universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Cyclic Arbitrage on Constant Function Market Makers

verfasst von | submitted by

Milutin Popović BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von | Supervisor:	Univ.-Prof. Dr. Radu Ioan Bot Privatdoz.

Acknowledgements

Firstly, I thank my supervisor Univ.-Prof. Dr. Radu Ioan Bot Privatdoz., for having an open mind about the topic of the thesis, for his time, availability, support and constructive feedback throughout the construction and conclusion of the thesis. I also express my gratitude to Assoz. Prof. Dr. Philipp Christian Petersen, for his openness towards the topic, for the early discussions and for the idea to use Reinforcement Learning on graphs to solve for the optimal walk. I thank the University of Vienna and the Vienna Scientific Cluster (VSC) for providing the compute hardware for the calculations. I thank my friends for encouraging me, in the first place, to write a thesis on this topic. And lastly, I thank my family, my friends and my girlfriend for their love and continuous support.

Abstract

Decentralized Exchanges (DEXs) are a new form of financial markets emerging on top of blockchain technology. Through these arises a competitive and complex environment of Maximal Extractable Value (MEV), where entities called Searchers race against the block time and compete against each other to find profitable opportunities when the market is at rest, and the next block is not yet committed. One of these opportunities is an n-cyclic arbitrage, on which this thesis focuses on. Through modeling of the building blocks of DEXs , governed by Constant Function Market Makers (CFMMs), we can derive a nonlinear convex optimization problem finding an optimal n-cyclic arbitrage trade. However this optimization problem neither specifies the order in which these trades should be executed, nor reduces optimization to specific, used DEXs. To combat this, we propose a Markov Decision Process and find an optimal policy through a Monte Carlo Tree Search REINFORCE algorithm guiding us to a profitable n-cyclic arbitrage walk based on the marginal exchange rates. The information of the found walk can be then applied to the nonlinear optimization problem to find an optimal trade in a timely manner.

Vorwort

Dezentrale Börsen (DEXs) sind eine neue Form von Finanzmärkten, die auf der Grundlage der Blockchaintechnologie entstehen. Durch diese ensteht ein wettbewerbsintensives und komplexes Umfeld des Maximal Extractable Value (MEV), in dem sogenannte Searcher, gegen die Blockzeit anlaufen und miteinander konkurrieren, um gewinnbringende Gelegenheiten zu finden. Einer dieser Gelegenheiten ist eine zyklische Arbitrage, auf die sich diese Arbeit konzentriert. Durch die Modellierung der Grundbausteine lässt sich ein nichtlineares konvexes Optimierungsproblem ableiten, das einen optimalen n-zyklischen Arbitrage Trade findet. Dieses Optimierungsproblem gibt jedoch weder die Ausführungsrheinfolge der Trades vor noch beschränkt es die Optimierung auf spezifisch genutzte DEXs. Um dies zu lösen, schlagen wir einen Markow Entscheidungsproblem vor und ermitteln eine optimale Srategie mithilfe eines Monte Carlo Baum Suche REINFORCE Algorithmus, der uns anhand der nominellen Wechselkurse zu einem profitablen n-zyklischen Arbitragepfad führt. Die Informationen des Pfades können dann durch das nichtlineare Optimierungsproblem genutzt werden, um zeitnah einen optimalen Trade zu bestimmen.

Contents

1 Introduction

Maximal Extractable Value (MEV), sometimes also called Miner Extractable Value or Block Proposer Extractable Value, refers to the maximal value that can be extracted from block production (excluding block reward and gas fees) by including, excluding and changing the order of transactions in a block and thereby creating an economic gain. The opportunities, among others, include arbitrage, frontrunning and liquidations. By definition, MEV exists on any decentralized blockchain. This thesis primarily focuses on the application found on the Ethereum blockchain. Nevertheless, everything can be applied to other blockchains with smart contract functionality. Maximizing MEV extraction requires sophisticated technicalities, custom software and high-end hardware, making it more likely for institutional centralized entities to outperform individuals. The centralization is a threat to the most important quality of blockchain technology, decentralization. There are multiple proposals on how to combat MEV centralization, however solutions that evolve completely removing MEV from the picture close one door but open multiple other ones.
Instead, the Ethereum Foundation [1] and Flashbots [2] implemented a reshape of 'MEV economics' through the Proposer-Builder-Separation (PBS) [3]. In short the PBS splits the MEV reward through multiple entities, each playing a distinct role in the ecosystem. The Searchers capture the bulk of the MEV profits. They identify the opportunities and commit bundle transactions to exploit them. The Block Builders construct the blocks by including, excluding and optimizing a set of selected transactions to produce the most profitable blocks. They earn a share of MEV from the Searchers, who are committing bids to the Block Builders to include the bundles. Block Builders can also earn MEV directly by reordering transactions. The last entity that takes a direct cut of MEV are Validators. They propose the most profitable block to the network and outsource block production to block builders via software like MEV-Boost [2]. They earn a share of MEV by competing bids of Block Builders to include their block. In multiple stages the PBS produces a highly competitive environment where usually the highest bid wins. Nevertheless, PBS is not a perfect solution. There have been cases where, in one month, 80% of the blocks have been produced by only two block builders and the majority of the opportunities found by a handful of Searchers.
In the scope of the Ethereum blockchain, this thesis is primarily applicable for Searchers. The aim of the thesis is to contribute to the decentralization of MEV by 'laying it out in the open'.
We focus on a specific opportunity arising from an emerging form of financial markets. These are implemented on Decentralized Exchanges (DEXs). In comparison to traditional markets, based on order books and centralized liquidity market makers, the DEXs are based on Automated Market Makers (AMMs) and decentralized liquidity providers.

AMMs, specifically Constant Function Market Makers (CFMMs), use a trading function and set of rules to automatically enable trading and liquidity provision for any user satisfying the rules. The opportunity in question in this thesis is an n-cyclic arbitrage over multiple DEXs, in which an optimized sequence of trades on n distinct DEXs is executed. The sequence starts from one specific asset, executes the trades, and ends on that same asset with a balance greater than before the trades. We model CFMMs to construct an optimization problem that maximizes the profit of an n-cyclic arbitrage, specifically for UniswapV2 and UniswapV3 CFMMs. In this scenario, Searchers races against the block time, trying to solve multiple of these optimization problems to find a profitable one. When the clock turns and a new block is included, the market equilibrium changes and the optimization problem becomes obsolete. One of the improvements in this case is to make a convex optimization problem. The original implementation allows for a convex relaxation. We can show that the relaxation has the same optimum as the original. This allows us to utilize more efficient convex solvers.
There are two further problems with the proposed optimization problem. These can be solved through one solution. The current model optimizes a trade vector over a set of DEXs, yet there are currently around $4 \cdot 10^5$ DEXs deployed on UniswapV2 and UniswapV3 [4] alone, not counting other implementations. It would be extremely inefficient to optimize over all of these DEXs, and it would still be inefficient and unprofitable to optimize on a restriction of a much smaller set of DEXs. The second problem is that the optimal trade vector does not specify in which order the trades should be executed. We propose a solution to these problems by finding a specific walk on a highly connected graph of DEXs. The graph has nodes representing assets traded on the DEXs. These nodes are connected by an edge if the underlying assets can be traded through a DEX. The weights of the edges are the so-called marginal exchange rates. The goal is to find a walk starting and ending at the same node, maximizing the product of the marginal exchange rates without repeating an edge. The information of this optimal walk can then be used to construct the convex optimization problem and execute the trade by the chronology of the walk. Any brute-force, depth/breadth-first-search solution would take too long. Hence, we propose a Markov Decision Process representing the walk optimization. We find an optimal policy through a modified policy gradient method, specifically a variation of the Monte Carlo Tree Search REINFORCE [5] algorithm inspired by AlphaZero [6].
The chapter outline of the thesis is the following.
In the second chapter we go through the background on first-order optimality conditions for constrained nonlinear optimization problems based on [7]. Through the characterization of optimal solutions based on tangent cones, we can show that an optimal solution, under certain constraint qualifications, implies the existence of the Lagrange multipliers satisfying the Karush-Kuhn-Tucker (KKT) conditions. These conditions are necessary for optimality. In the case where the objective function and the inequality restrictions are convex and the equality restrictions are affine linear, we can apply the Slater constraint qualification. Under this assumption, the KKT conditions are both necessary and sufficient for optimality.

In the third chapter, we model DEXs through CFMMs. Modelling of CFMMs and optimal trades has already been introduced in the literature, i.e. in [8] and [9] on which this chapter is based on. By gradually building the foundations of general trades, we can construct an optimization problem optimizing a trade over one CFMM. We can show that the convex relaxation of the problem, where we relax the equality constraints to inequality constraints, has the same optimal solution as the original [8]. We can inductively increase the number of CFMMs for which we optimize a trade vector, giving a general optimization problem for a n-cyclic arbitrage [9] and specifically model a n-cyclic arbitrage scenario on two asset constant product market markers.
In the fourth chapter, we give a brief introduction to Markov Decision Processes (MDPs), Graph Neural Networks (GNNs), Monte Carlo Tree Search (MCTS) and a policy gradient method called REINFORCE [5]. All of these will be utilized for construction of the algorithm used to train the policy parametrized by a deep graph neural network for finding an n-cyclic arbitrage walk.
In the fifth and final chapter, we model the underlying MDP of finding the n-cyclic arbitrage walk. We describe the specific architecture of GNN parametrizing the policy and the implementation of a modified REINFORCE MCTS parameter updates. Lastly, we give an overview of the results of the policy optimization.

2 Background on Nonlinear Optimization

In this chapter we will go through the necessary background to understand optimality conditions for nonlinear constrained optimization problems. We will introduce necessary optimality conditions. In the case we are optimizing a convex function under convex inequality constraints and affine linear equality constraints, can show that under certain qualifications the necessary conditions are also sufficient for optimality. These are all very well known results from the theory of nonlinear optimization, hence this chapter is based on the lecture notes in [7].

2.1 First Order Optimality Conditions

Generally we are considering a nonempty subset $X \subseteq \mathbb{R}^n$ for $n \in \mathbb{N}$ and a function $f : \mathbb{R}^n \to \mathbb{R}$ in the context of an optimization problem

$$\min_{x \in X} f(x). \tag{2.1}$$

A local minimum of this problem is an element $x^* \in X$, such that there is an $\varepsilon > 0$ ball, wrt. to the euclidean norm, i.e.

$$U_\varepsilon(x^*) = \{x \in \mathbb{R}^n : \|x - x^*\| < \varepsilon\}, \tag{2.2}$$

such that

$$f(x^*) \leq f(x) \qquad \forall x \in U_\varepsilon(x^*) \cap X. \tag{2.3}$$

A global minimum is an element $x^* \in X$ such that the inequality holds on the whole set X.

$$f(x^*) \leq f(x) \qquad \forall x \in X. \tag{2.4}$$

To better understand local minima we look at directions, from a given point x_0, from where we can move and still remain in X. In this sense we introduce a cone

$$T_X(x_0) := \left\{ d \in \mathbb{R}^n \,\middle|\, \exists (x^k)_{k\in\mathbb{N}} \subseteq X,\ \exists (t_k)_{k\in\mathbb{N}} \searrow 0 \ : \lim_{k\to\infty} \frac{x^k - x_0}{t_k} = d \right\}, \tag{2.5}$$

called the Bouligard tangent cone. It is nonempty since $0 \in T_X(x_0)$ for the choice of $x^k = x_0$ and $t_k = \frac{1}{k}$. It is really a cone since it is scaling invariant for $d \in T_X(x_0)$ also $\lambda d \in T_X(x_0)$, for all $\lambda > 0$, by multiplying $(t_k)_{k\in\mathbb{N}}$ with $\frac{1}{\lambda}$.

Proposition 2.1

let x^* be a local minimum of (2.1), where $f \in C^1(U_\varepsilon(x^*))$, for an $\varepsilon > 0$. Then

$$\nabla f(x^*)^T d \geq 0 \qquad \forall d \in T_X(x^*). \tag{2.6}$$

Proof. Fix $d \in T_X(x^*)$, then there are sequences $(x^k)_{k\in\mathbb{N}}$ and $(t_k)_{k\in\mathbb{N}}$ such that

$$\lim_{k\to\infty} \frac{x^k - x^*}{t_k} = d. \tag{2.7}$$

Multiplying by t_k gives

$$\lim_{k\to\infty} x^k - x^* = \lim_{k\to\infty} \frac{x^k - x^*}{t_k} t_k = \lim_{k\to\infty} dt_k = 0. \tag{2.8}$$

Since x^* is a local minimum of (2.1) we have that $f(x^*) \leq f(x)$ for all $x \in U_\delta(x^*) \cap X$, where $0 < \delta < \varepsilon$. So there is a $k_\delta \in \mathbb{N}$ such that $x^k \in U_\delta(x^*)$ for all $k \geq k_\delta$. By the mean value theorem there is a $\xi^k \in \{\lambda x^* + (1-\lambda)x^k | \lambda \in [0,1]\}$, such that

$$f(x^k) - f(x^*) = \nabla f(\xi^k)(x^k - x^*). \tag{2.9}$$

Let $\xi^k = \lambda_k x^* + (1-\lambda_k)x^k$, then for all $k \geq k_\delta$ we have

$$\begin{aligned} \|\xi^k - x^k\| &= \|\lambda_k x^* + (1-\lambda_k)x^k - x^*\| && (2.10)\\ &= |\lambda_k - 1|\|x^k - x^*\| && (2.11)\\ &\leq \|x^k - x^*\| \to 0. && (2.12) \end{aligned}$$

And thus $\xi^k \to x^*$. By the continuity of the gradient in the neighborhood of x^* we also have $\nabla f(\xi^k) \to \nabla f(x^*)$, thereby

$$\begin{aligned} \nabla f(x^*)^T d &= \lim_{k\to\infty} \frac{\nabla f(\xi^*)(x^k - x^*)}{t_k} && (2.13)\\ &= \lim_{k\to\infty} \frac{f(x^k) - f(x^*)}{t_k} && (2.14)\\ &\geq 0. && (2.15) \end{aligned}$$

□

The statement in Proposition 2.1, basically implies that at a local minimum x^* there is no tangent direction or 'descent direction' for which $\phi'(0) < 0$, where $\phi(t) := f(x^* + td)$.

Bringing everything together gives the first order optimality condition for unconstrained problems.

Theorem 2.1

Let x^* be a local minimum of (2.1) and $f \in C^1(U_\varepsilon(x^*))$ for an $\varepsilon > 0$. Then x^* is a critical point of f, i.e.

$$\nabla f(x^*) = 0. \tag{2.16}$$

Proof. Since x^* is a local minimum of (2.1) and $f \in C^1(U_\varepsilon(x^*))$ then x^* is also a local minimum in of f in $U_\varepsilon(x^*)$. By Proposition 2.1 we have that $\nabla f(x^*)^T d \geq 0$, for all $d \in T_{U_\varepsilon(x^*)}(x^*) = \mathbb{R}^n$. Fix $d \neq 0$ and plug in d and $-d$, giving $\nabla f(x^*)^T d = 0$. Hence $\nabla f(x^*) = 0$. □

From now on consider a particular set X, the feasible set, defining a constrained optimization problem through

$$X = \left\{ x \in \mathbb{R}^n \middle| \begin{array}{l} g_i(x) \leq 0,\ i = 1, \ldots, m \\ h_j(x) = 0,\ j = 1, \ldots, p \end{array} \right\}, \tag{2.17}$$

where the functions $g_i, h_j \in C^1(\mathbb{R})$ for all $i = 1, \ldots, m$ and $j = 1, \ldots, p$. Then we have a general constrained nonlinear optimization problem

$$\begin{aligned} &\min\ f(x) \\ &\text{s.t.: } g_i(x) \leq 0,\ i = 1, \ldots, m \\ &\qquad h_j(x) = 0,\ j = 1, \ldots, p \\ &\qquad x \in \mathbb{R}^n, \end{aligned} \tag{2.18}$$

i.e. $x \in X$, in the feasible set.

Generally the Bouligard tangent cone $T_X(x_0)$ for a $x_0 \in X$ is not a practical object, since it cannot be determined out of the box. Hence, we introduce the linearized tangent cone, which will serve as a replacement and can be easily determined for a given feasible set X. Also, in a specific setting the linearized tangent cone will be valid representation of the Bouligard tangent cone. For an element $x_0 \in X$ consider

$$\mathcal{A}(x_0) = \{ i = 1, \ldots, m |\ g(x_0) = 0 \}, \tag{2.19}$$

the set of active indices at x_0 and

$$\mathcal{I}(x_0) = \{1, \ldots, m\} \setminus \mathcal{A}(x_0), \tag{2.20}$$

the set of inactive indices at x_0. The linearized tangent cone of X at x_0 is defined as

$$T_{\text{lin}}(x_0) = \left\{ d \in \mathbb{R}^n \middle| \begin{array}{l} \nabla g_i(x_0)^T d \leq 0,\ i \in \mathcal{A}(x_0) \\ \nabla h_j(x_0)^T d = 0,\ j = 1, \ldots, p. \end{array} \right\}. \tag{2.21}$$

The linearized tangent cone is really a cone since the scaling with $\lambda > 0$ does not change the outcome of the equalities and inequalities.

