ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230573

Simplex Decomposition for Portfolio Allocation Constraints in Reinforcement Learning

David Winkel[a,b;*], **Niklas Strauß**[a,b], **Matthias Schubert**[a,b] **and Thomas Seidl**[a,b]

[a]LMU Munich, Germany
[b]Munich Center for Machine Learning (MCML)
ORCiD ID: David Winkel https://orcid.org/0000-0001-8829-0863,
Niklas Strauß https://orcid.org/0000-0002-8083-7323, Matthias Schubert https://orcid.org/0000-0002-6566-6343,
Thomas Seidl https://orcid.org/0000-0002-4861-1412

Abstract. Portfolio optimization tasks describe sequential decision problems in which the investor's wealth is distributed across a set of assets. Allocation constraints are used to enforce minimal or maximal investments into particular subsets of assets to control for objectives such as limiting the portfolio's exposure to a certain sector due to environmental concerns. Although methods for constrained Reinforcement Learning (CRL) can optimize policies while considering allocation constraints, it can be observed that these general methods yield suboptimal results. In this paper, we propose a novel approach to handle allocation constraints based on a decomposition of the constraint action space into a set of unconstrained allocation problems. In particular, we examine this approach for the case of two constraints. For example, an investor may wish to invest at least a certain percentage of the portfolio into green technologies while limiting the investment in the fossil energy sector. We show that the action space of the task is equivalent to the decomposed action space, and introduce a new reinforcement learning (RL) approach CAOSD, which is built on top of the decomposition. The experimental evaluation on real-world Nasdaq-100 data demonstrates that our approach consistently outperforms state-of-the-art CRL benchmarks for portfolio optimization.

1 Introduction

Portfolio optimization tasks belong to the family of resource allocation tasks in which an actor needs to allocate the available resources over a set of choices in each time step. Technically, resource allocation tasks can be considered multi-step decision problems with a standard-simplex action space describing all possible allocation choices, e.g., the set of all possible portfolio allocations in a financial setting. Policy gradient based RL can be used to optimize stochastic policies over the corresponding simplex action space and thus, they are often used to optimize portfolio allocation agents. For example, [20] proposes to use PPO [16] in combination with a Dirichlet action distribution for risk-aware portfolio optimization.

In many real-world financial settings, investors set maximum and minimum allocation weights to certain groups of assets for their portfolio. These constraints might originate from their client's investment guidelines, restrictions posed by the regulator, or the investors' economic opinion. For example, an investor might need to consider sustainability aspects in addition to generating economic returns. In this setting, the investor might be required to invest a minimal amount of 30% of the portfolio into green technologies and a maximum amount of 15% into companies belonging to the fossil energy sector. Allocation constraints reduce the set of allowed actions for the agent within the simplex action space, constraining the action space to a *subset of the original simplex action space* that can be described geometrically as a convex polytope. Unfortunately, directly modeling a suitable action distribution on this polytope, which can be used to formulate a parametrizable policy function, is inherently difficult. A viable alternative approach is constrained Reinforcement Learning (CRL), which penalizes constraint violations in order to teach the agent to avoid disallowed actions, e.g., [12, 21, 18]. However, most of these approaches cannot *guarantee* that no constraint will be violated, may exhibit unstable training behavior, or produce suboptimal results, especially if more than a single allocation constraint is needed.

In this paper, we propose an alternative approach that decomposes the original constraint action space into unconstrained sub-action spaces, each containing a subset of the assets. The actions from these sub-action spaces are then combined back into the original action space using a weighted Minkowski sum. We exploit that any constraint requiring a maximum investment into a subset of assets is equivalent to requiring a minimum investment in the inverse subset of assets. This allows us to consider only constraints requiring a minimum allocation to asset groups. For the case of two allocation constraints, we decompose the action space into four sub-action spaces. The first sub-action space invests into the assets that are restricted by both allocation constraints. The second and third sub-action spaces ensure the fulfillment of each constraint after the allocations in the first sub-action space. The final sub-action space freely distributes the remaining funds which were not needed to fulfill the constraints. The allocation of assets in each sub-action space can be parameterized using an unconstrained Dirichlet distribution. We employ PPO [16] to optimize the policy function over the joint distribution of the sub-action spaces. Our new approach CAOSD ensures a tractable computation of the joint probability and gradients of the sub-action spaces through an auto-regressive architecture. Additionally, we use a transformer-based encoder of the current pricing structure of the market which is based on the recent price development.

We demonstrate the effectiveness of our novel approach in port-

[*] Corresponding Author. Email: winkel@dbs.ifi.lmu.de

folio optimization tasks based on real-world Nasdaq-100 index data. The results show that our approach is able to generate significantly higher returns than state-of-the-art constraint RL methods in the overwhelming majority of cases.

The main contributions of this paper are the following:

- We introduce a novel decomposition for constrained simplex action spaces with two allocation constraints into several unconstrained sub-action spaces.
- We propose a new method named CAOSD utilizing this decomposition to effectively apply standard RL algorithms like PPO on a surrogate action space.
- We demonstrate that CAOSD is able to significantly outperform state-of-the-art CRL benchmark approaches on real-world market data.

Our paper is structured as follows: In Section 2, we give an overview of the related work and continue in Section 3 with a formal problem definition. Afterward, we introduce our decomposition and show how to utilize the decomposition for RL in Section 4. We proceed with an extensive experimental evaluation of our approach in Section 5 before concluding the paper in Section 6.

2 Related Work

Portfolio optimization tasks with allocation constraints can be formulated as constrained Markov decision processes (CMDPs) [3] and policies can be optimized using CRL approaches [12, 21, 18]. Note that our novel decomposition is able to parameterize the constraint action space directly, and thus, the task can be formulated as an (unconstrained) Markov decision process (MDP), for which optimal policies can be found through standard RL algorithms.

CMDPs have seen successful applications in fields such as network traffic [11], and robotics [10, 2]. Finance applications such as [4, 20] use a scalarized objective function to maximize the return while penalizing for risk to perform a multi-objective, i.e. risk/return, portfolio optimization. The allocation constraints in our setting allow also to control for risk as they can be used to limit the investor's exposure to risky sub-markets. [1] combine the risk/return optimization while also considering allocation constraints. They enforce these constraints by using a penalizing CRL approach stating that “there is no straightforward way to design an actor-network so that all proposed actions are compliant”. Our approach tackles this challenge and provides a way to do so on an actor-network level without the need for a penalty term.

