x_t|}_{\text{transac. costs}}, \tag{5}$$

where *bid* in this case represents the *best* bid or the highest price an investor is ready to pay, and similarly for the ask. $\frac{bid-ask}{2} \cdot |a_t - x_t|$ represent the transaction costs.

5.2 Nearest Neighbor Generative Model

A key element of applying MCTS, apart from the specific planning algorithm is also the generative model used to generate the simulations during the planning phase. A first alternative is to use classical models such as GBM or Vasicek SDEs, or econometric models such as ARIMA, with parameters calibrated to fit the training data. In this section, we propose a novel technique to generate MC simulations during the planning phase, based on a Nearest Neighbors [2] framework, to retrieve, from the historical data available, price sequences that are "similar" to the current price window in the state.

Formally, we consider time series of historical prices of the asset, $(p_1, p_2, \ldots, p_T)$ where T is the length of the time series. At time t, we observe the window of the last M prices of the asset, $(p_{t-M}, \ldots, p_{t-1})$. Since the length of the dataset T might cover multiple years, and the price of the asset might have changed substantially in these years we consider, instead of the series of prices, the series of price variations, $D = (\delta_1, \delta_2, \ldots, \delta_T)$ where $\delta_j = \frac{p_j - p_{j-1}}{p_{j-1}}$, where we consider $p_0 = p_1$. Our goal is to find the "closest neighbors" of the window $w_t = (\delta_{t-M}, \delta_{t-M+1}, \ldots, \delta_{t-1})$ in the partial dataset $D_t = (\delta_1, \delta_2, \ldots, \delta_{t-1})$, that is the historical data before time t. By finding the nearest neighbors of w_t, we can use the continuation of the windows as simulations during the rollout.

Specifically, given a rollout length N, we aim to retrieve the K nearest neighbors of w_t (relative to a distance measure d), $\{w_{t_i}\}_1^K$, where $M < t_i < t - N$ is the time index of the i-th neighbor. We split the time series D_t in overlapping windows of length M, generating the dataset X, where each row of the dataset is a window of length M (same as the state window), where $X_0 = (\delta_1, \ldots, \delta_M)$, $X_1 = (\delta_2, \ldots, \delta_{M+1})$ and the last row $X_{t-N} = (\delta_{t-N-M}, \ldots, \delta_{t-N})$. Note that, the last price in the dataset X, is the price at time $t - N$, meaning that the last price of the corresponding rollout simulation is the last timestep. Before starting the planning phase, we search in the dataset X, for the K nearest neighbors of the current window w_t. This K neighbors give K possible future continuations of the

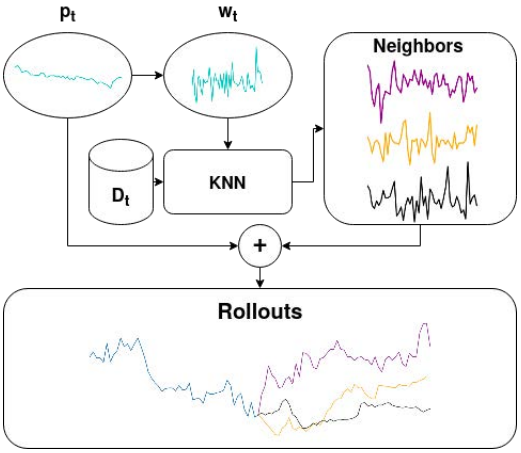

Figure 1: Visual representation of the NN generative model.

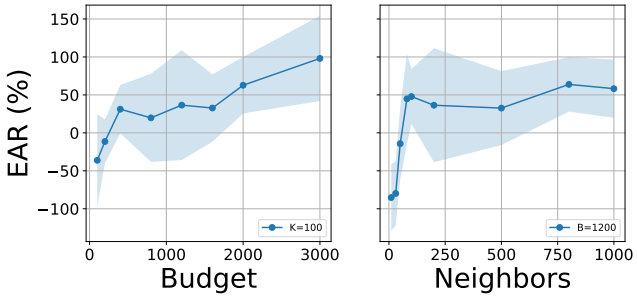

Figure 2: Expected annual return with no transaction costs, as a function of the search budget and the numbers of neighbors. Average over 50 runs, 95% CI.

price variations, which are used during planning. Figure 1 shows a visual representation of the NN model.