Lemma 2.1

Let $x_0 \in X$, then it holds that $T_X(x_0) \subseteq T_{\text{lin}}(x_0)$.

Proof. We start with a $x_0 \in X$ and $d \in T_X(x_0)$. Then there are sequences by the definition of the Bouligard tangent cone, $(x^k)_{k\in N}$ and $(t_k)_{k\in N} \searrow 0$, such that

$$\lim_{k\to\infty} \frac{x^k - x_0}{t_k} = d. \tag{2.22}$$

First we prove that $\nabla g_i(x_0)^T d \leq 0$, for all active indices $i \in \mathcal{A}(x_0)$. For this we use the mean value theorem, for all $k \geq 1$ there is a $\xi^k \in \{\lambda x_0 + (1-\lambda)x^k |\ \lambda \in [0,1]\}$ such that

$$g_i(x^k) - g_i(x_0) = \nabla g_i(\xi^k)^T (x^k - x_0). \tag{2.23}$$

Since i is active, we have $g_i(x_0) = 0$ and thus

$$\nabla g_i(\xi^k)^T (x^k - x_0) = g_i(x^k) \leq 0. \tag{2.24}$$

Dividing by t_k and passing the limit gives

$$0 \geq \lim_{k\to\infty} \nabla g_i(\xi^k)^T \frac{x^k - x_0}{t_k} = \nabla g_i(x_0)^T d. \tag{2.25}$$

For the second statement in $T_{\text{lin}}(x_0)$ we need to show that $\nabla h_j(x_0)^T d = 0$ for all $j = 1, \ldots, p$. For this we can use the same argumentation, by the mean value theorem for all $k \geq 1$ there is a $\xi^k \in \{\lambda x_0 + (1-\lambda)x^k |\ \lambda \in [0,1]\}$ such that

$$h_j(x^k) - h_j(x_0) = \nabla h_j(\xi^k)^T (x^k - x_0). \tag{2.26}$$

Since $h_j(x_0) = 0$ and $h_j(x^k) = 0$ we have that

$$\nabla h_j(\xi^k)^T (x^k - x_0) = 0. \tag{2.27}$$

Dividing by t_k and passing the limit gives

$$0 = \lim_{k\to\infty} \nabla h_j(\xi^k)^T \frac{x^k - x_0}{t_k} = \nabla h_j(x_0)^T d. \tag{2.28}$$

Thus $d \in T_{\text{lin}}(x_0)$. □

Lemma 2.2 *Lemma of Farkas*

Let $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. Then the two statements are equivalent

1. The system $Ax = b$ has a solution $x \geq 0$.
2. For all $d \in \mathbb{R}^m$ with $A^T d \geq 0$ it holds that $b^T d \geq 0$.

For the next result we also need to introduce the dual cone.

Definition 2.1

Let $K \subseteq \mathbb{R}^n$ be a cone. The dual cone of K is denoted by K^*, defined by

$$K^* = \{s \in \mathbb{R}^n | \; s^T d \geq 0 \; \forall d \in K\}. \tag{2.29}$$

Theorem 2.2

Let $x_0 \in X$, the it holds that

$$-(T_{lin}(x_0))^* = N_{\text{lin}}(x_0). \tag{2.30}$$

where

$$N_{\text{lin}} = \left\{ \sum_{i=1}^{m} \lambda_i \nabla g_i(x_0) + \sum_{j=1}^{p} \mu_j \nabla h_j(x_0) \;\middle|\; \begin{array}{l} \lambda_i \geq 0, \; i \in \mathcal{A}(x_0), \\ \lambda_i = 0, \; i \in \mathcal{I}(x_0), \\ \mu_j \in \mathbb{R}, \; j = 1, \ldots, p \end{array} \right\} \tag{2.31}$$

Proof. Direction $\supseteq$: Let $s \in N_{\text{lin}}(x_0)$ and $d \in T_{\text{lin}}(x_0)$. Then

$$-s^T d = -\sum_{i=1}^{m} \lambda_i \nabla g_i(x_0)^T d - \sum_{j=1}^{p} \mu_j \nabla h_j(x_0)^T d \tag{2.32}$$

$$\geq 0. \tag{2.33}$$

by $\lambda_i \geq 0$, $\nabla g_i(x_0)^T d \leq 0$ and $\nabla h_j(x_0)^T d = 0$. Hence $s \in -(T_{\text{lin}}(x_0))^*$.

Direction $\subseteq$: Let $s \in -(T_{\text{lin}}(x_0))^*$, then by definition $-s^T d \geq 0$ for all $d \in T_{\text{lin}}(x_0)$. We can use Lemma 2.1 with the matrix $A \in \mathbb{R}^{n \times (r+2p)}$, with active indices $\mathcal{A}(x_0) = \{i_1, \ldots, i_r\}$ defined as

$$A := \left(A_g, A_h, -A_h\right), \tag{2.34}$$

where

$$A_g = \left(-\nabla g_{i_1}(x_0), \ldots, -\nabla g_{i_r}(x_0)\right), \tag{2.35}$$

$$A_h = \left(\nabla h_1(x_0), \ldots, \nabla h_p(x_0)\right). \tag{2.36}$$

By Lemma 2.1 there is a $\nu = (\lambda, \mu_1, \mu_2) \in \mathbb{R}^r_+ \times \mathbb{R}^p_+ \times \mathbb{R}^p_+$ satisfying

$$A\nu = -s. \tag{2.37}$$

This is equivalent to

$$\sum_{i=1}^{m} -\lambda_i \nabla g_i(x_0) + \sum_{j=1}^{p} (\mu_1)_j \nabla h_j(x_0) + \sum_{j=1}^{p} -(\mu_2)_j \nabla h_j(x_0) = -s. \tag{2.38}$$

By setting $\mu = \mu_2 - \mu_1$ and the fact that $\lambda_i \geq 0$ for all $i \in \mathcal{A}(x_0)$ and $\lambda_i = 0$ for all $i \in \mathcal{I}(x_0)$ we have

$$s = \sum_{i=1}^{m} \lambda_i \nabla g_i(x_0) + \sum_{j=1}^{p} \mu_j \nabla h_j(x_0), \tag{2.39}$$

thus $s \in N_{\text{lin}}(x_0)$. □

Next we will introduce the KKT conditions

Definition 2.2

The function $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ defined as

$$L(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x) \tag{2.40}$$

is called the Lagrangian function of the optimization problem (2.18). The conditions

$$\begin{cases} \nabla_x L(x, \lambda, \mu) = 0, \\ \lambda \geq 0,\ g(x) \leq 0,\ \lambda^T g(x) = 0 \\ h(x) = 0 \end{cases} \tag{2.41}$$

are called KKT optimality conditions, where

$$\nabla_x L(x, \lambda, \mu) = \nabla f(x) + \lambda^T \nabla g(x) + \mu^T \nabla h(x) \tag{2.42}$$

$$= \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x) + \sum_{j=1}^{p} \mu_j \nabla h_j(x). \tag{2.43}$$

An element (x^*, λ^*, μ^*) fulfilling the KKT optimality conditions is called a KKT-point. The vectors λ^* and μ^* are called Lagrange multipliers associated with the corresponding restrictions, i.e. $g(x^*) \leq 0$ and $h(x^*) = 0$.

In the case there are no restrictions, the KKT optimality conditions are $\nabla f(x) = 0$. Thus the scope of these conditions is a generalization of the first order optimality for constrained optimization and these play are necessary conditions for a solution to be optimal. The KKT conditions together with Abadie constraint qualification are necessary for a optimality. However, in the case the problem is convex together with Slater constraint qualification the KKT-conditions are also sufficient.

Definition 2.3

Let $x_0 \in X$, where X is the feasible set from equation (2.18). Then x_0 fulfills Abadie constraint qualification (CQ) if

$$T_X(x_0) = T_{\text{lin}}(x_0). \tag{2.44}$$

A particularly nice thing about Abadie CQ is that it does not assume anything about the objective function f and a very nice result can be proven.

Theorem 2.3

Let x^* be local minimum of (2.18), fulfilling Abadie CQ. Then there exist Lagrange multipliers $\lambda^* \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^p$, such that (x^*, λ^*, μ^*) is KKT point of the constrained optimization problem.

Proof. The strategy is to utilize the cone duality and already known results. From Proposition 2.1 we know that $\nabla f(x^*) \in (T_X(x^*))^*$, and since x^* fulfills Abadie CQ it

holds that $(T_X(x^*))^* = (T_{\text{lin}}(x^*))^*$. Thus $-\nabla f(x^*) \in -(T_{\text{lin}}(x^*))^*$. From Theorem 2.1 we have that $-\nabla f(x^*) \in N_{\text{lin}}(x^*)$ and hence we can write

$$-\nabla f(x^*) = \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla g_i(x^*) + \sum_{j=1}^{p} \mu_j^* h_j(x^*), \tag{2.45}$$

where we know there are $\lambda_i^* \geq 0$ for all $i \in \mathcal{A}(x^*)$ and $\lambda_i^* = 0$ for all $i \in \mathcal{I}(x^*)$ and $\mu_j^* \in \mathbb{R}$ for all $j = 1, \ldots, p$. Then we expand the sum and rearrange to get

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_m^* \nabla g(x^*) + \sum_{j=1}^{p} \mu_j^* \nabla h_j(x^*) = 0 \tag{2.46}$$

which is the first condition in the KKT conditions

$$\nabla_x L(x^*, \lambda^*, \mu^*) = 0. \tag{2.47}$$

For the other conditions, we have for i is active $g_i(x^*) = 0$ and for i inactive $\lambda_i^* = 0$. Thus $\lambda_i^* \geq 0$, $g_i(x^*) \leq 0$, $\lambda_i g(x^*) \leq 0$. Also by construction of the problem $h_j(x^*) = 0$ for all $j = 1, \ldots, p$. □

2.2 Convex Case

Abadie CQ is an excellent result, giving us a hint of finding x^*. However, it assumes something about x^* before it is even found. In the case of convex optimization problems we can derive a relaxed global condition on the optimization problem which implies Abadie CQ. Firstly a brief introduction on convex functions

Definition 2.4

Let $U \subseteq \mathbb{R}^n$ be a convex set. The function $f : U \to \mathbb{R}$ is called convex if for all $x, y \in U$ and $\lambda \in [0, 1]$ it holds that

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y). \tag{2.48}$$

An important result we will use throughout is the following.

Proposition 2.2

Let $U \subseteq \mathbb{R}^n$ be a nonempty, open and convex set, $f : U \to \mathbb{R}$ a differentiable function on U. Then the following statements are equivalent

1. f is convex on U;
2. $\nabla f(x)^T (y - x) \leq f(y) - f(x)$ for all $x, y \in U$;
3. $(\nabla f(x) - \nabla f(y))^T (y - x) \geq 0$ for all $x, y \in U$;
4. if $f \in C^2(U)$, then $\Delta f(x)$ is positive semidefinite for all $x \in U$.

Proof. We show (1) ⇔ (2) ⇔ (3), (4) ⇒ (2) and (3) ⇒ (4). Starting off with (1) ⇒ (2) by definition we have that $\forall x, y \in U$ and $\lambda \in [0, 1]$ it holds that

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y). \tag{2.49}$$

Rearranging, dividing by λ and expanding the fraction by $x - y$ we get

$$\frac{f(y + \lambda(x - y)) - f(y)}{\lambda(x - y)}(x - y) \leq f(x) - f(y). \tag{2.50}$$

Letting $\lambda \downarrow 0$ we get

$$\nabla f(y)^T(x - y) \leq f(x) - f(y). \tag{2.51}$$

For (2) $\Rightarrow$ (1) we know that U is convex hence fix $x, y \in U$ then for all $\lambda \in [0, 1]$ there is a z in the line segment of x and y such that $z = \lambda x + (1 - \lambda)y$. We can construct two inequalities

$$\nabla f(z)^T(x - z) \leq f(x) - f(z) \tag{2.52}$$

$$\nabla f(z)^T(y - z) \leq f(y) - f(z). \tag{2.53}$$

By multiplying the first one with λ and second one with $(1-\lambda)$ and adding them together we get

$$\nabla f(z)^T(\lambda x + (1 - \lambda)y - z) \leq \lambda f(x) + (1 - \lambda)f(y) - f(z). \tag{2.54}$$

Inserting for z we get $\lambda x + (1 - \lambda)y - z = 0$ and thus

$$f(z) = f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y). \tag{2.55}$$

For (2) $\Rightarrow$ (3) we consider two inequalities for all $x, y \in U$

$$\nabla f(x)^T(y - x) \leq f(y) - f(x) \tag{2.56}$$

$$\nabla f(y)^T(x - y) \leq f(x) - f(y). \tag{2.57}$$

Adding them together gives

$$(\nabla f(x) - \nabla f(y))^T(x - y) \leq 0. \tag{2.58}$$

For (3) $\Rightarrow$ (2) we consider a function $g(t) = f(tx + (1 - t)y)$ defined on the line segment of $x, y \in U$ for $t \in [0, 1]$, well-defined since U is convex. The derivative of g is

$$g'(t) = \nabla f(tx + (1 - t)y)^T(y - x) \tag{2.59}$$

Note that by (3) we have that

$$\nabla f(x)^T(y - x) \leq \nabla f(y)^T(y - x), \tag{2.60}$$

for all $x, y \in U$. Now we use the Mean Value Theorem on g,

$$f(y) - f(x) = \int_0^1 g'(t)dt \tag{2.61}$$

$$= \int_0^1 \nabla f(tx + (1 - t)y)^T(y - x)dt \tag{2.62}$$

$$\geq \int_0^1 \nabla f(x)^T(y - x)dt \tag{2.63}$$

$$= \nabla f(x)^T(y - x). \tag{2.64}$$

For (4) $\Rightarrow$ (2) we use Taylor's Theorem, i.e. for all $x, y \in X$ we have

$$f(y) = f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}(y-x)^T\nabla^2 f(\xi)(y-x), \tag{2.65}$$

for some ξ in the line segment of x and y. We have that the Hessian is positive semidefinite $\frac{1}{2}(y-x)^T\nabla^2 f(\xi)(y-x) \geq 0$ thus rearranging gives

$$\nabla f(x)^T(y-x) \leq f(y) - f(x) \tag{2.66}$$

On the other hand we show (3) $\Rightarrow$ (4). We take the function $g(t) = f(tx + (1-t)y)$ defined on the line segment of $x, y \in U$ for $t \in [0,1]$. Note that g is convex if and only if $g''(t) \geq 0$ for all $t \in [0,1]$. Since f is convex so is g. The second derrivative of g w.r.t. to t is

$$g''(t) = (y-x)^T\nabla f(tx + (1-t)y)^T(y-x). \tag{2.67}$$

In particular for $g''(0)$ we have

$$g''(0) = (y-x)^T\nabla f(y)(y-x) \geq 0. \tag{2.68}$$

□

Throughout this section consider the following convex problem

$$\begin{aligned} &\min f(x) \\ &\text{s.t.: } g_i(x) \leq 0,\ i = 1, \ldots, m \\ &\qquad Ax = b, \\ &\qquad x \in \mathbb{R}^n, \end{aligned} \tag{2.69}$$

where f, g_i are convex and continuously differentiable functions for all $i = 1, \ldots, p$ and the function h are affine linear, i.e. $h(x) = Ax - b$, where $A \in \mathbb{R}^{p \times n}, b \in \mathbb{R}^p$. The feasible set X of problem (2.69) is a convex set in this case. An important condition for convex problems is the so called Slater constraint qualification (CQ).

Definition 2.5

The optimization problem (2.69) fulfills the Slater CQ if there exists an $x' \in \mathbb{R}^n$ such that

$$g_i(x') < 0,\ i = 1, \ldots, m \tag{2.70}$$
$$Ax' = b. \tag{2.71}$$

Given Slater CQ we can prove the following

Theorem 2.4

Let x^* be a local minimum of (2.69) and the problem fulfills Slater CQ. Then there exist Lagrange multipliers $\lambda^* \in \mathbb{R}^m$ and $\mu^* \in \mathbb{R}^p$ such (x^*, λ^*, μ^*) is a KKT point

of (2.69).

