FinRDDL: Can AI Planning be used for Quantitative Finance Problems?

Sunandita Patra[1], **Mahmoud Mahfouz**[1, 2], **Sriram Gopalakrishnan**[1], **Daniele Magazzeni**[1], **and Manuela Veloso**[1]

[1] J.P. Morgan AI Research
[2] Imperial College London
{sunandita.patra, mahmoud.a.mahfouz, sriram.gopalakrishnan, daniele.magazzeni, manuela.veloso}@jpmchase.com

Abstract

In this paper, we introduce FinRDDL, the first framework for financial portfolio management problems using a planning language. FinRDDL uses Relational Dynamic Influence Diagram Language (RDDL) to formalize the problems of asset allocation and optimal trade execution; two canonical quantitative finance problems. We show how the seminal works of Markowitz (1952) for asset allocation and Almgren and Chriss (2001) for optimal trade execution can be modelled using RDDL and provide preliminary results on how the performance of various AI actors with different planning strategies compare against existing methods. Finally, we highlight the limitations of existing planning algorithms for solving financial sequential decision making problems and discuss future research avenues.

1 Introduction

Sequential decision-making problems in finance have attracted significant academic interest from both the quantitative finance and artificial intelligence (AI) communities. Two canonical problems, in particular, have been the focus of research: asset allocation and optimal trade execution. Both problems fall within the context of financial portfolio management whereby asset allocation deals with the problem of deciding the weights of the financial assets to include in a portfolio and optimal trade execution deals with efficiently placing the corresponding trades in the market.

The primary goal of this paper is to investigate whether it is possible to formalize asset allocation and optimal trade execution problems in a planning formalism and utilize AI planning techniques to solve them. By examining the benefits and limitations of using AI planning methods, we aim to provide insights into whether this approach can offer an alternative to current approaches used in quantitative finance.

There are several challenges with using planning in this field, starting with the fact that the state and action space are often continuous. Finding optimal plans in such numerical planning domains is very hard; Helmert (2002) has shown that such problems are often not event decidable (i.e. it is not known if a solution exists). Dealing with a continuous space of possible states and actions can be difficult and computationally expensive to work with. As a result, practical approaches with continuous action spaces often require approximating the optimal policy using techniques such as function approximation, gradient-based optimization, or reinforcement learning.

Another significant challenge for using planning methodologies is that the underlying environment the plan is executed on is highly dynamic and affected by exogenous events. This dynamic nature of the environment can make it difficult to anticipate the long-term consequences of actions. One way the field of quantitative finance handles this issue is by using different statistical models to model and predict the variables of interest such as the asset price evolution (Hu and Øksendal 1998; Hachicha, Jarboui, and Siarry 2011).

Based on these challenges, we believe RDDL (Sanner et al. 2010) is an appropriate planning language for modelling financial trading problems because (a) it can capture a continuous state and action space, (b) it allows concurrent actions with possibly conflicting effects, and, (c) various mathematical models for predicting asset prices can be integrated in RDDL's conditional probabilistic state transition functions.

Our Contributions

- We formalize the asset allocation and optimal trade execution problems using RDDL. To the best of our knowledge, this is the first application of RDDL to model these two canonical quantitative finance problems.
- We experimentally evaluate five different acting strategies: uniform allocation, Greedy Markowitz, Monte Carlo Tree Search, Mixed-integer quadratic programming, and tfplan (Wu, Say, and Sanner 2017) for asset allocation.
- We provide a desiderata for the AI planning community highlighting the limitations of current planning approaches and how they can be improved to solve quantitative finance problems.

The paper is organized as follows: we first provide the background and related work in section 2 on asset allocation, optimal trade execution and RDDL. We then describe the FinRDDL framework, the AI planning techniques explored and the experimental evaluation for asset allocation in section 3 and optimal trade execution in section 4. We provide a desiderata for the AI planning community in section 5 and conclude the paper in section 6 with a summary of our work and suggestions for future research.

2 Background & Related Work

This section provides an overview of the financial terminology necessary to understand the asset allocation and optimal trade execution problems. We examine the relevant literature pertaining to these two problems. Additionally, we provide an outline of the fundamental components of RDDL.

2.1 Asset Allocation

Financial portfolio management is a multi-faceted process that relies on human expertise and historical data analysis for constructing portfolios of financial assets on behalf of individual or institutional investors. A key component of portfolio management is asset allocation. This is the task of optimally allocating a finite cash budget to a finite set of financial assets to meet a certain financial objective dictated by the investor. The objectives vary depending on the investor's goals, investment horizon and risk appetite. A portfolio is a collection of multiple financial assets characterized by its constituents (N assets) and a portfolio weights vector $\boldsymbol{w}_t$ at time t which is defined as:

$$\boldsymbol{w}_t = [w_{1,t}, \ldots, w_{N,t}]^T \in \mathbb{R}^N \text{ and } \sum_{i=1}^{N} w_{i,t} = 1 \quad (2.1)$$

where, $w_{i,t}$ represents the ratio of the total budget invested in asset i at time t.

Investors are typically concerned with price changes over time as opposed to absolute prices since they reflect their investment's profit and loss. Let $p_{i,t}$ denote the price of asset i at time t. The gross return $R_{i,t}$, simple return $r_{i,t}$ and log return $\rho_{i,t}$ of an asset i at time t are given by:

$$R_{i,t} \triangleq \frac{p_{i,t}}{p_{i,t-1}} \in \mathbb{R} \quad (2.2)$$

$$r_{i,t} \triangleq \frac{p_{i,t} - p_{i,t-1}}{p_{i,t-1}} = \frac{p_{i,t}}{p_{i,t-1}} - 1 = R_{i,t} - 1 \in \mathbb{R} \quad (2.3)$$

$$\rho_{i,t} \triangleq \ln(R_{i,t}) = \ln\left(\frac{p_{i,t}}{p_{i,t-1}}\right) = \ln(r_{i,t} + 1) \in \mathbb{R} \quad (2.4)$$

For a portfolio with multiple assets, a linear combination of each asset return weighted by the portfolio weights vector yields the portfolio return. For example, the portfolio log return is defined as:

$$\rho_t \triangleq \sum_{i=1}^{N} w_{i,t}\rho_{i,t} = \boldsymbol{w}_t^T \boldsymbol{\rho}_t = \ln\big(1 + \boldsymbol{w}_t^T \boldsymbol{r}_t\big) \in \mathbb{R} \quad (2.5)$$

where, $\boldsymbol{\rho}_t = [\rho_{1,t}, \rho_{2,t}, \ldots, \rho_{N,t}]^T \in \mathbb{R}^N$. Translating this to investments over *multi-periods* with an investment horizon H, we obtain the cumulative log return $\rho_{t\to H}$, an example of an objective an investor would seek to maximize,

$$\rho_{t\to H} \triangleq \ln\left(\prod_{i=t+1}^{H} R_i\right) = \sum_{i=t+1}^{H} \ln\left(R_i\right) = \sum_{i=t+1}^{H} \rho_i \in \mathbb{R} \quad (2.6)$$

The first attempt to suggest an optimization approach for asset allocation in a systematic way was introduced by Markowitz (1952). The key insight of this approach is that by combining assets with different expected returns and volatility, one can decide on a mathematically optimal allocation. Following this seminal work, the problem of asset allocation was studied extensively in the literature with a wide range of approaches investigated. We focus on prior work on *multi-period* asset allocation, a sequential decision making problem, where the portfolio weights are adjusted periodically (e.g. monthly) to meet the investor's objectives. This is in contrast to *single-period* asset allocation where the optimal portfolio weights are decided once and the portfolio is maintained over a fixed time period (e.g. 1 year).