There are different approaches to identify optimal policies in a CMDP setting. Penalty-based approaches include an additional penalty term into the objective function representing the constraints. An example is Lagrangian-based approaches such as [18, 5] that transform a constrained optimization problem into an unconstrained one by applying Lagrangian relaxation. A subsequent step solves a saddle point problem by optimizing the objective function and dynamically adjusting the penalty factor λ.

Alternative penalty-based approaches, such as [12], employ interior-point methods. Common drawbacks of penalty-based approaches are the need for additional hyperparameter tuning, expensive training loops, and the lack of guarantees for satisfying the constraints. An alternative approach is based on defining Trust Regions [2], which may produce constraint violations due to approximation errors. Furthermore, there are approaches based on prior knowledge [9] which pretrain a model to predict simple one-step dynamics of the environment. Other approaches are based on the use of Lyapunov functions to solve CMDPs by projecting either the policy parameter or the action onto the set of feasible solutions induced by state-dependent linearized Lyapunov constraints [6, 7]. However, this approach can be computationally expensive and, in some cases, numerically intractable, especially as the action space grows larger [7].

Our method relies on a novel decomposition of the action space into sub-spaces. Thus, approaches *factorizing the action spaces* are another important research area that is related to this work. In [17], the authors introduce action branching, which divides the action space into independent sub-action spaces. In contrast, our problem involves modeling sub-action spaces that depend on each other. Auto-regressive approaches, which can model dependencies between sub-action spaces [13, 19, 15], are another common method for the factorization of the action spaces. Unlike these works, our approach focuses on a novel decomposition of a *constrained simplex action space* to make the optimization problem easier to solve using standard RL methods.

3 Problem Setting

An MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the set of available actions, $\mathcal{P}$ the transition function describing the distribution over future states s' given a state-action pair (s, a), $\mathcal{R}$ is a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and γ is the discount factor.

For portfolio allocation tasks, $\mathcal{A}$ is defined as a simplex over a set of N assets $I = \{0, \dots, N-1\}$, i.e., $\mathcal{A} = \left\{a \in \mathbb{R}^N_{0,+} : \sum_{i=0}^{N-1} a_i = 1\right\}$. In other words, a_i describes the positive relative amount of capital assigned to the i^{th} asset.

An allocation constraint is defined by a subset $V \subseteq I$ of assets and a threshold value $c \in [0, 1]$ and implies that $\sum_{j \in V} a_j \geq c$ for all allocations $a \in \mathcal{A}$. Thus, at least the amount c of the available capital must be allocated to assets from the set V. Let us note that any *less-than* constraint can be rewritten into a greater-equal constraint and vice versa. In particular, assigning *at most* c into assets from V is equivalent to assigning *more than* $(1 - c)$ into the remaining assets $I \setminus V$, e.g., in a three asset setup with asset weights $x_1 + x_2 + x_3 = 1$ the greater-equal constraint $x_1 + x_2 \geq 0.3$ is equivalent to the constraint $x_3 < 0.7$. Thus, without the loss of generality, we will assume greater-equal constraints in the following. For portfolio allocation tasks with one or more allocation constraints, the action space is a convex polytope within the original simplex of all N-dimensional allocation vectors.

The goal of a constrained portfolio allocation task is to find a policy π_θ maximizing the expected reward $\mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T-1} \gamma^t \cdot r_{t+1}]$ where r_{t+1} is the direct reward observed for the t^{th} state transition of episodes τ sampled by π_θ while only using allowed allocations. In our setting, the reward corresponds to the direct economic returns of the entire portfolio. Finding a suitable formulation for π_θ becomes more and more complex with an increasing number of allocation constraints. In the following, we will formulate π_θ for up to two allocation constraints which can directly be used in combination with standard actor-critic and policy gradient RL algorithms. Thus, we consider the following action space with $V_1 \subseteq I$ and $V_2 \subseteq I$:

$$\mathcal{A}_{2C} = \Bigg\{a \in \mathbb{R}^N_{0,+} : \sum_{i \in I} a_i = 1, \sum_{j \in V_1} a_j \geq c_1,$$

$$\sum_{k \in V_2} a_k \geq c_2, 0 \leq c_1 \leq 1, 0 \leq c_2 \leq 1\Bigg\}$$

4 Constrained Allocation Optimization with Simplex Decomposition (CAOSD)

As mentioned in Section 3, the action space underlying our problem is a convex polytope within the standard simplex over a universe of n assets. Thus, directly defining a parameterizable probability distribution that could be used in a policy function for reinforcement learning is rather complex. To avoid this complexity, we propose to decompose the action space into four standard simplices over subsets of I. For a proper weighting, allocations taken from these four standard simplices add up to form a complete allocation action in the original action space. In the following, we will describe the simplices and how to compute weights, guaranteeing that the original and the decomposed action set are equivalent. Afterward, we will describe how a policy function can be defined on top of the decomposed action space and how policy gradient based reinforcement learning methods can be applied.

4.1 Action Space Simplex Decomposition

To formalize our approach, we begin by defining the basic elements of our decomposition, i.e., the padded standard simplices and their combination with the weighted Minkowski sum.

Definition 1. *Let $I = \{0, \ldots, N-1\}$ be a set of indices referring to respective dimensions in $\mathbb{R}^N$. Let S_K be a standard simplex in the subspace defined over the dimensions indicated by the index set $K \subseteq I$. Let $g_K : \mathbb{R}^{|K|} \rightarrow \mathbb{R}^N$ be a function that projects S_K into $\mathbb{R}^N$, by padding the entries for any elements in $\mathbb{R}^N$ in those dimensions with indices $I \setminus K$ with 0. Applying the function g on S_K then yields a **padded standard simplex (PSS)** defined as:*

$$PSS_K = g_V(S_K) = \Bigg\{ y \in \mathbb{R}^N_{0,+} : \sum_{j \in K} y_j = 1; y_i \geq 0 \; \forall i \in K; \quad y_j = 0 \quad \forall j \in I \setminus K \Bigg\} \quad \textit{for } K \neq \emptyset$$

and

$$PSS_K = g_V(S_K) = \Bigg\{ y \in \mathbb{R}^N_{0,+} : y_j = 0 \quad \forall j \in I \Bigg\} \quad \textit{for } K = \emptyset$$