5.3 Experiments

In this section, we present an experimental campaign evaluating the MCTS approach in the trading problem. We use as planner, QL-OL UCT described in Section 4. As a generative model, we use the Nearest Neighbor approach described in the previous section. Obviously K becomes an hyperparameter of our approach. At the beginning of each planning iteration, we sample one of these neighbors to use as a trajectory for the next rollout.

We concentrated our experiments on the EURUSD FX pair. We used a dataset of historical 1 minute prices from 2017 to 2019. In each episode, we sample a random date from the year 2019 and begin a trading episode for the next H minutes, where the horizon H is set to 200 in our episodes. We repeat this 50 times and present results as average return together with the 95% Confidence Intervals (CI).

For each episode, we use as dataset for the Nearest Neighbors model, the series of prices from 1st of January 2017 to the current date. We considered as neighbors only the price windows coming from hours in a window of 3 hours centered at the current hour of the day, meaning that if we are currently trading in 10:00, we consider only windows from the time 9:00 to 11:59. This yielded better results, compared to considering all the possible windows, since there appears to be some correlation in the return windows depending on the hour of the day. Furthermore, this decreases the computational costs of retrieving the neighbors.

In each timestep, we perform a tree search, using QL-OL UCT with planning budget B and by sampling K neighbors. Both B and K represent hyper-parameters. During planning, we decrease the decision frequency, meaning that every action chosen during the simulation phase (tree search) is repeated C times in the environment, before allowing to chose another action. This is done to effectively increase the planning horizon, without increasing the planning cost, as frequent changes of the position are often non-optimal, especially under the presence of transaction costs. In all our experiments we use $C = 5$. Moreover, the maximum tree-depth has been set to 5, to allow for fast tree-search construction with a low budget because of the quasi-real-time requirements of the application. This essentially means that the tree-search algorithm will optimize the return over the next 25 minutes given a tree depth of 5, where each level represents 5 minutes of transactions.

Results. In Figure 2 we evaluate our approach in a setting without transaction costs. This is done to evaluate whether the Nearest Neighbours model is able to accurately predict the future trends in the market. On the y-axis we present the expected annual return (EAR). On the left we present the EAR as a function of the planning budget B, measured as transitions sampled from the forward model. In this experiment, we fix the number of neighbors $K = 100$. While for very low budgets, of 100 and 200 transitions, corresponding to 20 and 40 future observed trajectories, the agent comes at a loss, we see that starting from really low budgets of 400 are able to achieve a profit. As we increase the budget, the returns improve (as we would expect). Finally, on the right we show the dependence of the returns on the number of neighbors K, while fixing the planning budget $B = 1200$ samples. Similarly to the previous experiments, a low number of neighbors yields a really low performance, as the trajectories seen during the simulation do not accurately reflect the true future trajectories, so the planning agent "overfits" these simulations. As the number of neighbors surpasses 100, we become profitable, and the return improves slightly with the increase in neighbors. Next, we evaluate our approach in the more realistic scenario of trading with transaction costs. In this scenario, we set the costs of Equation (5), $\frac{bid-ask}{2} = 10^{-5}$. Figure 3 shows the results of this experiment, where we vary the search budget and fix the number of neighbors to 100. Differently from the previous case, the addition of transaction costs has made the planning agent more careful. While, similarly to the previous case, for low budget values, the agent performs at a loss, when increasing the budget, even further than the previous case, the agents converges in a policy of not trading and insuring 0 profit (and loss). This probably happens because the search horizon is shorter than the true horizon of the interaction, making it inconvenient to trade with transaction costs because there is not enough time to observe the benefits of paying these costs. On the other hand, increasing the horizon search comes with increased computational demand for taking each