Boyd et al. (2017) presents a framework for single-period and multi-period optimization of asset allocation strategies, balancing expected return, risk, transaction and holding costs. They employ a convex optimization problem formulation, under the assumption that cost, risk, trading and holding functions and constraints are all convex. The method is used to plan a sequence of trades with future quantities estimated using predictions, without addressing the critical component of forecasting future quantities. Li, Uysal, and Mulvey (2022) describes a method for multi-period portfolio optimization that employs model predictive control with a risk-parity objective, and provide a successive convex programming algorithm that is more computationally efficient compared to previous methods. Their comprehensive comparison of models demonstrates that multi-period models outperform single-period models in out-of-sample periods with market impact costs, achieving higher Sharpe ratios for mean-variance and risk-parity formulations, respectively. Lastly, Blay et al. (2020) take an analogous approach to ours and use a simulator to capture market dynamics and compute a set of weights (allocations) for multi-period asset allocation. The weights are computed using gradient based optimization procedure, and their method computes a fixed allocation over all periods. This is like computing a static plan for a dynamic or probabilistic environment. Using RDDL, we can compute a policy that adapts it's allocation decisions (actions) based on the current state which includes prices, and current allocation amongst others.

2.2 Optimal Trade Execution

The optimal trade execution (OTE) problem is another classical problem in quantitative finance that refers to the challenges of buying or selling securities in a manner that minimizes transaction costs and maximizes trading profits. Consider a portfolio manager (PM) who wishes to adjust (rebalance) his/her portfolio by liquidating (selling) a fixed large block of a given security. The PM would send his order to a broker who is tasked with executing it in the market (*e.g.* on an exchange like NASDAQ). A common practice is to slice-and-dice the PM's order into smaller *child orders* to be executed within a given time horizon. This then turns the OTE problem into a sequential decision making problem of deciding the optimal sequence of child orders to place in the market to minimize the transaction costs.

The OTE problem is characterized by having an objective function (minimizing execution costs), a set of possible actions to take (order types, prices and quantities to place in the market) and market variables to take into considera-

tion when making the decision at every step of the execution time horizon. The problem is multi-faceted as there are several challenges to consider. At the *macroscopic* level the PM needs to decide the time horizon and total quantity to execute. At the *mesoscopic* level, the broker faces the *order scheduling problem* (OSP) and has to decide how to slice the metaorder across time and determine the volume to execute for each slice. Finally, the broker is also faced with the *order placement problem* (OPP) at the *microscopic* level where he/she needs to decide which order type to use and which market venue to send the order to (Bouchaud et al. 2018). In this paper, we focus only on the *order scheduling problem*.

The order scheduling problem has been widely studied in the literature and was first formalized by Bertsimas and Lo (1998) in their work on the optimal control of execution costs. The authors formulate the problem mathematically as a sequential decision making problem, define the concept of *best execution* and use the *implementation shortfall* (Perold 1988) as the cost objective to be minimized. They then use stochastic dynamic programming to solve the problem and derive a number of closed-form solutions under certain assumptions around the price dynamics. An important contribution of the paper is showing that breaking up a large trade into a a number of smaller trades of equal size is optimal when the price dynamics follow an arithmetic random walk, the price impact is linear in the trade size and its effect is permanent on future prices. Almgren and Chriss (2001) extended this work using a mean-variance approach similar to that used in asset allocation to allow for the minimization of the expected implementation shortfall and the variance (volatility) of the expected execution costs. Similar to (Bertsimas and Lo 1998), they also derive closed-form solutions of the optimal schedules under a set of assumptions around the price dynamics and market impact and show that there exists an optimal solution balancing trade off between the costs of trade execution and the speed at which the trades are sent to the market. Following the seminal works of Bertsimas and Lo (1998) and Almgren and Chriss (2001), optimizing trade execution became an active research topic for decades with researchers investigating different forms of price dynamics, market impact and optimal trading strategies (Bouchaud et al. 2018). See Donnelly (2022) for a recent review on this topic. Given the importance of this problem, we present a formulation of this problem in RDDL in section 4 for the planning and RL research community, and discuss the pertinent dynamics therein.

2.3 Relational Dynamic Influence Diagram Language (RDDL)

RDDL (Sanner et al. 2010) is a formal language used to model dynamic environments for decision-making problems. It allows for uncertainty in the environment dynamics (probabilistic transitions), and is geared toward planning problems. Multi period asset allocation and order execution are classic examples of such problems, and RDDL can be used to model them.

To model a problem using RDDL, we need to define the state and action fluents which can change at every discrete time step. Action fluents represent the decision variables. Actions can occur concurrently. Constants are declared as non-fluents. Two key components of an RDDL domain are the transition function and rewards associated with each state-action pair. The transitions capture the dynamics and uncertainty in the market, such as the probability of different price movements, while the reward considers the expected return, risk of the portfolio and transaction costs. The reward function can be defined to incorporate different investment goals, such as maximizing return, minimizing risk, or optimizing the trade-off between the two. We can also incorporate various constraints, such as transaction costs, liquidity requirements, and risk tolerance levels. These can be represented as additional variables or as part of the reward function.

Finally, any RDDL solver that supports continuous state and action spaces can be used to generate an optimal policy based on the model defined in this work. In the case of asset allocation, the policy provides guidance on the optimal allocation of assets between the different classes, taking into account the current market conditions and investment goals.

We have chosen not to use PPDDL (Younes and Littman 2004) due to its lack of support for concurrent actions that may result in conflicting effects. Given the stochastic nature of our problem, there is always some probability that such conflicts may occur. For instance, changing the weight of two assets in our portfolio may result in related market impacts. Enumerating the joint actions and their effects would be a possible solution, but is very difficult to maintain over time. Additionally, multiple exogenous events that may occur in a single day cannot affect the same fluent or state variable, further limiting the applicability of PPDDL.

With respect to planning in RDDL, the Prost planner (Keller and Eyerich 2012) is a probabilistic planning system that utilizes the UCT algorithm and several enhancements to improve its performance in domain-independent probabilistic planning. It has demonstrated significant improvement in benchmark domains. However, Prost cannot handle continuous state and action spaces (intended for discrete domains).

3 FinRDDL: Asset Allocation

This section builds upon the mathematical terminology established in section 2.1 for the asset allocation problem. The main focus of this section is to present the primary objective (reward), the RDDL model, and a preliminary experimental evaluation to compare different planning strategies for multi-period asset allocation using RDDL.