Definition 2. *Given n sets of vectors $Q_1, \ldots, Q_n$ in $\mathbb{R}^N$, the **weighted Minkowski sum** of $Q_1, \ldots, Q_n$ is generated by adding each combination of vectors from sets Q_i after applying a respective weighting factor z_i, i.e., $(Q_i)_{z_i} = \{z_i \cdot q_i | q_i \in Q_i\}$ with $i = \{1, \ldots, n\}$. We write the weighted Minkowski sum of the sets as $M_z = (Q_1)_{z_1} + \ldots + (Q_n)_{z_n} = \{z_1 \cdot q_1 + \ldots + z_n \cdot q_n | q_1 \in Q_1, \ldots, q_n \in Q_n\}$. We refer to $(Q_i)_{z_i}$ for $i = \{1, \ldots, n\}$ as the weighted Minkowski summands.*

Our approach identifies four PSSs and a weighting vector $z = [z_1, z_2, z_3, z_4]$ that can be combined as a weighted Minkowski sum $M_z = (PSS_1)_{z_1} + (PSS_2)_{z_2} + (PSS_3)_{z_3} + (PSS_4)_{z_4}$ such that $M_z = \mathcal{A}_{2C}$.

The first weighted Minkowski summand PSS_{K_1} is built over the intersection of assets $K_1 = V_1 \cap V_2$. Investing into these assets contributes to fulfilling both constraints. In fact, if $c_1 + c_2 > 1$, we need to invest at least a portion of $z_1 = c_1 + c_2 - 1$ into assets from $V_1 \cap V_2$ to avoid over investment.

The second weighted Minkowski summand PSS_{K_2} is defined over the assets in $V_1 = K_2$. To fulfill the first constraint, we have to make sure that we at least invest c_1 into assets in V_1. However, we need to consider that any capital z_1 already being allocated into PSS_{K_1} also contributes towards fulfilling this constraint. Correspondingly, PSS_{K_3} is defined over the assets in $K_3 = V_2$ and requires an investment of c_2 minus any allocation made to $V_1 \cap V_2$ from PSS_{K_1} and PSS_{K_2}. Finally, PSS_{K_4} is defined over the complete asset universe I. It covers the case, that not any available capital is needed to fulfill the allocation constraints. Thus, any remaining capital $1 - (z_1 + z_2 + z_3)$ can be freely allocated among the assets in I to maximize the economic return. An illustration of the four sets being covered by these weighted Minkowski summands can be found in Figure 1.

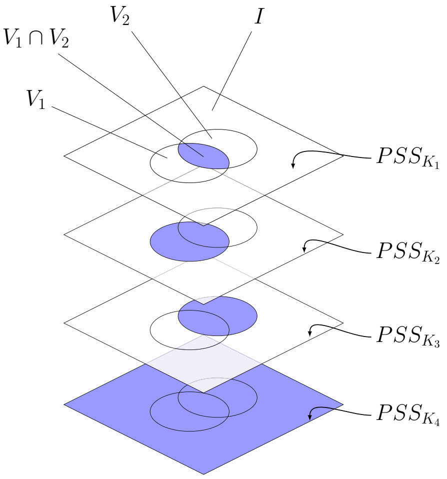

Figure 1: Set of padded variables represented as white area, set of modeled variables represented as colored area for each of the four PSSs.

To demonstrate the basic principle, consider an allocation task in which our capital must be allocated over five assets a_1, a_2, a_3, a_4 and a_5. The first constraint c_1 requires to allocate at least 30% weight to the group of variables with index $V_1 = \{1, 3\}$. The second constraint c_2 requires to allocate at least 50% weight to the group of variables with index $V_2 = \{2, 4\}$. Thus, the set of feasible solutions, i.e. the action space, is defined by the polytope

$$P_0 = \Bigg\{ a \in \mathbb{R}^5 : \sum_{i \in I} a_i = 1; \sum_{i \in V_1} a_i \geq 0.3; \sum_{i \in V_2} x_i \geq 0.5; \quad x_i \geq 0 \quad \forall i \in I = \{1, 2, 3, 4, 5\} \Bigg\}$$

Given the four PSS_{K_j} with the respective index sets of $K_1 = \emptyset$, $K_2 = V_1$, $K_3 = V_2$, $K_4 = I$ and the weighting vector $z = [z_1, z_2, z_3, z_4] = [0.0, 0.3, 0.5, 0.2]$. The corresponding weighted Minkowski sum M will equal P_0 as shown in the following: Any final allocation $a = [a_1, a_2, a_3, a_4, a_5] \in M$ is the vector sum of four vectors $\tilde{a}_j = [\tilde{a}_{1,j}, \tilde{a}_{2,j}, \tilde{a}_{3,j}, \tilde{a}_{4,j}, \tilde{a}_{5,j}] \in (PSS_{K_j})_{z_j} \subset \mathbb{R}^5$ for $j \in \{1, 2, 3, 4\}$, i.e. $a = \tilde{a}_1 + \tilde{a}_2 + \tilde{a}_3 + \tilde{a}_4$. The first sub-weighting vector $\tilde{a}_1$ will be $(0, 0, 0, 0, 0)$ due to $K_1 = \emptyset$ as we cannot invest into any assets. Any sub-weighting vector $\tilde{a}_2 \in (PSS_{K_2})_{z_2}$ will allocate a total of $z_2 = 0.3$ weight to the variables with an index in V_1, ensuring that any $a \in M$ will always satisfy the *lower bound* of the first constraint $c_1 = 0.3$. Any vector $\tilde{a}_3 \in (PSS_{K_3})_{z_3}$ will allocate a total of $z_3 = 0.5$ weight to the variables with an index in V_2, ensuring that any $a \in M$ will always satisfy the *lower bound* of the second constraint $c_2 = 0.5$. Any vector $\tilde{a}_4 \in (PSS_{K_4})_{z_4}$ will allocate the remainder of $z_4 = 0.2$ weight to any combination of variables in I, (a) ensuring that $\sum_{i \in I} a_i = 1$ and (b) potentially allocating additional weight to the variables with indices in V_1 and V_2 (since $V_1 \subseteq I$ and $V_2 \subseteq I$), allowing $a \in M$ to exceed the lower bounds of c_1 and c_2, i.e. $\sum_{i \in V_1} a_i \geq 0.3$ and $\sum_{i \in V_2} a_i \geq 0.5$. As a result, the sets M and P_0 have an identical H-representation (see Definition 3), i.e. being specified by the identical sets of constraints, making them identical polytopes.

