33rd International Conference on
Automated Planning and Scheduling
July 9 - 10, 2023, Prague, Czech Republic

FinPlan 2023

Preprints of the 4th Workshop on
Planning and Scheduling for
Financial Services (FinPlan)

Edited by:

Parisa Zehtabi, Alberto Pozanco, William Yeoh, Biplav Srivastava

Organization

Parisa Zehtabi
J.P. Morgan AI Research, UK

Alberto Pozanco
J.P. Morgan AI Research, Spain

William Yeoh
Washington University in St. Louis, USA

Biplav Srivastava
University of South Carolina, USA

Program Committee

Daniel Borrajo (J.P. Morgan AI Research, Spain)
Michael Cashmore (J.P. Morgan AI Research, UK)
Giuseppe Canonaco (J.P. Morgan AI Research, Spain)
Mark Feblowitz (IBM, USA)
Fernando Fernández (Universidad Carlos III de Madrid, Spain)
Andrew Murray (University of Strathclyde, UK)
Alberto Pozanco (J.P. Morgan AI Research, Spain)
Rui Silva (J.P. Morgan AI Research, UK)
Shirin Sohrabi (IBM, USA)
Biplav Srivastava (University of South Carolina, USA)
William Yeoh (Washington University in St. Louis, USA)
Parisa Zehtabi (J.P. Morgan AI Research, UK)

Foreword

Planning and scheduling are mature fields in terms of base techniques and algorithms to solve goal-oriented tasks. Planning approaches have been successfully applied to many domains including classical domains (e.g., logistics and Mars rovers) and, more recently, in oil and gas as well as mining industries. Similarly, scheduling approaches have also been applied to many industrial applications. However, very little work has been done in relation to the problems in the finance industry, which spans a diverse range of activities in financial markets, corporate finance, insurance, banking, and accounting. Recently, some large financial corporations have started AI research labs and researchers at those teams have found that there are plenty of open planning and scheduling problems to be tackled by the ICAPS community. For example, these include trading markets, workflow learning, generation and execution, transactions flow understanding, fraud detection, and customer journeys. In addition, planning problems tackled in other settings like dialog management and network penetration; and richer problem formations involving planning along with learning and scheduling.

FinPlan'23 is the fourth workshop on Planning and Scheduling for Financial Services held in conjunction with ICAPS, whose aim is to bring together researchers and practitioners to discuss challenges for Planning and Scheduling in Financial Services, and the opportunities such challenges represent to the planning research community.

Parisa Zehtabi, Alberto Pozanco, William Yeoh, Biplav Srivastava, July 2023

Contents

Predicting Customer Goals in Financial Institution Services: A Data-Driven LSTM Approach
Andrew Estornell, Stylianos Loukas Vasileiou, William Yeoh, Daniel Borrajo, Rui Silva **1**

Value Detection Rate: A Performance Metric for Payments Fraud Detection
Danial Dervovic, Saeid Amiri, Michael Cashmore **7**

Accelerating Benders Decomposition via RL Surrogate Models
Kyle Mana, Stephen Mak, Parisa Zehtabi, Michael Cashmore, Daniele Magazzeni, Manuela Veloso **12**

Deep Reinforcement Learning for Optimal Portfolio Allocation: A Comparative Study with Mean-Variance Optimization
Srijan Sood, Kassiani Papasotiriou, Marius Vaiciulis, Tucker Balch **21**

Surrogate Assisted Monte Carlo Tree Search in Combinatorial Optimization
Saeid Amiri, Parisa Zehtabi, Danial Dervovic, Michael Cashmore **31**

FinRDDL: Can AI Planning be used for Quantitative Finance Problems?
Sunandita Patra, Mahmoud Mahfouz, Sriram Gopalakrishnan, Daniele Magazzeni, Manuela Veloso **36**

Can LLMs be Good Financial Advisors?: An Initial Study in Personal Decision Making for Optimized Outcomes
Kausik Lakkaraju, Sai Krishna Revanth Vuruma, Vishal Pallagani, Bharath Muppasani, Biplav Srivastava **51**

Predicting Customer Goals in Financial Institution Services: A Data-Driven LSTM Approach

Andrew Estornell*,[1] Stylianos Loukas Vasileiou*,[1] William Yeoh,[1] Daniel Borrajo,[2] Rui Silva[2]

[1] Washington University in St. Louis
{aestornell,vstylianos, wyeoh}@wustl.edu
[2] J.P. Morgan AI Research
{daniel.borrajo,rui.silva}@jpmchase.com

Abstract

In today's competitive financial landscape, understanding and anticipating customer goals is crucial for institutions to deliver a personalized and optimized user experience. This has given rise to the problem of accurately predicting customer goals and actions. Focusing on that problem, we use historical customer traces generated by a realistic simulator and present two simple models for predicting customer goals and future actions – an LSTM model and an LSTM model enhanced with state-space graph embeddings. Our results demonstrate the effectiveness of these models when it comes to predicting customer goals and actions.

Introduction

The financial industry has experienced significant transformation in recent years, driven by rapid technological advancements, evolving customer expectations, and increased competition. As customers demand more personalized and convenient services, financial institutions are under pressure to develop a deeper understanding of their clients' needs and preferences. This has led to a growing interest in leveraging data-driven approaches to gain insights into customer behavior and predict future actions. Accurate goal prediction can help financial institutions provide targeted incentives, improve customer satisfaction, and ultimately foster loyalty and retention in an increasingly competitive landscape.

Planning-based approaches have been widely used for goal prediction, as they focus on modeling an agent's decision-making process and finding optimal sequences of actions to achieve specific objectives (Ramírez and Geffner 2010; Sohrabi, Riabov, and Udrea 2016; Keren, Gal, and Karpas 2019; Wayllace et al. 2016; Vered and Kaminka 2017). For example, Ramírez and Geffner (2010) propose a probabilistic plan recognition approach that utilizes classical planners to predict goals based on observed actions. Keren, Gal, and Karpas (2019) introduced the concept of goal recognition design, which aims to optimize the planning domain to facilitate the goal prediction process.

In this context, the work of Borrajo and Veloso (2020) presents an approach to address the challenges associated with predicting goals in complex real-world domains. In particular, they introduced a domain-independent simulator for generating synthetic customer behavior datasets, which can be used to model and analyze customer interactions with a financial institution, such as ATM or mobile app usage. By leveraging this simulator, they are able to create datasets that accurately represent the intricacies and dynamics of customer traces in a banking environment, providing a valuable foundation for the development and evaluation of goal prediction models.

Inspired by the work of Borrajo and Veloso (2020), we build upon their domain-independent simulator to generate synthetic customer behavior datasets. We then use this data to train two models – an LSTM model and an LSTM model enhanced with state-space graph embeddings. The LSTM-based models capitalize on the sequential nature of customer traces, capturing the intricate patterns present in customer interactions over time. By incorporating state-space graph embeddings into the LSTM model, we further enrich the model's understanding of the relationships and dependencies among various features within the dataset, which may lead to improved performance. This combination of LSTM models and state graph embeddings offers a more scalable and efficient solution in predicting customer goals and actions, while maintaining a high level of accuracy and robustness in the face of real-world complexities. Our results demonstrate the effectiveness of LSTM models and state graph embeddings in addressing the challenges faced by financial institutions when it comes to predicting customer goals and actions.

Related Work

Goal prediction (known as goal recognition in the planning literature) refers to the process of identifying an agent's objective among several possibilities, based on the agent's behavior, a model of the environment, and a sequence of observations. The first approach that explicitly addresses goal prediction is that of Baker, Saxe, and Tenenbaum (2009), who framed the problem as "inverse" planning. Ramírez and Geffner (2010) followed up on the work by proposing a generative approach that uses classical planning algorithms for inferring probability distributions over a set of possible goals given some observations. Recent research has extended the original problem formulation of recognizing plans in various domain models, such as continuous domain models and epistemic planning problems (Vered and Kaminka 2017;

*Equal contribution.

Sohrabi, Riabov, and Udrea 2016; Kaminka, Vered, and Agmon 2018; Shvo et al. 2020). In similar spirit, Keren, Gal, and Karpas (2019) provided an alternate view of goal recognition that focuses on modifying the domain model such that the goal recognition can be achieved with as few observations as possible. Extensions of the goal recognition design problem have been developed over various types of domain models and settings (Shvo and McIlraith 2020; Wayllace et al. 2016, 2020; Wayllace and Yeoh 2022).

However, planning-based approaches often struggle to scale with the size and complexity of real-world domains, due to factors such as large state spaces, partial observability, and dynamic environments. In contrast, learning-based approaches may perform better under these circumstances, especially when real-time or near-real-time predictions are required. Most notably, Borrajo, Gopalakrishnan, and Potluru (2020) illustrated the trade-offs between planning-based and learning-based approaches for goal recognition in different (planning) domains. Their results highlight that planning-based approaches perform better when there is a partial order of actions in plans, whereas learning techniques, such as LSTMs, can better capture the relationship between actions and goals if there is a relation not directly associated with goal achievement. While in this paper we also use a learning-based approach (e.g., LSTMs) for goal prediction, note that we focus on the specific task of predicting customer goals and actions in the context of financial institutions. Finally, there has been an interest in integrating planning-based approach with data-driven approaches. For example, Wilken et al. (2023) proposed a hybrid method that captures statistical relations between certain states of the environment and goals learned from past observations.

While in this paper we focus on goal prediction problems, i.e., predicting user goals and actions, we explore data-driven methodologies and focus on a problem related to financial industry. In that context, a number of works that tackle related problems have been investigated. Baeza-Yates et al. (2015) presented a method for predicting the next mobile app a user is going to open based on their usage history and the "wisdom" of the crowd, while several works have addressed the problem of customer churn in banking with various machine learning techniques (Yaghini, Zhiyan, and Fallahi 2011; Bilal Zorić 2016; Rahman and Kumar 2020). While these works and our work in this paper address challenges in the financial industry, the specific goals and techniques employed are different, i.e., we employ LSTM models with graph embeddings aimed at understanding customer behavior and preferences.

Methodology

In this section, we describe the dataset, data preparation, and models used for predicting customer goals and actions. In particular, we employ two models – an LSTM model exploiting a bag-of-words representation, and an LSTM model enhanced with a state graph embedding. Our approach is underpinned by a comprehensive semi-synthetic dataset, capturing intricate customer interactions with diverse banking interfaces.

Date and Time	Event
2022-02-28 18:06:08	mobile: login
2022-02-28 18:10:24	mobile: enter menu settings
2022-02-28 18:10:24	mobile: enter menu profile-maintenance
2022-02-28 18:14:40	mobile: change information on demographic
2022-02-28 18:18:56	mobile: log-off
2022-04-30 07:28:32	web: login
2022-04-30 07:28:32	web: enter menu credit-card
2022-04-30 07:28:32	web: get information on credit-card-transaction-history
2022-04-30 07:28:32	web: exit menu root-section
2022-04-30 07:32:48	web: enter menu settings
2022-04-30 07:37:04	web: enter menu alerts-maintenance
2022-04-30 07:41:20	web: change information on alerts-definition
2022-04-30 07:45:36	web: log-off

Table 1: Example trajectory of events for an agent of *medium income*, *low fail behavior*, and *digital interface*.

Customer Behavior Data

We utilize a semi-synthetic dataset of customer behavior generated via a domain-independent simulator proposed by Borrajo and Veloso (2020). The dataset consists of interactions between customers and a multitude of bank interfaces: bank-website, mobile app, teller counter, or ATM. Each interface offers customers a (potentially unique) set of actions such as making payments, checking rewards program information, modifying personal information, etc. A set of example actions are provided in Table 1, where we see a customer login to the mobile app and modify their personal information; the same customer then logs into the website two months later and checks their credit-card transaction history as well as modifying their alter settings. Approximately 300 actions are recorded per-customer.

Customer Goals: In addition to customer actions, a diverse set of goals can be induced from the dataset, such as:

- *check information*: The customer wants to get some information about their account (e.g., balance).
- *change information*: The customer wants to change the value of some data point (e.g., address).
- *operational goals*: The customer wants to perform some banking operations, such as deposit-cash, withdrawal, exchange, deposit-check, pay-bill, make-payment, and so on.

These goals correspond to the customers purpose for interacting with any of the bank interfaces. Customers can possess multiple goals; for example, the customer in Table 1 seeks to both check and change information while using the website (second interaction in the table).

Customer Types: The dataset also features labels for the type of customer, characterized by three attributes:

- *Income*: income level of the customer (high, medium, low, standard).
- *Fail behavior*: how frequently the customer's action correspond to errors (rarely, often, no-failure).
- *Digital behavior*: customers preferred interface type (traditional, digital, mixed).

These attributes represent the probabilities of using different channels (web, mobile app, ATM, banker, and teller), the failure rate of operations, and types of goals. For instance, a student might be categorized as low-rarely-digital, while a medium-class worker could be medium-rarely-mixed.

Data Preparation

Next we discuss our approach to data preprocessing. The predictive features $\mathbf{X}$ are the customer's historical actions (e.g. Table 1). The target features y are the customer's future actions, the customer's goal, or the customer's type.

Handcrafted Features: To increase the predictive efficacy of the data we first introduce several handcrafted features, and then discuss how both the handcrafted, and original features, are represented. The predictive features $\mathbf{X}$ are provided as Time (an integer) and Action (a string). We break the Event string into more meaningful features which are listed in Table 2. The Event string contains an indicator of which interface the customer is interacting with (web, mobile, teller, ATM), which we refer to as the *primary location* feature.