Objective. For the multi-period asset allocation problem, the objective is to find the optimal set of portfolio weights $w_{i,t}$ for all assets i over the investment horizon. Formally,

$$\begin{aligned} \underset{\substack{\{w_{i,t}\}_{i=1,\dots,N} \\ t=1,\dots,H}}{\operatorname{argmax}} \sum_{t=1}^{H}\sum_{i=1}^{N} & \Big(w_{i,t}\rho_{i,t} - \alpha_\sigma w_{i,t}\sigma \\ & \qquad -\alpha_\delta \left\| w_{i,t} - w_{i,t-1} \right\|_1 \Big) \\ \text{s.t. } \textstyle\sum_{i=1}^{N} w_{i,t} = 1 \text{ and} \quad & w_{i,t} \geq 0, \forall t = 1, \dots, H \end{aligned} \tag{3.1}$$

where, H is the investment time horizon, N is the number of available assets and $w_{i,t}$ is the weight of asset i at time t.

The expression maximizes the sum of the expected returns and minimizes the risk and transaction costs of all assets over all periods. The first constraint ensures that the asset weights at each period sum up to 1 and the second constraint enforces non-negativity of the weights. The scaling constants α_σ and α_δ determine the respective weightings of the three optimization metrics - return, risk, and transaction costs.

In order to simulate the evolution of asset prices over time, practitioners often rely on calibrated geometric Brownian motion models (GBM) (Abidin and Jaffar 2012). The model assumes that the asset price follows a stochastic process, where the changes in the asset price are normally distributed and the drift and volatility are estimated properties. The resulting equation describes a random walk with drift, where the expected value of the asset price grows over time but with increasing variability. The equation for geometric Brownian motion is:

$$dp_t = \mu p_t dt + \sigma p_t dW_t, \tag{3.2}$$

where, p_t is the asset price at time t, μ is the drift, or expected rate of return per unit time, σ is the volatility, or standard deviation of the asset returns per unit time and dW_t is a Wiener process, representing the random noise or error term. This stochastic differential equation (SDE) describes the change in the asset price over an infinitesimal time period dt.

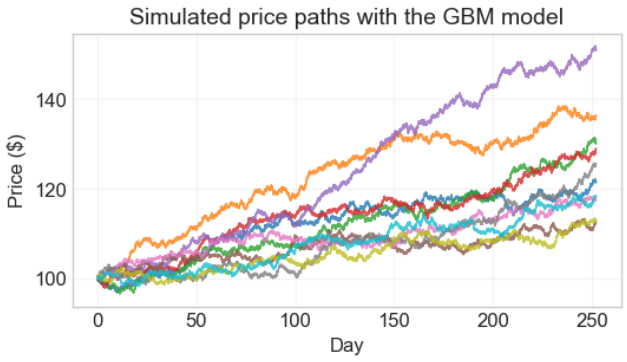

Figure 1: Different asset price paths following the Geometric Brownian Motion (GBM) model with a starting price of $100, drift ($\mu$) of 0.001 and volatility (σ) of 0.005.

3.1 RDDL Model

Following the discussion about RDDL from Section 2.3, we start by defining the objects, followed by the state and action fluents, state transition function, reward function and the constraints. Each asset is represented as an object, denoted by *asset* or $?a$ (the '?' symbol indicates a variable in RDDL). A domain instance can include any number of assets.

State variables: Each asset is associated with two state variables (fluents); one each for its current weight and price. In addition, there is a variable that tracks the current day (the timer or step index variable).

$$w(asset)(\text{real}) : \text{current weight of } asset \tag{3.3}$$
$$p(asset)(\text{real}) : \text{current price of } asset \tag{3.4}$$
$$\text{day(int)} : \text{current day} \tag{3.5}$$

Actions: An action-fluent *set_w*(*asset*) is defined for each asset, allowing to change the weight (allocation) of the corresponding asset. Notably, RDDL allows concurrent actions with potentially conflicting effects. This is particularly relevant when rebalancing the portfolio, as we may need to adjust the weights of multiple assets simultaneously.

State transition function: The weight of an asset $?a$ is updated as:

$$w'(?a) = \frac{set\_w(?a)}{\sum_{?b:asset} set\_w(?b)}, \quad \text{if } period \mid \text{day}, \tag{3.6}$$
$$= w(?a), \quad \text{otherwise}, \tag{3.7}$$

where $period$ is the reallocation period length in days. This ensures that asset weights are only updated at the reallocation points. A normalization step divides each asset's weight update by the total weight of all assets in the portfolio, ensuring that the asset weights sum up to 1, which is necessary for a valid portfolio.

The price of an asset $?a$ is updated following the Geometric Brownian motion as,

$$p'(?a) = p(?a).e^{\mu(?a)+\sigma(?a)\mathcal{N}(0,1)} \tag{3.8}$$

where, $\mu(?a)$ (drift) and $\sigma(?a)$ (volatility) are non-fluents set for each problem instance and $\mathcal{N}(0,1)$ samples from the standard Normal distribution. The current day simply progresses as,

$$\text{day}' = \text{day} + 1. \tag{3.9}$$

Reward. RDDL is limited to having only one objective function as the reward. Therefore, to incorporate multiple factors such as log return, portfolio risk, and transaction costs, we combine them into a single expression and scale them using hyperparameters. These hyperparameters, which are non-fluents, are denoted by α_σ for risk and α_δ for transaction costs. Specifically, the reward for a step is computed as the sum over all assets:

$$\text{Reward} = \sum_{?a:asset} w(?a)\{\ln \frac{p'(?a)}{p(?a)} - \alpha_\sigma \sigma(?a) \tag{3.10}$$
$$- \alpha_\delta \left| w'(?a) - w(?a) \right|\} \tag{3.11}$$

State Invariants. We ensure that the asset weights always stay in the range [0,1] with the constraint,

$$\forall_{?a:asset}(w(?a) \geq 0 \wedge w(?a) \leq 1)$$

The full RDDL domain descriptions and example instances are included in the appendix.

3.2 Planning Strategies

In this section, we discuss four potential planning algorithms to solve the RDDL formulation of the asset allocation problem.

Markowitz Mean Variance Optimization (MMVO) Mean-variance optimization (Markowitz 1952) is a classical approach for the single period asset allocation problem that maximize portfolio returns while minimizing portfolio risk. It uses estimates of the expected returns and covariance

of different assets and then creates a quadratic program with asset weights as variables to solve for.

For a universe of N assets with associated historical data, we first obtain the empirical estimates of the expected returns $\boldsymbol{\rho} = [\rho_1, \rho_2, \ldots, \rho_N] \in \mathbb{R}^N$ and the covariance matrix of the asset returns $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N}$, where, ρ_i is the sample mean log return of asset i and σ_{ij} is the covariance of assets i and j. We then compute the portfolio weights vector $\boldsymbol{w} \in \mathbb{R}^M$ as:

$$\begin{array}{ll} \max\limits_{\boldsymbol{w}} & \boldsymbol{w}^T\boldsymbol{\rho} \text{ or } \min_w \boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{s.t.} & \mathbf{1}_N^T\boldsymbol{w} = 1 \wedge \boldsymbol{w} \succcurlyeq 0 \end{array} \tag{3.12}$$

where, $\boldsymbol{w}^T\boldsymbol{\Sigma}\boldsymbol{w}$ represents the portfolio volatility (*risk*) and $\boldsymbol{w}^T\boldsymbol{\rho}$ is the portfolio expected return. Equation 3.12 is a quadratic program that can be solved numerically using gradient-based algorithms (Boyd, Boyd, and Vandenberghe 2004).