Proof. We show that x^* fulfills Abadie CQ, i.e. $T_{\text{lin}}(x^*) \subseteq T_X(x^*)$. Let $d \in T_{\text{lin}}(x^*)$ and choose an $x' \in X$ satisfying Slater CQ. Set $d' := x' - x^*$, by Proposition 2.2 (2) we have for all $i \in \mathcal{A}(x^*)$ the following

$$\begin{aligned} \nabla g_i(x^*)^T d' &= \nabla g_i(x^*)^T (x^* + d' - x^*) && (2.72)\\ &\leq g_i(x^* + d') - g_i(x^*) && (2.73)\\ &< 0, && (2.74) \end{aligned}$$

where $g(x') < 0$ by Slater CQ and $g_i(x^*) = 0$ for all active indices which we considered. Also

$$Ad' = Ax' - Ax^* = b - b = 0. \tag{2.75}$$

Let $d(\tau) = d + \tau d'$ for all $\tau > 0$, note that $d = \lim_{\tau \to 0} d(\tau)$. Then we have

$$\nabla g_i(x^*)^T d(\tau) = \nabla g_i(x^*)^T d + \tau \nabla g_i(x^*)^T d' < 0, \tag{2.76}$$

for all $i \in \mathcal{A}(x^*)$ and

$$Ad(\tau) = Ad + \tau Ad' = 0. \tag{2.77}$$

We show that $d(\tau) \in T_X(x^*)$ for all $\tau > 0$, which concludes the proof by the closure of $T_X(x^*)$. The cone $T_X(x^*)$ in the case X is convex can be written as $T_X(x^*) = \text{cl}(\text{cone}(X - x^*))$. Let

$$\begin{aligned} x^k &= x^* + \frac{1}{k} d(\tau), && (2.78)\\ t_k &= 1/k, && (2.79) \end{aligned}$$

for $k \in \mathbb{N}$. This gives

$$\frac{x^k - x^*}{t_k} = d(\tau) \tag{2.80}$$

To finish the proof, we need to show that $x^k \in X$. We show that $g_i(x^k) \leq 0$ for all $i = 1, \ldots, m$ and $Ax^k - b = 0$ for all k large enough. Indeed for all $i \in \mathcal{A}(x^*)$ by the Mean Value Theorem, we can find for each $k \geq 1$ a ξ^k in the line segment of x^* and x^k such that

$$\begin{aligned} g_i(x^k) &= g_i(x^k) - g_i(x^*) && (2.81)\\ &= \nabla g_i(\xi^k)^T (x^k - x^*) && (2.82)\\ &= \frac{1}{k} \nabla g_i(\xi^k)^T d(\tau). && (2.83) \end{aligned}$$

Thus there is a $k_0^i \geq 1$ such that

$$kg_i(x^k) < 0 \quad \forall k \geq k_0^i, \tag{2.84}$$

and therefore

$$g_i(x^k) < 0 \quad \forall k \geq k_0^i, \tag{2.85}$$

For $i \in \mathcal{I}(x^*)$, by definition $g_i(x^*) < 0$ and $x^k \to x^*$ by letting k go to infinity, i.e.

$$\lim_{k\to\infty} g_i(x^k) = g_i(x^*) < 0. \tag{2.86}$$

So there is a $k_1^i \geq 1$ such that

$$g_i(x^k) < 0 \quad \forall k \geq k_1^i. \tag{2.87}$$

Taking $k_0 = \max_{i=1,\dots,m}\{k_0^i, k_1^i\}$ shows the first part. Lastly

$$Ax^k = Ax^* + \frac{1}{k}Ad(\tau) = b, \tag{2.88}$$

since $x^* \in X$ and $Ad(\tau) = 0$. This shows $d(\tau) \in T_X(x^*)$ and by continuity $d \in T_X(x^*)$. □

Thus Salter CQ implies Abadie CQ and KKT-conditions in this case are also sufficient for optimality.

Theorem 2.5

Let (x^*, λ^*, μ^*) be a KKT point of (2.69). Then x^* is a local (global) minimum of (2.69).

Proof. Since $x^* \in X$ and (x^*, λ^*, μ^*) a KKT point of (2.69) and by Proposition 2.2 (2) we have

$$f(x) \geq f(x^*) + \nabla f(x^*)^T(x - x^*) \tag{2.89}$$

$$= f(x^*) + \left(-\sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*)^T - \sum_{j=1}^{p} \mu_j^* a_j^T \right)(x - x^*) \tag{2.90}$$

$$= f(x^*) - -\sum_{i\in\mathcal{A}(x^*)}^{m} \lambda_i^* \nabla g_i(x^*)^T(x - x^*), \tag{2.91}$$

where a_j is the j-th row of A. Using the convexity of g and the constraint of $g(x) \leq 0$ we get

$$f(x) \geq f(x^*) - \sum_{i\in\mathcal{A}(x^*)}^{m} \lambda_i^* \left(g_i(x) - g_i(x^*)\right) \tag{2.92}$$

$$= f(x^*) - \sum_{i\in\mathcal{A}(x^*)}^{m} \lambda_i^* g_i(x) \tag{2.93}$$

$$\geq f(x^*). \tag{2.94}$$

□

3 Constant Function Market Makers

On top of blockchain technology, a new form of financial markets has found its roots, they are called Decentralized Exchanges (DEXs). As in any financial market, DEXs allow agents to trade cryptocurrencies, which we will refer to throughout as an asset. Thorough a decentralized ledger, stored and copied on multiple computers (blockchain), the asset ownership is being tracked by a storage slot of a smart contract. All agents holding an asset have an entry in the storage slot, equivalent to a database entry in the asset's smart contract with their wallet address and an amount. In this chapter we go through the underlying principles of DEXs and their properties by mathematical modeling and optimization to arrive at optimal trades and ultimately model arbitrage scenarios. In recent years a lot of groundwork on modeling has been done in [8] and on optimal trades in [9], on which this chapter is based on. We give an extensive description in a specific case of optimal arbitrage in a network of CFMMs all based on [4] and [10].

3.1 Basics

A DEX operates based on a constant function market maker (CFMM), which consists of a trading function $\varphi : \mathbb{R}^n_+ \to \mathbb{R}_+$, along with a set of conditions. The conditions define the way an agent interacts with a DEX. Here $n \geq 2$ is the number of assets an agent can trade, these are labeled $\{1, \ldots, n\}$. The CFMM is a function of the reserves $y \in \mathbb{R}^n_+$, where $y_i \in \mathbb{R}_+$ is the reserve of asset $i \in \{1, \ldots, n\}$ [8]. The reserves are the liquidity of the DEX and generally lie in its own smart contract portfolio. DEXs are also sometimes referred to simply as liquidity pools. It might sound confusing but the terms DEX, CFMM and liquidity pool can be used interchangeably. There are two underlying ways an agent can interact with a DEX:

- Change liquidity
- Trade

We will only look at the trading interaction which lies in the scope of the thesis.

An agent initializes a trade on a DEX, in which a set amount of assets is traded for another [8]. A set amount of assets given is denoted by $x \in \mathbb{R}^n_+$, and the set amount of assets received by $\tilde{x} \in \mathbb{R}^n_+$. Then $x_i, \tilde{x}_i \in \mathbb{R}_+$ is the amount of given and received asset $i \in \{1, \ldots, n\}$ respectively. A trade can either be accepted or rejected, depending on an acceptance principle of the underlying CFMM. If the trade is accepted the state of the

pool is updated by changing the reserves

$$y \mapsto y + x - \tilde{x}. \tag{3.1}$$

A necessary condition for trade acceptance is that $y \geq 0$ after the trade. In the case the trade is rejected the reserves do not change. Throughout we assume that the vectors x and $\tilde{x}$ have disjoint support [8]. This is a valid assumption, since when trading, the agent is never giving and receiving the same asset, but rather one or multiple different ones. The support of a vector $v \in \mathbb{R}^n_+$ is $\text{supp}(v) := \{i \mid v_i > 0\}$, thus the assumption is $\text{supp}(x) \cap \text{supp}(\tilde{x}) = \emptyset$.

A trade $(x, \tilde{x})$ is accepted, if the trading function φ stays constant during the trade [8]

$$\varphi(y + \gamma x - \tilde{x}) = \varphi(y). \tag{3.2}$$

Depending on the CFMM, a trade has a fee $\gamma \in (0, 1]$. Then $(1 - \gamma)x$ of the set amount of given assets is distributed to liquidity providers as a compensation for providing liquidity. The discounted amount γx is actually being exchanged for $\tilde{x}$, but the reserves are updated by the full amount $x - \tilde{x}$.

Throughout assume that the trading function φ is concave, increasing and differentiable, as it is implemented by Uniswap [4], Balancer [11], Sushiswap [12], CurveFI [13] and many others. Most of them are also positively homogeneous $\varphi(\alpha y) = \alpha\varphi(y)$, for all $\alpha \geq 0$ [8].

3.1.1 CFMMs Examples

The weighted sum trading function [8] is

$$\varphi(y) = p^T y = \sum_{i=1}^{n} p_i y_i, \tag{3.3}$$

where p_i is called the price of asset i. The trade acceptance condition is equivalent to

$$\begin{aligned} & \varphi(y + \gamma x - \tilde{x}) = \varphi(y) \\ \Leftrightarrow \quad & p^T(y + \gamma x - \tilde{x}) = p^T y \\ \Leftrightarrow \quad & \gamma p^T x = p^T \tilde{x}. \end{aligned} \tag{3.4}$$

The weighted geometric mean trading [8] function

$$\varphi(y) = \prod_{i=1}^{n} y_i^{w_i}, \tag{3.5}$$

where $w_i > 0$ and $\sum_{i=1}^{n} w_i = 1$. The trade acceptance condition can be simplified by splitting the product (ln of sum) in the indices of $\text{supp}(x)$ and $\text{supp}(\tilde{x})$, [8]

$$\varphi(y + \gamma x - \tilde{x}) = \varphi(y)$$

$$\prod_{\substack{i \in \text{supp}(x) \\ j \in \text{supp}(\tilde{x}) \\ k \notin \text{supp}(x) \cup \text{supp}(\tilde{x})}} (y_i + \gamma x_i)^{w_i} (y_j - \tilde{x}_j)^{w_j} (y_k)^{w_k} = \prod_{i=1}^{n} (y_i)^{w_i}$$

$$\prod_{\substack{i \in \text{supp}(x) \\ j \in \text{supp}(\tilde{x})}} \left(1 + \frac{\gamma x_i}{y_i}\right)^{w_i} \left(1 - \frac{\tilde{x}_j}{y_j}\right)^{w_j} = 1. \tag{3.6}$$

In the case we are trading one for one of the tradeable assets, we have $|\text{supp}(x)| = |\text{supp}(\tilde{x})| = 1$, as in [8]. Let i and j be the corresponding indices of the giving and receiving asset then an analytic expression can be explicitly derived for x_i (or $\tilde{x}_j$) by taking the power of w_i (or w_j) and rearranging equation (3.6) gives

$$x_i = \frac{y_i}{\gamma} \left(\frac{y_j^{w_j/w_i}}{(y_j - \tilde{x}_j)^{w_j/w_i}} - 1 \right) \tag{3.7}$$

$$\tilde{x}_j = y_j \left(1 - \frac{y_i^{w_i/w_j}}{(y_i + \gamma x_i)^{w_i/w_j}} \right). \tag{3.8}$$

Note that the weighted geometric mean is positively homogeneous because the weights sum up to one, i.e. $\sum_{i=1}^{n} w_i = n$

Curve [14] market makers use the following trading function

$$\varphi(y) = \sum_{i=1}^{n} y_i - \alpha \prod_{i=1}^{n} y_i^{-1}, \tag{3.9}$$

where $\alpha > 0$. This trading function is used for stable coin markets to reduce price impact without large counter liquidity. An aggregated generalization is a combination between the geometric weighted mean and the (un)weighted sum [8] is

$$\varphi(y) = (1 - \alpha) \sum_{i=1}^{n} y_i + \alpha \prod_{i=1}^{n} y_i^{w_i}, \tag{3.10}$$

where $\sum_{i=1}^{n} w_i = 1$, $w_i > 0$ and $\alpha \in [0, 1]$. For $\alpha = 0$, the trading function becomes the negative weighted sum and for $\alpha = 1$ the geometric weighted mean.

3.1.2 Concentrated Liquidity

Another implementation of AMMs are CFMMs with concentrated liquidity like UniswapV3 [10]. The idea is to split liquidity provision from the whole sublevel set $\varphi(y + \gamma x - \tilde{x}) =$

$\varphi(y)$ to multiple smaller sets. The sets are determined by ticks $\{p(i)\}_{i\in\mathbb{Z}}$ with the rate usually set to $p(i) = 1.0001^i$, where $p(i)$ is the so-called marginal exchange rate of the tick i. Every tick range

$$(p(i), p(i+1)], \tag{3.11}$$

has its own reserves (liquidity) and all of them are governed by a single trading function φ depending on the liquidity bound by the tick range. The trade is executed in a specific tick range when the marginal exchange rate is in the this specific tick range. Then the liquidity of this specific tick range is used for the trade.

The liquidity providers specify an upper and a lower bound $(p(U), p(L)]$ and provide liquidity on a depth $\tilde{k} = \varphi(y)$. They earn the fee of the trades, which is split by the weighted amount of liquidity provided in the tick range by each liquidity provider instead of on the whole trading curve.

When the marginal exchange rate raises above a trick range during the trade, the tick range is flipped to the next one and the rest of the trade is executed on the new tick range. This however is associated with more transaction cost for the agent conducting the trade.

3.2 First order approximations

For fix reserves $y \in \mathbb{R}^n_+$ let $\nabla\varphi(y) = P$ be the vector of unscaled prices. We proceed as in [8]. Note that P is positive in all entries because φ is by assumption increasing. The entries are given by the gradient

$$P_i = \frac{\partial\varphi(y)}{\partial y_i}. \tag{3.12}$$

The reported price also called the marginal exchange rate is defined as ration between the unscaled prices and the last unscaled price

$$p_i = \frac{P_i}{P_n}, \tag{3.13}$$

where $p_n = 1$ always.

Recall the trade acceptance condition (3.18), first order Taylor expansion of $(y+\gamma x-\tilde{x})$ around y gives us

$$\varphi(y + \gamma x - \tilde{x}) - \varphi(y) \simeq \nabla\varphi(y)^T(\gamma x - \tilde{x}) \tag{3.14}$$

$$= P^T(\gamma x - \tilde{x}), \tag{3.15}$$

rearranging gives

$$\gamma \sum_{i\in\text{supp}(x)} P_i x_i \simeq \sum_{i\in\text{supp}(\tilde{x})} P_i \tilde{x}_i, \tag{3.16}$$

Thus, dividing by P_n

$$\gamma \sum_{i \in \text{supp}(x)} p_i x_i \simeq \sum_{i \in \text{supp}(\tilde{x})} p_i \tilde{x}_i, \tag{3.17}$$

also holds. Note that this is a first order approximation [8], similarly to a midpoint price in an order book. The actual trades are completely determined by the trade acceptance condition. It is an reassuring fact, that the trade can approximated by the marginal exchange rates which is the usual intuition. We will later see, that by concavity of the trade function this will hold in an inequality.