Moreover, the Event string contains information indicating when the customer navigates through the interface, e.g., in Table 1 the customer has Event string "web: enter menu credit-card", which indicates that the customer has left the web home-page (which they arrived at when logging in) and navigated to a menu of credit-card options. We refer to these features as *secondary location* features.

Defining secondary locations allows the model to more effectively learn which actions are available to customers at any given time, e.g., customers cannot modify their credit-card information without having first entered the credit-card menu. The combination of the primary and secondary location features can be interpreted as the customer's current state.

In addition to the state features, we also define corresponding *action* features that fall into three categories: *transitioning-actions* that indicate the customer changing primary (or secondary) locations (e.g., "web: enter menu credit-card"); *information-gaining-actions* that indicate the customer obtaining new information (e.g. "web: get information on credit-card-transaction-history"); and *modification-actions* that indicate the customer modifying their information (e.g. "web: change information on alerts-definition"). Full details on the state-action features are provided in Table 2. The predictive features $\mathbf{X}$ are thus represented in terms of these state-action pairs and a customer trajectory consists are a list of state-action pairs.

Feature Name	Possible Values
Primary location	*web, mobile, teller, banker, ATM*
Secondary location (web and mobile menus)	*credit-card, credit-score, offers, rewards, operations, settings, alerts-maintenance, contact-us, account-documents, profile-maintenance*
Transitioning actions	*login, log-off, enter enter-menu, exit, exit-menu*
Information-gaining actions	*alerts-definition, alerts-history, atm-branches, balance, benefits, demographic, documents, faq, help-call, help-email, offers, credit-card-trans-history, credit-card-trans-summary, limit-credit-card, credit-score-history, messages, rewards-activity, rewards-use-points, credit-score-summary, trans-history, trans-summary*
Modification actions	*demographic, password, user-id, limit-credit-card, alerts-definition*

Table 2: List of handcrafted features.

Graph Representation: In addition to a bag-of-word representation (Harris 1954), we also utilize a state graph embedding that will allow for better learning of each of our objectives.

In the graph embedding, nodes correspond to state features (i.e., primary and secondary location features). Nodes are connected by an edge if there exists a transition (i.e., an action that allows a customer to move between the two corresponding states). In addition to neighbor information, the node features of the graph are given as four binary indicators: the customer has visited this node in the past, the customer is currently at this node, the customer performed an information-gaining-action at this node, the customer performed a modification-action at this node. These indicators are refereed to respectively as $\langle$*past-nodes*, *ego-node*, *info-gain*, *modification*$\rangle$.

The intuition behind the advantage of this graph embedding is that it allows the model to have a less myopic understanding of the way in which actions affect the customer's location in each interface, which in turn allows the model to have a less myopic understanding of which actions will be available to the customer at the future time-steps as well as possible paths to nodes which may achieve the customer's goal.

Predictive Models

To capture the temporal patterns in customer interactions and learn the intricate relationships between actions and goals over time, we utilized the architecture of LSTM networks (Hochreiter and Schmidhuber 1997). Specifically, we used a bag-of-words and one-hot encoding to represent the events as well as constructed a state graph to use as a GNN embedding in the LSTM model. The graph was constructed by examining all customer trajectories in the training dataset and defining nodes and edges as any state or transition action which appeared in more than 10 times. We found 10 to be the best frequency threshold when defining the graph as this value provided an effective balance between including irrelevant states and edges (which the model would then need to learn to ignore) and excluding important states and edges that will increase the myopicness of model predictions. Figure 1 shows an example of bag-of-words representation and state-space graph representation, respectively, generated from the datasets.

	id	time	is_digital	is_traditional	atm	mobile	web	teller	banker	login	log-off	enter
0	2	1677088680	0	1	1	0	0	0	0	1	0	0
1	2	1677088920	0	1	1	0	0	0	0	0	1	0
3000	16	1677167640	1	0	0	1	0	0	0	1	0	0
3001	16	1677167760	1	0	0	1	0	0	0	0	0	0
3002	16	1677168000	1	0	0	1	0	0	0	0	1	0

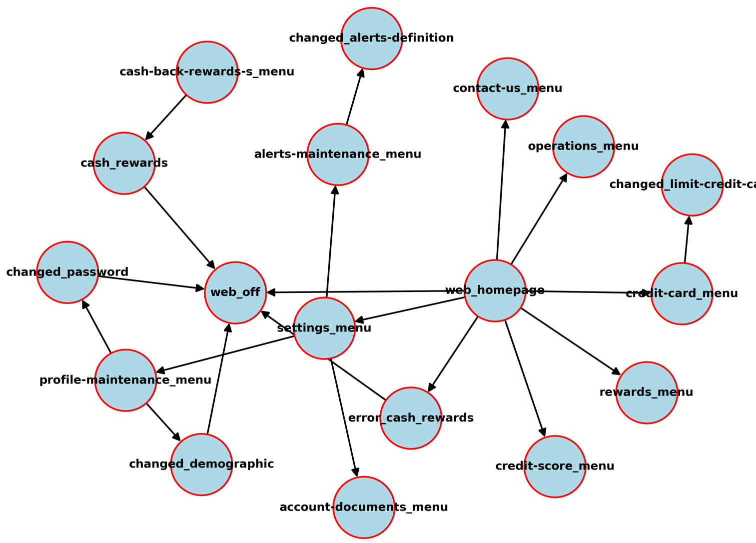

Figure 1: An example of bag-of-words and one-hot encoding representation, and a state-space graph representation.

Predictive Objective: Our primary focus will be on the following predictive tasks:

- *Goal Prediction:* Given n_H historical actions, what goal does the customer have?
- *Type Prediction:* Given n_H historical actions, what is the customer's type?
- *Trajectory Prediction:* Given n_H historical actions, what are the customer's next n_F future actions?

Experimental Results

In this section, we present the experimental results of our two proposed models – LSTM and GNN+LSTM. Our experimental setup consisted of the semi-synthetic dataset with 12 thousand customer interactions, split into 70% training, 15% validation, and 15% test sets. Each data point in our dataset represents a sequence of a customer's previous 20 events. The LSTM and GNN+LSTM models were trained using the Adam optimizer (Kingma and Ba 2014), with a learning rate of 0.01, over 5000 epochs with early stopping based on the validation set performance. The primary metrics used for comparison were prediction accuracy for customer goals, agent types, and future events.

Goal Prediction: Table 3 shows the accuracy for each approach when predicting the goal of the customer. We see that both the LSTM and GNN+LSTM models are capable of accurately predicting customer goals, however neither method

Model	Accuracy	
	Check Info	Change Info
LSTM	71%	68%
GNN+LSTM	77%	75%

Table 3: Accuracy for each approach when predicting agent type. LSTM corresponds to the LSTM approach with a bag-of-words embedding and GNN+LSTM corresponds to the LSTM approach with the graph representation and GNN encoding layer . For each prediction, the model views the customer's most recent 20 events.

achieves greater than 80% accuracy. With that said, graph embedding does offer a significant improvement to model efficacy when compared to the bag-of-words embedding.

Type Prediction: In Table 4, we see the accuracy when predicting customer type.[1] Both LSTM and GNN+LSTM have high efficacy (roughly 90% or more) when predicting both the agents failure rate type and preferred interface. In the case of preferred interface, customers tend to exclusively use their preferred interface, meaning that this predictive task becomes significantly easier once observing historical events from each customer.

Trajectory Prediction: In Table 5, we see the accuracy

[1] The ground truths for each customer type (income, fail behavior, digital behavior) were stated in the dataset.

Model	Accuracy		
	Income	Fail Behavior	Digital Behavior
LSTM	63%	89%	97%
GNN+LSTM	70%	90%	97%

Table 4: Accuracy for each approach when predicting customer type. For each prediction, the model views the customer's most recent 20 events.

Model	Accuracy		
	length 1	length 5	length 15
LSTM	52%	40%	33%
GNN+LSTM	67%	62%	49%

Table 5: Accuracy for approach when predicting the next 1, 5, and 15 customer events. For each prediction, the model views the customer's most recent 20 events.

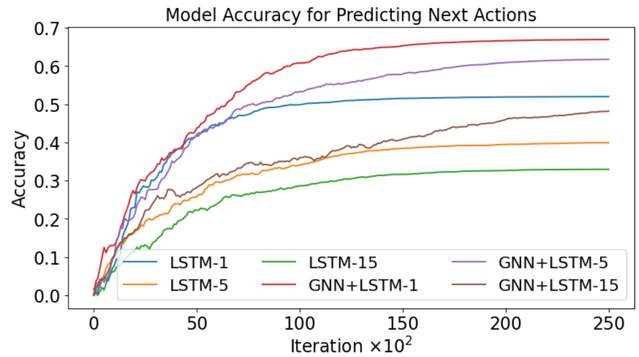

Figure 2: Loss function of LSTM and GNN+LSTM approaches when predicting the next 1, 5, and 15 actions of the customer when viewing the customer's last 20 actions.

when predicting the future events of customers. While the accuracy on this task are notably lower than the other tasks, the space of possible predictions (i.e., the space of possible events) is far greater in trajectory prediction than in those other tasks. As is expected we see that as the model is required to forecast customer events farther into the future, its predictive efficacy decreases. However, we again see a large improvement in predictive efficacy when using the graph embedding over the bag-of-word embedding.

When examining the bag-of-words approach with an LSTM model and the graph approach with a combination of a GNN and LSTM, we observe consistent and significant improvement with the graph embedding. This is due in part to the non-myopic nature of the graph embedding in that it explicitly encodes all possible paths customers can take through each of the interfaces; as such the model to reason more efficiently about the ways in which current, or past events, influence future events. This is also visible in Figure 2, where we plot the loss functions of the LSTM and GNN+LSTM models.

Conclusions

We have provided a pipeline for predicting customer behavior, type, and objective, from observations of customers interacting with multiple bank interfaces (web, mobile, teller, ATM). This pipeline is comprised of both a feature extraction procedure which takes reordered customer "events" and builds a state-action graph as well as where states and actions are the result of hand-crafted features (Table 2). Second we use a combined GNN and LSTM architecture to make use of both the temporal and structural nature of customer interactions. We found this approach to be effective at each of the three predictive tasks, and to have consistent improvement over both baselines.

Future Work: Currently we are exploring extensions of our graph based approach to not only predict customer behavior, but also to modify customer behavior. For example, the bank may desire to migrate customers towards digital interfaces, rather than in-person interfaces, i.e., modifying customer type. To modify the behavior of customers which prefer in-person interfaces, we can first learn a state-action graph for those customers, which in turn allows us to surmise the perceived cost that those individuals have for using digital interfaces (unfamiliarity with mobile apps may increase and individual's perceived cost of taking actions within the bank's mobile app). With these perceived costs, the bank can then use targeted rewards (e.g., cash rewards for using the mobile app) to incentivize customers to use the mobile app. After using the mobile app several times, individuals would become more familiar with app and would thus have a lower perceived cost of using the mobile app compared to in-person interfaces. Banks could then use this reward shaping technique to help migrate customers to a desired interface with greater precision than simply offering blind rewards to all individuals. Behavior prediction, which is the current focus of this paper, is an essential first-step in modifying behavior as the ability to efficiently shape behavior is directly tied to the ability to predict that behavior.

Acknowledgements

This work was funded in part by J.P. Morgan AI Research. This paper was prepared for informational purposes 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.

References

Baeza-Yates, R.; Jiang, D.; Silvestri, F.; and Harrison, B. 2015. Predicting the next app that you are going to use. In *WSDM*, 285–294.

Baker, C. L.; Saxe, R.; and Tenenbaum, J. B. 2009. Action understanding as inverse planning. *Cognition*, 113(3): 329–349.

Bilal Zorić, A. 2016. Predicting customer churn in banking industry using neural networks. *Interdisciplinary Description of Complex Systems: INDECS*, 14(2): 116–124.

Borrajo, D.; Gopalakrishnan, S.; and Potluru, V. K. 2020. Goal recognition via model-based and model-free techniques. In *ICAPS Workshop on Financial Planning (FinPlan)*.

Borrajo, D.; and Veloso, M. 2020. Domain-independent generation and classification of behavior traces. *arXiv preprint arXiv:2011.02918*.

Harris, Z. S. 1954. Distributional structure. *Word*, 10(2-3): 146–162.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation*, 9(8): 1735–1780.

Kaminka, G.; Vered, M.; and Agmon, N. 2018. Plan recognition in continuous domains. In *AAAI*, 6202–6210.

Keren, S.; Gal, A.; and Karpas, E. 2019. Goal Recognition Design in Deterministic Environments. *JAIR*, 65: 209–269.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Rahman, M.; and Kumar, V. 2020. Machine learning based customer churn prediction in banking. In *ICECA*, 1196–1201.

Ramírez, M.; and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI*, 1121–1126.

Shvo, M.; Klassen, T. Q.; Sohrabi, S.; and McIlraith, S. A. 2020. Epistemic plan recognition. In *AAMAS*, 1251–1259.

Shvo, M.; and McIlraith, S. A. 2020. Active goal recognition. In *AAAI*, 9957–9966.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan Recognition as Planning Revisited. In *IJCAI*, 3258–3264.

Vered, M.; and Kaminka, G. A. 2017. Heuristic Online Goal Recognition in Continuous Domains. In *IJCAI*, 4447–4454.