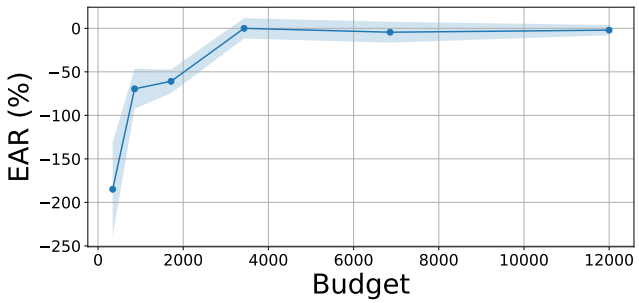

Figure 3: Expected annual return with transaction costs as a function of the search budget. Average over 50 runs, 95% CI.

decision. Nonetheless, even for low budgets of 3000 samples during the search (which allow for near real-time response), the agent does not suffer losses. In future works, we aim to tackle the problem of mismatching horizons during search and interaction, with the aim of increasing the search horizon without suffering costs in the response time. Potentially by using more intelligent rollout policies, that will focus the tree construction in the relevant parts.

6 OPTION HEDGING

In this section, we focus on the application of MCTS to the option hedging problem faced by market makers. We consider the case of a single vanilla equity call option. Vanilla options are contracts that offer the buyer the right to buy or sell a certain amount (the option's notional) of the underlying asset at a predefined price (the strike) at a certain future time (the maturity). The investor pays at inception the option premium in order to buy the option. This offers the investor the opportunity to gain money if the price of the underlying asset surpasses the strike. On the other hand, this comes with the risk of losing the payed premium if the price is lower than the strike at expiry. We take the point of view of an options market maker, who, in order to manage the risk generated by having a net positive or negative inventory of options needs to hedge the delta risk. Option pricing and hedging builds on the Black & Scholes (B&S) model [5] which is based on a strong set of assumptions that tend to be unrealistic [24]: hedging is assumed to be cost-less and continuous.

6.1 Problem Formulation

B&S is the main option pricing model used in practice, and so the benchmark considered in this work. In this framework, the underlying behaves as Geometric Brownian Motion (GBM), thus let S_t be the underlying at time t, then it can be described as $\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sigma S_t \mathrm{d}W_t$ where W_t is Brownian motion, μ the drift (which we assume to be 0 throughout the paper without loss of generality) and σ the volatility. Let C_t be call option price at time t, T the time *of* maturity, $T - t$ the Time *To* Maturity (TTM), K the strike price, σ and μ coincide with those of the GBM. The B&S call price C_t is:

$$C_t(S_t) = \Phi(d_t)S_t - \Phi(e_t)Ke^{\mu(T-t)},$$
$$d_t = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(\mu + \frac{\sigma^2}{2}\right)(T-t)\right],$$
$$e_t = d_t - \sigma\sqrt{T-t},$$

where Φ is the cumulative distribution function of the standard normal distribution. We introduce $\frac{\partial C_t}{\partial S}$, which is known as the option delta and for our position (a long call of unitary notional) is bounded between 0 and 1. In particular when $T-t$ is relatively small and $\frac{S_t}{K} \ll 1$, $\frac{\partial C_t}{\partial S} \to 0$ and $C_t \to 0$; instead if $\frac{S_t}{K} \gg 1$, $\frac{\partial C_t}{\partial S} \to 1$ and $C_t \to S_t$. A trader who has a long position in a call option will endure a profit swing of $C_{t+k}(S_{t+k}) - C_t(S_t)$ for a time-lag of k. A delta hedge is a strategy to limit this profit movement by buying or selling a certain quantity of the underlying.

The Profit and Loss (P&L) in one timestep of hedging the option variation is:

$$\rho_{t+k} = \underbrace{C_{t+k}(S_{t+k}) - C_t(S_t)}_{\text{Option variation}} - \underbrace{h_t \times (S_{t+k} - S_t)}_{\text{hedge variation}} - \underbrace{m \times c(n)}_{\text{transac. costs}} . \quad (6)$$

Where n is the difference between two consecutive portfolios. If $h_t = \frac{\partial C_t}{\partial S_t}$, $k \to 0$ and $c(n) = 0$ the B&S model assures a zero profit. Similarly to the trading case, we define proportional transaction costs which depend on the bid-ask spread of the underlying instrument:

$$c(n) = |n| \times \frac{bid - ask}{2}. \quad (7)$$

Embedding in an MDP. In order to use online planning, it is necessary to define the characteristics of the MDP:

- the action a_t is the current hedge portfolio, replaces h_t,
- the state $s_t = (S_t, C_t, \frac{\partial C_t}{\partial S_t}, a_{t-1})$,
- the reward $\mathcal{R}(s_t, a_t) = f(\rho_t)$, with ρ_t as in Equation (6).