In the following, we will introduce a weighting scheme selecting z which guarantees general equivalence between M and P_0. First, we formalize our constrained action space A_{2C} as convex polytope and introduce its H-representation.

Definition 3. *A convex polytope P in $\mathbb{R}^n$ is defined as a polytope that additionally is also a convex set. P can be viewed as the set of solutions to a system of linear inequalities, i.e., the intersection of a finite number of closed half-spaces, called P's half-space representation (H-representation):*

$$\begin{array}{ccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1n}x_n & \geq & b_1 \\
a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2n}x_n & \geq & b_2 \\
\vdots & & \vdots & & \vdots & & \vdots \\
a_{m1}x_1 & + & a_{m2}x_2 & + \cdots + & a_{mn}x_n & \geq & b_m
\end{array}$$

The general formulation on how to identify the four PSS_{K_j} with their respective index sets K_j and the definition of the weighting vector z can be found in Theorem 1. Let us note that we define the weighting vector z as an autoregressive function with $z = [z_1, z_2(z_1), z_3(z_1, z_2, y_2), z_4(z_1, z_2, y_2, z_3)]$ which results in an adaptive weighting vector depending on each current combination of elements in PSS_{K_j} for all $j = \{1, 2, 3, 4\}$.

Theorem 1. *Any polytope P defined by the system*

$$\sum_{i \in I} x_i = 1; \quad x_i \geq 0 \quad \forall i \in I; \quad \sum_{i \in V_1} x_i \geq c_1; \quad \sum_{i \in V_2} x_i \geq c_2$$

with $I = \{0, \ldots, N-1\}$, $V_1 \subseteq I$ and $V_2 \subseteq I$ can be expressed as a weighted Minkowski sum with the four weighted Minkowski summands with $y_{i,j} = [y_{0,j}, \ldots, y_{N-1,j}] \in (PSS_{K_j})_{z_j}$:

$$\begin{aligned}
(PSS_{K_1})_{z_1} &: K_1 = V_1 \cap V_2 \textit{ and } z_1 = \max(0, c_1 + c_2 - 1) \\
(PSS_{K_2})_{z_2} &: K_2 = V_1 \textit{ and } z_2 = \max(0, c_1 - z_1) \\
(PSS_{K_3})_{z_3} &: K_3 = V_2 \textit{ and } z_3 = \max(0, c_2 - z_1 - z_{2,\cap}) \\
&\qquad \textit{where } z_{2,\cap} = \sum_{i \in V_1 \cap V_2} y_{i,2} \\
(PSS_{K_4})_{z_4} &: K_4 = I \textit{ and } z_4 = 1 - z_1 - z_2 - z_3
\end{aligned}$$

Proof. Showing that two convex polytopes have an equivalent H-representation, i.e. an equivalent system of linear inequalities describing them, proves that they are identical.

When calculating the weighted Minkowski sum M of the four PSSs, we can deduce that for any element $x = (y_1 + y_2 + y_3 + y_4) \in M$ with $y_1 \in (PSS_{K_1})_{z_1}$, $y_2 \in (PSS_{K_2})_{z_2}$, $y_3 \in (PSS_{K_3})_{z_3}$ and $y_4 \in (PSS_{K_4})_{z_4}$ the following constraints are fulfilled:

The contribution to the variables $\sum_{V_1} x_i$ by $(PSS_{K_1})_{z_1}$ and $(PSS_{K_2})_{z_2}$ will always be $\max(0, c_1 + c_2 - 1) + \max(0, c_1 - z_1) = c_1$ while $(PSS_{K_3})_{z_3}$ and $(PSS_{K_4})_{z_4}$ can optionally contribute positive weight, resulting in

$$\begin{aligned}
\sum_{i \in V_1} x_i = & \underbrace{\sum_{i \in V_1 \cap K_1} y_{i,1}}_{= z_1 = \max(0, c_1 + c_2 - 1)} + \underbrace{\sum_{i \in V_1 \cap K_2} y_{i,2}}_{= z_2 = \max(0, c_1 - z_1)} + \underbrace{\sum_{i \in V_1 \cap K_3} y_{i,3}}_{\geq 0} \\
& + \underbrace{\sum_{i \in V_1 \cap K_4} y_{i,4}}_{\geq 0} \geq c_1.
\end{aligned}$$

The contribution to the variables $\sum_{V_2} x_i$ by $(PSS_{K_1})_{z_1}$, $(PSS_{K_2})_{z_2}$, $(PSS_{K_3})_{z_3}$ will always be $z_1 + z_{2,\cap} + \max(0, c_2 - z_1 - z_{2,\cap}) = c_2$ while $(PSS_{K_4})_{z_4}$ can optionally contribute positive weight, resulting in

$$\begin{aligned}
\sum_{i \in V_2} x_i = & \underbrace{\sum_{i \in V_2 \cap K_1} y_{i,1}}_{= z_1} + \underbrace{\sum_{i \in V_2 \cap K_2} y_{i,2}}_{= z_{2,\cap}} + \underbrace{\sum_{i \in V_2 \cap K_3} y_{i,3}}_{= \max(0, c_2 - z_1 - z_{2,\cap})} \\
& + \underbrace{\sum_{i \in V_1 \cap K_4} y_{i,4}}_{\geq 0} \geq c_2.
\end{aligned}$$

The total weight contribution from all four $(PSS_{K_j})_{z_j}$ for $j = \{1, 2, 3, 4\}$ to all variables $\sum_I x_i$ will be always $z_1 + z_2 + z_3 + (1 - z_1 - z_2 - z_3) = 1$, i.e.

$$\begin{aligned}
\sum_{i \in I} x_i = & \underbrace{\sum_{i \in I \cap K_1} y_{i,1}}_{= z_1} + \underbrace{\sum_{i \in I \cap K_2} y_{i,2}}_{= z_2} + \underbrace{\sum_{i \in I \cap K_3} y_{i,3}}_{= z_3} \\
& + \underbrace{\sum_{i \in I \cap K_4} y_{i,4}}_{= z_4 = 1 - z_1 - z_2 - z_3} = 1.
\end{aligned}$$

Additionally, we check the constraints for the single variables x_i for $i \in I$. Since for $y_j = [y_{0,j}, \ldots, y_{N-1,j}]$ with $j = \{1, 2, 3, 4\}$ all single variables $y_{i,j}$ with $i \in I$ are defined to be greater equal than zero, it follows that

$$x_i = \underbrace{y_{i,1}}_{\geq 0} + \underbrace{y_{i,2}}_{\geq 0} + \underbrace{y_{i,3}}_{\geq 0} + \underbrace{y_{i,4}}_{\geq 0} \geq 0 \quad \forall i \in I$$

which shows the equivalence of the two sets of convex closed half-spaces defining P and M, which proofs that $P = M$. □

4.2 Task optimization via Reinforcement Learning

After describing the simplex decomposition of the action space, we will now define a stochastic policy function based on our novel decomposition which can be used for policy optimization with policy gradient based RL approaches.