Wayllace, C.; Hou, P.; Yeoh, W.; and Son, T. C. 2016. Goal recognition design with stochastic agent action outcomes. In *IJCAI*, 3279–3285.

Wayllace, C.; Keren, S.; Gal, A.; Karpas, E.; Yeoh, W.; and Zilberstein, S. 2020. Accounting for Partial Observability in Stochastic Goal Recognition Design: Messing with the Marauder's Map. In *ECAI*.

Wayllace, C.; and Yeoh, W. 2022. Stochastic Goal Recognition Design Problems with Suboptimal Agents. In *AAAI*, 9953–9961.

Wilken, N.; Cohausz, L.; Schaum, J.; Lüdtke, S.; and Stuckenschmidt, H. 2023. Investigating the Combination of Planning-Based and Data-Driven Methods for Goal Recognition. *arXiv preprint arXiv:2301.05608*.

Yaghini, M.; Zhiyan, T.; and Fallahi, M. 2011. A Prediction Model for Recognition of Bad Credit Customers in Saman Bank Using Neural Networks. In *ICDATA*, 1.

Value Detection Rate: A Performance Metric for Payments Fraud Detection

Danial Dervovic[1], Saeid Amiri[2], Michael Cashmore[1]

[1]JP Morgan AI Research. Edinburgh, UK
[2]JP Morgan AI Research. New York City, NY, USA
danial.dervovic@jpmchase.com, saeid.amiri@jpmchase.com, michael.cashmore@jpmorgan.com

Abstract

Any plan for preventing fraud in financial transactions networks requires a well specified metric for success. In the industry, a well-known metric used to benchmark payments fraud detection algorithms is *Value Detection Rate (VDR)*. This metric explicitly considers monetary savings due to prevented fraud in contrast to metrics in the academic literature. In this short note we motivate and introduce this metric via a formal description and show it constitutes a consistent asymptotically unbiased statistical estimator to a relevant probabilistic query about test data drawn iid from the transaction population. We show via experiments on two publically available fraud datasets that fraud prevention policies guided by VDR have smaller financial losses than those guided by the popular Precision-at-k ($P@k$) metric. Moreover, we empirically show the VDR estimate converges given a practical number of samples, recapitulating the limiting case proven formally.

Introduction

Payments Fraud is a damaging problem across industry, eroding trust, impacting customer loyalty and employee morale, alongside financial losses incurred due to the fraud itself and attendant regulatory action. Indeed the problem is widespread, with 71% of surveyed organisations being targeted by fraudsters in 2021 according to the study by the Association for Financial Professionals (2022). In the 2022 survey by PwC (2022), responding organisations reported total losses of US$42B.

Fraudsters typically belong to one of several archetypes (Saporta and Maraney 2022; Association for Financial Professionals 2022) where often there is a sequential component to their actions. For example, a fraudster may initially attempt several smaller fraudulent transactions involving a compromised account before attempting a larger transaction. On the defender's side, constraints on how many resources can be used to fight fraud are balanced against how much fraud can be prevented (Hassanzadeh et al. 2021).

These characteristics of the payments fraud prevention problem suggest this problem is well-suited to AI Planning methods and sequential decision-making algorithms, yet the literature is scant. The works by Shen and Kurshan (2021); Vimal et al. (2021) formulate fraud prevention as an MDP and use Deep Reinforcement Learning to solve this problem. Rigter et al. (2022) formulate an online task allocation problem that applies to fraud prevention as a hybrid MDP and solve it using a dynamic programming approach with state abstraction. The paper by Dervovic et al. (2021) considers fraud prevention as a constrained sequential decision-making problem, solved via a dynamic threshold based policy. Planning-based techniques are used by Borrajo, Veloso, and Shah (2021) for the – related, but different – task of Anti-Money Laundering.

Indeed, the vast majority of literature on applying AI to fraud detection uses Machine Learning or Data Mining approaches. This literature is too extensive to present here, so we reference a number of surveys on these approaches (Bolton and Hand 2002; Phua et al. 2010; Lucas and Jurgovsky 2020; Ali et al. 2022; Narayan, Madhu Kumar, and Chacko 2023).

In this work we describe the fraud detection problem and present an important metric, VDR, for evaluating fraud detection policies. Moreover, we show that VDR has several desirable theoretical and experimental characteristics.

Formulating Fraud Detection

Fraud detection is usually presented as a supervised learning problem with a large class imbalance. Approaches to imbalanced learning are extensively surveyed (Abd Elrahman and Abraham 2013; Wagle and Manoj Kumar 2023). Concretely, individual transactions are modelled as samples drawn from a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ corresponds to transaction features available to the defender or decision-maker responsible for detecting fraud. The data labels $\mathcal{Y} = \{0, 1\}$ are binary, with 1 corresponding to a true fraud and 0 corresponding to a non-fraudulent transaction. Each sample $(X, Y) \sim \mathcal{D}$ corresponds to one transaction. Notably, the fraudulent ($Y = 1$) population is orders of magnitude smaller than the non-fraudulent population ($Y = 0$). The defender has a corpus of training data drawn from $\mathcal{D}$. Their goal is then to prevent as many fraudulent transactions as possible at test time – with transactions drawn again from $\mathcal{D}$. In practise, the training data will be taken before a certain time period and test from the time period immediately following (Saporta and Maraney 2022; Le Borgne et al. 2022).

This testing procedure is carried out to allow for violations of the iid assumption on the data distribution, that is, the test data being drawn from some alternative distribution $\mathcal{D}'$. In this case the decision-maker wishes to be confident that any policy trained on data from $\mathcal{D}$ still performs well on $\mathcal{D}'$. The performance in fraud prevention is measured along a number of different axes in the literature.

Existing Performance Metrics

The fraud detection problem is usually stated in terms of unbalanced supervised classification, so most metrics are based on the test-set confusion matrix based on a fixed threshold, or threshold free methods such as Receiver Operator Characteristic (ROC) and Precision-Recall (PR) curves, with the latter being more popular (Boyd, Eng, and Page 2013; Saito and Rehmsmeier 2015). The work by Davis and Goadrich (2006) discusses the relationship between PR and ROC curves with respect to imbalanced classification. Hassanzadeh et al. (2021) explicitly consider tradeoffs in these metrics when there are constraints on the number of positive label assignments the defender can give – as is often the case in practise. The framework of *Cost-sensitive learning* (Elkan 2001) seeks to minimise the misclassification cost between positive and negative examples, even on an example-by-example basis (Bahnsen, Aouada, and Ottersten 2014), but often the methods are tuned to a specific classification algorithm. The Precision-@-k metric (Fan and Zhu 2011), or $P@k$, selects the transactions with top-k highest predicted fraud probability by a model and evaluates the precision on this set of data. Currently there is no consensus on the best metric to use for evaluating fraud detection policies (Le Borgne et al. 2022), but instead one must decide on the tradeoffs one is willing to make.

That being said, what is missing from these evaluation metrics is considerations of monetary costs incurred by undetected fraudulent transactions. The Value Detection Rate (VDR) is a metric used in industry that explicitly considers this. Namely, maximising VDR corresponds to minimising monetary costs. In this brief note we formally introduce VDR. We proceed to show some theoretical and experimental properties of the VDR metric, namely its estimator is asymptotically unnbiased and consistent. We empirically validate these theoretical bounds on public payments fraud datasets. Furthermore, we show that an inspection policy that is based on VDR minimises financial costs due to missed fraudulent transactions in contrast to existing fraud detection metrics.

Value Detection Rate

Value Detection Rate (VDR) is a quantity taking values in $[0, 1]$ designed to capture the financial damage prevented by a plan or policy. Normalising to $[0, 1]$ allows a fair performance comparison of the same policy across different data slices.

We assume that there is a trained classifier $\hat{y} : \mathcal{X} \to [0, 1]$, where the output $\hat{y}(x)$ is interpreted as the classifier's subjective probability that the transaction $x \in \mathcal{X}$ is fraudulent. For brevity we define the function $r : \mathcal{X} \to \mathbb{R}_{++}$ that denotes the monetary value of a transaction.

The classifier subjective probability and transaction value allow us to define a *scoring function*, $\Psi : \mathcal{X} \to \mathbb{R}_{++}$,

$$\Psi(x) = \hat{y}(x) \cdot r(x), \tag{1}$$

that is, the scoring function is the monetary value of the transaction weighted by the classifier's subjective probability that the transaction is fraudulent. Suppose we have a validation set, $D_n = \{(x_j, y_j)\}_{j=1}^n$, drawn iid from $\mathcal{D}$. We fix a proportion of transactions that are to be inspected $\epsilon \in (0, 1)$ and define the *inspection function*, $\psi_\epsilon : \mathcal{X} \to \{0, 1\}$,

$$\psi_\epsilon(x) = \mathbb{1}\{\Psi(x) \geq \Psi_\epsilon\}, \tag{2}$$

where

$$\Psi_\epsilon := \min\left\{\Psi(x_j) \,\middle|\, j \in \operatorname*{arg\,max}_{S \subset \{1,\dots,n\}} \left\{\sum_{i \in S} \Psi(x_i) \,\middle|\, |S| \leq \epsilon n\right\}\right\}. \tag{3}$$

The score threshold Ψ_ϵ represents the score needed to be included in the top ϵ-scoring fraction of the validation set D_n. The inspection function is the classifier that marks transactions for inspection ($\psi_\epsilon = 1$) that exceed the score threshold Ψ_ϵ. We note that while the mathematical formula is complicated, this is easy to implement – solving the inner optimisation problem by sorting $\Psi(x_i)$ in descending order and choosing the top $\lfloor \epsilon n \rfloor$ transactions is sufficient.

Given an inspection function ψ_ϵ we want to know how effective this inspection function is at surfacing fraud, prioritising the most valuable transactions. Let $(X, Y) \sim \mathcal{D}$ be the random variable corresponding to a transaction, along with its fraud label. The monetary value of a transaction, assuming it is fraudulent, is given by $\mathbb{E}[r(X) \mid Y = 1]$. The expected fraudulent value captured by our inspection function ψ_ϵ is given by $\mathbb{E}[\psi_\epsilon(X) \cdot r(X) \mid Y = 1]$. We construct a dimensionless quantity that summarises the effectiveness of an inspection function ψ_ϵ, the Value Detection Rate (VDR), VDR_ϵ.

Definition 1 (Value Detection Rate)**.** Let $\psi : \mathcal{X} \to \{0, 1\}$ be a fraud classifier. Then the Value Detection Rate (VDR) of ψ over $\mathcal{D}$ is defined as

$$\mathrm{VDR}(\psi) = \frac{\mathbb{E}_{(X,Y)\sim\mathcal{D}}[\psi(X) \cdot r(X) \mid Y = 1]}{\mathbb{E}_{(X,Y)\sim\mathcal{D}}[r(X) \mid Y = 1]},$$

where for brevity we denote the VDR of an inspection function ψ_ϵ as defined in Eq. (2) by $\mathrm{VDR}_\epsilon \equiv \mathrm{VDR}(\psi_\epsilon)$.

Intuitively, VDR encompasses the notion that we want any fraud prevention scheme to capture as much of the fraudulent monetary value as possible given a randomly sampled transaction. Note that a perfect classifier ψ will achieve $\mathrm{VDR}(\psi) = 1$. Moreover, the trivial classifier $\psi : x \mapsto 1$ also has a perfect VDR score – it is the restriction on inspecting only an ϵ-fraction of transactions while also measuring fraudulent value captured that renders VDR_ϵ an effective metric for measuring performance.

We note that there is an additional metric used in the industry, *Total Detection Rate (TDR)*, corresponding to the special case where we impose $r : x \mapsto 1$. TDR is very similar to Precision@k, where the sole difference is k is a constant number of transactions to inspect vs a fraction ϵ, indeed the metrics are identical when $k = \lceil \epsilon n \rceil$.

Estimating VDR

Let us now assume we have test data $D_m = \{(x_j, y_j)\}_{j=1}^m$ drawn iid from $\mathcal{D}$. How can we estimate VDR_ϵ using D_m. An estimator that immediately springs to mind is the following one.

$$\widehat{\mathrm{VDR}}_\epsilon := \frac{\sum_{i=1}^m \psi_\epsilon(x_i) r(x_i) y_i}{\sum_{i=1}^m r(x_i) y_i} \tag{4}$$

We shall see that this estimator is the appropriate one to use, as it is asymptotically unbiased and consistent. We first state a lemma that will greatly help in proving these propositions.

Lemma 2. *(Cochran 1977, Theorem 6.4) Suppose we have random variables U, V with finite variances S_u^2, S_v^2, correlation coefficient $\rho_{u,v}$, and finite means μ_U, μ_V such that $\mu_V \neq 0$. The limiting distribution of $\hat{\mu}_u / \hat{\mu}_v := \sum_{i=1}^n u_i / \sum_{i=1}^n v_i$, from random samples of size n from an infinite population, is normal $\mathcal{N}(\mu, \sigma^2)$ with*

$$\mu = \frac{\mu_U}{\mu_V}, \quad \sigma^2 = \frac{1}{n}\frac{\mu_U^2}{\mu_V^2}\left[\frac{S_u^2}{\mu_U^2} - \frac{2\rho_{u,v} S_u S_v}{\mu_U \mu_V} + \frac{S_v^2}{\mu_V^2}\right].$$

We can now show asymptotic unbiasedness and consistency. Note in the following we assume that the distribution $r(X)$ has a finite mean and variance.