6.2 Our Approach

In this section, we describe the approach we took to tackle the option hedging problem with QL-OL UCT. We choose an underlying asset and an option on that underlying. We observe from the market the current underlying price p_t and option price o_t , we then calculate the B&S implied volatility σ_t and plan with QL-OL UCT using as generative model a GBM with starting price p_t and volatility σ_t. Once the search is over, we select the action $a_t = \max_a Q\left(\mathcal{N}_{0,0}, a\right)$. Then we start over by observing the new market information and continue until option expiry.

As it would be quite expensive from a planning point of view to search the entire action space, we took the following steps to facilitate planning using available information:

- the rollout policy is the delta hedge;
- the search looks only a few steps ahead, as using the accounting formulation makes it unnecessary to see what happens at expiry thus reducing the necessary budget;
- $a_t = \frac{\partial C_t}{\partial S_t}$ is always explored;
- the action search space is reduced to being between the current portfolio and the delta + ϵ.

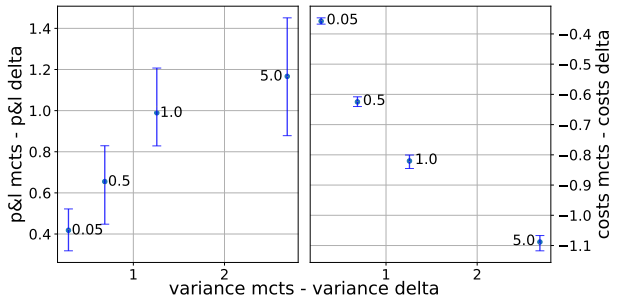

Figure 4: P&l of the MCTS agent w.r.t. the delta hedge (left) and trading costs generated by MCTS agent w.r.t delta hedge (right). Average of 2000 simulations, 95% CI. Results in EUR, annualized and for a single option.

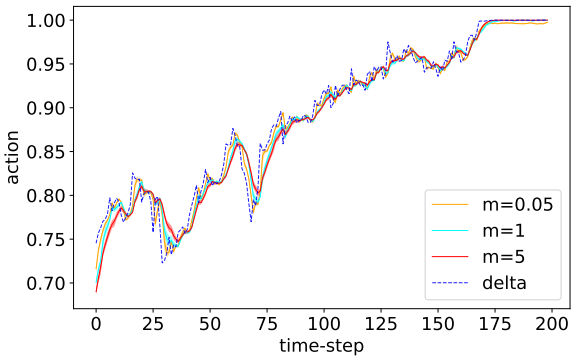

Figure 5: Action on SX7E, single option with strike 90 and expiry 17/06/2021, starting 25 working days before expiry.

With such a formulation, the MCTS algorithm searches around the delta hedge action in order to understand if there is an action which is superior to the delta hedge action given our reward formulation.

6.3 Experiments

In this section, we perform an empirical evaluation QL-OL UCT on the option hedging environment. Initially, we observe the performance on simulated data and in a second experiment, we evaluate the performance on real data always comparing to the delta hedge. One of the used metrics use is the cumulated p&l over the entire life of the option we are considering, where the step p&l is defined as in Equation (6). We also refer to the cumulated p&l as the returns. The results are annualized for ease of comparison and expressed in terms of a single option, but are linear so can be multiplied by the number of options.