We optimize the policy on a surrogate action space $\tilde{\mathcal{A}} = \tilde{\mathcal{A}}_1 \times \tilde{\mathcal{A}}_2 \times \tilde{\mathcal{A}}_3 \times \tilde{\mathcal{A}}_4$, which is the Cartesian product of the four sub-action spaces. These sub-action spaces correspond to the four PSS_{K_j} when decomposing $\mathcal{A}_{2C}$ as introduced in the previous section. An action $\tilde{a} \in \tilde{\mathcal{A}}$ can be mapped into the original action space by using the weighted asset-wise sum over the sub-action spaces: $a = z_1 \cdot \tilde{a}_1 + z_2 \cdot \tilde{a}_2 + z_3 \cdot \tilde{a}_3 + z_4 \cdot \tilde{a}_4$. Note that each surrogate action $\tilde{a}$ maps to one particular action a in the original action space $\mathcal{A}$ but not vice versa. In other words, the mapping is surjective but not bijective. Based on this property, we can show that any optimal policy on the surrogate action set $\tilde{\mathcal{A}}$ is optimal on the original action space $\mathcal{A}$ as well.

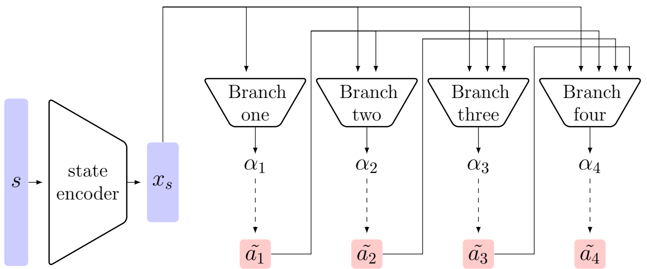

Figure 2: Auto-regressive architecture. The dashed arrows represent the process of sampling from a Dirichlet distribution to generate a sub-action $\tilde{a}_j$.

Theorem 2. *Given the constraint allocation task $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ as defined in Section 3 with the surrogate action space $\tilde{\mathcal{A}}$ corresponding to the action decomposition defined in Theorem 1. Any optimal policy over the surrogate action space $\tilde{\mathcal{A}}$, $\pi^*_{\tilde{\mathcal{A}}}$ with $Q(s, \pi^*_{\tilde{\mathcal{A}}}(s)) \geq Q(s, \pi_{\tilde{\mathcal{A}}}(s))$ for any state $s \in \mathcal{S}$ and any policy $\pi_{\tilde{\mathcal{A}}}(s)$, implies an optimal policy $\pi^*_{\mathcal{A}}$ on the original action space $\mathcal{A}$.*

Proof. We know from Theorem 1 that there exist weights z which allow to represent any $a \in \mathcal{A}$ by at least one surrogate action $\tilde{a}$. In addition, we know that for any state $s \in \mathcal{S}$ and any action $a \in \mathcal{A}$, the state-action value function $Q(s, a)$ is the same for any $\tilde{a}$ mapping to a as the reward received for performing $\tilde{a}$ is provided by the environment via performing the joint action a.

Thus, for any optimal policy $\pi^*_{\tilde{\mathcal{A}}}$, there exists a corresponding policy $\pi^*_{\mathcal{A}}$ providing the same Q-values. Now assume that $\pi^*_{\mathcal{A}}$ is not optimal and thus, there would be a policy $\hat{\pi}_{\mathcal{A}}$ with $Q(s, \hat{\pi}_{\mathcal{A}}(s)) > Q(s, \pi^*_{\mathcal{A}}(s))$ for at least one state $s \in \mathcal{S}$. Since the mapping between the $\tilde{\mathcal{A}}$ and $\mathcal{A}$ is surjective, there must exist a decomposition of $\hat{\pi}_{\mathcal{A}}(s)$ to the surrogate action space yielding higher Q-values than $\pi^*_{\tilde{\mathcal{A}}}(s)$ which contradicts the optimality of $\pi^*_{\tilde{\mathcal{A}}}$. □

Theorem 2 shows that any well-performing policy in $\tilde{\mathcal{A}}$ can be mapped to an equally well-performing policy over $\mathcal{A}$.

After introducing the surrogate action space, we will introduce our stochastic policy function over $\tilde{\mathcal{A}}$. We model each sub-action space with a Dirichlet distribution with a parameter vector α_{K_j}, which is obtained by a neural network. Our policy function that is based on the decomposition described in Theorem 1, requires an iterative computation of the weighting vector z. Our algorithm for sampling an action is detailed in Algorithm 1. In each step, we sample the asset allocation for PSS_{K_j} by the corresponding Dirichlet distribution Dir_j and then determine the corresponding weight z_j. In the end, the weighted asset-wise sum is computed and returned as joint action which is applied to the environment.

An overview of our architecture is depicted in Figure 2. We first create a representation x_s of the observation s using a transformer model. Each sub-action space is parameterized by an MLP using the representation x_s as well as all sampled surrogate actions from the previous sub-action spaces. This structure allows a tractable computation and optimization of the joint surrogate action $\tilde{a}$ probability: $P(\tilde{a}_1, \tilde{a}_2, \tilde{a}_3, \tilde{a}_4|x_s) = P(\tilde{a}_1|x_s) \cdot P(\tilde{a}_2|\tilde{a}_1, x_s) \cdot P(\tilde{a}_3|\tilde{a}_2, \tilde{a}_1, x_s) \cdot P(\tilde{a}_4|\tilde{a}_3, \tilde{a}_2, \tilde{a}_1, x_s)$.

We employ a policy gradient approach based on the PPO algorithm introduced by [16]. Note that our method can also be used with other policy gradient based RL methods.