3.3 Swaps

The most common type of trade interaction with a DEX is a swap [8], i.e. trading one asset for another. Denote these two assets with index i and index j, $x = \delta e_i$, $\tilde{x} = \tilde{\delta} e_j$, where $\delta, \tilde{\delta} \in \mathbb{R}_+$ and e_i, e_j are the canonical basis vectors of $\mathbb{R}^n$, $i \neq j$ (disjoint support). The trade acceptance condition in this case is

$$\varphi(y + \gamma \delta e_i - \tilde{\delta} e_j) = \varphi(y). \tag{3.18}$$

It follows from the assumptions on φ, that the left-hand side of the equation $\varphi(y+\gamma\delta e_i - \tilde{\delta} e_j)$ is increasing in δ and decreasing in $\tilde{\delta}$, [8]. Hence, for all $\delta \geq 0$ there is at most one $\tilde{\delta}$ which satisfies the trade acceptance condition. Respectively for all $\tilde{\delta} \geq 0$ there is at most one δ satisfying the trade acceptance condition. Thereby δ and $\tilde{\delta}$ have a one-to-one function correspondence, and two asset trades are uniquely determined by either δ or $\tilde{\delta}$. By concavity we get that

$$\begin{aligned} 0 &= \varphi(y + \gamma \delta e_i - \tilde{\delta} e_j) - \varphi(y) && (3.19) \\ &\leq \nabla_y \varphi(y)(\gamma \delta e_i - \tilde{\delta} e_j) && (3.20) \\ &\leq p_i \gamma \delta - p_j \tilde{\delta}. && (3.21) \end{aligned}$$

Thus

$$p_j \tilde{\delta} \leq p_i \gamma \delta. \tag{3.22}$$

It is natural to define a function $F : \mathbb{R}_+ \to \mathbb{R}$, the forward exchange function [8]. Where $F(\delta)$ is unique to one $\tilde{\delta}$. The function F is defined for all valid trades. Additionally, it increasing since φ is increasing. Also, F is non-negative, because $F(0) = 0$ and it is concave. To show concavity we use the implicit function theorem on the trade acceptance condition, as in [8]

$$f(\delta, F(\delta)) = \varphi(y + \gamma \delta e_i - F(\delta) e_j) - \varphi(y) = 0, \tag{3.23}$$

denote $y' = y + \gamma\delta e_i + F(\delta)e_j$. The chain rule gives us

$$F'(\delta) = -\frac{\partial_\delta f(\delta, F(\delta))}{\partial_{\tilde{\delta}} f(\delta, F(\delta))} \tag{3.24}$$

$$= \gamma \frac{\partial_{y_i}\varphi(y')}{\partial_{y_j}\varphi(y')} \tag{3.25}$$

Together with the properties of φ, it can be shown that F is concave [8], by showing that

$$F(\delta') \leq F'(\delta)(\delta' - \delta) + F(\delta), \tag{3.26}$$

holds for all valid $\delta, \delta' \geq 0$. Let $y'' = y + \gamma\delta' e_i - F(\delta')e_j$ and note that $\varphi(y) = \varphi(y') = \varphi(y'')$ as part of the trade acceptance condition. Since φ is a concave function

$$\varphi(y'') \leq \nabla\varphi(y')^T(y'' - y') + \varphi(y'), \tag{3.27}$$

or simply

$$\nabla\varphi(y')^T(y'' - y') \geq 0. \tag{3.28}$$

We can insert for y'' and y' to get

$$0 \leq (\delta' - \delta)\partial_{y_i}\varphi(y') - (F(\delta') - F(\delta))\partial_{y_j}\varphi(y') \tag{3.29}$$

$$0 \leq (\delta' - \delta)\frac{\partial_{y_i}\varphi(y')}{\partial_{y_j}\varphi(y')} - (F(\delta') - F(\delta)) \tag{3.30}$$

$$F(\delta') \leq F'(\delta)(\delta' - \delta) + F(\delta'). \tag{3.31}$$

Respectively we define the reverse exchange function $G : \mathbb{R}_+ \to \mathbb{R}$, $G(\tilde{\delta})$ uniquely determined by δ satisfying the trade acceptance condition [8]. If no such δ exists then the function is undefined. The reverse exchange function is non-negative and increasing because of φ, but unlike the forward exchange function it is convex [8]. To show convexity, we can again use the implicit function theorem and apply a similar argumentation to the forward exchange function, i.e.

$$G'(\tilde{\delta}) = -\frac{\partial_{\tilde{\delta}} f(G(\tilde{\delta}), \tilde{\delta})}{\partial_\delta f(G(\tilde{\delta}), \tilde{\delta})} \tag{3.32}$$

$$= \frac{1}{\gamma}\frac{\partial_{y_j}\varphi(y')}{\partial_{y_i}\varphi(y')}. \tag{3.33}$$

We need to show that

$$G(\tilde{\delta}') \geq G'(\tilde{\delta})(\tilde{\delta}' - \tilde{\delta}) + G(\tilde{\delta}), \tag{3.34}$$

for all valid $\tilde{\delta}, \tilde{\delta}' \geq 0$. We set $y'' = y + \gamma G(\tilde{\delta}')e_i - \tilde{\delta}'e_j$ and use the concavity of the trading function φ together with the trade acceptance condition to get

$$\nabla\varphi(y')^T(y'' - y') \geq 0 \tag{3.35}$$

$$(G(\tilde{\delta}') - G(\tilde{\delta}))\partial_{y_i}\varphi(y') - (\tilde{\delta}' - \tilde{\delta})\partial_{y_j}\varphi(y') \geq 0 \tag{3.36}$$

$$(G(\tilde{\delta}') - G(\tilde{\delta})) - (\tilde{\delta}' - \tilde{\delta})\frac{\partial_{y_j}\varphi(y')}{\partial_{y_i}\varphi(y')} \geq 0 \tag{3.37}$$

$$G(\tilde{\delta}') \geq -(\tilde{\delta}' - \tilde{\delta})G'(\tilde{\delta}') + G(\tilde{\delta}) \tag{3.38}$$

For two asset trades on the geometric weighted mean trading function in equation (3.5), an analytic form of F and G has been derived in equations (3.7), (3.8),

$$F(\delta) = y_j\left(1 - \frac{y_i^{w_i/wj}}{(y_i + \gamma\delta)^{w_i/w_j}}\right) \tag{3.39}$$

$$G(\tilde{\delta}) = \frac{y_i}{\gamma}\left(\frac{y_j^{w_j/w_i}}{\left(y_j - \tilde{\delta}\right)^{w_j/w_i}} - 1\right), \tag{3.40}$$

for $\tilde{\delta} < y_j$ or else the trade is invalid. For the weighted sum in equation (3.3) the forward and reverse exchange functions are

$$F(\delta) = \gamma\frac{p_i}{p_j}\delta \tag{3.41}$$

$$G(\tilde{\delta}) = \frac{1}{\gamma}\frac{p_j}{p_i}\tilde{\delta}, \tag{3.42}$$

In the case there is no analytical expression for the forward and reverse exchange function we can compute the mapping numerically. Since the trade is uniquely defined by one variable either δ or $\tilde{\delta}$, we can fix one to compute a solution to the trade acceptance condition numerically (3.18), [8]. Recall the Newton method, for a continuously differentiable function f with an initial guess x_0 in an adequate neighborhood of the root, with the iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{3.43}$$

converges to an a. We can frame the trade acceptance condition as a function of $\tilde{\delta}$ or δ the other respectively to arrive at the iterations

$$\delta_{k+1} = \delta_k - \frac{\varphi(y + \gamma\delta_k e_i - \tilde{\delta}e_j) - \varphi(y)}{\partial_{y_i}\varphi(y + \gamma\delta_k e_i - \tilde{\delta}e_j)}, \tag{3.44}$$

$$\tilde{\delta}_{k+1} = \tilde{\delta}_k + \frac{\varphi(y + \gamma\delta e_i - \tilde{\delta}_k e_j) - \varphi(y)}{\partial_{y_j}\varphi(y + \gamma\delta e_i - \tilde{\delta}_k e_j)}. \tag{3.45}$$

with optimal values at $\delta^*, \tilde{\delta}^*$ repsectively. To solve for $\tilde{\delta}^*$, fix δ and we can choose the iterates from the ε ball defined by the Lipschitz continuity of the derivative. Assuming φ is continuously differentiable, the derivative is also Lipschitz continuous, i.e. there exists a constant $L > 0$ such that

$$|f'(\tilde{\delta}) - f'(\tilde{\delta}^*)| \leq L|\tilde{\delta} - \tilde{\delta}^*|. \tag{3.46}$$

Here f is the function we are solving for $f(\tilde{\delta}) = \varphi(y + \gamma\delta e_i - \tilde{\delta}e_j) - \varphi(y)$. Then we can set

$$\varepsilon := \frac{\partial_{y_j}\varphi(y_j - \tilde{\delta}^*)}{2L}. \tag{3.47}$$

Assuming that $f'(\tilde{\delta}^*) = \partial_{y_j}\varphi(y_j - \tilde{\delta}^*) \neq 0$, we ensure that by concavity and monotonicity that the $f'(\tilde{\delta}) \neq 0$ for all δ in the ball of ε around the solution, and also that

$$\frac{1}{\partial_{y_j}\varphi(y_j - \tilde{\delta}^*)} \leq \frac{1}{\partial_{y_j}\varphi(y_j - \tilde{\delta})} \leq c, \tag{3.48}$$

for a $c \geq 0$ where $|\tilde{\delta} - \tilde{\delta}^*| = \varepsilon$. Thus

$$|f'(\tilde{\delta}) - f'(\tilde{\delta}^*)| \leq \frac{1}{2}|f'(\tilde{\delta}^*)| \tag{3.49}$$

holds for all $\tilde{\delta}$ in the epsilon ball of the solution. Then by the mean value theorem and Lipschitz continuity for all $\tilde{\delta}$ in the ε ball around the solution we have

$$\begin{aligned}
&|f(\tilde{\delta}) - f(\tilde{\delta}^*) - f'(\tilde{\delta})(\tilde{\delta} - \tilde{\delta}^*)| \leq && (3.50)\\
&\leq |f(\tilde{\delta}) - f(\tilde{\delta}^*) - f'(\tilde{\delta})(\tilde{\delta} - \tilde{\delta}^*)| + |f'(\tilde{\delta}) - f'(\tilde{\delta}^*)||\tilde{\delta} - \tilde{\delta}^*| && (3.51)\\
&\leq \int_0^1 |f'(\tilde{\delta}^* + t(\tilde{\delta} - \tilde{\delta}) - f'(\tilde{\delta}^*)||\tilde{\delta} - \tilde{\delta}^*|dt + |f'(\tilde{\delta}) - f'(\tilde{\delta}^*)||\tilde{\delta} - \tilde{\delta}^*| && (3.52)\\
&\leq \int_0^1 L(1-t)|\tilde{\delta} - \tilde{\delta}^*|^2 dt + |f'(\tilde{\delta}) - f'(\tilde{\delta}^*)||\tilde{\delta} - \tilde{\delta}^*| && (3.53)\\
&\leq \frac{1}{4c}|\tilde{\delta} - \tilde{\delta}^*| + \frac{1}{2c}|\tilde{\delta} - \tilde{\delta}^*| = \frac{3}{4c}|\tilde{\delta} - \tilde{\delta}^*|. && (3.54)
\end{aligned}$$

For a $\tilde{\delta}_0$ in the epsilon ball the first iterate is also in the ball

$$\begin{aligned}
|\tilde{\delta}_1 - \tilde{\delta}^*| &\leq \left|\tilde{\delta}_0 - \frac{f(\tilde{\delta})}{f'(\tilde{\delta})} - \tilde{\delta}^*\right| && (3.55)\\
&\leq |f'(\tilde{\delta}_0)|^{-1}|f(\tilde{\delta}_0) - f(\tilde{\delta}^*) - f'(\tilde{\delta}_0)(\tilde{\delta}_0 - \tilde{\delta}^*)| && (3.56)\\
&\leq \frac{3}{4}|\tilde{\delta}_0 - \tilde{\delta}^*| && (3.57)
\end{aligned}$$

Repeating this argument $k \geq 0$ times we get

$$|\tilde{\delta}_{k+1} - \tilde{\delta}^*| \leq \left(\frac{3}{4}\right)^k |\tilde{\delta}_0 - \tilde{\delta}^*| \tag{3.58}$$

Thus the iterates of the Newton method $(\tilde{\delta}_k)_{k\in\mathbb{N}}$ lie in the ε ball of the solution, the iteration is well defined since the derivatives are non zero and the iterates converge to $\tilde{\delta}^*$ as $k \to \infty$. To solve for δ^*, fix $\tilde{\delta}$, the proof stays the same.

3.4 General & Optimal Trades on one CFMM

For general trades [8], the agent chooses a trade $(x, \tilde{x})$ through the solution of an optimization problem. The optimization problem maximizes the change in the traders' portfolio $x - \tilde{x}$ through a utility function. The agent prefers a change in holdings that maximize a utility function $U : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty\}$. For two trades $(x, \tilde{x})$ and $(x', \tilde{x}')$ the agent prefers the trade $(x, \tilde{x})$ if

$$U(\tilde{x} - x) > U(\tilde{x}' - x'). \tag{3.59}$$

We map U to $\{-\infty\}$ when the trade does not pass the trade acceptance condition or the change in the portfolio is absolutely not preferred. We choose the utility function to be increasing, concave, and thereby differentiable [8]. By the trade acceptance condition, the trade that maximizes the utility can be framed as a constrained optimization problem

$$\begin{aligned} \max \quad & U(\tilde{x} - x) \\ \text{s.t.:} \quad & \varphi(y + \gamma x - \tilde{x}) = \varphi(y) \\ & x \geq 0 \qquad \tilde{x} \geq 0 \end{aligned} \tag{3.60}$$

We would prefer the problem in equation (3.60) to be convex [8]. Then it can be solved efficiently fast using convex methods. The feasible set of the optimization problem is

$$D := \left\{x \in \mathbb{R}^n_+,\ \tilde{x} \in \mathbb{R}^n_+ \mid \varphi(y + \gamma x - \tilde{x}) - \varphi(y) = 0\right\} \tag{3.61}$$

For $(x_1, \tilde{x}_1), (x_2, \tilde{x}_2) \in D$ and $\lambda \in [0, 1]$ reveals for $(\lambda x_1 + (1-\lambda)x_2, \lambda\tilde{x}_1 + (1-\lambda)\tilde{x}_2)$ that

$$\varphi\Big(y + \gamma(\lambda x_1 + (1-\lambda)x_2) - (\lambda\tilde{x}_1 + (1-\lambda)x_2\Big) - \varphi(y) = \tag{3.62}$$

$$= \varphi\Big(y + \lambda(\gamma x_1 - \tilde{x}_1) + (1-\lambda)(\gamma x_2 - \tilde{x}_2)\Big) - \varphi(y), \tag{3.63}$$

cannot be reduced further, thus the convex combination generally does belong to D . But, if we consider the relaxation from equality in D to inequality $\geq$, i.e.

$$D^r := \left\{x \in \mathbb{R}^n_+,\ \tilde{x} \in \mathbb{R}^n_+ \mid \varphi(y + \gamma x - \tilde{x}) - \varphi(y) \geq 0\right\}, \tag{3.64}$$

together with the concavity of φ and $y = \lambda y + (1-\lambda)y$, we can reduce the equation (3.63) further.

$$\varphi\Big(y + \lambda(\gamma x_1 - \tilde{x}_1) + (1-\lambda)(\gamma x_2 - \tilde{x}_2)\Big) - \varphi(y) \tag{3.65}$$
$$= \varphi\Big(\lambda(y + \gamma x_1 - \tilde{x}_1) + (1-\lambda)(y - \gamma x_2 - \tilde{x}_2)\Big) - \varphi(y) \tag{3.66}$$
$$\geq \lambda\varphi(y + \gamma x_1 - \tilde{x}_1) + (1-\lambda)\varphi(y + \gamma x_2 - \tilde{x}_2) - \varphi(y). \tag{3.67}$$

For $(x_1, \tilde{x}_1), (x_2, \tilde{x}_2) \in D^r$ it holds that

$$\lambda\varphi(y + \gamma x_1 - \tilde{x}_1) + (1-\lambda)\varphi(y + \gamma x_2 - \tilde{x}_2) - \varphi(y) \geq 0 \tag{3.68}$$
$$\lambda\left(\varphi(y + \gamma x_1 - \tilde{x}_1) - \varphi(y)\right) + (1-\lambda)\left(\varphi(y + \gamma x_2 - \tilde{x}_2) - \varphi(y)\right) \geq 0, \tag{3.69}$$

for all $\lambda \in [0, 1]$. Hence, D^r is convex. So the relaxation of problem (3.60), i.e.

$$\begin{aligned} \max \quad & U(\tilde{x} - x) \\ \text{s.t.:} \quad & \varphi(y + \gamma x - \tilde{x}) \geq \varphi(y) \\ & x \geq 0 \qquad \tilde{x} \geq 0 \end{aligned} \tag{3.70}$$

can be solved using convex methods. It is remains to show that any solution of the convex relaxation (3.70) is a feasible solution to the original (3.60). We proceed as in [8], if a solution of the relaxation (3.70) satisfies the equality condition $\varphi(y + \gamma x - \tilde{x}) = \varphi(y)$ then it is also solution to the original (3.60). In the case of strict inequality

$$\varphi(y + \gamma x - \tilde{x}) > \varphi(y). \tag{3.71}$$

Since φ is increasing in x and decreasing in $\tilde{x}$, we can decrease x by $\varepsilon > 0$ and increase $\tilde{x}$ by $\tilde{\varepsilon} > 0$ until

$$\varphi(y + \gamma(x - \varepsilon) - (\tilde{x} + \tilde{\varepsilon})) \geq \varphi(y). \tag{3.72}$$

But then the utility of $(x - \varepsilon, x + \tilde{\varepsilon})$ is bigger than that of $(x, \tilde{x})$. Thus, $(x, \tilde{x})$ is not an optimal solution.

Furthermore the relaxed problem fulfills Slater CQ. By the choice of $\tilde{x} = 0$ we can choose any $x \in \mathbb{R}^n_+$ such that by the monotonicity assumptions on φ we have

$$\varphi(y + \gamma x) - \varphi(y) > \varphi(y + \gamma x - \tilde{x}) - \varphi(y) \geq 0. \tag{3.73}$$

Thereby, given a local minimum of the relaxed optimization problem we can find Lagrange multipliers, forming a KKT-point satisfying the KKT conditions 2.1. And on the other hand if we find a KKT point we will have a local minimum.