Proposition 3 (VDR Asymptotic Unbiasedness)**.** *The estimator $\widehat{\mathrm{VDR}}_\epsilon$ is asymptotically unbiased, that is*

$$\mathbb{E}\left[\widehat{\mathrm{VDR}}_\epsilon - \mathrm{VDR}_\epsilon\right] \to 0 \quad as \quad m \to 0.$$

Proof. The estimator $\widehat{\mathrm{VDR}}_\epsilon$ is a ratio. Consider first the expectation of the numerator of (4),

$$\mathbb{E}[\textstyle\sum_{i=1}^m \psi_\epsilon(x_i) r(x_i) y_i] = \sum_{i=1}^m \mathbb{E}[\psi_\epsilon(x_i) r(x_i) y_i], \tag{5}$$

by linearity of expectations

$$= \mathbb{E}[\psi_\epsilon(x_i) r(x_i) \mathbb{1}\{y_i = 1\}] \tag{6}$$

collecting terms in the sum and rephrasing the y_i variables

$$= \mathbb{E}[\psi_\epsilon(X) r(X) \mid Y = 1]\mathbb{P}(Y = 1). \tag{7}$$

Similarly, for the denominator of (4) we have

$$\mathbb{E}[\textstyle\sum_{i=1}^m r(x_i) y_i] = m\mathbb{E}[r(X) \mid Y = 1]\mathbb{P}(Y = 1). \tag{8}$$

Thus the ratio of expectations of the numerator and denominator of (4) is $\mathbb{E}[\psi_\epsilon(X) r(X) \mid Y = 1] / \mathbb{E}[r(X) \mid Y = 1] = \mathrm{VDR}_\epsilon$. Indeed, from Lemma 2 we have for large m that the expectation of the ratio is the ratio of the expectations and so $\widehat{\mathrm{VDR}}_\epsilon$ constitutes an unbiased estimator of VDR_ϵ for a sufficiently large test set D_m. □

Proposition 4 (VDR Consistency)**.** *The estimator $\widehat{\mathrm{VDR}}_\epsilon$ is consistent, that is for all $\varepsilon > 0$*

$$\lim_{m\to\infty} \mathbb{P}\left[\left|\widehat{\mathrm{VDR}}_\epsilon - \mathrm{VDR}_\epsilon\right| > \varepsilon\right] = 0.$$

Proof. From the proof of Proposition 3 and Lemma 2 we have that $\widehat{\mathrm{VDR}}_\epsilon$ follows a normal distribution for large m, with mean VDR_ϵ. Notice that the variance is $O(1/m)$, from which consistency immediately follows. □

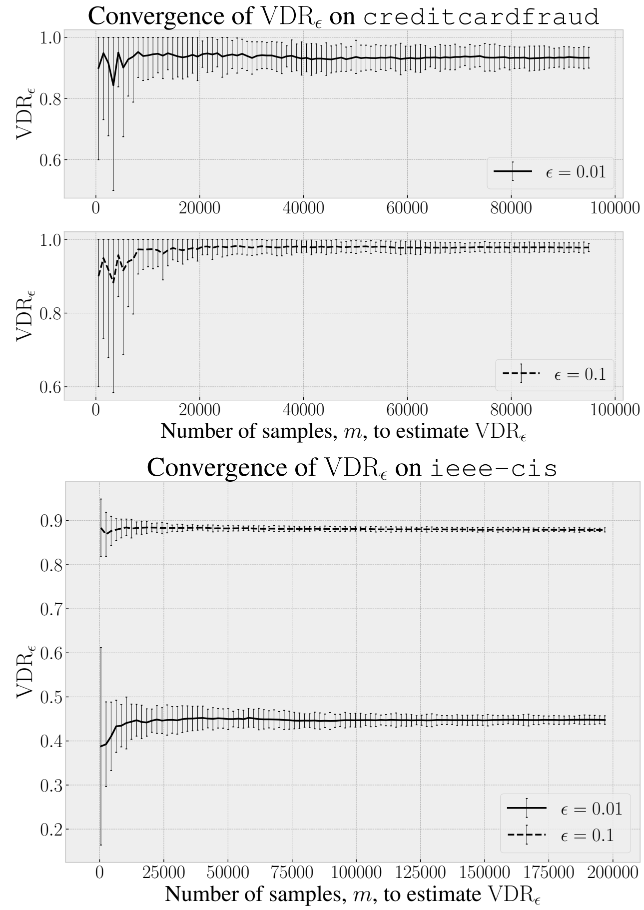

Figure 1: Plot of size of test set D_m against VDR_ϵ showing convergence for large number of samples on datasets `creditcardfraud` (top two panels) and `ieee-cis` (bottom panel). These give a practical estimate for the rates of convergence of Propositions 3 and 4. Reported error bars are standard deviation over 20 samples.

Experiments

In this section we conduct two experiments: *Experiment 1. VDR Convergence*; and *Experiment 2. VDR Effectiveness*. Our domain is fraud in financial transactions – there are few public datasets with realistic exemplars of this data (Le Borgne et al. 2022). We choose the `creditcardfraud` (Dal Pozzolo et al. 2015) and `ieee-cis` (IEEE-CIS 2019) datasets as they are based on real data, provide realistic features and explicit monetary values for transactions. Dataset preparation was identical to that in (Dervovic et al. 2021) and the model used was XGBoost (Chen and Guestrin 2016) with default hyperparameters.

In Experiment 1 (Figure 1) we vary the size of the test set D_m used to estimate VDR_ϵ for $\epsilon \in \{0.01, 0.1\}$, with m varying from 5000 to the size of the respective validation sets. There are 20 replicates sampled without replacement for each value of m, of which there were 100 linearly spaced values. From Figure 1 we see for both

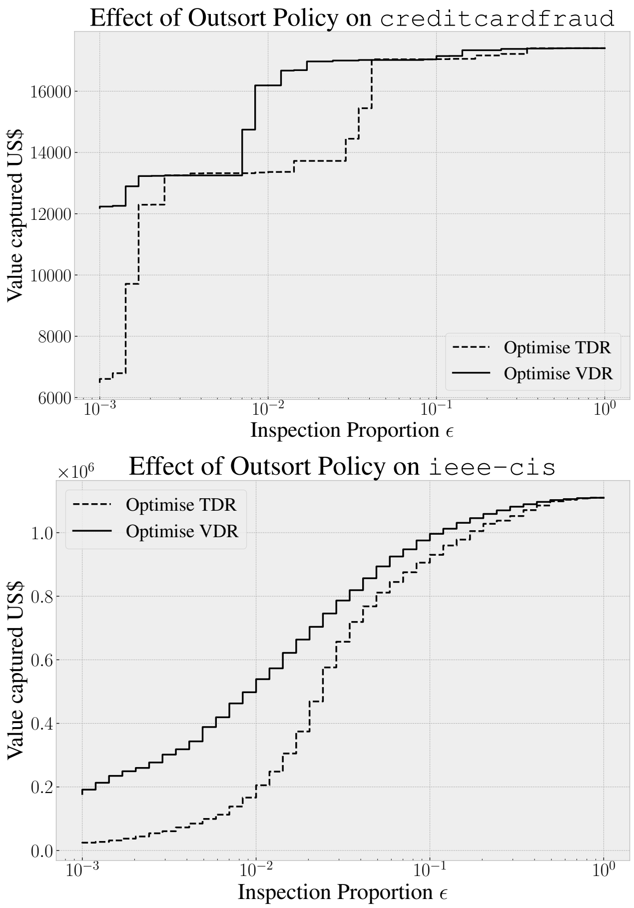

Figure 2: Curves of Fraudulent US$ captured against outsort rate $\epsilon \in (0, 1)$ using two different policies: Total Detection Rate (equiv. to $P@k$) and VDR. Top panel is `creditcardfraud` dataset and bottom panel is `ieee-cis` . Notice that using an outsorting policy informed by VDR captures more fraudulent value over a wide range of outsort rates. The x-axis is log-scaled showing this effect persists over multiple orders of magnitude.

`creditcardfraud` and `ieee-cis` that the VDR_ϵ estimates are converging to approximately normal after a reasonable number of samples m, as evidenced by the stabilisation in the size of the error bars. This empirically validates the theoretical asymptotic bounds established in Propositions 3 and 4.

In Experiment 2 we consider fraud prevention via static threshold policies, namely for a given threshold τ, there are two policies:

$$\begin{aligned} \text{VDR policy} &:= \text{Inspect } x \text{ iff } \Psi(x) \geq \tau, \\ \text{TDR policy} &:= \text{Inspect } x \text{ iff } \hat{y}(x) \geq \tau. \end{aligned} \tag{9}$$

If a truly fraudulent transaction is inspected its monetary value is said to be captured and if it is not inspected the money is assumed to be lost by the decision maker. The VDR policy is implictly optimising VDR as we are thresholding based on the value-weighted model score, whereas the TDR policy is thresholding solely on model score. The TDR policy stands in for optimising existing fraud detection metrics such as $P@k$.

For the VDR policy we take $\tau_\epsilon^{(\mathrm{VDR})} = \Psi_\epsilon$ as defined in Eq. (3) and for the TDR policy we assume $\tau_\epsilon^{(\mathrm{TDR})}$ takes the form

$$\tau_\epsilon^{(\mathrm{TDR})} := \min\left\{ \hat{y}(x_j) \,\middle|\, j \in \operatorname*{arg\,max}_{S \subset \{1,\dots,n\}} \left\{ \sum_{i \in S} \hat{y}(x_i) \,\middle|\, |S| \leq \epsilon n \right\} \right\}. \tag{10}$$

Both thresholds $\tau_\epsilon^{(\mathrm{VDR})}$ and $\tau_\epsilon^{(\mathrm{TDR})}$ are computed with respect to the model training data.

In Figure 2 we plot the fraudulent value captured in the test set using the VDR and TDR informed policies, over a range of ϵ values in $(0, 1)$ covering several orders of magnitude. For the `ieee-cis` dataset we see that the VDR policy strictly dominates the TDR policy for all ϵ and dominates for the `creditcardfraud` dataset. This confirms that using VDR to inform a fraud prevention policy gives superior monetary savings due to avoided fraud than existing metrics.

Conclusion

In this short paper we formally introduce a metric used for fraud detection in industry, VDR, and motivate its use in prevention plans and policies. VDR is shown to have several desirable statistical properties. We encourage the use of VDR in works by the academic community on planning for fraud detection and prevention.

References

Abd Elrahman, S. M.; and Abraham, A. 2013. A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013): 332–340.

Ali, A.; Abd Razak, S.; Othman, S. H.; Eisa, T. A. E.; Al-Dhaqm, A.; Nasser, M.; Elhassan, T.; Elshafie, H.; and Saif, A. 2022. Financial Fraud Detection Based on Machine Learning: A Systematic Literature Review. *Applied Sciences*, 12(19).

Association for Financial Professionals. 2022. 2022 AFP® Payments Fraud and Control Survey. Underwritten by JP Morgan. https://www.afponline.org/publications-data-tools/reports/survey-research-economic-data/Details/payments-fraud.

Bahnsen, A. C.; Aouada, D.; and Ottersten, B. 2014. Example-Dependent Cost-Sensitive Logistic Regression for Credit Scoring. In *Proc. ICMLA*, 263–269.

Bolton, R. J.; and Hand, D. J. 2002. Statistical Fraud Detection: A Review. *Statist. Sci.*, 17(3): 235–255.

Borrajo, D.; Veloso, M.; and Shah, S. 2021. Simulating and Classifying Behavior in Adversarial Environments Based on Action-State Traces: An Application to Money Laundering. In *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20. New York, NY, USA: Association for Computing Machinery.

Boyd, K.; Eng, K. H.; and Page, C. D. 2013. Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals. In *Proceedings of the 2013th European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part III*, ECMLPKDD'13, 451–466.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 785–794. New York, NY, USA: Association for Computing Machinery.

Cochran, W. G. 1977. *Sampling Techniques, 3rd Edition.* John Wiley.

Dal Pozzolo, A.; Caelen, O.; Johnson, R. A.; and Bontempi, G. 2015. Calibrating Probability with Undersampling for Unbalanced Classification. In *2015 IEEE Symposium Series on Computational Intelligence*, 159–166.

Davis, J.; and Goadrich, M. 2006. The Relationship between Precision-Recall and ROC Curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, 233–240. New York, NY, USA: Association for Computing Machinery.

Dervovic, D.; Hassanzadeh, P.; Assefa, S.; and Reddy, P. 2021. Non-parametric stochastic sequential assignment with random arrival times. *arXiv preprint arXiv:2106.04944*.

Elkan, C. 2001. The Foundations of Cost-Sensitive Learning. In *Proc. IJCAI*, 973–978.

Fan, G.; and Zhu, M. 2011. Detection of rare items with target. *Statistics and Its Interface*, 4(1): 11–17.

Hassanzadeh, P.; Dervovic, D.; Assefa, S.; Reddy, P.; and Veloso, M. 2021. Tradeoffs in streaming binary classification under limited inspection resources. In *Proceedings of the Second ACM International Conference on AI in Finance*, 1–9.

IEEE-CIS, I. C. I. S. 2019. IEEE-CIS Fraud Detection. https://www.kaggle.com/c/ieee-fraud-detection/datasets.

Le Borgne, Y.-A.; Siblini, W.; Lebichot, B.; and Bontempi, G. 2022. *Reproducible Machine Learning for Credit Card Fraud Detection - Practical Handbook.* Université Libre de Bruxelles.