The state encoder is composed of three fully connected layers of size 512, 256, and 128 with ReLU activation functions that feed into a GTrXL element, allowing also to handle tasks that require memory. GTrXL is based on [14] and is specifically designed to utilize transformers in an RL setting. The GTrXL element is composed of a single *transformer unit* with a single encoder layer and a single decoder layer with four attention heads and an embedding size of 64. The different branching modules are all made up of two fully connected layers of size 64 and 32, respectively, with a ReLU activation function after the first layer and a softplus activation function for the final output layer.

Algorithm 1 Action Generation using the Simplex Decomposition

```
Input: Index set of all $N$ assets in the investable universe $I = \{0, 1, ..., N-1\}$; Two allocation constraints $C_1 : \sum_{i \in V_1} x_i \geq c_1$ and $C_2 : \sum_{i \in V_2} x_i \geq c_2$
Define:
$K_1 = V_1 \cap V_2$, $K_2 = V_1$, $K_3 = V_2$ and $K_4 = I$, $f_j$ is an autoregressive policy network branch for $j = \{1, 2, 3, 4\}$
Begin action generation:
calculate $\alpha_1$ from $f_1(x_s)$, sample $\tilde{a}_1$ from $Dir(\alpha_1)$
set $z_1 = \max(0, c_1 + c_2 - 1)$
set $q_{V_1 \cap V_2} := z_1 \sum_{i \in V_1 \cap V_2} x_{i,1}$
calculate $\alpha_2$ from $f_2(x_s, \tilde{a}_1)$, sample $\tilde{a}_2$ from $Dir(\alpha_2)$
set $z_2 = \max[0, c_1 - z_1]$
update $q_{V_1 \cap V_2} := q_{V_1 \cap V_2} + z_2 \sum_{i \in V_1 \cap V_2} x_{i,2}$
calculate $\alpha_3$ from $f_3(x_s, \tilde{a}_1, \tilde{a}_2)$, sample $\tilde{a}_3$ from $Dir(\alpha_3)$
set $z_3 = \max[0, c_2 - q_{V_1 \cap V_2}]$
calculate $\alpha_4$ from $f_4(x_s, \tilde{a}_1, \tilde{a}_2, \tilde{a}_3)$, sample $\tilde{a}_4$ from $Dir(\alpha_4)$
set $z_4 = 1 - z_1 - z_2 - z_3$
calculate action $a$ by adding the weighted sub-actions:
    $z_1 \cdot \tilde{a}_1 + z_2 \cdot \tilde{a}_2 + z_3 \cdot \tilde{a}_3 + z_4 \cdot \tilde{a}_4 = a$
```

5 Experiments

5.1 Constrained Portfolio Optimization Tasks

We evaluate our approach in the financial setting on various constrained portfolio optimization tasks. The environment is based on [20] and uses real-world data from the Nasdaq-100 index that has been processed by the qlib package.[1] The data is used to estimate the parameters of a hidden Markov model (HMM), which is then used to generate trajectories. The monthly closing stock prices from January 1, 2010 to December 31, 2020 are included in the data set. An additional data set containing monthly closing prices from January 1, 2021 to December 31, 2021 is exclusively used to backtest the approaches. The environment's investment universe consists of 12 assets plus the special asset cash. Cash has neither a positive or negative return and remains stable over time. The remaining 12 assets are chosen at random from a pool of 35 pre-selected assets from the Nasdaq-100 data set. The assets were pre-selected based on the fact that they have been a member of the index at least since January 1, 2010, and there were no missing data entries.

In the following, we provide a detailed description of the portfolio optimization task. An agent is required to invest his wealth into N different assets based on asset allocation decisions made at each time step of the investment horizon T. The *constrained* **action space** for this task is described by $\mathcal{A}_{2C}$ as defined in Section 3, where the sets V_1 and V_2 contain the indices of assets affected by the respective constraint. These allocation constraints in the financial setting can be motivated by various factors, such as the need to invest *at least* a

[1] https://github.com/microsoft/qlib/tree/main

certain percentage of the portfolio into a group of assets in a specific sector or with a certain risk classification.

The **observation space** is defined as $\mathcal{O} = \mathcal{W} \times \mathcal{V} \times \mathcal{U}$. The set $\mathcal{W} \subseteq \mathbb{R}$ represents the agent's current absolute level of wealth, the set $\mathcal{V} \subseteq \mathbb{R}^N$ describes the current portfolio allocation, and the set $\mathcal{U} \subseteq \mathbb{R}^N$ is the observed single asset economic returns from the previous time step.

The total portfolio return $r = \vartheta_{PF} - tc$ is the agent's **reward** for each time step and is defined as the realized portfolio's return ϑ_{PF} minus any transaction costs tc that occurred. The portfolio's return is a random variable $\Theta_{PF} = a^\intercal \Theta$ based on the random vector $\Theta = [\Theta_0, \dots, \Theta_{N-1}]$ representing the economic returns of the single assets and the deterministic vector a that represents the portfolio's allocation weights selected by the agent. The cumulative portfolio total return over the investment horizon of 12 months, i.e. $T = 12$ time steps, is defined as

$$\nu = \sum_{t=0}^{T-1} r_{t+1}$$

and in the following referred to as the *annualized total portfolio return*.

5.2 Experimental Setup

As previously stated, two evaluation environments are used: (1) the *simulation environment* and (2) the *backtesting environment*. We conduct a total of 100 experiments, each with a unique constraint configuration, i.e., a unique combination of two allocation constraints. Each constraint configuration is evaluated on both evaluation environments with the goal of comparing the performance of the approaches for different constraint configurations (a) on the environment the agents were originally trained on and (b) on *unseen, real world* data.