Solving for conflicting objectives gives rise to the *efficient frontier* which shows the set of optimal portfolios with the highest levels of return for a given level of risk and vice versa. In the context of multi-period asset allocation, this approach reduces to greedily maximizing the return (or minimizing risk) at each rebalancing point as it occurs.

Monte Carlo Tree Search (MCTS) MCTS (Kocsis and Szepesvári 2006) a well known algorithm for decision making in stochastic dynamic environments. The general idea is to incrementally build the search tree by simulating potential asset price evolution ($\boldsymbol{p}_t$) over time t via an RDDL simulator. Starting with an initial portfolio and an empty search tree, each iteration traverses the tree from the root node to a leaf node using the Upper Confidence Bound (UCB) (Kocsis and Szepesvári 2006) to select the most promising combination of assets weights $\boldsymbol{w}_t$ at rebalancing point t. The selected node is expanded by adding children representing changes in $\boldsymbol{w}_t$. Then, a rollout is simulated starting from the newly added node using randomly chosen changes in $\boldsymbol{w}_t$. The statistics of all visited nodes are updated based on the outcome of the simulated evolution of $\boldsymbol{p}_t$. The steps are repeated until $t = H$ (the investment horizon). The set of optimal weights for each period, $\{\boldsymbol{w}_t^*\}_{t=1}^H$ are selected based on the statistics accumulated during the search, typically by choosing $\{\boldsymbol{w}_t\}_{t=1}^H$ with the highest expected reward. We have currently discretized the action space in steps of 0.1 in our implementation to expand a node. (Mansley, Weinstein, and Littman 2011) and (Kim et al. 2020) are possible techniques one can adapt to do MCTS in continuous spaces.

Mixed Integer Quadratic Programming (MIQP) Boyd et al. (2017) multi-period asset allocation uses a MIQP convex optimization formulation used for optimizing investment portfolios over multiple time periods at once. The algorithm determines the optimal allocation of all assets across all reallocation periods that maximizes the total expected reward from Equation 3.1. To estimate returns and volatility, we utilized the RDDL simulator by playing it and obtaining relevant data. These estimates are fed into the objective expression.

tf-plan. Wu et al. (Wu, Say, and Sanner 2017) developed the tf-plan algorithm which does symbolic gradient optimization on RNNs to effectively plan in high-dimensional mixed discrete and continuous nonlinear domains. tf-plan uses Tensorflow with RMSProp optimization and is competitive with MILP-based optimization on piecewise linear planning domains and outperforms state-of-the-art interior point methods for nonlinear domains, demonstrating a new frontier for highly scalable planning in nonlinear hybrid domains. However, since it does not support ln (and other math functions) in its RDDL domain description, we use a modified step reward function from Equation 3.11 as:

$$\text{Reward} = \sum_{?a:\textit{asset}} w(?a)\{\frac{p'(?a)}{p(?a)} - \alpha_\sigma \sigma(?a) \tag{3.13}$$

$$- \alpha_\delta \left| w'(?a) - w(?a) \right|\} \tag{3.14}$$

3.3 Experimental Evaluation

With our RDDL model (section 3.1 and potential planning algorithms (section 3.2) we compare the performance of the following AI actors for asset allocation using distinct planning strategies:

- *Uniform Actor.* Always distributes the total wealth uniformly among the different assets.
- *Greedy Markowitz Actor.* At every rebalancing point, it ignores the transaction costs and strictly maximizes the reward for the current period (Markowitz 1952).
- *MCTS Actor.* Does Monte Carlo Tree Search with a discretized search space (Kocsis and Szepesvári 2006) to decide the optimal set of asset weights.
- *MIQP Actor.* Formulates and solves the reward maximization problem across multiple periods as a convex optimization problem (Boyd et al. 2017).
- *tf-plan Actor.* Follows the recommendation of the tf-plan algorithm (Wu, Say, and Sanner 2017).

While our RDDL model is general enough for N assets, we do our preliminary tests on a synthetic universe consisting of two assets (Table 1). Each asset follows a Geometric Brownian Motion (Figure 1) of prices with a specific return (mean) and risk (variance). We assume that the investment horizon H is known in advance and we set it to 300 days. The horizon consists of five re-balancing opportunities, at an interval of 60 days. By doing a set of exploratory experiments, we set the values of the two hyperparameters, $\alpha_\sigma = 0.1$ (for risk), and $\alpha_\delta = 0.001$ (for transaction costs). We ran our experiments on a c4.8xlarge Amazon EC2 instance with 36 vCPUs, 60 GiB RAM and a maximum timeout of 10 mins for plan computation. We use pyRDDLGym (Taitler et al. 2022) to create a gym (Brockman et al. 2016) environment from the RDDL file. We test the strategies of the AI actors on the gym environment to track the step rewards and the total cumulative rewards.

Table 1: Synthetic Asset Data Configuration for our experimental evaluation.

Asset	**Price at** $t = 0$	**Drift** μ	**Volatility** σ
Asset 1	10	0.001	0.010
Asset 2	50	0.001	0.005

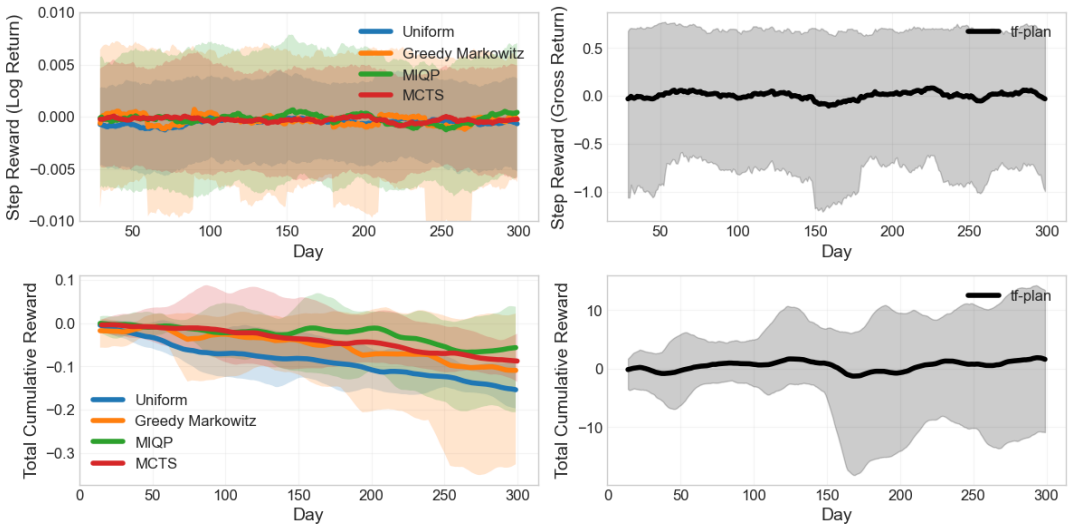

Figure 2: The top plots show the observed daily step reward for AI actors with different planning strategies. The bottom plot shows the total cumulative reward upto that day. Higher rewards indicate better performance. The plots on the right show the step and cumulative rewards for the tf-plan actor. tf-plan uses gross return instead of log returns in its reward function due its lack of support for the RDDL ln function.