Lucas, Y.; and Jurgovsky, J. 2020. Credit card fraud detection using machine learning: A survey. *arXiv preprint arXiv:2010.06479*.

Narayan, A.; Madhu Kumar, S. D.; and Chacko, A. M. 2023. A Review of Financial Fraud Detection in E-Commerce Using Machine Learning. In Bhateja, V.; Yang, X.-S.; Chun-Wei Lin, J.; and Das, R., eds., *Intelligent Data Engineering and Analytics*, 237–248. Singapore: Springer Nature Singapore.

Phua, C.; Lee, V.; Smith, K.; and Gayler, R. 2010. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*.

PwC. 2022. Global Economic Crime and Fraud Survey 2022. https://www.pwc.com/gx/en/services/forensics/economic-crime-survey.html.

Rigter, M.; Dervovic, D.; Hassanzadeh, P.; Long, J.; Zehtabi, P.; and Magazzeni, D. 2022. Optimal Admission Control for Multiclass Queues with Time-Varying Arrival Rates via State Abstraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 9918–9925.

Saito, T.; and Rehmsmeier, M. 2015. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3): e0118432.

Saporta, G.; and Maraney, S. 2022. *Practical Fraud Prevention: Fraud and AML Analytics for Fintech and ECommerce, Using SQL and Python*. O'Reilly Media, Incorporated.

Shen, H.; and Kurshan, E. 2021. Deep Q-Network-Based Adaptive Alert Threshold Selection Policy for Payment Fraud Systems in Retail Banking. In *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20. New York, NY, USA: Association for Computing Machinery.

Vimal, S.; Kayathwal, K.; Wadhwa, H.; and Dhama, G. 2021. Application of Deep Reinforcement Learning to Payment Fraud. In *Multi-Armed Bandits and Reinforcement Learning: Advancing Decision Making in E-Commerce and Beyond*, KDD '21.

Wagle, P. P.; and Manoj Kumar, M. V. 2023. A Comprehensive Review on the Issue of Class Imbalance in Predictive Modelling. In Shetty, N. R.; Patnaik, L. M.; and Prasad, N. H., eds., *Emerging Research in Computing, Information, Communication and Applications*, 557–576. Singapore: Springer Nature Singapore.

Accelerating Benders Decomposition via Reinforcement Learning Surrogate Models

Kyle Mana*, Stephen Mak*†, Parisa Zehtabi, Michael Cashmore, Daniele Magazzeni, Manuela Veloso

{kyle.mana, parisa.zehtabi, michael.cashmore, daniele.magazzeni, manuela.veloso}@jpmorgan.com, sm2410@cam.ac.uk

Abstract

Stochastic optimization (SO) attempts to offer optimal decisions in the presence of uncertainty. Often, the classical formulation of these problems becomes intractable due to a) the number of scenarios required to capture the uncertainty and b) the discrete nature of real-world planning problems. To overcome these tractability issues, practitioners turn to decomposition methods that divide the problem into smaller more tractable sub-problems. The focal decomposition method of this paper is Benders decomposition (BD), which decomposes stochastic optimization problems on the basis of scenario independence. In this paper we propose a method of accelerating BD with the aid of a surrogate model in place of an $\mathcal{NP}$-hard integer master problem. Through the acceleration method we observe 30% faster average convergence when compared to other accelerated BD implementations. In a working example, we introduce an RL agent as a surrogate and solve stochastic inventory management problems.

Introduction

Optimization is frequently subject to conditions of uncertainty. If this uncertainty is not sufficiently accounted for by a solution, even minor perturbations in the environment can devalue results and lead to catastrophic outcomes. While uncertainty can often be simulated or even parameterized, *solving* over that uncertainty offers incredible complexity. To make optimal decisions that consider both the uncertainty and constraints of a system, the field of SO is often applied. SO considers a distribution of possible scenarios rather than a deterministic event, and seeks an optimal outcome across the range of possibilities.

A common challenge for stochastic optimization is tractability. Generating an optimal decision that considers its outcome across a large number of scenarios can be extremely costly. To combat these computational issues, a common approach is to decompose the problem into simpler and independent sub-problems that can be combined to retrieve a global certificate of optimality. In this paper, we offer an adaptation of the common decomposition method of Benders decomposition (BD).

*Denotes equal contribution

†Work done while at JP Morgan AI Research

Despite wide usage since its introduction, BD suffers from two well-known practical limitations. First, in discrete space BD relies on an $\mathcal{NP}$-hard mixed-integer master problem (MIMP). This MIMP is responsible for making *global* decisions that are homogeneous across scenarios. Second, with each iteration a set of scenario-specific sub-problems (SP) generate gradient approximations that are passed to the MIMP as constraints (or cuts). The result is a MIMP with complexity that scales linearly with the number of required iterations as constraints are added.

Given these deficiencies, accelerating BD has become a compelling research problem. In production routing applications, Adulyasak et al. (2015) implement lower-bound lifting inequalities to tighten initial lower bounds, and exploit scenario grouping to reduce added complexity at each iteration. Baena et al. (2020) attempt to localize the loss approximation of BD by restricting each iteration to a subspace centered around strong past solutions. Crainic et al. (2016) aid initial iterations by including an informative subset of scenarios within the MIMP. Lee et al. (2021) offer a machine learning approach to predicting constraint importance; retaining only important cuts and limiting MIMP complexity. Each of these proposals has shown computational benefits, but remain solely dependent on the expensive MIMP to generate successive solutions. In contrast, Poojari and Beasley (2009) replace the MIMP with a genetic algorithm to produce faster feasible solutions. Although the heuristic produces fast master problem (MP) solutions, it is still reliant on SP approximations to undertstand scenario loss, and offers feasible as opposed to certifiably optimal solutions.

Our proposal introduces a surrogate model to quickly generate solutions to the discrete MP rather than relying on the MIMP. This surrogate generates fast solutions to unseen problems after learning the loss of decisions in similar stochastic environments. At varying rates, the MIMP is still run to retrieve the certificate of optimality offered by BD. In total, our contributions are:

- A generalized method of accelerating BD that retrieves optimal solutions to stochastic optimization problems while drastically reducing run times.
- A solution selection method that uses cuts from BD sub-problems to inform selection of future surrogate MP solutions; offering a further unification of the surrogate

MP within the BD framework.

- A worked inventory management problem with detailed implementation of the acceleration method. We offer an explicit Benders formulation, and leverage an RL model as our surrogate MP.
- Experiments showing a 30% reduction in run-time vs alternative acceleration methods.

Background

A widely used form of stochastic optimization is Sample Average Approximation (SAA). SAA aims to approximate loss over the distribution of possible scenarios using simulation. In SAA, R scenarios are simulated, with each simulation yielding its own deterministic sub-problem with a loss function $f(x, w, D_r)$, where x is a set of global decisions (universal across all scenarios), w is a cost vector, and D_r is a set of scenario-specific parameters. The total loss of the problem is then computed as an average of the loss across all scenarios,

$$\ell(x) = \frac{1}{R}\sum_{\forall r \in R} f(x, w, D_r) \tag{1}$$

Despite success in a number of optimal planning domains, the struggles of scaling SO problems are well documented. For example, Gendreau et al. (1996) note that when solving stochastic vehicle routing problems, practitioners commonly resort to comparing heuristics as exact methods become intractable. To combat scalability issues, decomposition methods are commonly employed to solve large-scale SO problems. Here we introduce the principles of Benders decomposition. Consider an SAA problem of the form:

$$\min_{x,y} c^T x + \frac{1}{R}\sum_{\forall r \in R} w^T y_r \tag{2}$$

subject to

$$Ax = b \tag{3}$$

$$Bx + D_r y_r = g, \qquad \forall r \in R \tag{4}$$

$$x \in \mathbb{Z}, y_r \in \mathbb{Z}^+, \qquad \forall r \in R \tag{5}$$

where x is again our set of global decisions, A, b, and B are parameters that define constraints on x, c is the cost of global decisions, D_r and g are scenario-specific parameters, y_r is a set of decisions made independently within each scenario, and w is a cost applied to each scenario-specific decision. In this formulation, $w^T y_r$ is equivalent to (1). The first step of BD is to separate the global decision variables x and scenario-specific decision variables y_r. This leaves us with a master problem

$$\{\min_{x,\theta} c^T x + \frac{1}{R}\sum_{\forall r \in R} \theta_r : Ax = b, x \in \mathbb{Z}^+\} \tag{6}$$

and a collection of R sub-problems, where for each $r \in R$ we have

$$\{\min_{y_r} w^T y_r : D_r y_r = g - Bx^*, y_r \in \mathbb{R}^+\} \tag{7}$$

The sub-problems accept a fixed x^* based on the solution to (6), and are solved to obtain optimal sub-problem decisions y_r. Note that BD introduces a set of auxiliary variables $\theta_r, \forall r \in R$ to the master problem (6). This auxiliary variable, frequently called the recourse variable, is responsible for tracking an approximation of the sub-problem loss that has been moved to (7). Let us assume the sub-problem is always feasible. This is not a necessary assumption, but simplifies the following description of BD.

Note that integrality on y_r has been relaxed in the sub-problem. This relaxation is necessary for Benders decomposition, and only possible when a) the sub-problem variables were not discrete to begin with or b) the decomposition results in a totally-unimodular sub-problem structure. Taking the dual of the sub-problem, we get:

$$\{\max_{q_r} q_r^T (g - Bx^*) : q_r^T D_r \leq w\} \tag{8}$$

The dual sub-problem has three essential properties. First, through strong duality the optimal value of (8) is equivalent to the optimal value of (7) at x^*. Second, the objective function (8) is linear with respect to the master problem decisions x. And lastly, with the optimal dual values of q_r^* we can establish

$$\{min_{y_r} w^T y_r : D_r y_r = g - Bx\} \geq q_r^{*T}(g - Bx), \forall x \in \mathbb{R}, \forall w \in \mathbb{R} \tag{9}$$

via weak duality. With these traits established, we see that the optimal dual SP objective $q_r^{*T}(g - Bx)$ can be included as a valid constraint on θ_r in the MIMP. These constraints serve as a sub-gradient approximations of the SP loss. For each SP solution, we can update the MIMP with the valid constraint of $\theta_r \geq q_r^{*T}(g - Bx)$ and re-solve for a new x. This process is repeated until the SP's do not offer any strengthening constraints on θ_r, indicating convergence and full approximation of SP loss. Figure 1 offers a visual representation of this process.

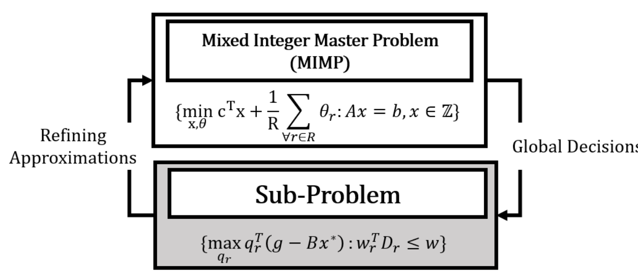

Figure 1: Iterative procedure of Benders decomposition, alternating between a MIMP (6) and SP (8).

Reinforcement Learning

Reinforcement learning (RL) offers a powerful approach to solving combinatorial problems. Delarue et al. (2020) gives one such example of RL applied to combinatorial problems, solving notoriously challenging capacitated vehicle routing problems using value-based methods. As shown in Delarue et al., the benefit of RL-based methods is that after learning the optimal policy they can generate actions in discrete space very quickly, albeit without a guarantee of optimality.

RL is typically based on the Markov Decision Process (MDP) framework as described by Sutton and Barto (2018). This can be defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is a set of transition probabilities from state s to the next state s', and $\mathcal{R}$ is the reward function. In temporal environments, we can adopt the notation of $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ for the state and action of a given time step t.

In RL, an agent attempts to learn the optimal action in a given state. Performance is measured by the collective rewards over future states and actions. The behaviors of the agent are updated based on prior experience, and can collectively be defined by a policy, $\boldsymbol{\pi}(s, a)$. RL algorithms can be broadly partitioned into two classes: value-based and policy based. In a value-based implementations, the policy $\boldsymbol{\pi}(s, a)$ is selected using value-function approximation methods, where

$$Q^{\boldsymbol{\pi}}(s_t, a_t) = \sum_{\forall j \in T} \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}[\gamma_t \mathcal{R}(s_{t+j}, a_{t+j})] \quad (10)$$

is the expected reward of an action, γ_t is a discount rate placed on future reward, and an optimal policy is deterministically selected based on $argmax_{\boldsymbol{\pi}} Q^{\boldsymbol{\pi}}$.

Rather than estimating the value-function Q and generating policies based on actions that maximize that approximation, policy-based reinforcement learning aims to optimize a functional representation of the policy $\boldsymbol{\pi}(s, a)$. We define the functional representation of a policy as $\boldsymbol{\pi}_{\boldsymbol{\beta}}(s, a)$, where $\boldsymbol{\beta}$ is a set of learned parameters. Importantly, in policy-based learning the agent optimizes the parameters $\boldsymbol{\beta}$ to generate a stochastic policy. This stochastic policy respects the fact that the cumulative reward for an action may not be deterministic, and consequentially a single best action may not exist.