Simulated Data. In the simulated results, we consider hedging a long position in a vanilla call option, considering any other type of vanilla option and positioning would have been equivalent. The characteristics of the GBM are: $\sigma = 0.45$, $\mu = 0$, starting price 100 EUR, the same as the strike (*i.e.* the option is ATM). The TTM is initially 20 days with 7 time-steps per day, thus the resulting option price at inception is ~ 4.7 EUR and the delta is ~ 0.5. There are 7 rebalancing steps per day, thus one every hour, so a total of 140 steps. The TTM is calculated as year fraction, considering the actual amount of time between every two steps which is different at the end of each working day and during weekends. The $\frac{bid-ask}{2}$ of the transaction costs is considered as 0.05 and we use a zero financing cost. In the following experiments, the results are calculated on 2000 independently run scenarios, a budget of 3000, and a search depth of 6. Figure 4 shows the results obtained on the simulated data, the x-axis is the difference between the variance of the returns generated by the MCTS minus that generated by the delta hedge in both graphs. The graph on the left has as y-axis the difference of the returns between the MCTS agent and the delta hedge while the graph on the right represents the difference in trading costs generated by the MCTS agent and the delta hedge. Each point represents the average of the 2000 independent runs, the blue bars are 95% CI. The number annotated next to the each point represents the coefficient m from Equation (6) used during the search, at each real step in the environment $m = 1$.

We can see that as we increase the coefficient m, we are increasing the returns, decreasing the costs and increasing the risk generated by the MCTS agent. Another characteristic, which can be noticed by comparing the left and right graph, is that most of the over-performance of the MCTS agent w.r.t the delta hedge is given by the reduction in costs.

Real Data. We considered two datasets, one with the SX7E future and 4 options on this future with strike 89, 90, 91, 92 (and starting price 4.9, 4.5, 4.1 and 3.7 EUR respectively) and expiry 17/06/2021 with 8 timesteps per day one each hour from 10am to 5pm CET, starting from 11/03/2021 for a total of 551 timesteps. The second the time series of the ICLN ETF, with an option with strike 23 and unitary starting price 1.6 USD. The expiry was 17/06/2021, there are 8 timesteps per day from 2.30pm to 9.30pm CET starting on 15/04/2021 for a total of 360 timesteps. The $\frac{bid-ask}{2} = 0.05$ for the SX7E future and 0.01 for the ICLN ETF. Given the stochasticity of a MCTS agent, the optimal approach is to run the search multiple times and take the average. The figures and tables compare the behavior of the delta hedge with that of a MCTS agent averaged over 30 searches with 95% CI. In Figure 5 we can see the policy learnt by MCTS agents when changing the parameter m. Specifically, we can see that as m increases the policy becomes smoother, as we are reducing the hedging costs. The delta hedging strategy is the least smooth.

The cumulated p&l results are summarized in Table 1, which shows for each different option and each parameter m, the difference in $P\&L$ ($\Delta P\&L$) and in trading costs (Δ costs) of the MCTS agent with respect to the delta hedge averaged over 30 runs, with the corresponding 95% CI. We ideally would like to achieve $\Delta P\&L > 0$ and as large as possible, with a negative Δcosts. We can see that while not all the agents are capable of statistically significant higher p&l than the delta hedge, all of them achieve the objective of lowering the trading costs of the underlying and as m increases, transaction costs become lower. In our opinion, the main reason for which the results on real data are not as encouraging as those on simulated data is that option price jumps may be more severe than what the generative model predicts. Another reason is that

	$m = 0.05$				$m = 0.5$				$m = 1$				$m = 5$			
	Δp&l	±CI	Δcosts	±CI	Δp&l	±CI	Δcosts	±CI	Δp&l	±CI	Δcosts	±CI	Δp&l	±CI	Δcosts	±CI
ICLN23	-0.127	0.046	-0.090	0.003	-0.084	0.086	-0.149	0.003	-0.100	0.063	-0.155	0.002	-0.078	0.053	-0.158	0.002
SX7E89	-0.003	0.032	-0.066	0.005	0.045	0.043	-0.168	0.003	-0.041	0.030	-0.203	0.004	-0.034	0.041	-0.254	0.002
SX7E90	0.007	0.055	-0.060	0.003	0.009	0.042	-0.154	0.003	0.011	0.047	-0.185	0.003	-0.006	0.039	-0.234	0.003
SX7E91	-0.024	0.034	-0.074	0.004	0.006	0.045	-0.177	0.004	-0.042	0.040	-0.213	0.004	-0.107	0.044	-0.264	0.003
SX7E92	0.037	0.046	-0.072	0.005	0.040	0.058	-0.178	0.004	-0.005	0.054	-0.212	0.003	-0.047	0.044	-0.268	0.003

Table 1: Behavior of average MCTS agent with respect to the delta in terms of terminal p&l and costs, 30 runs, 95% CI. Figures are in EUR (USD for ICLN), annualized and for a single option.

these are results on a single scenario, and more data is necessary to achieve statistically significant results.