To measure the performance of the different actors, we observe three parameters, (1) the step reward every day (we simply call it reward), (2) the total cumulative reward from the beginning until day t, and (3) how the weights of assets change with time.

Reward. The step reward for a specific day is determined by evaluating the reward function (Equation 3.11) for that particular day. In Figure 2 (top-left), the step rewards are displayed for all actors except for tf-plan. This is due to the fact that the return component of the reward function contains a natural logarithm function, which is not supported by the version of RDDL used by tf-plan. Consequently, we computed the step reward for the tf-plan actor (Figure 2 top-right) by using the gross return in the reward expression, which resulted in more spread out values compared to the other actors.

Total Cumulative Reward. To determine the overall performance of each actor, we computed the total cumulative reward, which is the sum of all step rewards accumulated up to that point. The comparison of the actors in terms of cumulative rewards earned is shown in Figure 2 (bottom). It is evident that the Greedy Markowitz actor experiences a significant dip at the rebalancing periods, as it completely disregards transaction costs. In contrast, all other actors outperform the uniform strategy. Notably, the MIQP actor exhibits the best performance overall. However, it is worth noting that a true comparison of the tf-plan actor with other actors can only be made using normalized reward functions. Figure 2 (bottom-right) displays the promising performance of the tf-plan actor, but normalization is required for fair comparison with the other actors.

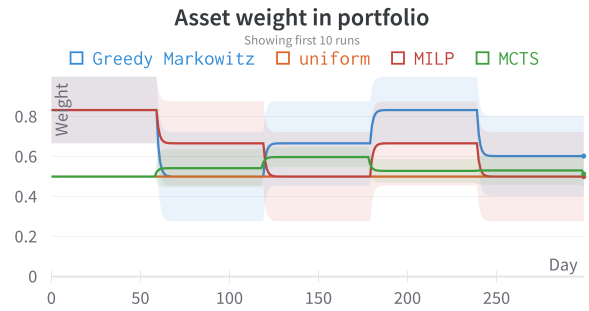

Figure 3: This plot shows how the weight (ratio of the total wealth) of a single portfolio asset changes with time. Please note that the transitions in weights are sharp jumps but they are smoothed out for improved readability.

Asset Weights In Figure 3, we observe how the weight of a specific portfolio asset changes over time. The plot compares the performance of different actors in terms of minimizing jumps in it's share of the portfolio over time. The solid lines show the average weights. The MCTS actor's strategy appears to result in more gradual changes in asset weight over time, which may lead to a more stable and consistent performance in asset allocation. In contrast, the other actors exhibit more significant and abrupt changes in asset weight over time, which may result in a higher degree of volatility and unpredictability in performance.

Discussion In the context of asset allocation, the performance of different actors is highly dependent on various hyperparameters. For the MCTS actor, the discretization step is a critical hyperparameter that significantly impacts the

quality of the solution. Additionally, the number of loops and rollouts are other key parameters that need to be tuned carefully for optimal performance. tf-plan presented limitations in terms of the lack of support for the modulus and natural logarithmic function and the inability to set a reallocation period in the domain. In this case, actions are applicable at every step, and state invariants and action preconditions are not directly honored from the RDDL.

The choice of reward hyperparameters significantly impacts the overall performance of each actor. As next steps, we plan to evaluate the performance of different actors with varying numbers of assets, horizon sizes, and rebalancing window sizes. Additionally, we will conduct experiments using real stock exchange data to evaluate the practical utility of the various actors. Finally, we plan to conduct a computation time analysis of each approach to evaluate their practical feasibility.

4 FinRDDL: Optimal Trade Execution

Once the asset weights are decided in the portfolio, if changes are needed from the previous state, then a trade has to be placed in the market and executed to implement the desired portfolio rebalancing changes.

Consider an investor that wishes to liquidate (sell) X units of a security between times $t = 0$ and $t = T$. The execution time horizon T is divided into N intervals of length $\tau = T/N$ to define the discrete times $t_k = k\tau$ for $k = 0, \ldots, N$. At time t_k, the *market* price of the security is denoted by S_k, the *execution* price of the broker's order is $\tilde{S}_k$. The number of units we plan to hold at time t_k is denoted by x_k and the number of units to sell is n_k. Finally, the rate of trading (execution speed) during the interval t_{k-1} to t_k is given by $v = n_k/\tau$.

The scheduling problem is then formulated as the problem of finding the optimal sequence $\{n_k\}_{k=0}^N$ that maximizes the revenue obtained from selling X with N orders:

$$\begin{aligned} \max_{\{n_k\}_{k=0}^N} \quad & \mathbb{E}\left[\sum_{k=1}^N \tilde{S}_k n_k\right] \\ \text{s.t. } \textstyle\sum_{k=1}^N n_k \quad & = X \end{aligned} \tag{4.1}$$

We follow the formulation of the price dynamics in (Almgren and Chriss 2001) where the market price evolves according to two factors; volatility and market impact. The volatility of the security price is a manifestation of market forces that occur randomly and are not affected by our trading. The market impact, on the other hand, is the component that accounts for the change in the market price S_k and the execution price $\tilde{S}_k$ which is impacted by our trading. Assuming that the security price evolves according to a the discrete arithmetic random walk, we define S_k and $\tilde{S}_k$ as:

$$S_k = S_{k-1} + \sigma\sqrt{\tau}\xi_k - \tau g(v), \quad \tilde{S}_k = S_{k-1} - h(v) \tag{4.2}$$

where, σ is a constant representing the volatility of the asset, ξ_k are draws from independent random variables each with zero mean and unit variance, $g(v)$ is the *permanent* impact function that represents the change in the market price S_k caused by our trading and $h(v)$ is the *temporary* impact function that we use to obtain the actual execution price. Assuming that we are selling and both impact functions to be *linear* in the rate of trading $v = n_k/\tau$, we get:

$$g(v) = \gamma v, \quad h(v) = \varepsilon n_k + \eta v \tag{4.3}$$

where γ is the linear permanent impact constant, ε is a constant defining the fixed cost of selling and η is the linear temporary impact constant.

To be able to measure the quality (and optimality) of the sequence of orders in the generated schedule, a reference price such as S_k at $t = 0$ is typically used. A common measure for transaction costs is the *implementation shortfall* (Perold 1988) which is the difference between the revenue if we were sell X at a given price (*e.g.* S_0) and the actual revenue obtained with the sequence of orders $\{n_k\}_{k=0}^N$.

$$\begin{aligned} C &= XS_0 - \sum_{k=1}^N n_k \tilde{S}_k \\ &= XS_0 - \sum_{k=1}^N \left(\sigma\sqrt{\tau}\xi_k - \tau\gamma v\right) x_k - \sum_{k=1}^N \varepsilon n_k + \frac{\eta}{\tau} n_k^2 \end{aligned} \tag{4.4}$$

The unconstrained quadratic optimization problem with a Lagrange multiplier λ representing risk aversion is:

$$\min_{\{n_k\}_{k=0}^N} \quad \left(\mathbb{E}\left[C\right] + \lambda \mathbb{V}\left[C\right]\right) \tag{4.5}$$

The closed-form solutions for this problem with linear price impact are equations 17 and 18 in (Almgren and Chriss 2001). The optimal solutions with different values of λ and τ are shown in figure 4.