Each experiment uses a different random seed as well as a randomly generated constraint configuration. A constraint configuration is made up of two allocation constraints C_j of the form $\sum_{i \in V_j} a_i \geq c_j$ with $j \in \{1, 2\}$ where V_j represents the set of affected assets and c_j the constraint's threshold value. To generate both allocation constraints at random, we sample the number of affected assets between 1 and 12. We rule out the possibility of selecting 0 or 13 assets since any greater equal allocation constraint would be either infeasible or trivial. The sampled number defines the number of specific assets which are then sampled from the list of 13 assets, i.e. the investment universe, without replacement resulting in V_j. Subsequently c_j is sampled from a uniform distribution in the interval $[0, 1]$. The process is repeated for the second allocation constraint as well resulting in a randomly generated constraint configuration. In a final step it is verified that the resulting polytope P as defined in Section 3 is not an empty set, i.e. a system that does not have a feasible solution.[2]

We compare our CAOSD approach to four other approaches, one of which is a naive random approach and three of which are state-of-the-art CRL approaches. The CRL approaches typically model constraint violations on a trajectory level, which means that they constrain the expected discounted sum of costs that occurred in each time step [3]. However, they can be adjusted to model allocation constraints that must be satisfied at each time step. This can be done by defining the costs at each time step in such a way that they return a positive value if an allocation constraint is violated and zero otherwise. When a violation occurs in any time step, the discounted sum of costs will be greater than zero. Therefore, we can constrain every time step in the trajectory implicitly by imposing a constraint on the trajectory level that the expected discounted sum of costs needs to be less than or equal to zero.

The first CRL benchmark approach is the Lagrangian-based RCPO introduced by [18]. The second benchmark approach is IPO proposed by [12] that uses an interior-point method to optimize the policy. The third approach is P3O by [21], a first-order optimization approach that uses an unconstrained objective in combination with a penalty term equaling the original constraint objective. The benchmark approaches are implemented in the RLlib framework based on their papers.[3] The code for all approaches is made publicly available.[4] All agents were trained on a cluster using various types of commercially available single GPUs. All approaches were extensively tuned in terms of hyperparameters using a grid search. Additional information on the hyperparameter tuning process can be found in the Appendix. During evaluation, RL agents take the action with the highest likelihood.

In addition to the three benchmark approaches we also utilize a random approach. This approach uniformly draws actions, i.e. asset allocations, from the constrained polytope. Efficient uniform sampling from a polytope is a surprisingly complex task, therefore we follow [8] to obtain uniform samples from the constrained action space. Using several rollouts of this baselines allows us to establish an estimate of the difficulty of an experiment, since the possible returns are highly dependent on the allocation constraints.

5.3 Evaluation

In our evaluation, we first compare the performance of our approach and the benchmarks over the entire set of experiments, which demonstrates the effectiveness of our approach in various settings. Afterward, we discuss the performance and convergence during training and take a detailed look at a single experiment. A key metric of the evaluation is the agents' mean annualized total portfolio returns. We define the mean annualized total portfolio return for each of the five approaches, i.e. $app = \{RCPO, IPO, P3O, CAOSD, RDM\}$, and each of the environments, i.e. $env = \{sim, bt\}$, as $\bar{\nu}_{app}^{env} = \frac{1}{J} \sum_{j=0}^{J-1} \nu_{app,j}^{env}$ where J is the number of evaluation trajectories. For the simulation environment $J = 1000$ trajectories per approach are evaluated after the agents' training is completed.

In backtesting – with the exception of the random approach – only the single real-world trajectory is evaluated to measure the agents' performance since their evaluation is deterministic. The random approach is treated differently since its evaluation remains stochastic due to its previously mentioned design to always sample uniformly an action from P. To reduce the variance in the results, we evaluate $\bar{\nu}_{RDM}^{bt}$ on $J = 1000$ rollouts during backtesting.

We use two measures to evaluate the performance of the approaches over all experiments. The first measure, $\bar{\theta}_{app}^{env}$, is the average of the mean annualized return of each approach over all experiments. More formally, $\bar{\theta}_{app}^{env} = \frac{1}{N} \sum_{i=0}^{N-1} \bar{\nu}_{app,i}^{env}$, where $N = 100$ is the number of experiments. Since the return that can be achieved in each experiments varies greatly depending on the constraint configuration, our second measure is defined as the difference of returns

[2] This can be checked by determining whether the V-representation of P contains at least one vertex.

[3] https://docs.ray.io/en/master/rllib/index.html

[4] https://github.com/DavWinkel/SimplexDecompositionECAI

	$\bar{\theta}^{env}_{app}$	Upper 95% CI	Lower 95% CI
Simulation			
RCPO	0.292	0.299	0.284
IPO	0.212	0.217	0.207
P3O	0.305	0.314	0.296
CAOSD	**0.327**	0.335	0.319
Random	0.209	0.217	0.201
Backtesting			
RCPO	0.497	0.521	0.474
IPO	0.35	0.365	0.334
P3O	0.522	0.552	0.492
CAOSD	**0.552**	0.582	0.522
Random	0.333	0.355	0.311

Table 1: Evaluation of $\bar{\theta}^{env}_{app}$ and its 95% confidence interval for all approaches in both environments for N=100 experiments after training is completed.

of each approach to the performance of the random baseline in the same experiment: $\bar{\delta}^{env}_{app} = \frac{1}{N}\sum_{i=0}^{N-1} \bar{\nu}^{env}_{app,i} - \bar{\nu}^{env}_{RDM,i}$.

	$\bar{\delta}^{env}_{app}$	Upper 95% CI	Lower 95% CI
Simulation			
RCPO	0.082	0.093	0.071
IPO	0.003	0.012	-0.007
P3O	0.096	0.108	0.084
CAOSD	**0.118**	0.129	0.106
Backtesting			
RCPO	0.164	0.196	0.132
IPO	0.017	0.046	-0.012
P3O	0.189	0.227	0.151
CAOSD	**0.219**	0.258	0.179

Table 2: Evaluation of $\bar{\delta}^{env}_{app}$ and its 95% confidence interval for the non-random approaches in both environments for N=100 experiments after training is completed.

Table 1 and Table 2 show the performance of the approaches for both metrics in both environments as well as their corresponding 95% confidence intervals. The CAOSD approach shows considerable improvements over the other approaches in both metrics and both environments. These improvements are statistically significant on a 95% confidence interval. The P3O approach ranks second in both environments for both metrics before RCPO. IPO is only able to outperform the random approach in the backtesting environment while producing similar performance results to the random approach in the simulation environment.

In the second part of the evaluation, we will discuss the performance of the agents *during training* on a representative experiment. The experiment has a constraint configuration with the two allocation constraints $C_1 : \sum_{i\in V_1} a_i \geq 0.23$ with V_1 containing the indices referring to the company stocks [BIDU, QCOM] and $C_2 : \sum_{i\in V_2} a_i \geq 0.32$ with V_2 referring to the indices of the companies [ADBE, SBUX, QCOM] (see Appendix for a detailed list of the environment's investment universe). During training, an evaluation with $J = 200$ trajectories is performed every 80000 environment steps.