3.4.1 Optimality Condition

The KKT-conditions of the minimization of equation (3.70), i.e.

$$\min \quad -U(\tilde{x}-x) \tag{3.74}$$
$$\text{s.t.:} \quad (x,\tilde{x}) \in D^r, \tag{3.75}$$

Since Slater CQ is satisfied so is Abadie CQ, given some local minimum then there is a KKT point $(x,\tilde{x},\lambda,\omega,\mu)$, which is a sufficient condition for optimality in the convex case with Slater CQ. Hence, $(x,\tilde{x})$ is a local minimum of the optimization problem and any $(x,\tilde{x},\lambda,\omega,\mu)$, as in [8], satisfying

$$\begin{cases} \nabla_x L(x,\tilde{x},\lambda,\omega,\mu)=0, \\ \nabla_{\tilde{x}} L(x,\tilde{x},\lambda,\omega,\mu)=0, \\ \lambda \geq 0,\ g(x,\tilde{x}) \leq 0,\ \lambda g(x,\tilde{x})=0, \\ \omega \geq 0,\ \omega x = 0, \\ \mu \geq 0,\ \mu\tilde{x}=0 \end{cases}, \tag{3.76}$$

is a KKT point. Here $L : \mathbb{R}^n \times \mathbb{R}_+ \times \mathbb{R}^n_+ \times \mathbb{R}^n_+ \to \mathbb{R}$ is the Lagrangian, $\lambda \in \mathbb{R}_+, \omega \in \mathbb{R}^n_+, \mu \in \mathbb{R}^n_+$ are the Lagrange multipliers and $g(x,\tilde{x}) := \varphi(y) - \varphi(y+\gamma x - \tilde{x})$ a notational convention. The Lagrangian of the problem is

$$L(x,\tilde{x},\lambda,\omega,\mu) = -U(\tilde{x}-x) + \lambda g(x,\tilde{x}) - \omega x - \mu\tilde{x}, \tag{3.77}$$

the first two conditions give

$$0 = \nabla_x L = \nabla U(\tilde{x}-x) - \lambda\gamma\nabla\varphi(y') - \underbrace{\omega}_{\geq 0}, \tag{3.78}$$
$$0 = \nabla_{\tilde{x}} L = -\nabla U(\tilde{x}-x) + \lambda\nabla\varphi(y') - \underbrace{\mu}_{\geq 0}, \tag{3.79}$$

where $y' = y - \gamma x + \tilde{x}$. The two inequalities give

$$\lambda\gamma\nabla\varphi(y') \leq \nabla U(\tilde{x}-x) \leq \lambda\nabla\varphi(y'). \tag{3.80}$$

Specific interpretation for this condition comes when looking the trade that is not increasing the utility, e.g. for a solution $x = \tilde{x} = 0$. Then the condition, after dividing with λP_n becomes

$$\gamma p \leq \frac{1}{\lambda P_n}\nabla U(0) \leq p. \tag{3.81}$$

The set of prices p which satisfy this condition is denoted by

$$K := \left\{ p \in \mathbb{R}^n_+ \,|\, \exists \alpha \geq 0 :\ \gamma p \leq \alpha U(0) \leq p \right\}, \tag{3.82}$$

in literature referred to as the no-trade cone [8] for the utility U of trade function φ. The set is a cone since multiplying by any $p \in K$ by an $\eta > 0$, by division $\frac{\alpha}{\eta}$ is still bigger or equal to zero. The Set K is also convex, for any $p_1, p_2 \in K$ and $\eta \in [0,1]$, $\eta p_1 + (1-\eta)p_2 \in K$, because the condition is additive and a K a cone. The cone is nonempty since $0 \in K$ with the choice of $\alpha = 0$.

In the case of linear utility $U(\psi) = z^T\psi$, with $z \in \mathbb{R}^n_+$, [8] we have that

$$\gamma p \leq \frac{1}{\lambda P_n} z \leq p, \tag{3.83}$$

hods component wise.

3.5 Multi-DEX Trades

When trading with multiple DEXs, the model is given by [9]. The agent interacts with multiple CFMMs. In light of multi asset trades (3.70) a generalization to an interaction with multiple CFMMS can be made. Consider assets $j \in \{1, \dots, n\}$, labeled as such. Consider CFMMs $i \in \{1, \dots, m\}$, each trading $N_i \subseteq \{1, \dots, n\}$ assets which are specifically ordered based on the construction of the CFMM i and trading function $\varphi_i : \mathbb{R}_+^{|N_i|} \to \mathbb{R}$ with liquidity y_i and trading fee γ_i. To link the ordering of N_i and asset labels i, consider the binary matrix $A_i \in \mathbb{R}^{n \times |N_i|}$ with

$$(A_i)_{kj} = \begin{cases} 1 & \text{for } k \in N_i,\ k = j \\ 0 & \text{else,} \end{cases} \tag{3.84}$$

for all $i \in \{1, \dots, m\}$. The agent interacts with each CFMM i through a trade $(x_i, \tilde{x}_i)$, the mapping to the asset indices j is done through the matrices A_i. Hence, the global trade vector interacting with all CFMMs is $\psi \in \mathbb{R}^n$ is defined as

$$\psi = \sum_{i=1}^{m} A_i(\tilde{x}_i - x_i). \tag{3.85}$$

The implication of $\psi_k \geq 0$, means that the agent does not give any amount of asset k at the end of the trade to the set of CFMMs. The agent still might trade asset k, but receives a positive amount. Analogous to multi-asset trades, the agent judges a trade with a utility function

$$U : \mathbb{R}^n \to \mathbb{R} \tag{3.86}$$

$$\psi \mapsto U(\psi), \tag{3.87}$$

where U is concave, differentiable and increasing. In the case of $U(\psi) = -\infty$ the trade is not acceptable, and the agent does not trade [9]. E.g. if the agent wants to increase the holdings $h \in \mathbb{R}^n_+$ after the trade, then $U(\psi) = \psi + h \geq 0$ is a constraint satisfying the condition of increasing the holdings. For any trade which does not satisfy the constraint,

the utility is $-\infty$, hence not acceptable.

A set of optimal trades with multiple CFMMs, maximizing the utility [9] is

$$\max \quad U\left(\sum_{i=1}^{m} A_i(\tilde{x}_i - x_i)\right) \tag{3.88}$$

$$\text{s.t.:} \quad (x_i, \tilde{x}_i) \in D_i \ \ \forall i \in \{1, \ldots, m\}, \tag{3.89}$$

where

$$D_i = \left\{(x, \tilde{x}) |\ x, \tilde{x} \in \mathbb{R}_+^{|N_i|}, \ \ \varphi_i(y_i + \gamma_i x - \tilde{x}) = \varphi_i(y_i)\right\} \tag{3.90}$$

It is already shown that the convex relaxation of the optimization problem satisfies the solution to the original, the proof stays the same component-wise for all CFMMs $i \in \{1, \ldots, m\}$. The convex relaxation of the problem is

$$\max \quad U\left(\sum_{i=1}^{m} A_i(\tilde{x}_i - x_i)\right) \tag{3.91}$$

$$\text{s.t.:} \quad (x_i, \tilde{x}_i) \in D_i^r \ \ \forall i \in \{1, \ldots, m\}, \tag{3.92}$$

where

$$D_i^r = \left\{(x, \tilde{x}) |\ x, \tilde{x} \in \mathbb{R}_+^{|N_i|}, \ \ \varphi_i(y_i + \gamma_i x - \tilde{x}) \geq \varphi_i(y_i)\right\} \tag{3.93}$$

Similar to the argumentation in section 3.4.1 the Slater CQ is satisfied for all CFMM φ_i $i \in \{1, \ldots, m\}$ by setting $\tilde{x}_i = 0$. Since the problem is convex and Slater CQ is satisfied the KKT conditions are necessary and sufficient for optimality. The Lagrangian of the problem is

$$L(x, \tilde{x}, \nu, \lambda, \omega, \mu) = -U(x, \tilde{x}) + \nu^T \left(\psi - A(x - \tilde{x})\right) + \lambda^T g(x, \tilde{x}) - \omega^T x - \mu^T \tilde{x}. \tag{3.94}$$

The KKT conditions [9] give

$$0 = \nabla_{(x,\tilde{x})} L(x, \tilde{x}, \nu, \lambda, \omega, \mu) = -\nabla U(x, \tilde{x}) + \nu = 0 \tag{3.95}$$

$$\Rightarrow \nabla U(x, \tilde{x}) = \nu. \tag{3.96}$$

Also for all $i \in \{1, \ldots, m\}$ we have

$$\nabla_{x_i} L(x, \tilde{x}, \nu, \lambda, \omega, \mu) = A_i^T \nabla U(x, \tilde{x}) - \lambda_i \gamma \nabla \varphi_i(y_i + \gamma_i x_i - \tilde{x}_i) - \underbrace{\omega}_{\geq 0} \tag{3.97}$$

$$\nabla_{\tilde{x}_i} L(x, \tilde{x}, \nu, \lambda, \omega, \mu) = -A_i^T \nabla U(x, \tilde{x}) + \lambda_i \nabla \varphi_i(y_i + \gamma_i x_i - \tilde{x}_i) - \underbrace{\omega}_{\geq 0}. \tag{3.98}$$

Thus,

$$A_i^T \nu \geq \lambda_i \gamma \nabla \varphi_i(y_i + \gamma_i x_i - \tilde{x}_i) \tag{3.99}$$

$$A_i^T \nu \leq \lambda_i \nabla \varphi_i(y_i + \gamma_i x_i - \tilde{x}_i). \tag{3.100}$$

Combining gives

$$\lambda_i \gamma \nabla \varphi_i(y_i + \gamma_i x_i - \tilde{x}_i) \leq A_i^T \nu \leq \lambda_i \nabla \varphi_i(y_i + \gamma_i x_i - \tilde{x}_i). \tag{3.101}$$

When the optimal trade is zero, the agent does not trade. This is the case when $\psi = 0$ and $x_i = \tilde{x}_i = 0$ for all $i \in \{1, \ldots, m\}$ [9]. Thus, the no-trade cones are defined through the following inequality component-wise

$$\gamma_i \lambda_i p_i \leq A_i^T \nabla U(0,0) \leq \lambda p_i \qquad \forall i \in \{1, \ldots, m\}, \tag{3.102}$$

where $p_i = \nabla \varphi_i(y_i)$ are the marginal exchange rates of CFMM i at liquidity y_i, [9].

3.6 Cyclic Arbitrage

Let us go back to the general trading problem without transaction fees in equation (3.91). It is important to note, that the solution does specify in which order the trades should be executed and that the computational effort will grow larger with more CFMMs. Hence, in a large space of CFMMs it is a separate problem of finding cyclic arbitrage walks. This walk will give the order of the trades and specify for which CFMM the optimization problem needs to be framed, dropping the rest.

We consider now that the agent has found a cyclic arbitrage [9] path of $m \in \mathbb{N}$ CFMMs. For an arbitrage trade, the agent needs the trade to be valid, i.e. $\sum_{i=1}^m A_i(\tilde{x}_i - x_i) \neq 0$. The agent also needs $\sum_{i=1}^m A_i(\tilde{x}_i - x_i) \geq 0$, because at the end of a set of trades, the arbitrage trade receives a positive amount of at least one asset. This hold only for $\sum_{i=1}^m A_i(\tilde{x}_i - x_i)$, i.e. at the end of the trade. The utility function needs to be bound in the domain $\mathbb{R}_+^n$. There is an arbitrage, if there is a nonzero solution. An optimal arbitrage is one that maximizes the utility under the constraints i.e. $U\left(\sum_{i=1}^m A_i(\tilde{x}_i^* - x_i^*)\right) > U(0)$ with $x^*, \tilde{x}^*$ the optimal solution. As discussed we have the follwing optimization problem for the general multiasset optimal trade [9]

$$\max \quad U\left(\sum_{i=1}^m A_i(\tilde{x}_i - x_i)\right) \tag{3.103}$$

$$\text{s.t.:} \quad (x_i, \tilde{x}_i) \in D_i^r \ \ \forall i \in \{1, \ldots, m\}, \tag{3.104}$$

Again the solution where $\sum_{i=1}^m A_i(\tilde{x}_i - x_i) = 0$ is valid, but in this case there is no arbitrage and the agent does not trade.

3.6.1 Arbitrage on two Asset Constant Product Market Makers

A common case is an n cyclic arbitrage under linear utility

$$U(x, \tilde{x}) = 1^T \left(\sum_{i=1}^m A_i(\tilde{x}_i - x_i)\right), \tag{3.105}$$

with n CFMMs $\varphi_i(y_i) = \sqrt{y_{i,1}y_{i,2}}$, with n distinct assets via

$$1 \xrightarrow{\varphi_1} 2 \xrightarrow{\varphi_2} 3 \xrightarrow{\varphi_3} \dots \xrightarrow{\varphi_{n-1}} n \xrightarrow{\varphi_n} 1. \tag{3.106}$$

W.l.o.g. for each trade let $(x_i, \tilde{x}_i) = (\delta_1 e_1, \tilde{\delta} e_2)$, where e_1, e_2 are the basis vectors in $\mathbb{R}^2$. Then the matrices $A_i \in \mathbb{R}^{n\times 2}$ take the shape

$$A_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix},\ A_2 = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 0 \end{pmatrix},\ \dots\ A_{n-1} = \begin{pmatrix} 0 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \end{pmatrix},\ A_n = \begin{pmatrix} 0 & 1 \\ \vdots & \vdots \\ 1 & 0 \end{pmatrix}. \tag{3.107}$$

Also, the general trade vector is

$$\sum_{i=1}^{m} A_i(\tilde{x}_i - x_i) = \begin{pmatrix} \tilde{\delta}_n - \delta_1 \\ \tilde{\delta}_1 - \delta_2 \\ \vdots \\ \tilde{\delta}_{n-1} - \delta_n \end{pmatrix}. \tag{3.108}$$

The convex constraints component wise can be written as

$$y_{i,1}y_{i,2} - (y_{i,1} + \gamma_i\delta_i)(y_{i,2} - \tilde{\delta}_i) \leq 0. \tag{3.109}$$

This can be rewritten into a quadratic form by simplifying

$$\begin{aligned} & y_{i,1}y_{i,2} - (y_{i,1} + \gamma_i\delta_i)(y_{i,2} - \tilde{\delta}_i) && (3.110)\\ & = y_{i,1}\tilde{\delta}_i - \gamma_i y_{i,2}\delta_i + \gamma_i\delta_i\tilde{\delta}_i && (3.111)\\ & = \begin{pmatrix} -\gamma_i y_{i,2} & y_{i,1} \end{pmatrix} \begin{pmatrix} \delta_i \\ \tilde{\delta}_i \end{pmatrix} + \gamma_i \begin{pmatrix} \delta_i & \tilde{\delta}_i \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta_i \\ \tilde{\delta}_i \end{pmatrix}. && (3.112) \end{aligned}$$

Set $\xi_i = \begin{pmatrix} \delta_i & \tilde{\delta}_i \end{pmatrix}^T \in \mathbb{R}^2$, $c_i = \begin{pmatrix} -\gamma_i y_{i,2} & y_{i,1} \end{pmatrix}^T \in \mathbb{R}^2$ and $B = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$. Then the constraint can be written into quadratic form

$$c\xi_i + \left(\sum_{i=j}^{m} A_j(\tilde{x}_j - x_j) \right)_i^T A\xi_i \leq 0. \tag{3.113}$$

Together with $R = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ the optimization problem becomes a quadratically constrained linear problem.

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} 1^T A_i R \xi_i && (3.114)\\ \text{s.t.:} \quad & \sum_{i=1}^{n} A_i R \xi_i \geq 0 && (3.115)\\ & \xi_i \geq 0,\ \ c_i^T \xi_i + \gamma_i \xi_i^T B \xi_i \leq 0\ \ \forall i \in \{1, \dots, n\}. && (3.116) \end{aligned}$$

Additionally, it is possible use matrix factorization to factor $\sum_{i=1}^{n} A_i R \xi_i$ into $A\xi$ for $A \in \mathbb{R}^{n \times 2n}$ with entries $-1, 0, 1$, together with $\xi = \begin{pmatrix} \delta_1 & \tilde{\delta}_1 & \dots & \delta_n & \tilde{\delta}_n \end{pmatrix}^T \in \mathbb{R}^{2n}$. We can also pull the inequality constraint into a single constraints under the same ξ with a $c \in \mathbb{R}^{2n}$ and $D \in \mathbb{R}^{2n \times 2n}$ consisting of n blocks of $\gamma_i B$, to get

$$\begin{aligned} \max \quad & 1^T A\xi && (3.117) \\ \text{s.t.:} \quad & c \odot \xi + \xi \odot D\xi \leq 0 && (3.118) \\ & \xi \geq 0, && (3.119) \end{aligned}$$

where $\odot$ denotes the componentwise multiplication.