The RDDL domain file developed for the optimal trade execution problem is detailed in listing 3 in the appendix. The state is defined to be a combination of the agent and market variables and includes the time elapsed t, quantity executed x_k, market price S_k and the execution price $\tilde{S}_k$. The action at a given state is the number of shares to sell n_k. The transition functions and the reward are given by equations 4.2 and 4.4 respectively.

5 Desiderata for AI Planning for Quantitative Finance Problems

Through this study, we gained valuable insights into the potentials and limitations of AI planning for financial trading problems. Based on our analysis, we have identified the main areas where AI planning can be improved:

Support for multi-dimensional rewards. In our formulation for asset allocation, we combined risk, returns, and transaction costs into a single reward expression. However, the hyperparameters α_σ and α_δ need to be set effectively, making it a sensitive task. Multi-objective planning algorithms (Zhang et al. 2022; Do and Kambhampati 2003; Yu, Kirley, and Buyya 2007) can be particularly helpful in this regard. To the best of our knowledge, RDDL or pyRDDLgym do not support multi-objective rewards as of yet.

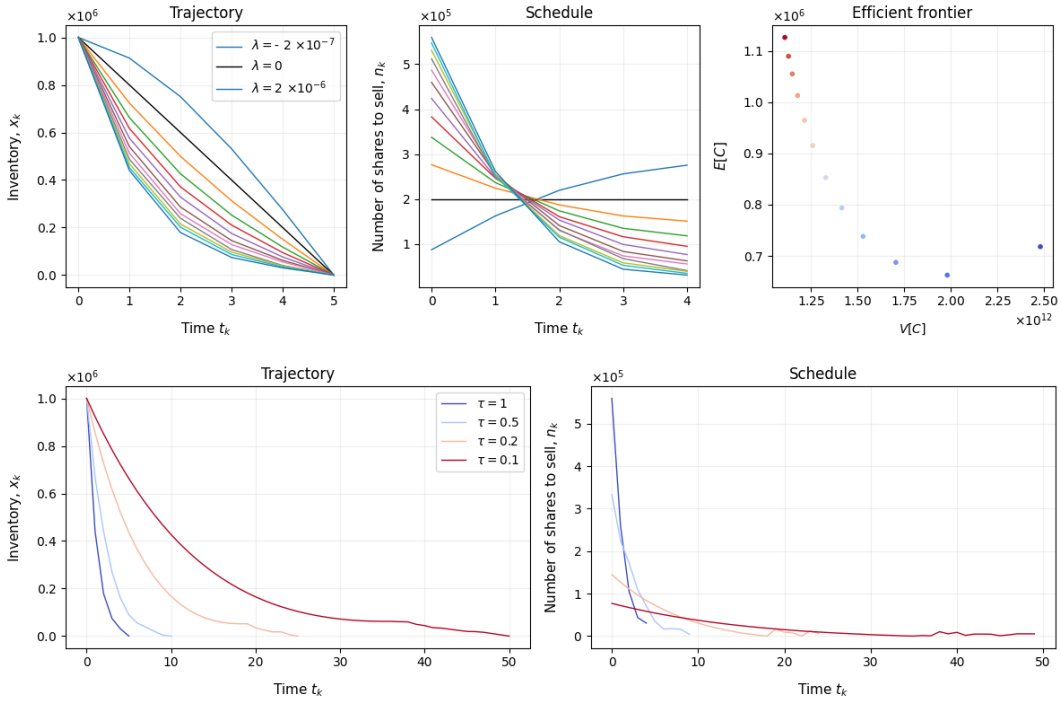

Figure 4: Trade trajectories and schedules with different risk-aversion values λ (top) and trading speed τ (bottom). Our RDDL domain which can simulate this behavior is provided in listing 3 in the appendix.

Challenges and opportunities of planners with continuous state-action spaces in dynamic environments. While the RDDL language supports continuous state and action spaces, most planners assume the search space to be discrete. For instance, the popular probabilistic planning system, Prost planner (Keller and Eyerich 2012), uses a discretization approach to represent continuous state spaces as a finite set of discrete states by default. To address this limitation, Prost provides advanced features that allow users to model continuous state spaces more accurately, which can be explored. Planners that support PDDL+ such as DiNo (Piotrowski et al. 2016) can support continuous state and action spaces (albeit they discretize it during solving). However, they do not support handling concurrent actions with conflicting effects.

Improving the MCTS actor. The current MCTS actor can be enhanced in several ways to improve its performance. One way is to modify the reward function to include risk metrics such as Value at Risk or Expected Shortfall and optimize the Pareto frontier of risk and return. Another way is to parallelize the rollouts to reduce the computational time required for rollouts, particularly in applications with large search spaces. Additionally, incorporating continuous and/or state and action space through techniques such as (Mansley, Weinstein, and Littman 2011; Yee et al. 2016) can make MCTS more flexible and applicable to a wider range of problems.

Bridging the gap between simulation and reality. While the current RDDL formulation poses a challenging planning problem, real-world trading problems involve even more complex features that are often overlooked in the model. For instance, the total wealth is a critical parameter that can significantly impact the asset weights. Additionally, illiquid assets may have trade restrictions that need to be taken into account. Order execution happens in continuous time with durative actions. It may be beneficial to address asset allocation and order execution as an integrated planning and acting problem, rather than solving them separately. The investment horizon can be unknown and multiple orders may need to be scheduled and executed together.

6 Conclusion

In this study, we formulated the asset allocation and optimal trade execution as planning problems using the RDDL domain modelling language. We tested several AI actors with different planning strategies on the developed RDDL instance for asset allocation. Finally, we critically analyzed the current limitations of automated planning for financial trading problems, and how they can be addressed. By combining RDDL modeling with planning, we have the potential to generate strong trading policies that are tailored to the specific needs and objectives of the investor. The use of RDDL for modelling asset allocation represents an important research avenue in portfolio management that allows for more sophisticated and effective decision-making in a complex and dynamically evolving market environment.

References

Abidin, S. N. Z.; and Jaffar, M. M. 2012. A review on Geometric Brownian Motion in forecasting the share prices in Bursa Malaysia. *World Applied Sciences Journal*, 17(1): 82–93.

Almgren, R.; and Chriss, N. 2001. Optimal execution of portfolio transactions. *Journal of Risk*, 3: 5–40.

Bertsimas, D.; and Lo, A. W. 1998. Optimal control of execution costs. *Journal of financial markets*, 1(1): 1–50.

Blay, K.; Gosh, A.; Kusiak, S.; Markowitz, H.; Savoulides, N.; and Zheng, Q. 2020. Multiperiod Portfolio Selection: A Practical Simulation-Based Framework. *Journal of Investment Management*, 18(4): 94–129.

Bouchaud, J.-P.; Bonart, J.; Donier, J.; and Gould, M. 2018. *Trades, quotes and prices: financial markets under the microscope*. Cambridge University Press.