Work from Sutton et al. (1999) introduces an optimization procedure for policy-based RL that updates the parameter set $\boldsymbol{\beta}$ via an estimate of the policy gradient. A powerful variation of policy-based optimization was introduced by Schulman et al. (2017) to avoid detrimentally large policy updates. In their version of policy-based optimization, the policy changes are regulated by limiting the reward of policy variation. Their method, titled Proximal Policy Optimization (PPO), updates the objective function to clip the reward of policy updates where the ratio $|\frac{\boldsymbol{\pi}_{\boldsymbol{\beta}}^{new}(a_t|s_t)}{\boldsymbol{\pi}_{\boldsymbol{\beta}}^{old}(a_t|s_t)}|$ extends beyond some ϵ.

Policy-based RL is a more applicable form of RL for our proposal, as it enables a set of diverse actions to be generated in a given state. Given a requirement for non-deterministic actions, our working example implements a PPO RL algorithm with a multi-layer neural network serving as our agent. The parameter set of this network, $\boldsymbol{\beta}$, defines our policy $\boldsymbol{\pi}_{\boldsymbol{\beta}}$.

Accelerating Benders Decomposition

With background on BD and RL provided, we introduce our proposed method of accelerating Benders decomposition. First, we will offer specifics on how a surrogate model is used in place of the MIMP. Then, we will introduce three possible mechanisms for selecting actions from the surrogate model. Lastly, we will offer a more thorough coverage of the theoretical benefits that the surrogate model provides, and known deficiencies of BD that it addresses.

Surrogate-MP

Recall the iterative procedure outlined in figure 1. The SP's can be solved efficiently using any standard LP solver, but each iteration calls back to a complex MIMP. Not only is the MIMP $\mathcal{NP}$-hard, but its complexity scales linearly with the number of iterations as a new constraint is added from the SP. Given these mechanics, there is a strong desire to a) increase the speed of each master problem iteration and b) decrease the total number of calls to the MIMP required. We achieve both results by periodically introducing a faster surrogate model in place of the MIMP (figure 2). This surrogate model can be any model that has learned to map the stochastic input space to the discrete decision space with intentions of minimizing the problem loss. Later in the paper, we introduce an RL agent as our surrogate model to generate master problem solutions. We call this framework Surrogate-MP.

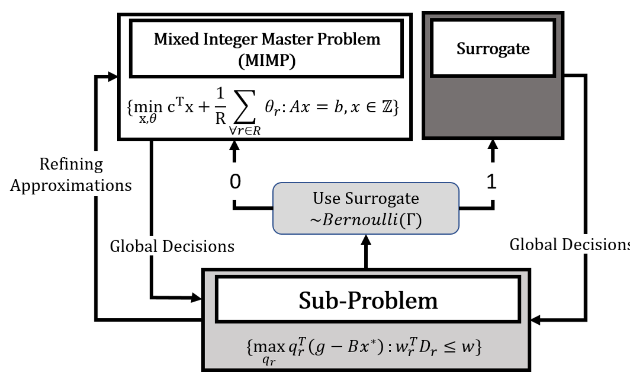

Figure 2: Iterative procedure of Surrogate-MP.

Note in this modified schema that with each iteration, the decision to use the surrogate in place of the MIMP is drawn from a Bernoulli distribution with a control parameter Γ. If a value of 1 is returned from the Bernoulli distribution, the surrogate is used to generate global decisions. Otherwise, the standard MIMP is run and the optimality gap can be confirmed. Regardless of whether the MIMP or surrogate are used, global decisions are passed to the sub-problem and loss approximating cuts are added.

Controlling Surrogate Usage

The surrogate model usage can be controlled in a variety of ways, and we offer three forms of control. These variants are aimed at answering 1) How frequently should we use the surrogate? 2) How can we be sure the surrogate solutions are useful for convergence? 3) If surrogate actions are non-deterministic, how can we decide which action are best to use? The three methods we implement are a greedy selection, weighted selection, and informed selection. Each of these methods assume the surrogate has generated a non-deterministic batch of actions for the given environment.

Greedy Selection The greedy selection process first evaluates every surrogate solution in a batch against expectations over the horizon to estimate solution performance. At each iteration, the decision to use the surrogate is made with some probability. If the surrogate is used, we select the top performing solution from the batch and use it as our MP solution. The solution is then removed from the batch and the process is continued.

Weighted Selection Rather than deterministically selecting actions based on their performance against an expectation, we can perform weighted random sampling. We again use the calculated loss of action i evaluated against an expected outcome, which we call ℓ_i. However, instead of selecting $argmin_i(\ell_i)$ as in the greedy method, we create a probability vector, where $p(i) = \frac{\frac{1}{\ell_i}}{\sum_{\forall i \in I} \frac{1}{\ell_i}}$. Using this probability vector, we perform weighted sampling from the batch of actions each time the surrogate is called.

Informed The final proposal is observed to be strongest in our experiments, and incorporates feedback from the BD sub-problems. With informed selection, surrogate solutions are selected using the constraint set currently placed on θ_r. The benefit of utilizing the constraint matrix to select surrogate solutions is that these constraints inherently motivate exploration to either a) minimal or b) poorly approximated regions of the convex loss. Given final convergence is defined by a binding subset of these constraints, it is necessary to explore these minimal or poorly approximated regions.

To describe the method, we introduce a constraint matrix $A_r \in \mathbb{R}^{I \times N}$ which contains the sub-gradient approximations imposed on θ_r, and a row vector of constant values $c_r \in \mathbb{R}^I$ that is added to each sub-gradient approximation. I refers to the iteration number of BD, N refers to the number of MP decision variables, and r refers to the scenario.

Note each iteration generates a new set of sub-gradient approximations that are added to the matrix. As mentioned, these are the same sub-gradients that are applied to θ_r in the master problem, and are generated using our dual sub-problem. On a given iteration, we have a batch of M solutions that have been generated by the surrogate. Decisions for this batch are represented by matrix $D \in \mathbb{Z}^{N \times M}$. We begin by computing the loss approximations of each gradient, for each of the M solutions. This is given as $TCA_r \in \mathbb{R}^{I \times M}$.

$$TCA_r = A_r \cdot O + (\mathbf{c_r} \cdot 1^{1 \times I})^T \tag{11}$$

The TCA_r matrix contains approximations of the sub-problem loss for each of the M solutions, generated by each of the I constraints currently placed on θ_r. We can now take the maximum value for each column M as the approximated cost of solution m. In LP terms, this maximum value relates to the binding constraint on θ_r in the MIMP, and is thus our true approximation of SP cost at that point. We represent this approximation (ℓ_{mr}) as:

$$\ell_{m,r} = \max_{\forall i \in I}(TCA_r)_{im} \tag{12}$$

Now we fully approximate the expected loss for each of the M solutions by taking an average across all R scenarios, and adding the fixed loss of that decision (denoted f_m).

$$\ell_m = \frac{1}{R} \sum_{\forall r \in R} \ell_{m,r} + f_m \tag{13}$$

the surrogate solution that minimizes the problem

$$argmin_m \ell_m \tag{14}$$

is then taken as our MP solution, and passed to the sub-problem for constraint generation.

Benefits of Surrogate-MP

The benefits of using a surrogate model with learned actions in place of the MIMP is based on two central principles.

1. The time required to generate solutions from a pre-trained surrogate model is negligible compared to the time required to solve a large scale MIP.
2. The surrogate model has learned its actions from past exposure to the stochastic environment. As a result, sub-problem loss is expressed in surrogate model solutions regardless of how well θ_r approximates SP loss. This means that even early iterations of the surrogate model will be highly reflective of sub-problem loss.

The first benefit is fairly self-explanatory; we desire faster MP solutions, and the surrogate provides them. The second benefit is more nuanced and worth expanding. We recall the general form MIMP (6), where θ_r offers an approximation of sub-problem loss that is refined through linear constraints generated by (8). It is well observed that this approximation can converge quickly if global decisions are localized to the optimal region, but it can also be very slow if global decisions are far from the optimal region or the cuts poorly approximate the loss (Crainic et al. (2016), Baena et al. (2020)). At initialization, θ_r has not received any feedback from the SP, and is instead bound by some heuristic or known lower bound (commonly $\theta_r \geq 0$ for non-negative loss). Given the lack of information initially imparted on θ_r, the MP generates global solutions that lack consideration of SP loss and can be very distant from the optimal region. Similar to a gradient based algorithm with a miss-specified learning rate, this can lead BD to oscillate around the minimal region or converge slowly, wasting compute and adding complexity with minimal benefit to the final solution (Baena et al. (2020)).

The surrogate mitigates this major issue by generating global decisions that reflect an understanding of their associated SP loss without requiring strong loss approximations on θ_r. As a result, initial global decisions generated by the surrogate are localized to the minimal region and cuts can quickly approximate the minimum of the convex loss. These two fundamental benefits are the basis for a 30% reduction in run-times, observed in experiments with the working example that follows.

Working Example

Let us introduce an inventory management problem (IMP) as a working example. In the proposed IMP, we assume the required solutions must a) choose a delivery schedule from a finite set, b) decide an order-up-to amount (order = order-up-to - current inventory) for each order day, and c) place costly emergency orders if demand cannot be met with current inventory. For simplicity we consider a single-item, single-location ordering problem where there is a requirement to satisfy all demand using either *planned schedules*, or more costly *just-in-time emergency orders*. The demand estimate is generated using a forecast model with an error term from an unknown probability distribution.

Adaptations of the general form IMP are applied in industries ranging from financial services, to brick-and-mortar retail. In e-commerce, vendors make decisions to either assume the holding costs associated with stocking inventory near demand locations, or use more costly fulfillment options to meet consumer needs (Arslan et al. (2021)). In commercial banking, cash must be held at physical locations and made available to customers when needed, with a compounding cost of capital being applied to any unused cash (Ghodrati et al. (2013)). Or in commodities trading, physical assets may need to be purchased and held until a desired strike price is realized in the future (Goel et al. (2011)).

SO Formulation and Decomposition

To model the IMP as a SO mixed-integer problem we introduce the following notation: let T be set of days t, R be set of scenarios r, and S define a finite set of schedules s. Holding cost of an item (per unit-of-measure, per day) is h, the cost of emergency services (per unit) is e, the penalty applied to over-stocking (per unit over-stocked) is q, and f_s is the fixed cost of a schedule. Capacity is defined by m and starting inventory by y. The parameter w_{st} indicates whether schedule s orders on day t. Demand on day t under scenario r is n_{tr}.

The decision space is defined by seven sets of variables. The decision to use schedule s is made using variable $u_s \in \{0,1\}$. The order-up-to amount is decided by $a_t \in \mathbb{Z}^+$, and $k_{tr} \in \mathbb{Z}$ is the required order quantity to meet the order-up-to amount. Inventory on hand is monitored by $d_{tr} \in \mathbb{Z}^+$, the units of holding space required to stock the inventory is $p_{tr} \in \mathbb{Z}^+$, the required emergency order quantity is $o_{tr} \in \mathbb{Z}^+$, and $v_{tr} \in \mathbb{Z}^+$ is the number of units that inventory is over-filled by (all defined $\forall t \in T, \forall r \in R$). The formulation of our IMP is

$$min \sum_{\forall s \in S}(u_s f_s) + \frac{1}{R}\sum_{\forall r \in R}\sum_{\forall t \in T}(p_{tr}h + o_{tr}e + v_{tr}q) \quad (15)$$

subject to:

$$d_{tr} = y - n_{tr} + k_{tr} - v_{tr} + o_{tr}, t = 0, \forall r \in R \quad (16)$$

$$d_{tr} = d_{t-1,r} + k_{tr} - n_{tr} + o_{tr} - v_{tr}, \forall t \in \{1,...,T\}, \forall r \in R \quad (17)$$

$$p_{tr} \geq y + k_{tr} - v_{tr}, t = 0, \forall r \in R \quad (18)$$

$$p_{tr} \geq a_t, \forall t \in \{1,...,T\}, \forall r \in R \quad (19)$$

$$p_{tr} \geq p_{t-1,r} - a_t, \forall t \in \{1,...,T\}, \forall r \in R \quad (20)$$

$$y + k_{tr} - v_{tr} \leq m, t = 0, \forall r \in R \quad (21)$$

$$d_{t-1,r} + k_{tr} - v_{tr} \leq m, \forall t \in \{1,...,T\}, \forall r \in R \quad (22)$$

$$k_{tr} = a_t - y\sum_{\forall s \in S} u_s w_{st} \quad (23)$$

$$k_{tr} \geq a_t - d_{t-1,r}, \forall t \in \{1,...,T\}, \forall r \in R \quad (24)$$

$$k_{tr} \leq a_t - d_{t-1,r} + (1 - \sum_{\forall s \in S} u_s w_{st})m, \forall t \in \{1,...,T\}, \forall r \in R \quad (25)$$

$$k_{tr} \leq a_t, \forall t \in T, \forall r \in R \quad (26)$$

$$k_{tr} \geq -\sum_{\forall s \in S} u_s w_{st} \times m, \forall t \in T, \forall r \in R \quad (27)$$

$$v_{tr} \leq a_t, \forall t \in T, \forall r \in R \quad (28)$$

$$a_t \leq \sum_{\forall s \in S} u_s w_{st} \times m, \forall t \in T \quad (29)$$

$$\sum_{\forall s \in S} u_s = 1 \quad (30)$$

The objective (15) minimizes the sum of planned schedule costs and the average of holding costs, emergency order costs, and over-fill costs across the R scenarios. Flow constraints (16) and (17) balance inflow and outflow of inventory through demand and deliveries. The holding cost is enforced by constraints (18), (19), and (20). Constraints (21) and (22) mandate that inventory cannot be filled beyond its capacity. Lastly, constraints (23), (24), (25), (26), (27), (28), and (29) ensure an order exactly fills the inventory to the optimal order-up-to-amount, and that orders are only placed on scheduled days. (30) guarantees exactly one schedule is selected.