4 Reinforcement Learning and Graph Neural Networks (GNNs)

In this chapter we introduce the framework on which finding the approximate solution of the arbitrage walk is going to be based. We are going to give a brief introduction to Monte Carlo Tree Search (MCTS), Reinforcement Learning and Graph Neural Networks. We will apply these concepts to a model of a network of CFMM and find an optimal walk. This chapter is largely based on [15, Chapter17] for the foundations of Reinforcement Learning and [5] for the REINFORCE algorithm. The combination of REINFORCE with MCTS was inspired by [6].

4.1 Markov Decision Process

Generally in comparison to supervised learning, reinforcement learning is not trained directly with labeled data, instead it learns through an agent which takes actions and is rewarded based on the outcome of these actions. The idea is to, in expectation, increase the rewards of the agent throughout the learning process by 'trial and error'. Hence, reinforcement learning is better suited for problems with large solution spaces and decision making settings, where we have dynamical environments. The combinatorial complexity of these problems is too big, such that supervised learning would lack labeled data. For instance the Shannon number is 10^{120}, which is the lower bound of all possible positions in chess. In this chapter we will go through the foundation of reinforcement learning based on [15, Chapter 17]. The learning scenario is based on the agent collecting information by interacting with the environment. In response to an action the agent will enter a new state and will rewarded depending wrt. some reward function 4.1. For example the agent is rewarded a score value of $+1$ if set of actions leads to a win in a decision based game. At each state the agent's objective is to maximize the expected reward. This translates to taking, in expectation, the best action at a given state, called the policy.

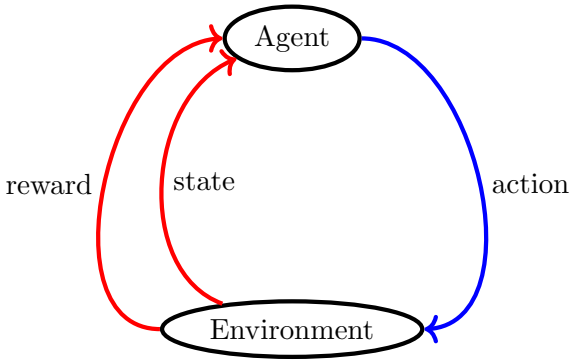


Figure 4.1: Reinforcement learning scenario

At each step the agent faces a decision to choose an action leading to a new state and collect reward associated with it. However, at the same time the agent is faced with a dilemma of whether to maybe choose an action to an unexplored state which would potentially lead to a higher reward or stick with an already known-to-work action. In reinforcement learning this is called 'exploration versus exploitation' trade off and is manifested and controlled in the scope of the learning algorithm.

Another key difference to traditional supervised learning, reinforcement learning does not sample features based on a fixed distribution. Essentially the training and testing phases are not separated, but rather combined in each step. The two main learning settings are distinguished, the setting where the full environmental model is known and where the environmental model is unknown. The mathematical model which describes the agent's iteration with the environment is called the Markov Decision Process (MPD).

Definition 4.1 *Markov decision process [15, Chapter 17]*

The MDP is defined by 4 tuple (S, A, P_a, R_a):

- The set of all states S, with a start state $s_0 \in S$,
- The set of all actions A,
- The transition probabilities P_a, $\mathbb{P}[s'|s,a] = \int_{S'} P_a(s,s')ds'$
- The rewards R_a, with probabilities $\mathbb{P}[r'|s,a]$

The transition probability $\mathbb{P}[s'|s,a]$ is the probability of landing in state s', when being initially at state s and taking an action a. Similarity, the reward probability is explained in the same way. The structure of state transition only depends on the last state and the action taken and not on the whole history of states and actions, hence the Markov property and the naming of the model. Usually, and in the scope of the thesis, the state transitions and rewards are deterministic. In this case the transition and reward probability is 1 and the MDP is called deterministic with the reward being $r(s,a)$.

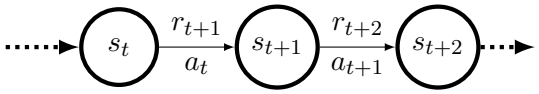


Figure 4.2: Deterministic discrete MDP [15]

As per the discrete (episodic) model, the agent preforms actions throughout a set of decision epochs $t \in 0, \ldots, T$, shown in Figure 4.2. This most commonly adopted model for a wide range of problems. The MDP is called finite when both S and A are finite sets. Furthermore, the MDP has a finite time horizon when T is finite.
Generally we would like to find a to find a good policy for the decisions, i.e. probabilities at each state telling us which action to take. The policy is a function of states, returning a probability distribution over the possible actions from the given state. We naturally choose the action with the highest probability.

Definition 4.2 *Policy [15, Chapter 17]*

A policy is a mapping $\pi : S \to \Delta(A)$, onto $\Delta(A)$ the set of probability distributions over A.

The policy is deterministic if:

$$\forall s \in S \;\; \exists! a \in A : \quad \pi(a|s) = 1, \tag{4.1}$$

where $\pi(a|s)$ is the probability of landing in action a given state s, the notation '|' serves as a notation for a probability distribution over $a \in A(s)$ for all $s \in S$. In deterministic case we have that $\mathrm{im}(\pi) = A$, and we can say $\pi : S \to A$ with $\pi(s) = a$.

Together with the definition of a policy, the aim of reinforcement learning can be expressed as trying to find a policy that maximizes the total in return in expectation. For a discrete policy the total return is

$$\sum_{t=0}^{T} \gamma^t r(s_t, \pi(s_t)) \tag{4.2}$$

where $r(s_t, \pi(s_t))$ is the reward function for the next state and $\gamma \in [0, 1]$ a discount factor for future rewards. We say that an MDP has a finite horizon if $T \to \infty$. To quantify what the optimal policy is, we define the value of the policy at a given state.

Definition 4.3 *Policy value [15, Chapter 17]*

The value $V_\pi(s)$ of the policy π at state $s \in S$ is defined as the expected value of the total reward over the policy distribution $\pi(s_t)$ of state $s_t \in S$ at epoch $t \leq T$

$$V_\pi(s) = \mathbb{E}_{a_t \sim \pi(s_t)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \middle| s_0 = s \right]. \tag{4.3}$$

Thereby the optimal policy can be defined in a standard way

Definition 4.4 *Optimal policy [15, Chapter 17]*

The policy π^* is called optimal if for any policy π it holds that

$$V_{\pi^*}(s) \geq V_\pi(s) \quad \forall s \in S. \tag{4.4}$$

Similarly to the policy value we can quantify the pair (s, a) associated to a policy π by a value, the so called state-action value

Definition 4.5 *State-Action value [15, Chapter 17]*

The state-action value function Q_π associated with a policy π is defined on all state-action pairs $(s, a) \in S \times A$

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}[r(s,a)]\mathbb{E}_{a_t \sim \pi(s_t)} \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) \middle| \, s_0 = s, \; a_0 = a \right] \\ &= \mathbb{E}\left[r(s,a) + \gamma V_\pi(s_1) \middle| \, s_0 = s, \; a_0 = a \right]. \end{aligned} \tag{4.5}$$

Note that the expected value of the state-value function at (s, a) over the policy distribution at state s is the policy value at s, i.e.

$$\mathbb{E}_{a \sim \pi(s)} \left[Q_\pi(s, a) \right] = V_\pi(s). \tag{4.6}$$

Now that the framework for is set, it can be shown that there exists an optimal policy. That is the so called policy improvement theorem.

Theorem 4.1 *Policy Improvement Theorem [15, Chapter 17]*

For any two policies π' and π, it holds that

$$\begin{aligned} &\mathbb{E}_{a \sim \pi'(s)} \left[Q_\pi(s, a) \right] \geq \mathbb{E}_{a \sim \pi(s)} \left[Q_\pi(s, a) \right] \quad \forall s \in S \\ &\implies V_{\pi'}(s) \geq V_\pi(s) \quad \forall s \in S. \end{aligned} \tag{4.7}$$

Leading us to the Bellman's conditions.

Theorem 4.2 *Bellman's optimality condition [15, Chapter 17]*

A policy π is optimal if and only if for all state-action pairs $(s, a) \in SxA$ with $\pi(s)(a) > 0$ it holds that

$$a \in \arg\max_{a' \in A} Q_\pi(s, a'). \tag{4.8}$$

Theorem 4.3 *Existence of optimal deterministic policy [15, Chapter 17]*

Any finite MDP has an optimal deterministic policy.

Now we consider an optimal deterministic policy π^* with its policy value and state-action value functions V^* and Q^*. The optimal policy for all states is

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s, a). \tag{4.9}$$

We only needs to know the state-action value function Q^* to determine the optimal policy, and do not need to have information about the transition and reward probabilities. Using the definition of the state-value function for Q^*, $V^*(s) = Q^*(s, \pi^*(s))$, gives us system of equations called *Bellman equations*:

$$V^*(s) = \max_{a \in A} \left\{ \mathbb{E}[r(s,a)] + \gamma \sum_{s' \in S} \mathbb{P}[s'|s,a] V^*(s') \right\} \quad \forall s \in S. \tag{4.10}$$

Theorem 4.4 *Bellman Equations [15, Chapter 17]*

The policy value $V_\pi(s)$ of a policy π at state s for an MDP with infinite horizon is subjected to the following linear system of equations

$$V^*(s) = \mathbb{E}_{a_1 \sim \pi(s)}[r(s,a_1)] + \gamma \sum_{s' \in S} \mathbb{P}[s'|s,\pi(s)] V^*(s') \quad \forall s \in S. \tag{4.11}$$

For a finite horizon , we can rewrite to the linear system [15, Chapter 17] with

$$v = \gamma P v + r, \tag{4.12}$$

where

$$p_{s,s'} = \mathbb{P}[s'|s,\pi(s)] \quad \forall s, s' \in S \quad \text{is the transition probability matrix,} \tag{4.13}$$
$$v_s = V_\pi(s) \quad \forall s \in S \quad \text{is the value vector and} \tag{4.14}$$
$$r_s = \mathbb{E}[r(s,\pi(s))] \quad \forall s \in S \quad \text{is the reward vector.} \tag{4.15}$$

For an MDP with finite horizon with a known probability matrix P and a known reward expectation r, we have a unique solution for the value vector by matrix inversion [15, Chapter 17]

$$v^* = (I - \gamma P)^{-1} R, \tag{4.16}$$

where I is the appropriate identity matrix.

Proofs for the theorems can be found in [15, Chapter 17].

4.2 Monte Carlo Tree Search (MCTS)

MCTS is a search algorithm for decision processes like MDPs. The algorithm focuses on finding the most favorable course of action by uniformly expanding the search tree at each step, and applying random actions/decisions until reaching a terminal state. The outcome is then communicated back to the start node via backpropagation. In the last decade MCTS has been combined with learning based methods like AlphaZero [6] to produce agents which can beat humans in Chess, GO and other games. There are many

variation of MCTS and in this chapter we introduce the basic one.

The core of the algorithm is based on approximating the true value of an action through random simulation, which ultimately adjusts the policy efficiently to derive a best-first strategy [16]. One search iteration is divided into four steps:

1. Selection Phase: Also called the Tree-Traversal Phase. Here we start at the root node state and walk down until reaching a leaf node state with a selection policy. The leaf node state is a state which is not yet fully expanded, meaning it has children that are not visited or it is not a terminal state. The selection policy of the walk is based on the upper confidence bound ($UCB1$) formula. The idea is to weight out and balance the 'exploration vs exploitation' trade-off. We select an action, leading to then next node state, which has the biggest $UCB1$ value. The standard $UCB1$ [5] formula is

$$UCB1 = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N_i}{n_i}}, \tag{4.17}$$

where w_i can be the number of wins of the node or the sum of the backpropagated rewards, n_i the number of visits of the node, N_i the number of visits of the parent node and $C \in R$ is a constant, usually chosen to be $C = \sqrt{2}$.

2. Node Expansion Phase: If the node state is not a terminal one, a new extra node is added or multiple ones into the search tree according to the available actions.

3. Simulation Phase: Also called the Roll-out state. A simulation/roll-out evolves around the current node selecting actions randomly until reaching a terminal state.

4. Backpropagation Phase: The reward of the simulation is used to update the information in the nodes traversed. The information backpropagated is the outcome/reward of the terminal state at the end of the simulation and also the visit count of the nodes.

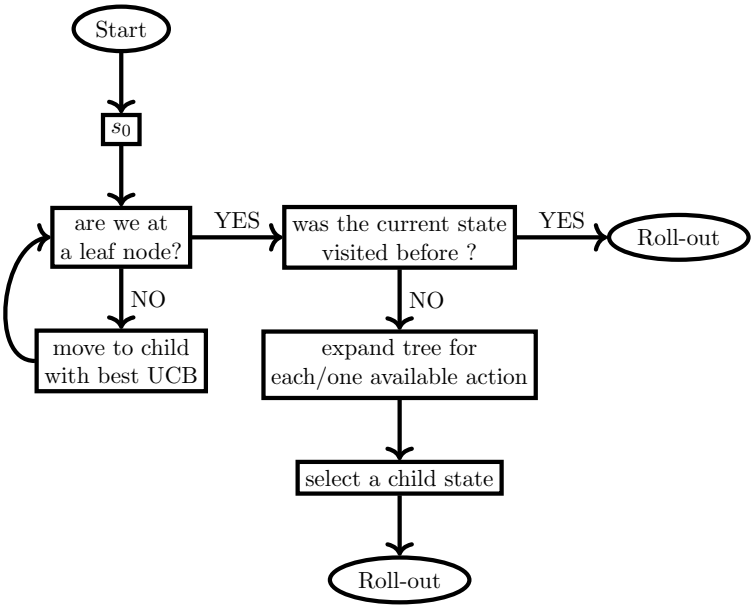


Figure 4.3: MCTS algorithmic diagram

The general algorithmic structure is described in [16], denoted in 1. For clarity purposes the diagram of the algorithm is in 4.3. The search starts with an input state s_0

Algorithm 1 MCTS Algorithm

Input: state s_0
  while within computational budget **do**
    $s_k \leftarrow \text{SelectionPolicy}(s_0)$
    $r \leftarrow \text{RollOut}(s_k)$
    $\text{BackPropagate}(s_k, r)$
  end while
Output: $\pi^{\text{MCTS}}(s_0)$

selecting the next state based on the selection policy, based on the biggest $UCB1$ value. The state s_k is reached after the selection phase and the expansion, upon which a simulation is run giving the terminal result/outcome r. The outcome is then communicated back from s_k back to the root node in the path taken to reach s_k from the root node s_0. At the end of a MCTS search we output a policy distribution $\pi^{\text{MCTS}}(s_0)$ at the root state. The policy distribution of the MCTS is usually determined by the visit counts of the available actions. It could also be determined by the expected values of the available actions.

The MCTS algorithm is a very diverse algorithm. It allows for a wide array of variations and improvements, from changing the selection policy to droping one of the four basics steps and utilizing learning parameters.

4.3 Graph Neural Networks

In this section we go through an overview of Graph Neural Networks (GNNs) [17]. GNNs are Neural Networks (NNs) which operate over graph structured data. Let us further denote a graph as a tuple (E, V), where E are edges and V are the nodes of the graph. Also let A be the adjacency matrix of the graph, where $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ otherwise. Finally, we denote X to be the set of node feature.
There are multiple types of GNNs some more complex than others [17]. The core principles are taken from the foundational, convolutional NNs. An image can be seen as a grid graph, where each node corresponds to a pixel and is connected to its neighboring pixels. Simple convolution in images aggregates pixel by applying a simple operation (kernel) on its neighborhood, thereby updating the pixels value. Similarly, in a graph the convolutional GNN will take the information of the neighboring node features (possible also edge features) to update the underlying node by applying some function/transformation on them 4.4.