7 CONCLUSIONS

In this work we analyzed a MCTS approach to trading and option hedging. Specifically we created a MCTS algorithm, coined QL-OL UCT, which is capable of working with real market data *i.e* with continuous stochastic states, and in the case of option hedging also continuous actions. We applied this algorithm to the trading setting, where we used a novel generative model using historical data and a clustering approach. We saw how this approach behaved on a real-data where we consider the EURUSD FX contract. While we manage to consistently achieve profit, even for small planning budgets without considering transactions costs, adding these costs causes the agents to decide not to trade. In future works, we plan to consider alternate generative models and tree-search procedures to allow to achieve a profit also with the addition of transaction costs. We applied the same algorithm in the option hedging context, testing it also on real data. The results on simulated data are encouraging as the MCTS agent achieves a superior performance than the delta hedge in terms of cumulated profit, but further work needs to be done in order to improve the result on real data where we are not working in an average scenario but there is only one realization which needs to be successfully hedged.

We intend to explore further avenues, specifically working on improving the generative models, for both the trading and option hedging environments. Finally, we intend to extend alphazero [19], which has achieved astonishing experimental results, in order to make it compatible with stochastic states and continuous actions.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[3] Lorenzo Bisi, Pierre Liotet, Luca Sabbioni, Gianmarco Reho, Nico Montali, Cristiana Corno, and Marcello Restelli. Foreign exchange trading: A risk-averse batch reinforcement learning approach. In *ICAIF 2020*, 2020.

[4] Lorenzo Bisi, Luca Sabbioni, Edoardo Vittori, Matteo Papini, and Marcello Restelli. Risk-averse trust region optimization for reward-volatility reduction. In *IJCAI-20*, 7 2020.

[5] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3), 1973.

[6] Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, 2019.

[7] Adrien Couetoux. *Monte Carlo tree search for continuous and stochastic sequential decision making problems*. PhD thesis, 2013.

[8] Thomas Degris and Olivier Sigaud. *Factored Markov Decision Processes*, chapter 4, pages 99–126. John Wiley & Sons, Ltd, 2013.

[9] Michael AH Dempster and Vasco Leemans. An automated fx trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30(3), 2006.

[10] Damien Ernst et al. An application of deep reinforcement learning to algorithmic trading. Technical report, arXiv, 2020.

[11] Thomas G Fischer. Reinforcement learning in financial markets-a survey. Technical report, FAU Discussion Papers in Economics, 2018.

[12] Carl Gold. Fx trading via recurrent reinforcement learning. In *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.*, pages 363–370. IEEE, 2003.

[13] Igor Halperin. The qlbs q-learner goes nuqlear: fitted q iteration, inverse rl, and option portfolios. *Quantitative Finance*, 2019.

[14] Chien Yi Huang. Financial trading as a game: A deep reinforcement learning approach. *arXiv*, 2018.

[15] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[16] Erwan Lecarpentier, Guillaume Infantes, Charles Lesire, and Emmanuel Rachelson. Open loop execution of tree-search algorithms. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2362–2368. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[17] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. Finrl: A deep reinforcement learning library for automated stock trading in quantitative finance. *arXiv*, 2020.

[18] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676), 2017.

[20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[21] Oleg Szehr. Hedging of financial derivative contracts via monte carlo tree search. *arXiv*, 2021.

[22] Edoardo Vittori, Michele Trapletti, and Marcello Restelli. Option hedging with risk averse reinforcement learning. In *ICAIF*, 2020.

[23] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[24] Orhun Hakan Yalincak. Criticism of the black-scholes model: But why is it still used? *SSRN*, 2012.

[25] Jin Zhang and Dietmar Maringer. Using a genetic algorithm to improve recurrent reinforcement learning for equity trading. *Computational Economics*, 47(4):551–567, 2016.