For BD, we note that $\mathbf{a}$ and $\mathbf{u}$ are schedule and order-up-to decisions that must be made the same across all scenarios. As a result, $\mathbf{a}$, $\mathbf{u}$, (29), and (30) are contained in the MIMP while the remaining decision variables and constraints are delegated to the scenario specific sub-problems. For brevity, we omit the primal sub-problem formulation and directly introduce the cut-generating dual sub-problem formulation. We define the dual variables in line with their related constraints: $\boldsymbol{\alpha} \in \mathbb{R}$ [(16), (17)], $\boldsymbol{\gamma} \in \mathbb{R}^+$ [(18), (19)], $\boldsymbol{\omega} \in \mathbb{R}^+$ (20), $\boldsymbol{\phi} \in \mathbb{R}^+$ [(21),(22)], $\boldsymbol{\xi^0} \in \mathbb{R}$ (23), $\boldsymbol{\xi^{lb}} \in \mathbb{R}^+$ (24), $\boldsymbol{\xi^{ub}} \in \mathbb{R}^-$ (25), $\boldsymbol{\sigma} \in \mathbb{R}^-$ (26), $\boldsymbol{\pi} \in \mathbb{R}^+$ (27), $\boldsymbol{\beta} \in \mathbb{R}^-$ (28).

Master Problem

$$\min_{a,u,\theta} \sum_{\forall s \in S} (u_s \times f_s) + \frac{1}{R} \sum_{\forall r \in R} \theta_r \tag{31}$$

$s.t.$

$$a_t \leq \sum_{\forall s \in S} u_s w_{st} \times m, \forall t \in T \tag{32}$$

$$\sum_{\forall s \in S} u_s = 1 \tag{33}$$

$$\theta_r \geq 0, \forall r \in R \tag{34}$$

Dual Sub-problem (solved independently for each scenario r)

$$\begin{aligned}
\max_{\alpha,\phi,\xi^0,\xi^{lb},\xi^{ub},\sigma,\pi} \quad & \alpha_{0r}(y - n_0 r) + \\
& \gamma_{0r} y + \\
& \xi^0_r (a_0 - y \times \sum_{\forall s \in S} u_s w_{s0}) + \\
& \phi_{0r}(m - y) + \\
& \sum_{t=1}^{T} (-\alpha_{tr} n_{tr} + \gamma_{tr} a_t - \omega_{tr} a_t + \phi_{tr} m + \\
& \xi^{lb}_{tr} a_{tr} + \xi^{ub}_{tr}(a_{tr} + (1 - \sum_{\forall s \in S} u_s w_{st}) m)) + \\
& \sum_{\forall t \in T} (\beta_{tr} a_t + \sigma_{tr} a_t - \pi_{tr}(\sum_{\forall s \in S} u_s w_{st} \times m))
\end{aligned} \tag{35}$$

$s.t.$

$$\alpha_{tr} \leq 0, t = T \tag{36}$$

$$\alpha_{tr} - \alpha_{t+1,r} + \phi_{t+1,r} + \xi^{ub}_{t+1,r} + \xi^{lb}_{t+1,r} \leq 0, \forall t \in \{0, ..., T-1\} \tag{37}$$

$$\gamma_{tr} - \omega_{t+1,r} \leq h, t = 0 \tag{38}$$

$$\gamma_{tr} + \omega_{tr} \leq h, t = T \tag{39}$$

$$\gamma_{tr} - \omega_{t+1,r} + \omega_{tr} \leq h, \forall t \in \{1, ..., T-1\} \tag{40}$$

$$\alpha_{tr} + \beta_{tr} - \phi_{tr} + \gamma_{tr} \leq q, t = 0 \tag{41}$$

$$\alpha_{tr} + \beta_{tr} - \phi_{tr} \leq q, \forall t \in \{1, ..., T\} \tag{42}$$

$$-\alpha_{tr} \leq p, \forall t \in T \tag{43}$$

$$\xi^0_r - \gamma_{tr} + \phi_{tr} - \alpha_{tr} + \sigma_{tr} + \pi_{tr} = 0, t = 0 \tag{44}$$

$$\xi^{lb}_{tr} + \xi^{ub}_{tr} + \phi_{tr} - \alpha_{tr} + \sigma_{tr} + \pi_{tr} = 0, \forall t \in \{1, ..., T\} \tag{45}$$

Let us refer to the polyhedron defined by MP constraints at iteration i as $\mathcal{P}_i$. The master problem generates optimal decisions $\mathbf{a}^*$ and $\mathbf{u}^*$ given the current approximation of sub-problem costs on $\boldsymbol{\theta}$. The objective function of the dual sub-problem (referred to as $\mathcal{L}(\mathbf{a}, \mathbf{u}, r)$, where r is the scenario) is updated with $\mathbf{a}^*$ and $\mathbf{u}^*$, and the sub-problem is solved. Recalling the mechanics of BD, the optimal solution to the dual sub-problem has two valuable properties: a) as a numeric value it defines the true scenario specific costs, and b) as a function it offers a sub-gradient on θ_r. The master problem polyhedron is then updated to $\mathcal{P}_{i+1} = \mathcal{P}_i \cap \{\mathbf{u}, \mathbf{a}, \boldsymbol{\theta} : \theta_r \geq \mathcal{L}^*(\mathbf{a}, \mathbf{u}, r)\}$, where $\mathcal{L}^*(\mathbf{a}, \mathbf{u}, r)$ refers to the optimized loss function of the sub-problem iteration. This process is repeated until convergence, with each iteration of the MP being solved over a more refined approximation of sub-problem costs.

RL Surrogate - Formulation

We leverage an RL agent as the surrogate model in our Surrogate-MP implementation. The state of our IMP is represented by the tuple $s_t = \langle d, h, e, q, m, \mu, \sigma, \mathbf{w}, \mathbf{o}, \mathbf{r} \rangle \in \mathcal{S}$, where t is a time step over the horizon T. Parameters d, h, e, q, $\mathbf{w}$, and m directly follow the definitions introduced in the SO Formulation and Decomposition section (page 5). Additional state parameters include μ as the expected demand, and σ as the estimated standard deviation of demand. A vector $\mathbf{o}$ tracks orders over the time horizon T. All future orders are set to zero, and past orders are taken from actions as they are performed. Similarly, a vector $\mathbf{r}$ tracks the forecast errors from past observations. All future error observations are set to zero, and events are populated as they are observed by the state.

The actions are represented by $\langle \mathbf{k}_t, \mathbf{u}_t \rangle \in \mathcal{A}$ which denotes (a) the quantity to order, and (b) the schedule to adhere to, at time t respectively. Note that the schedule must be determined at the beginning of the horizon, and thus only $\mathbf{u}_{t=0}$ is relevant. This is enforced through action masking and for simplicity we will refer to $\mathbf{u}_{t=0}$ as $\mathbf{u}$. The reward is negative cost, as defined by the objective (15).

As previously mentioned, we use PPO to optimize a multi-layer neural network as our agent. The network is a feed-forward neural network with two hidden layers and two linear output layers. The linear output layers return the log-odds that define our stochastic action space. We standardize the network inputs (the state) to be mean centered with unit variance, and generate $\mathbf{k}_t$ and $\mathbf{u}_t$ sequentially $\forall t \in T$.

The agent is presented with an initial state s_0 and must select a scheduling action to take. This scheduling action, $\mathbf{u}$, relates to a binary vector $\mathbf{w} \in \{0,1\}^T$ that defines whether an order is possible on day t. If $\mathbf{w}_t = 1$, an order can be placed, otherwise the agent cannot order. This schedule becomes part of the state, over-writing the initial zero vector $\mathbf{w}$.

With the schedule defined, the agent must generate a second action for state s_0; this time selecting an order amount. The repeated visitation of state s_0 is necessary as the selected schedule $\mathbf{w}$ has now become part of the state. While we have not temporally shifted, the state has changed.

After a second visitation of s_0, the agent sequentially traverses the horizon T. With each time step, an ordering decision is made and either accepted or masked depending on the schedule vector $\mathbf{w}$. Updates to the state include population of order quantities, residual updates, and an update of the inventory on hand based on observed demand and order amounts. If an order is scheduled, we retrieve the order-up-to amounts, denoted as $\mathbf{a}$ in the SO Formulation section, by adding inventory on hand at the end of $t-1$ to the ordering decision k_t. If an order is not scheduled the order-up-to amount is 0. After traversing the full horizon T, the agent will have selected a schedule $\mathbf{u} \in \{0,1\}^S$ and have a vector of order-up-to quantities $\mathbf{a} \in \mathbb{Z}^T_{\geq 0}$ from the agent. These two decision vectors, $\mathbf{u}$ and $\mathbf{a}$ are the essential ingredients required by the BD sub-problem. With these vectors, we can solve (35), generate sub-gradient approximations on θ_r, and further refine our approximation of true, stochastic, sub-problem cost.

Experiments

To evaluate the Surrogate-MP method, we implement our IMP formulation across 153 independent cases using real-world data. Each experiment was performed with a sample size of 500 scenarios ($R = 500$), a horizon of 28 days ($T = 28$), and 169 possible schedules ($S = 169$). The resultant problem has a high dimensional discrete decision space, consisting of scheduling and ordering decisions. In total, the decision space is $\mathbb{Z}^{70,197}$. Experiments were run on a 36 CPU, 72 GB RAM c5.9xlarge AWS instance. For solving the integer master problem and linear sub-problems, we leveraged the CPLEX commercial solver with default settings, allowing for distribution across the 36 CPU machine. We experimented with all three surrogate solution selection methods: greedy, random, and informed. For every implementation of Surrogate-MP, we deactivate calls to the surrogate after the optimality gap is $\leq$ 5%. The intuition behind deactivating the surrogate model is that as the gap percent shrinks, the MIMP must be used to retrieve the certificate of optimality.

As a benchmark, we evaluate our method against a baseline implementation of Benders decomposition. Accelerations implemented in the baseline include scenario group cuts (Adulyasak et al. (2015)) and partial decomposition (Crainic et al. (2016)). We did not compare against a generic implementation of Benders decomposition due to tractability issues.

Results

All three implementations (greedy, random, and informed) produced faster convergence than the benchmark BD implementation. Random implementation performed 14.96% (104.51s average run-time) faster than the baseline, greedy implementation achieved 19.43% (99.92s average run-time) faster performance, and the informed surrogate implementation performed 30.45% faster (85.47s average run-time). The convergence rates are displayed in figure 3.

Method	Avg Run-time (seconds)
No Surrogate	122.90
Random Surrogate	104.51
Greedy Surrogate	99.92
Informed Surrogate	85.47

In addition to acheiving faster average convergence, Surrogate-MP outperformed the baseline BD implementation across the majority of instances. Surrogate-MP with informed selection achieved better convergence rates on 135 of the 153 instances (88.24%, figure 4).

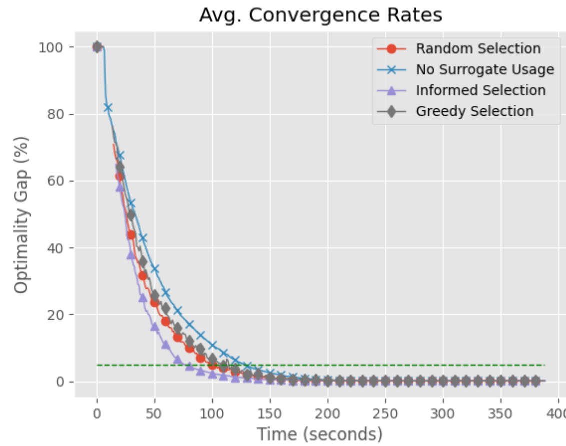

Figure 3: Convergence rates of a baseline BD, and Surrogate-MP with three selection methods (greedy, random, informed).

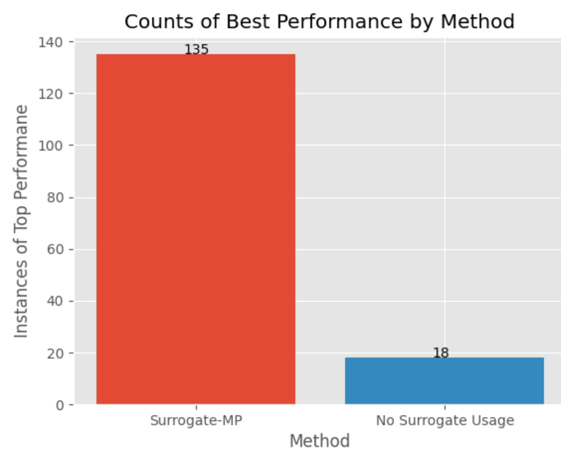

Figure 4: Count of instances with faster convergence between Surrogate-MP and a baseline BD implementation.

Acknowledging the strong performance of Surrogate-MP with informed selection, we continued our experiments by testing different frequencies of surrogate model usage. We leveraged three different rate parameters Γ that control whether to use the surrogate model during each iteration ($[0, 1] \sim \text{Bernoulli}(\Gamma)$). We experimented with $\Gamma = 0.25, 0.50, \text{and } 0.75$. For every value of Γ we use Surrogate-MP with informed selection. We observe in figures 5 and 6 that more frequent surrogate model usage results in improved convergence, with optimal convergence rates being generated by $\Gamma = 0.75$.