Boyd, S.; Boyd, S. P.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Boyd, S.; Busseti, E.; Diamond, S.; Kahn, R. N.; Koh, K.; Nystrup, P.; Speth, J.; et al. 2017. Multi-period trading via convex optimization. *Foundations and Trends® in Optimization*, 3(1): 1–76.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. .

Do, M.; and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20: 155–194.

Donnelly, R. 2022. Optimal Execution: A Review. *Applied Mathematical Finance*, 1–32.

Hachicha, N.; Jarboui, B.; and Siarry, P. 2011. A fuzzy logic control using a differential evolution algorithm aimed at modelling the financial market dynamics. *Information Sciences*, 181(1): 79–91.

Helmert, M. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables. In *AIPS*, 44–53.

Hu, Y.; and Øksendal, B. 1998. Optimal time to invest when the price processes are geometric Brownian motions. *Finance and Stochastics*, 2(3): 295–310.

Keller, T.; and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 22, 119–127.

Kim, B.; Lee, K.; Lim, S.; Kaelbling, L.; and Lozano-Pérez, T. 2020. Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9916–9924.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Machine Learning: ECML 2006: 17th European Conference on Machine Learning Berlin, Germany, September 18-22, 2006 Proceedings 17*, 282–293. Springer.

Li, X.; Uysal, A. S.; and Mulvey, J. M. 2022. Multi-period portfolio optimization using model predictive control with mean-variance and risk parity frameworks. *European Journal of Operational Research*, 299(3): 1158–1176.

Mansley, C.; Weinstein, A.; and Littman, M. 2011. Sample-based planning for continuous action markov decision processes. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 21, 335–338.

Markowitz, H. 1952. Portfolio selection. *The Journal of Finance*, 7(1): 77–91.

Perold, A. F. 1988. The implementation shortfall: Paper versus reality. *Journal of Portfolio Management*, 14(3): 4.

Piotrowski, W. M.; Fox, M.; Long, D.; Magazzeni, D.; and Mercorio, F. 2016. Heuristic Planning for PDDL+ Domains. In *AAAI Workshop: Planning for Hybrid Systems*, volume 16, 12.

Sanner, S.; et al. 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32: 27.

Taitler, A.; Gimelfarb, M.; Gopalakrishnan, S.; Mladenov, M.; Liu, X.; and Sanner, S. 2022. pyRDDLGym: From RDDL to Gym Environments. *arXiv preprint arXiv:2211.05939*.

Wu, G.; Say, B.; and Sanner, S. 2017. Scalable planning with tensorflow for hybrid nonlinear domains. *Advances in Neural Information Processing Systems*, 30.

Yee, T.; Lisỳ, V.; Bowling, M. H.; and Kambhampati, S. 2016. Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. In *IJCAI*, 690–697.

Younes, H. L.; and Littman, M. L. 2004. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2: 99.

Yu, J.; Kirley, M.; and Buyya, R. 2007. Multi-objective planning for workflow execution on grids. In *2007 8th IEEE/ACM International Conference on Grid Computing*, 10–17. IEEE.

Zhang, H.; Salzman, O.; Kumar, T. S.; Felner, A.; Ulloa, C. H.; and Koenig, S. 2022. A* pex: Efficient approximate multi-objective search on graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 394–403.

Acknowledgments

This paper was prepared for informational purposes in part by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("J.P. Morgan"), and is not a product of the Research Department of J.P. Morgan. J.P. Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

Appendix: FinRDDL

In this appendix, we present comprehensive RDDL descriptions along with an example instance for both the asset allocation and order execution problems. For asset allocation, we provide two distinct models: one with continuous action space and another with a discretized action space.

Asset Allocation (Continuous)

Listing 1 is the domain for the asset allocation problem with continuous action space. This model is utilized by various actors, including the uniform actor, Greedy Markowitz actor, and the MIQP actor. The tf-plan actor employs a slightly modified version of this domain that excludes the use of the log function when computing the rewards.

Listing 1: Asset Allocation RDDL domain with a continuous action space

```
////////////////////////////////////////////////////////////////////////////////
// A simple continuous state-action MDP for the asset allocation problem with actions
    describing the continuous portfolio weights.
////////////////////////////////////////////////////////////////////////////////

domain asset_allocation_continuous {

    requirements = { continuous, integer-valued };

    types {
        asset : object;
    };

    pvariables {
        // number of days between asset weight updates
        REALLOCATION-PERIOD : { non-fluent, int, default = 60 };
        mean(asset) : { non-fluent, real, default = 0.0001 };
        risk(asset) : { non-fluent, real, default = 0.0001 };

        // scaling factors for the risk term and transaction costs
        alpha_risk : { non-fluent, real, default = 0.5 };
        alpha_trans_cost : { non-fluent, real, default = 0.5 };

        // weight of asset
        weight(asset) : { state-fluent, real, default = 0 };
        // price of the asset
        price(asset) : { state-fluent, real, default = 0 };
        day : { state-fluent, int, default = 0 }; // current day

        set_weight(asset) : { action-fluent, real, default = 0 };
    };

    cpfs {
        // portfolio weights should sum to 1
        weight'(?a) = if (mod[day, REALLOCATION-PERIOD] == 0)
                    then set_weight(?a) / sum_{?b: asset} set_weight(?b)
                    else weight(?a);

        // price evolution using a Geometric Brownian Motion
        price'(?a) = price(?a) * exp[mean(?a) + risk(?a) * Normal(0, 1)];
        day' = day + 1;
    };

    reward = sum_{?a: asset} [weight(?a) * (ln[price'(?a)/price(?a)]
                - alpha_risk * risk(?a)
                - alpha_trans_cost * abs[weight'(?a) - weight(?a)])];

    state-invariants {
        forall_{?a : asset} (weight(?a) >= 0 ^ weight(?a) <= 1);
        forall_{?a : asset} (price(?a) >= 0);
    };

```

```
    action-preconditions {
        forall_{?a : asset} (set_weight(?a) >= 0 ^ set_weight(?a) <= 1);
    };
}
////////////////////////////////////////////////////////////////////////////////////

non-fluents aa_con_0 {
    domain = asset_allocation_continuous;

    objects {asset : {a0,a1};};

    non-fluents {
        mean(a0) = 0.0001;
        risk(a0) = 0.005;
        mean(a1) = 0.0001;
        risk(a1) = 0.005;
    };
}
////////////////////////////////////////////////////////////////////////////////////

instance inst_aa_con {
  domain = asset_allocation_continuous;

  non-fluents = aa_con_0;

  init-state {
        weight(a0) = 0.5;
        weight(a1) = 0.5;
        price(a0) = 10;
        price(a1) = 50;

  };

  max-nondef-actions = pos-inf;
  horizon = 300;
    discount = 1.0;
}
```

Asset Allocation (Discretized action space)

Listing 2 is the domain for the asset allocation problem with discretized action space. This model is used by the MCTS actor.