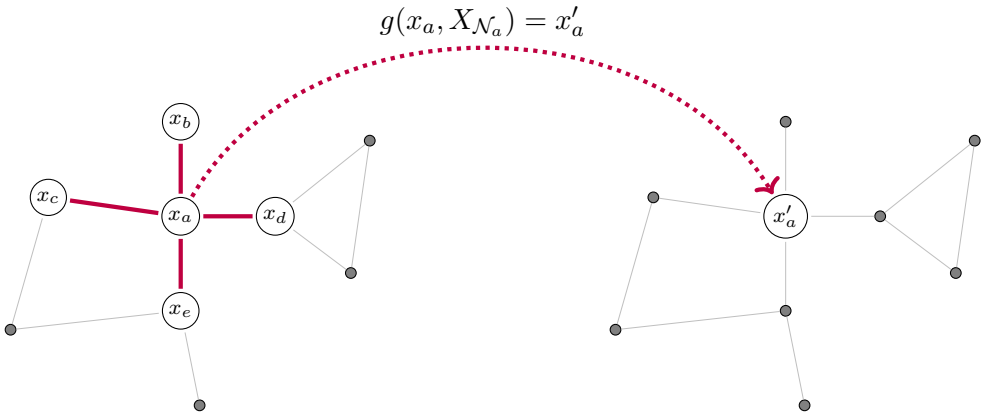


Figure 4.4: Convolutional GNNs: simple example

For each node $i \in V$ we can define its 1-hop neighborhood

$$\mathcal{N}_i := \{j : (i, j) \in E \ \wedge \ (j, i) \in E\}. \tag{4.18}$$

The features of the 1-hop neighborhood $\mathcal{N}_i$ of the node are

$$X_{\mathcal{N}_i} = \{\{x_j : j \in \mathcal{N}_i\}\}. \tag{4.19}$$

The function g operates on the features of the node itself and on the features of its 1-hop neighborhood $g\left(x_i, X_{\mathcal{N}_i}\right)$. Suitable are functions that preserve the invariance or equivariance of the graph. When applying a permutation matrix P we need to permute the rows and the columns of the adjacency matrix A, which results to PAP^T. This leads us to viable functions that satisfy

$$f(PX, PAP^T) = f(X, A) \quad \text{Invariance} \tag{4.20}$$

$$f(PX, PAP^T) = Pf(X, A) \quad \text{Equivariance} \tag{4.21}$$

By applying the local function g on all nodes and their neighborhoods we get

$$f(X, A) = \begin{bmatrix} g(x_1, X_{\mathcal{N}_1}) \\ \vdots \\ g(x_n, X_{\mathcal{N}_n}) \end{bmatrix}. \tag{4.22}$$

Thus, g needs to be permutation invariant and not depend on the order in $X_{\mathcal{N}_i}$, then f will be equivariant. Conventionally the local function g is referred to as 'diffusion' , 'propagation' or 'message passing' function and the global function f as a 'GNN Layer' [17, Chapter 5]. The majority of the GNN Layer types can be distinguished into three types, *Convolutional*, *Attentional* and *Message-passing* GNN. They are all different, in the sense how g incorporates the neighboring features.
The convolutional type [17, Chapter 5] as described above for clarification, aggregates the features of the neighbors with additional fixed weights c_{ij} after applying some function ψ on them

$$x_i' = g\left(x_i, \bigoplus_{j \in \mathcal{N}_i} c_{ij}\psi(x_j)\right). \tag{4.23}$$

The attention type [18] uses features of neighbors to apply implicit weights, via attention. The attention coefficient for each pair is computed via a function $\alpha_{ij} = a(x_i, x_j)$, usually a variation of the softmax function resulting in

$$x_i' = g\left(x_i, \bigoplus_{j \in \mathcal{N}_i} a(x_i, x_j)\psi(x_j)\right). \tag{4.24}$$

The message passing type [17, Chapter 5] computes arbitrary vectors 'messages', these are then sent through the edges back to the original node which gives

$$x_i' = g\left(x_i, \bigoplus_{j \in \mathcal{N}_i} \psi(x_i, x_j)\right). \tag{4.25}$$

In practice the functions g and ψ are learnable and are represented by some appropriate NN structure and the $\bigoplus$ operator is a nonparametric operation e.g. sum, mean, maximum or a recurrent neural network [17, Chapter 5].

We focus on the attention type, introduced in the famous GAT paper [18]. Given a set of node features $(h_i)_{i=1}^N \subset \mathbb{R}^d$, the GAT layer produces new features $(h_i')_{i=1}^N \subset \mathbb{R}^{Kd'}$ with dimension Kd'. The computation of the attention coefficient is the following

$$\alpha_{ij} = \operatorname{softmax}_j\left(\operatorname{abs}(e_{ij})\right), \tag{4.26}$$

where an attention mechanism $a : \mathbb{R}^{d'} \times \mathbb{R}^{d} \to \mathbb{R}$ computes the attention coefficients through a learnable matrix $W \in \mathbb{R}^{d' \times d}$

$$e_{ij}^{(k)} = a(Wh_i, Wh_j). \tag{4.27}$$

To stabilize the learning process, a multi-head attention mechanism is employed with K independent attention heads through

$$h_i' = \|_{k=1}^{K} \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^{k} W^k h_j \right), \tag{4.28}$$

where σ is an activation function, α_{ij}^{k} is normalized attention coefficient computed by the k-th attention head a^k with the corresponding learnable matrix W^k and $\|$ is the concatenation operator. The usual choice for a is single layer feed forward neural network with LeakyReLU activation function.

4.4 REINFORCE

In the following section we introduce a policy gradient method called REINFORCE [5, Chapter 13] on a discrete MDP. For convenience we set the discount factor $\gamma = 1$ throughout the explanation but it can will be later incorporated into the algorithm. The idea is to have a parametrization of the policy π_θ through some learnable parameters θ, which is differentiable wrt. θ and finite in the state action space. We would like to find the optimal policy by updating the weights via some performance measure $J(\theta)$ at the start of an episode s_0 through

$$J(\theta) = V_{\pi_\theta}(s_0), \tag{4.29}$$

where V_{π_θ} is the policy value function of the parametrazed policy π_θ. We would like to maximize the performance measure by going in the estimated direction of the gradient ascent of J wrt. θ, such that the updates are of the form

$$\theta_{t+1} = \theta_t + \alpha \mathbb{E}[\nabla J(\theta_t)], \tag{4.30}$$

where α is the learning rate and expectation is over some appropriate distribution. To get an appropriate estimation of the gradient, we take a look at the gradient of the value function, by equation (4.6) we write the policy value as function of the state value

$$\begin{aligned} \nabla V_{\pi_\theta}(s) &= \nabla \mathbb{E}_{\pi_\theta} \left[Q_{\pi\theta}(s, a) \right] && (4.31) \\ &= \nabla \left(\sum_a \pi_\theta(s|a) Q_{\pi_\theta}(s, a) \right) && (4.32) \\ &= \sum_a \Big(\nabla \pi_\theta(a|s) Q_{\pi_\theta}(s, a) + \pi_\theta(a|s) \nabla Q_{\pi_\theta}(s, a) \Big), && (4.33) \end{aligned}$$

where the gradient here is always wrt. θ. By the definition of the state-action function 4.1 we have

$$\nabla V_{\pi_\theta}(s) = \sum_a \Big(\nabla \pi_\theta(a|s) Q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla \sum_{s',r'} \mathbb{P}[s',r'|s,a] \left(r + V_{\pi_\theta}(s')\right) \Big) \tag{4.34}$$

$$= \sum_a \Big(\nabla \pi_\theta(a|s) Q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s'} \mathbb{P}[s'|s,a] \nabla V_{\pi_\theta}(s) \Big). \tag{4.35}$$

We can keep applying the same calculation to the term in $\nabla V_{\pi_\theta}(s)$ to get an expectation under the probabilities of landing in some state x from the state s in k steps under π, i.e.

$$\nabla V_{\pi_\theta}(s) = \sum_{x \in S} \sum_{k \in \mathbb{N}} \mathbb{P}[s \to x, k, \pi] \sum_a \nabla \pi_\theta(a|s) Q_\pi(s,a). \tag{4.36}$$

Then we have that

$$\nabla J(\theta) = \nabla V_{\pi_\theta}(s_0) \tag{4.37}$$

$$= \sum_s \sum_{k \in \mathbb{N}} \mathbb{P}[s_0 \to s, k, \pi] \sum_a \nabla \pi_\theta(a|s) Q_\pi(s,a) \tag{4.38}$$

$$= \sum_s \eta(s) \sum_a \nabla \pi_\theta(a|s) Q_\pi(s,a) \tag{4.39}$$

$$= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi_\theta(a|s) Q_\pi(s,a) \tag{4.40}$$

$$= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla \pi_\theta(a|s) Q_\pi(s,a) \tag{4.41}$$

$$\propto \sum_s \mu(s) \sum_a \nabla \pi_\theta(a|s) Q_\pi(s,a). \tag{4.42}$$

Where the constant to which the last equation is proportional to is the average length of an episode. The distribution μ is the on-policy distribution over the regular π [5, Chapter 9, 10], thus

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_a Q_{\pi_\theta}(s_t, a) \nabla \pi_\theta(a|s_t) \right] \tag{4.43}$$

by replacing s by some sample s_t. Doing the same for a to $a_t \sim \pi$ we can expand by π_θ to preserve equality

$$\nabla J(\theta) = \mathbb{E}_\pi \left[\sum_a Q_{\pi_\theta}(s_t, a) \pi_\theta(a|s_t) \frac{\nabla \pi_\theta(a|s_t)}{\pi_\theta(a|s_t)} \right] \tag{4.44}$$

$$= \mathbb{E}_\pi \left[Q_{\pi_\theta}(s_t, a_t) \frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right] \tag{4.45}$$

$$= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right], \tag{4.46}$$

where G_t is the sum of the discounted rewards from $t+1$ to the terminal state T final weights update for the REINFORCE method is

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}, \tag{4.47}$$

where for the expression $\frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}$, we can just compute the gradient of natural logarithm, i.e. $\nabla \ln \pi_\theta(a_t|s_t)$. Hence the reinforce algorithm has the following structure Furthermore we can generalize the gradient updates to include a baseline function $b(s)$

Algorithm 2 REINFORCE

Input: Policy parametrization π_θ, step-size α, initial parameters θ
  while within computational budget **do**
    Generate an episode $(s_0, a_0), \ldots, (s_T, a_T)$ through π_θ
    for $t = 0, 1, \ldots, T-1$ **do**
      $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
      $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla \ln \pi_\theta(a_t|s_t)$
    end for
  end while

[5, Chapter 13.4], instead of updating G_t we centralize the rewards by

$$G_t - B(s_t), \tag{4.48}$$

where B is a baseline function, giving a faster convergence. This is called the REINFORCE with baseline algorithm and there are a verity of options when construing the baseline function, some work better then others.

4.5 REINFORCE with MCTS

In this section we give a brief overview of how we can utilize MCTS with REINFORCE. Instead of doing a one-per-one episode, we are going to compute batches of episodes and average out the Monte Carlo improved policy gradients to compute the updates. There are two separate phases for making the algorithm work with purely reinforcement learning, in the one phase the model plays with itself and gathers information about the environment and gathers the 'data'. In the second phase the information of the self play part is used to update the weights of the policy parametrization 4.5.

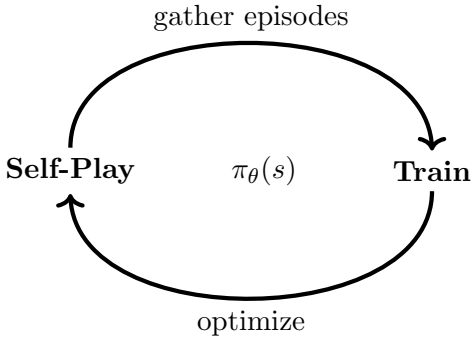


Figure 4.5: Self-Play loop

The policy is parametrized through through the parameters θ of a deep neural network π_θ, which takes a state as input and outputs the transition probability vector $\pi_\theta(s)$ for each action. In the self-play phase the model uses the MCTS algorithm to traverse down the MDP tree from the root node s_0. The key difference from standard MCTS is the selection strategy, the UCB selection formula used is

$$UCB = Q(s,a) + U(s,a), \tag{4.49}$$

where $Q(s,a)$ is the state-action value through backpropagation of the reward, and

$$U(s,a) = C\pi_\theta(a|s)\sqrt{\frac{\sum_i N(s,a_i)}{1+N(s,a)}}, \tag{4.50}$$

where C is the exploration-exploration constant, $\pi_\theta(a|s)$ is the probability output for action a at state s of the parametrized policy, $N(s,a)$ the node visit count and $\sum_i N(s,a_i)$ is the visit count of the parent. Generally it is non-trivial to find a suitable value for the constant C, theoretically it is set to $\sqrt{2}$ in the multi-armed bandit attack and MDP environments which have rewards spread evenly in the interval $[0,1]$. To avoid time-consuming tuning of the constant C we normalize the Q values during backpropagation. For the normalization we keep track of two quantities, the Q value of the state action and the best value b. The Q value does not exceed 1 and at the start of each search the best value of the root node is set to $b_0 = 1$ and else 0. Continuously during the backpropagation phase if the best child value exceeds the best value of the parent and is bigger than 0, then b_p (parent best value) is updated to the better value and the Q value is then set to

$$Q_c = \frac{r_c}{b_p}, \tag{4.51}$$

where the subscript c denotes the child and the subscript p the parent node. The reward r_c is the estimated reward of the child from the simulation phase, and b_p is the best value of the parent.

After an MCTS search we get the MCTS improved policy at state s for action a ,

$$\pi^{\text{MCTS}}(a|s) = \frac{Q(s,a)}{\sum_{a'} Q(s,a')} \tag{4.52}$$

In the case there are not Q values found above 0 then we opt to use the standard MCTS improved policy of visit counts. We sample an action from the probability distribution $\pi^{\text{MCTS}}(s)$ while training to avoid the policy becoming deterministic. However, In the testing or production phase we choose the action with the highest probability through

$$a = \arg\max_a \pi^{\text{MCTS}}(a|s). \tag{4.53}$$

In the training phase, after playing N_{play} times, for each time step of each episode we gather and compute the averages of $\nabla \ln \pi_\theta$ and G_t for all episodes on all states to update the parameters as in REINFORCE 2.

5 N-Cyclic Arbitrage Walks with MCTS based REINFORCE

In this chapter we go through the RL framework and neural network model used to find profitable walks in the CFMM arbitrage problem and present results. Firstly we introduce the model for the underlying MDP, states, actions and rewards. The policy is parametrized via a deep GNN of the line graph of the problem and the training uses the setup from chapter 4.5.

5.1 Agent and Environment

The goal is to find a walk in a graph that does not use repeated edges (trail), starts and ends at the same node, i.e. is circular, and is profitable. The graph is represented by ERC20 [19] tokens (referred to as assets) as nodes and decentralized exchange pools, CFMM with two assets, as edges (referred to as pools). Each pool has a swap fee, which usually is in the range of $0.3\% - 1\%$, and a marginal exchange rate. The marginal exchange rate, unique to the pool, is inversely bidirectional. Meaning, while swapping token 0 for token 1 in the pool the usual marginal exchange rate is used, but when swapping from token 1 to token 0 the inverse marginal exchange rate is used. In both cases the referred term is marginal exchange rate. 'Used' here is meant in the context of the learning setup, in the actual setting the automated market maker formula is used to determine the amount exchanged 3. Throughout we will use the term 'pool' and edge, 'asset' and node interchangeably.

The agent starts from a given node (asset) and chooses actions from a set of valid actions. Each action represents a trade, where the agent swaps an asset for another asset through a specific pool. The valid actions rule out the edges that have already been used. The idea behind this is the following: If there is an arbitrage opportunity, and the trades are conducted in a chronological order, at each swap the pool's 'capacity' will be exerted and the pools' exchange rate will be in balance with the overall market. Hence, the pool has no liquidity for an arbitrage to be used again.

We terminate the process of trades, when the agent has arrived back to the starting point, then the product of the exchange rates is calculated to give the profit of the walk. In this case, at each trade a trading penalty can utilized to account for the fees of each transaction. We also terminate the process of trades, when the agent arrives at a dead end, meaning the agent has not reached the starting point and has no valid actions left.

To bring it all together the MDP is based on a multi bidirectional graph $G = (V, E, W)$, where V is the set of nodes (assets), E the set of edges (pools) and W the set of edge weights (prices based on the specific pool). An edge $e_{ij}^k \in E$ represents the pool k trading assets i j, it has the weight

$$w_{ij}^k = \frac{p_i^k}{p_j^k}, \tag{5.1}$$

where p_i^k and p_j^k are the marginal exchange rates of the assets i and j in pool k respectively as defined in equation (3.13). The edge e_{ji}^k has the weight

$$w_{ji}^k = \frac{1}{w_{ij}^k} = \frac{p_j^k}{p_i^k}. \tag{5.2}$$

The index k denotes the k-th pool trading asset i and asset j, since there could be multiple pools trading the same asset. The underlying MDP is a 4 tuple (S, A, P, R), deterministic, and is specified by :

Actions. The action space A are all the possible actions the agent can take on the graph defined above. The agent acts by taking an action $a_t \in A$ at step t by selecting an edge e_{ij}^k leading to adjacent nodes and thereby transitioning from node i to node j at pool k. The action is valid if the edge e_{ij}^k has not been selected throughout the episode.