Figure 3 shows the agents' mean annualized total portfolio return during training on the y-axis and the number of environment steps on the x-axis. The figure also shows the 95% confidence interval of the mean annualized total portfolio return for each of the approaches seen as the shaded areas around the lines. Note that we only show the *training performance* for the simulation environment, since there is no training in the backtesting environment. The CAOSD approach had the best training performance in the experiment shown in Figure 3, followed by P3O and RCPO. The IPO approach is not able to improve the mean annualized total portfolio return during training and stays comparable to the random approach.

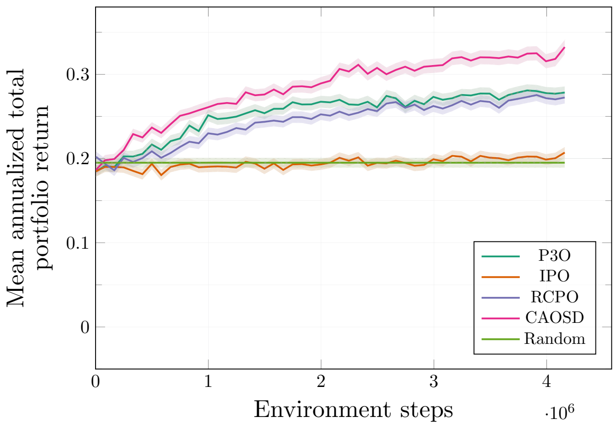

Figure 3: Mean annualized total portfolio return during training and its 95% confidence interval. This figure is best viewed in color.

6 Conclusion

In this paper, we examine portfolio optimization tasks with allocation constraints that require investing at least a certain portion of the available capital into a subset of assets. The task covers many real-world use-cases such as investors wanting to limit their exposure to certain groups of assets due to risk concerns, or investors who want to reflect aspects such as sustainability or social responsibility in their portfolio allocation. We examine settings that consider two allocation constraints and present CAOSD which decomposes the constrained action space into multiple unconstrained sub-action spaces. We show that the weighted Minkowski sum of these sub-action spaces is equivalent to the original action space if weights are chosen properly. Based on the decomposition we introduce a stochastic policy function that computes proper weights with an auto-regressive pattern. To optimize the policy for a given task, we apply a transformer-based state encoder and employ PPO [16] to train our agent. In the experimental part, we apply our approach to a variety of constrained portfolio optimization tasks, each characterized by a different set of constraints. We significantly outperform state-of-the-art approaches from CRL on real-world market data which demonstrates the effectiveness of our proposed method.

While this work shows decomposition with up to two allocation constraints, we will investigate decompositions for a greater number of constraints in future work. This increases the complexity of the possible relationship structures between the sets of choices, necessitating increasingly complex decompositions. In another line of future work, we want to examine the application of our approach to other tasks than portfolio optimization.

References

[1] Carlo Abrate, Alessio Angius, Gianmarco De Francisci Morales, Stefano Cozzini, Francesca Iadanza, Laura Li Puma, Simone Pavanelli,

Alan Perotti, Stefano Pignataro, and Silvia Ronchiadin, 'Continuous-action reinforcement learning for portfolio allocation of a life insurance company', in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 237–252. Springer, (2021).

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel, 'Constrained policy optimization', in *International conference on machine learning*, pp. 22–31. PMLR, (2017).

[3] Eitan Altman, *Constrained Markov decision processes: stochastic modeling*, Routledge, 1999.

[4] Eric André and Guillaume Coqueret, 'Dirichlet policies for reinforced factor portfolios', *arXiv preprint arXiv:2011.05381*, (2020).

[5] Shalabh Bhatnagar and K Lakshmanan, 'An online actor–critic algorithm with function approximation for constrained markov decision processes', *Journal of Optimization Theory and Applications*, **153**(3), 688–708, (2012).

[6] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh, 'A lyapunov-based approach to safe reinforcement learning', *Advances in neural information processing systems*, **31**, (2018).

[7] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh, 'Lyapunov-based safe policy optimization for continuous control', *arXiv preprint arXiv:1901.10031*, (2019).

[8] Mario Vazquez Corte and Luis V Montiel, 'Novel matrix hit and run for sampling polytopes and its gpu implementation', *arXiv preprint arXiv:2104.07097*, (2021).

[9] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa, 'Safe exploration in continuous action spaces', *arXiv preprint arXiv:1801.08757*, (2018).

[10] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine, 'Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates', in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396. IEEE, (2017).

[11] Chen Hou and Qianchuan Zhao, 'Optimization of web service-based control system for balance between network traffic and delay', *IEEE Transactions on Automation Science and Engineering*, **15**(3), 1152–1162, (2017).

[12] Yongshuai Liu, Jiaxin Ding, and Xin Liu, 'Ipo: Interior-point policy optimization under constraints', in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4940–4947, (2020).

[13] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson, 'Discrete sequential prediction of continuous actions for deep rl', *arXiv preprint arXiv:1705.05035*, (2017).

[14] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al., 'Stabilizing transformers for reinforcement learning', in *International conference on machine learning*, pp. 7487–7498. PMLR, (2020).

[15] Thomas Pierrot, Valentin Macé, Jean-Baptiste Sevestre, Louis Monier, Alexandre Laterre, Nicolas Perrin, Karim Beguir, and Olivier Sigaud. Factored action spaces in deep reinforcement learning, 2021.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, 'Proximal policy optimization algorithms', *arXiv preprint arXiv:1707.06347*, (2017).

[17] Arash Tavakoli, Fabio Pardo, and Petar Kormushev, 'Action branching architectures for deep reinforcement learning', in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, (2018).

[18] Chen Tessler, Daniel J Mankowitz, and Shie Mannor, 'Reward constrained policy optimization', in *International Conference on Learning Representations*, (2018).

[19] Ermo Wei, Drew Wicke, and Sean Luke, 'Hierarchical approaches for reinforcement learning in parameterized action space', in *2018 AAAI Spring Symposium Series*, (2018).

[20] David Winkel, Niklas Strauß, Matthias Schubert, and Thomas Seidl, 'Risk-aware reinforcement learning for multi-period portfolio selection', in *European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, (2022).

[21] Linrui Zhang, Li Shen, Long Yang, Shixiang Chen, Xueqian Wang, Bo Yuan, and Dacheng Tao, 'Penalized proximal policy optimization for safe reinforcement learning', in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3744–3750, (2022).