Listing 2: Asset Allocation RDDL domain with a discrete action space

```
////////////////////////////////////////////////////////////////////////////////////
// A simple discrete state-action MDP for the asset allocation problem with actions
    describing the discrete portfolio weights.
////////////////////////////////////////////////////////////////////////////////////

domain asset_allocation_discrete {

    requirements = {  };

    types {
        asset : object;
    };

    pvariables {
        // number of days between asset weight updates
        REALLOCATION-PERIOD : { non-fluent, int, default = 60 };
        mean(asset) : { non-fluent, real, default = 0.0001 };
        risk(asset) : { non-fluent, real, default = 0.0001 };

        // scaling factors for the risk term and transaction costs
        alpha_risk : { non-fluent, real, default = 0.5 };
```

```
        alpha_trans_cost : { non-fluent, real, default = 0.5 };

        // weight of asset
        weight(asset) : { state-fluent, real, default = 0 };
        // price of the asset
        price(asset) : { state-fluent, real, default = 0 };
        day : { state-fluent, int, default = 0 }; // current day

        set_weight(asset) : { action-fluent, int, default = 1 };
    };

    cpfs {
        // portfolio weights should sum to 1
        weight'(?a) = if (mod[day, REALLOCATION-PERIOD] == 0)
                    then set_weight(?a) / sum_{?b: asset} set_weight(?b)
                    else weight(?a);

        // price evolution using a Geometric Brownian Motion
        price'(?a) = price(?a) * exp[mean(?a) + risk(?a) * Normal(0, 1)];
        day' = day + 1;
    };

    reward = sum_{?a: asset} [weight(?a) * (ln[price'(?a)/price(?a)]
                - alpha_risk * risk(?a)
                - alpha_trans_cost * abs[weight'(?a) - weight(?a)])];

    state-invariants {
        forall_{?a : asset} (weight(?a) >= 0 ^ weight(?a) <= 1);
        forall_{?a : asset} (price(?a) >= 0);
    };

    action-preconditions {
        forall_{?a : asset} (set_weight(?a) >= 1 ^ set_weight(?a) <= 10);
    };
}
////////////////////////////////////////////////////////////////////////////////

non-fluents aa_dis_0 {
    domain = asset_allocation_discrete;

    objects {asset : {a0,a1};};

    non-fluents {
        mean(a0) = 0.0001;
        risk(a0) = 0.005;
        mean(a1) = 0.0001;
        risk(a1) = 0.005;
    };
}

////////////////////////////////////////////////////////////////////////////////

instance inst_aa_dis {
  domain = asset_allocation_discrete;

  non-fluents = aa_dis_0;

  init-state {
        weight(a0) = 0.5;
        weight(a1) = 0.5;
        price(a0) = 10;
        price(a1) = 50;

  };
```

```
  max-nondef-actions = pos-inf;
  horizon = 300;
    discount = 1.0;
}
```

Order Execution (Discretized action space)

Listing 3 is the domain for the order execution problem based on Almgren and Chriss, 2001. The action is the number of units of the security to sell at each discrete time step and the goal is to minimize the implementation shortfall (eq 4.4).

Listing 3: Order Execution RDDL domain with a discrete action space

```
////////////////////////////////////////////////////////////////////////////////
// A simple discrete state-action MDP for the order execution problem with actions
    describing the discrete number of shares to sell.
////////////////////////////////////////////////////////////////////////////////

domain order_execution_discrete {

    requirements = { continuous, integer-valued };

    types {
        security : object;
    };

    pvariables {
        // Initial Share Price ($)
        S_0(security) : { non-fluent, real, default = 50 };

        // Bid-Ask Spread
        BAS(security) : { non-fluent, real, default = 0.125 };

        // Annual Volatility (%)
        AV(security) : { non-fluent, real, default = 0.3 };

        // Median Daily Trading Volume (number of shares)
        MDTV(security) : { non-fluent, real, default = 5000000 };

        // Initial holdings (number of shares to sell)
        X(security) : { non-fluent, int, default = 10000000 };

        // Number of trades
        N : { non-fluent, int, default = 5 };

        // Risk aversion parameter
        lambda : { non-fluent, real, default = 0.000002 };

        // Market price of the security
        market_price(security) : { state-fluent, real, default = 0 };

        // Execution price of the security
        exec_price(security) : { state-fluent, real, default = 0 };

        // Time elapsed
        t : { state-fluent, int, default = 0 };

        // Quantity executed
        x(security) : { state-fluent, int, default = 0 };

        // Quantity to sell (our action)
        n(security) : { state-fluent, int, default = 0 };

        set_n(security) : { action-fluent, int, default = 0 };
    };

```

```
    cpfs {
        // Length of the interval
        tau = T / N;

        // Daily Volatility
        dv(?s) = AV(?s) / sqrt[250];

        // Scaled Volatility ( ($ / share) / (day)^0.5 )
        sigma(?s) = dv(?s) * S_0(?s);

        // Linear Permanent Impact Constant ($/share)/share
        gamma(?s) = BAS(?s) / (0.1 * MDTV(?s));

        // Fixed Cost of buying/selling ($/share)
        epsilon(?s) = BAS(?s) / 2;

        // Price Impact for Each 1% of Daily Volume Traded
        eta(?s) = BAS(?s) / (0.01 * MDTV(?s));

        n(?s) = set_n(?s)

        // rate of trading (execution speed)
        v = n(?s) / tau;

        // Permenant Impact (eq 4.3)
        perm_impact(?s) = gamma(?s) * v;

        // Temporary Impact (eq 4.3)
        temp_impact(?s) = epsilon(?s) + eta(?s) * v;

        // Market price evolution (eq 4.2)
        market_price'(?s) = market_price(?s)
                            + sigma(?s) * sqrt[tau] * Normal(0, 1)
                            - tau * perm_impact(?s);

        // Execution price evolution (eq 4.3)
        exec_price'(?s) = exec_price(?s) - temp_impact(?s);
    };

    // eq 4.4
    reward = X(?s) * S_0(?s) - n(?s) * exec_price'(?s);

    state-invariants {
        forall_{?s : security} (x(?s) <= X(?s) ^ n(?s) <= X(?s));
        forall_{?s : security} (market_price(?s) >= 0);
    };

    action-preconditions {
        forall_{?s : security} (x(?s) <= X(?s) ^ set_n(?s) <= X(?s));
    };

    termination {
        forall_{?s : security} (x(?s) >= X(?s));
    };
}
////////////////////////////////////////////////////////////////////////////////////////

non-fluents oe_dis_0 {
    domain = order_execution_discrete;

    objects {security : {s0};};

    non-fluents {
        S_0(s0) = 50;
        BAS(s0) = 0.125;
```

```
        AV(s0) = 0.3;
        MDTV(s0) = 5000000;
        X(s0) = 10000000;
        N = 5;
        lambda = 0.000002;
    };
}

////////////////////////////////////////////////////////////////////////////////////////////

instance inst_oe_dis {
  domain = order_execution_discrete;

  non-fluents = oe_dis_0;

  init-state {
        market_price(s0) = S_0(s0);
        exec_price(s0) = S_0(s0);
        t = 0;
        x(s0) = 0;
        n(s0) = 0;
  };

  max-nondef-actions = pos-inf;
  horizon = 5; // Execution time horizon (number of time periods)
    discount = 1.0;
}
```