States. The state space S, and a state $s_t \in S$ is the edge set traversed by the agent up to step t of the episode. The state s_t is terminal if there are no valid actions or the agent has reached the origin node. If the agent has not reached a terminal state then the agent selects an action $a_t \in A$ and transitions to the next state $s_{t+1} = s_t \cup a_t$ recursively.

Transition. The transition probabilities P, where $\mathbb{P}[s'|s, a] = 1$ is the probability of landing in state s' by taking action a at state s. The MDP is deterministic.

Rewards. The reward probabilities R, since the MDP is deterministic the reward function at state s_t is $r(s_t)$. Throughout the MDP, the agent is rewarded $r(s_t) = 0$, except when at a terminal state. All together the agent is rewarded by

$$r(s_t) = \begin{cases} \sum_{w_{ij}^k \in s_t} \ln(w_{ij}^k \cdot \tau) & \text{terminal state} \\ -1 & \text{no valid actions} \\ 0 & \text{else,} \end{cases} \tag{5.3}$$

where w_{ij}^k are the weights of the walk of s_T, i.e. marginal exchange rates, $\tau \in [0, 1]$ is the trade penalty which incorporates a rough estimate of the transaction cost for each trade. Throughout the training we set $\tau = 1$, however in production it might be useful to set an adaptive value to get an estimate if the overall trade will be profitable after paying the protocol-transaction fees. If the sum of the logarithm of the marginal exchange rates is bigger then zero then the trade is profitable and otherwise it is not. The natural logarithm comes here as a big convenience to characterize profitable and

nonprofitable trades,it also avoids 256-bit integer overflows, (most of the blockchains do not use floating point number, but unsigned 256-bit integers) which needs special attention when coding in production. In summary the agent is only rewarded if end node matches the start node.

5.2 Policy Parametrization

We are going to parametrize the policy π_θ, such that given a state s the output of $\pi_\theta(s)$ will be a probability vector of size $n = |E|$. This probability vector will be masked based on the valid actions at state s and then normalized. To archive this we will encode the state s through the line graph $L(G)$ of the graph used in the MDP. The nodes of the line graph $L(G)$ represent the edges of the graph G. Two nodes are connected by an edge in $L(G)$ if and only if their corresponding edges in G share a common endpoint in G. Instead of considering both edges e_{ij}^k, e_{ji}^k, we will only consider the corresponding pool connection as an edge. A node of the line graph $L(G)$ will have the following attributes

- Natural logarithm of the marginal exchange rate corresponding to the pool
- Binary entry stating if the pool was used 1 or not used 0
- Binary entry stating if agent is currently using asset 0 of the pool, 1 or not 0.
- Binary entry stating if agent is currently using asset 1 of the pool, 1 or not 0.

Given a state s the input to the policy parametrization will be encoded through graph as stated in the list above. The architecture of the deep GNN parametrization of the policy is as follows. Given the node attributes, the encoder computes the node embeddings h_i^0 for all nodes i through a learned liner projection to dimension $d_h = 128$. The embeddings are then updated via. N Residual Multihead Attention Layers [20]. Each layer consists of two sublayers, a multihead attention layer (GAT) with 4 heads, for message passing between the nodes which is flattened by a linear connection back to the original dimension d_h. And a node-wise fully connected feed-forward layer (FF) with 512 hidden dimension. Both sublayers have a Batch Norm (BN) and a residual connection similar to [21]. The following is a formalization of the residual block

$$\hat{h}_i = \mathrm{BN}^l\left(h_i^{(l-1)} + \mathrm{GAT}_i^l(h_1^{(l-1)}, \ldots, h_n^{(l-1)})\right) \tag{5.4}$$

$$h_i^{(l)} = \mathrm{BN}^l\left(\hat{h}_i + \mathrm{FF}_i^l(\hat{h}_i)\right). \tag{5.5}$$

After which we will do a contextual embedding by concatenating the graph embedding $\hat{h}^{(N)} = \frac{1}{n}\sum_{i=1}^n h_i^{(N)}$, the node embedding of the initial state $h_i^{\pi(s_0)}$ and the vectors h_i^N to the vector

$$h_i^{(c)} = \hat{h}^{(N)} \| h_i^{\pi(s_0)} \| h_i^{(N)}, \tag{5.6}$$

where $\|$ is the concatenation operator. The contextual embeddings $h_i^{(c)}$ are passed into the policy head. The policy head consists of GAT layer with 4 heads passed to a Batch Norm layer and ultimately compressed down to an output dimension of 2 per node

through a linear connection with Relu activation function

$$\pi_\theta(s) = \text{softmax}\left(W_p \cdot \text{ReLU}\left(\text{BN}_p(\text{GAT}_p(h_1^{(c)}, \dots, h_n^{(c)}))\right)\right), \tag{5.7}$$

where $W_p \in \mathbb{R}^{2\times d_h}$ is the learned linear connection, and the index p denotes the distinct weights used for the policy head. To get the policy vector we flatten and pass the softmax at the end.

5.3 Algorithm and Training

The training is purely unsupervised based on the MCTS and REINFORCE algorithm as described in Section 4.5. At each self play The marginal exchange rates of the prices are chosen randomly, some block number b on the Ethereum blockchain. The next state is chosen based on N_{MCTS} searches with selection probabilities computed through π_θ. In each iteration the agent plays the network N_{play} times until reaching a terminal state. Upon reaching a terminal state: the state, the action, the reward and the policy vector of the MCTS result are saved. After the self play stage the model parameters are updated based on the self play data. We fully utilize the REINFORCE algorithm updates through 2, where the loss function can be written as

$$L(\theta) = -\frac{1}{N_{\text{batch}}} \sum_{i=1}^{N_{\text{batch}}} \left(G_i - B(b_i, s_{it}^{b_i})\right) y_i \ln \pi_\theta(s_{it}^{b_i}). \tag{5.8}$$

Values of G_i are the sum of the rewards as in the REINFORCE algorithm, and y_i a one hot vector with 1 at the index of the action taken sampled from the MCTS improved policy. We utilize a baseline function $B(b_i)$ dependent on the block b_i. During the training we gather the rewards of each block, we then compute the average and multiply it by the discount factor γ with exponent dependant on the state $s_{it}^{b_i}$ as per G_i. E.g. the first state of an episode the exponent would be T and at the last one 1.
We use the MCTS version described in 4.5. The action taken at the end of an MCTS is determined by the MCTS improved policy π^{MCTS} based on (4.52). Instead of doing one rollout per state we initiate N_{sim} number of rollouts. The full algorithm utilizing MCTS with REINFORCE is the following in pseudocode 3.

Algorithm 3 RL Algorithm

```
while within computational budget do
    for 1, ..., N_Epoch do
        for i = 1, ..., N_play do
            Initialize state s = s_0 at block b ~ Uniform(B)
            while not terminal do
                π^MCTS(s) ← MCTS(s)
                s ← a ~ π^MCTS(s)
                save s, π^MCTS(s) and a
            end while
        end for
        θ ← Adam(θ, ∇L(θ))
    end for
end while
```

The code can be found at [22, Github].

The weighs of the model are updated using the Adam optimizer [23] with a learning rate of $\alpha = 10^{-4}$, a weight decay of $\lambda = 10^{-4}$, forgetting factors $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$ to prevent division by zero. We set the discount rate constant $\gamma = 0.98$ from REINFORCE 2. We use $N_{\text{batch}} = 25$, $N_{\text{play}} = 200$, $N_{\text{MCTS}} = 100$ with $N_{\text{sim}} = 100$ (rollout simulations for value estimation) with $N_{\text{Epoch}} = 100$ iterations. The marginal exchange rates are pulled from the block range $t \in [18 \cdot 10^6, 21.8 \cdot 10^6]$. The number of residual block layers is $N = 4$, these do not share weights. The UCB exploration-exploitation parameter C in the MCTS algorithm for choosing the next child in the tree is set to $C = 1.4$.

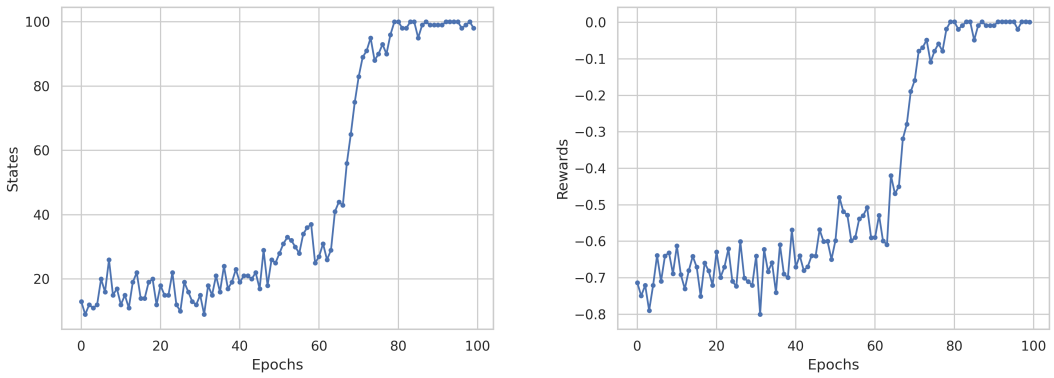
5.4 Data

The data used is publicly available on the Ethereum mainnet blockchain and can be accessed through an archive node query. The pools used are all of UniswapV2 [4] and UniswapV3 [10]. The difference between V2 pools and V3 pools is that the V3 pools have concentrated liquidity pools, but both the follow constant product function. At the current time of writing there are around $4.3 \cdot 10^5$ pools and approximately $4.1 \cdot 10^5$ different ERC20 [19] tokens in these pools. To narrow down the meaningful pools, firstly we use pools that have been created before block $18 \cdot 10^6$ and secondly we use a degree filter. Assets which have a smaller degree then 4 are filtered out. The question still remains if these pools are still in use currently. Hence, the pools with low liquidity are also filtered out, by looking at their two token reserves, the WETH value of the reserves is crucial here. If the reserve is worth less then 100 WETH it is filtered out. We also look at the gradient of the marginal exchange rates for all the pools, and filter out the pools with low price change. For the training process the marginal exchange rates of the pools are extracted from block $18 \cdot 10^6$ until the current block every 3600 blocks which

is equal to around every 12 hours ($\sim 12s$ is the time until the next block). During each self-play, a random block is sampled from which the marginal exchange rates are taken. Each state starts with the agent holding the WETH asset, from which the agent chooses a pool to conduct a swap with WETH to another token until reaching a terminal state. The code for the data extraction can be found at [22, Github].

5.5 Results

Firstly we will look at the pure REINFORCE method on one block. We just apply the Algorithm from 2 on the MDP. Instead of updating each episode, we gather $N_{\text{batch}} = 100$ episode batches to update the model weights. Figure 5.1 shows the progress. Sub-figure 5.1a shows the number of successful states per epoch. We label a successful state if the states has reached the origin node at the end of the walk. After about 60 epochs the GNN learns to reach the origin node rapidly. Sub-figure 5.1b shows the average rewards per epoch, correlating with the successful states. The problem however is that the agents gets stuck at a local optimum and does not reach a maximal reward of $r_{\text{max}} = 0.11$, rather pivots at $r_{\text{agent}} = 0.0015$.



(a) Successful states

(b) Average rewards

Figure 5.1: REINFORCE on one block

With the help of the MCTS and a baseline we can avoid the local optimum. The learning curve, however, can vary a lot based on the initial parameters, blocks chosen for training and paths found by the MCTS. After 100 training iterations/epochs we arrive at a steady precentige wise validation gap wrt. true optimals, on the profits on an average of 20 runs, of 0.942% and a mean squared error of 0.031, for REINFORCE with MCTS. The true optimal profits are calculated by a Depth-First-Search on the MDP and compared to the profits found by model at after training for 100 epochs on 100 different blocks (price data), which do not change throughout the validation process. In total this approach is enough to give an edge to a Searcher in a last second call. The results can be

further improved and learning process stabilized using additional supervised learning. Calculations were performed using supercomputer resources provided by the Vienna Scientific Cluster (VSC)

Bibliography

[1] Ethereum Foundation. *Ethereum Foundation*. Jan. 4, 2025. URL: https://ethereum.org/.

[2] Flashbots. *Flashbots*. Jan. 4, 2025. URL: https://www.flashbots.net/.

[3] Ethereum Foundation. *Proposer-Builder separation*. Jan. 4, 2025. URL: https://ethereum.org/en/roadmap/pbs/.

[4] Uniswap. *Uniswap*. Feb. 7, 2025. URL: https://docs.uniswap.org/.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[6] David Silver et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. arXiv: 1712.01815 [cs.AI]. URL: https://arxiv.org/abs/1712.01815.

[7] Radu Ioan Boţ. *Nonlinear Optimization Lecture Notes*. University of Vienna, 2022/2023.

[8] Guillermo Angeris et al. *Constant Function Market Makers: Multi-Asset Trades via Convex Optimization*. 2021. arXiv: 2107.12484 [math.OC]. URL: https://arxiv.org/abs/2107.12484.

[9] Guillermo Angeris et al. *Optimal Routing for Constant Function Market Makers*. 2022. arXiv: 2204.05238 [math.OC]. URL: https://arxiv.org/abs/2204.05238.

[10] Hayden Adams et al. *Uniswap v3 Core*. 2021. URL: https://app.uniswap.org/whitepaper-v3.pdf.

[11] Balancer. *Balancer*. Feb. 7, 2025. URL: https://balancer.fi/.

[12] Sushiswap. *ShusiSwap*. Feb. 7, 2025. URL: https://www.sushi.com.

[13] CurveFI. *CurveFI*. Feb. 7, 2025. URL: https://www.sushi.com.

[14] Michael Egorov. *StableSwap - efficient mechanism for Stablecoin liquidity*. 2019. URL: https://berkeley-defi.github.io/assets/material/StableSwap.pdf.

[15] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. Chap. 17.

[16] Cameron B. Browne et al. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: 10.1109/TCIAIG.2012.2186810.

[17] Michael M. Bronstein et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges". In: *CoRR* abs/2104.13478 (2021). arXiv: 2104.13478. URL: https://arxiv.org/abs/2104.13478.

[18] Petar Veličković et al. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML]. URL: https://arxiv.org/abs/1710.10903.

[19] Ethereum Foundation. *ERC20 Token Standard*. Feb. 7, 2025. URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/.

[20] Shaked Brody, Uri Alon, and Eran Yahav. *How Attentive are Graph Attention Networks?* 2022. arXiv: 2105.14491 [cs.LG]. URL: https://arxiv.org/abs/2105.14491.

[21] Wouter Kool, Herke van Hoof, and Max Welling. *Attention, Learn to Solve Routing Problems!* 2019. arXiv: 1803.08475 [stat.ML]. URL: https://arxiv.org/abs/1803.08475.

[22] Milutin Popović. *Master Thesis*. Jan. 4, 2025. URL: https://github.com/miksa234/master.

[23] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[24] Oleg Arenz. "Monte Carlo Chess". en. bachelor thesis. Darmstadt: Technische Universität Darmstadt, Nov. 2022, 35 Seiten. DOI: https://doi.org/10.26083/tuprints-00022930. URL: http://tuprints.ulb.tu-darmstadt.de/22930/.

[25] Raghuram Ramanujan, Ashish Sabharwal, and Bart Selman. "Understanding Sampling Style Adversarial Search Methods". In: *CoRR* abs/1203.4011 (2012). arXiv: 1203.4011. URL: http://arxiv.org/abs/1203.4011.

[26] Marco Grassia and Giuseppe Mangioni. "wsGAT: Weighted and Signed Graph Attention Networks for Link Prediction". In: *CoRR* abs/2109.11519 (2021). arXiv: 2109.11519. URL: https://arxiv.org/abs/2109.11519.

[27] Qi Wang and Yongsheng Hao. "Routing optimization with Monte Carlo Tree Search-based multi-agent reinforcement learning". In: *Applied Intelligence* 53 (Aug. 2023), pp. 1–16. DOI: 10.1007/s10489-023-04881-1.

[28] S.P. Boyd and L. Vandenberghe. *Convex Optimization*. Berichte über verteilte messysteme pt. 1. Cambridge University Press, 2004. ISBN: 9780521833783. URL: https://books.google.pt/books?id=mYm0bLd3fcoC.

[29] Theo Diamandis, Guillermo Angeris, and Alan Edelman. *Convex Network Flows*. 2024. arXiv: 2404.00765 [math.OC]. URL: https://arxiv.org/abs/2404.00765.

[30] Ethereum Foundation. *EVM Optcodes*. Jan. 4, 2025. URL: https://ethereum.org/en/developers/docs/evm/opcodes/.