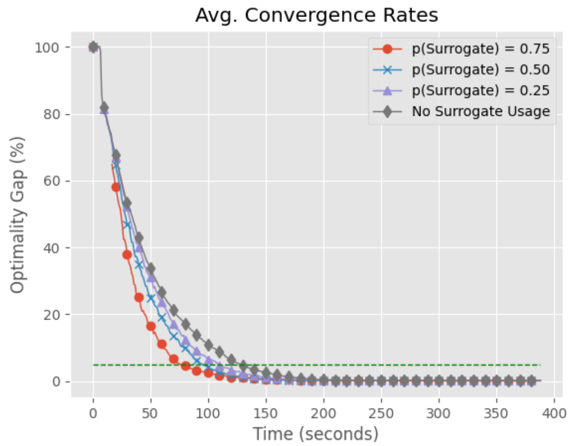

Figure 5: Convergence rates for different levels of informed surrogate usage. The dotted line indicates a gap of 5% (the point at which we deactivate the Surrogate-MP).

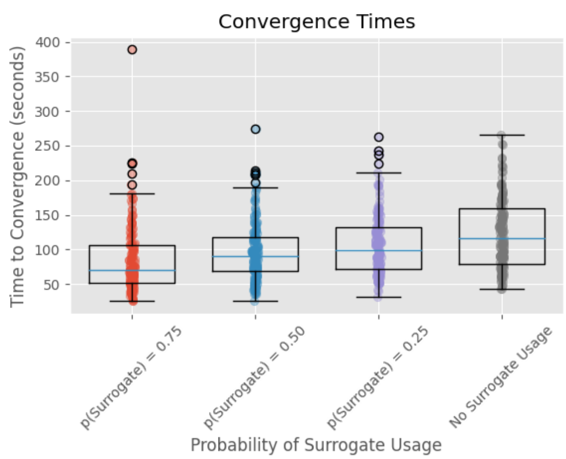

Figure 6: Convergence instances of BD accelerated by an informed Surrogate-MP, with different surrogate usages.

Conclusion & Future Work

In conclusion, by inserting a surrogate model in place of the MIMP we achieve a drastic reduction in convergence time. The proposed method is generalizable to any BD implementation, retreives certificates of optimality, and any surrogate capable of generating MP solutions can be used. We leverage an RL agent as our surrogate, and display results showing superiority in 88.24% of instances with a 30% reduction in average run time.

Observing the performance of our method, a promising extension of this work would be to design stronger integration between the surrogate model, SP, and MP. We took steps toward integration with the informed method of selecting surrogate solutions, and realized promising results. Some opportunities for integration we leave unexplored would be to directly inform the surrogate model on the strength of past solutions, offer sub-gradient information as a feature, or redesign the surrogate objective function to focus on weakly approximated areas of the SP loss as opposed to mirroring the BD objective directly. We are additionally eager to observe the performance of Surrogate-MP on other discrete SO problems.

References

Adulyasak, Y.; Cordeau, J.-F.; and Jans, R. 2015. Benders Decomposition for Production Routing Under Demand Uncertainty. Operations Research, 63(4): 851–867.

Arslan, A. N.; Klibi, W.; and Montreuil, B. 2021. Distribution network deployment for omnichannel retailing. European Journal of Operational Research, 294(3): 1042–1058.

Baena, D.; Castro, J.; and Frangioni, A. 2020. Stabilized Benders Methods for Large-Scale Combinatorial Optimization, with Application to Data Privacy. Management Science, 66(7): 3051–3068.

Crainic, T. G.; Hewitt, M.; Maggioni, F.; and Rei, W. 2016. Partial Benders Decomposition Strategies for Two-Stage Stochastic Integer Programs.

Delarue, A.; Anderson, R.; and Tjandraatmadja, C. 2020. Reinforcement learning with combinatorial actions: An application to vehicle routing. Advances in Neural Information Processing Systems, 33: 609–620.

Gendreau, M.; Laporte, G.; and Séguin, R. 1996. Stochastic vehicle routing. European journal of operational research, 88(1): 3–12.

Ghodrati, A.; Abyak, H.; and Sharifihosseini, A. 2013. ATM cash management using genetic algorithm. Management Science Letters, 3(7): 2007–2041.

Goel, A.; and Gutierrez, G. J. 2011. Multiechelon procurement and distribution policies for traded commodities. Management Science, 57(12): 2228–2244.

Lee, M.; Ma, N.; Yu, G.; and Dai, H. 2021. Accelerating Generalized Benders Decomposition for Wireless Resource Allocation. IEEE Transactions on Wireless Communications, 20(2): 1233–1247.

Poojari, C.; and Beasley, J. 2009. Improving benders decomposition using a genetic algorithm. European Journal of Operational Research, 199(1): 89–97.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. CoRR, abs/1707.06347.

Sutton, R. S.; and Barto, A. G. 2018. Reinforcement Learning: An Introduction. The MIT Press, second edition.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Solla, S.; Leen, T.; and Müller, K., eds., Advances in Neural Information Processing Systems, volume 12. MIT Press.

Deep Reinforcement Learning for Optimal Portfolio Allocation: A Comparative Study with Mean-Variance Optimization

Srijan Sood, [1] **Kassiani Papasotiriou,** [1] **Marius Vaiciulis,**[2] **Tucker Balch** [1]**.**

[1] J.P. Morgan AI Research

[2] J.P. Morgan Global Equities; Oxford-Man Institute of Quantitative Finance

{srijan.sood, kassiani.papasotiriou, marius.vaiciulis, tucker.balch}@jpmorgan.com

Abstract

Portfolio Management is the process of overseeing a group of investments, referred to as a portfolio, with the objective of achieving predetermined investment goals and objectives. Portfolio Optimization is a key component that involves allocating the portfolio assets so as to maximize returns while minimizing risk taken. Portfolio optimization is typically carried out by financial professionals who use a combination of quantitative techniques and investment expertise to make decisions about the portfolio allocation.

Recent applications of Deep Reinforcement Learning (DRL) have shown promising results when used to optimize portfolio allocation by training model-free agents on historical market data. Many of these methods compare their results against basic benchmarks or other state-of-the-art DRL agents but often fail to compare their performance against traditional methods used by financial professionals in practical settings. One of the most commonly used methods for this task is Mean-Variance Portfolio Optimization (MVO), which uses historical timeseries information to estimate expected asset returns and covariances, which are then used to optimize for an investment objective.

Our work is a thorough comparison between model-free DRL and MVO for optimal portfolio allocation. We detail the specifics of how to make DRL for portfolio optimization work in practice, also noting the comparable tweaks needed to get MVO operational. Backtest results display strong improved performance of the DRL agent in terms of many performance metrics, including Sharpe ratio, maximum drawdowns, and absolute returns.

Introduction

Portfolio management is a key issue in the financial services domain. It constitutes allocating funds across a diverse variety of assets, typically to generate uncorrelated returns while minimizing risk and operational costs. Portfolios can constitute holdings across asset classes (cash, bonds, equities, etc.), or can also be optimized within a specific asset class (e.g., picking the appropriate composition of stocks for an equity portfolio). Investors may choose to optimize for various performance criteria, often centered around maximizing portfolio returns relative to risk taken. Since the advent of Modern Portfolio Theory (Markowitz 1952), a lot of progress has been made in both theoretical and applied aspects of portfolio optimization. These range from improvements in the optimization process, to the framing of additional constraints that might be desirable to rational investors (Cornuejols and Tütüncü 2006; Li and Hoi 2014; Kalayci et al. 2017; Ghahtarani, Saif, and Ghasemi 2022). Recently, the community has tapped the many advancements in Machine Learning (ML) to aid with feature selection, forecasting and estimation of asset means and covariances, as well as using gradient based methods for optimization.

Concurrently, the past decade has witnessed the success of Reinforcement Learning (RL) in the fields of gaming, robotics, natural language processing etc. (Silver et al. 2017; Nguyen and La 2019; Su et al. 2016). The sequential decision making nature of Deep RL, along with its success in applied settings, has captured the attention of the finance research community. In particular, some of the most popular areas of focus of the application of DRL in finance have been on automated stock trading (Yang et al. 2020; Théate and Ernst 2021; Zhang, Zohren, and Roberts 2020; Wu et al. 2020), risk management through deep hedging (Buehler et al. 2019; Du et al. 2020; Cao et al. 2021; Benhamou et al. 2020b) and portfolio optimization. In the upcoming section, we'll examine the landscape of DRL in portfolio optimization and trading problems. While these approaches exhibit improved performance than previous studies, they do have some shortcomings. For instance, some generate discrete asset trading signals which limit their use in broader portfolio management. Additionally, majority of these approaches compare results against ML or buy-and-hold baselines, and don't consider classical portfolio optimization techniques, such as Mean-Variance Optimization.

In our work, we aim to compare a simple and robust DRL framework, that was designed around risk-adjusted returns, with one of the traditional finance methods for portfolio optimization, MVO. We train policy gradient based agents on a multi asset trading environment that simulates the US Equities market (using market data replay), and create observation states derived from the observed asset prices. The agents optimize for risk-adjusted returns, not dissimilar to the traditional MVO methods. We compare the performance of the DRL strategy against MVO through a series of systematic backtests, and observe improved performance along many performance metrics, including risk adjusted returns, max drawdown, and portfolio turnover.

Related Work

There is a lot of recent research interest into the application of Deep RL in trading and portfolio management problems. For portfolio optimization, a lot of the research focuses on defining various policy network configurations and reports results that outperform various traditional baseline methods (Wang et al. 2019; Liang et al. 2018; Lu 2017; Jiang and Liang 2017; Wang et al. 2021; Deng et al. 2016; Cong et al. 2021). Other work explores frameworks that inject information in the RL agent's state by incorporating asset endogenous information such as technical indicators (Liu et al. 2020; Sun et al. 2021; Du and Tanaka-Ishii 2020) as well as exogenous information such as information extracted from news data (Ye et al. 2020; Lima Paiva et al. 2021).

The current benchmarks for DRL frameworks typically involve comparing results against other DRL or ML approaches, a buy-and-hold baseline, or market/index performance. However, these benchmarks may be overly simplistic or provide only a relative comparison. To truly gauge the effectiveness of a DRL agent, it would be more meaningful to benchmark it against methodologies used by financial professionals in practice, such as Mean Variance Optimization (MVO).

While there are some approaches that compare DRL performance with MVO (Li et al. 2019; Koratamaddi et al. 2021; i Alonso, Srivastava et al. 2020), the comparison simply serves as another baseline, and the methodology is not clearly described because an in-depth comparison is not the primary focus of their study. To our knowledge, there is only one study that goes into a robust in-depth comparison of MVO and DRL (Benhamou et al. 2020a). However, across all these studies, there is usually a discrepancy between the reward function used to train the RL agent, and the objective function used for MVO (for e.g., daily returns maximization vs risk minimization). In order to make a fair comparison, it is crucial that both approaches optimize for the same goal. Additionally, some of these approaches provide exogenous information (e.g., signals from news data) to the DRL agent, which makes for a biased comparison with MVO. Additionally, none of these works provide implementation details for the MVO frameworks they used for their comparison. We aim to address these issues by conducting a robust comparison of Deep RL and Mean-Variance Optimization for the Portfolio Allocation problem.

Background

The goal of portfolio optimization is to continuously diversify and reallocate funds across assets with the objective of maximizing realized rewards while simultaneously restraining the risk. In practice, portfolio management often aims to not only maximize risk-adjusted returns but also to perform as consistently as possible over a given time interval (e.g. on a quarterly or yearly basis).

Markowitz introduced the modern portfolio theory (MPT) (Markowitz 1952), a framework that allows an investor to mathematically balance risk tolerance and return expectations to obtain efficiently diversified portfolios. This framework relies on the assumption that a rational investor will prefer a portfolio with less risk for a specified level of return and concludes that risk can be reduced by diversifying a portfolio. In this section, we will introduce Mean Variance Optimization (MVO) – one of the main techniques of the MPT – which we later compare to the performance of our DRL framework. Additionally, we introduce RL preliminaries, describing the technique independent of portfolio optimization.

Mean-Variance Portfolio Optimization

Mean-Variance Optimization (MVO) is the mathematical process of allocating capital across a portfolio of assets (optimizing portfolio weights) to achieve a desired investment goal, usually: 1. Maximize returns for a given level of risk, 2. Achieve a desired rate of return while minimizing risk, or 3. Maximize returns generated *per* unit risk. Risk is usually measured by the volatility of a portfolio (or asset), which is the variance of its rate of return. For a given set of assets, this process requires as inputs the rates of returns for each asset, along with their covariances. As the true asset returns are unknown, in practice, these are estimated or forecasted using various techniques that leverage historical data.

This task is then framed as an optimization problem, single or multi-objective, which can be solved in a variety of ways (Cornuejols and Tütüncü 2006; Kalayci et al. 2017; Ghahtarani, Saif, and Ghasemi 2022).

A typical procedure is to solve it as a convex optimization problem and generate an efficient frontier of portfolios such that no portfolio can be improved without sacrificing some measure of performance (e.g., returns, risk). Let w be the weight vector for a set of assets, μ be the expected returns, the portfolio risk can be described as $w^T\Sigma w$, for covariance matrix Σ. To achieve a desired rate of return μ^*, we can solve the portfolio optimization problem:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & w^T\Sigma w \\ \text{subject to} \quad & w^T\mu \geq \mu^*, \\ & w_i \geq 0, \\ & \sum w_i = 1 \end{aligned}$$

Varying μ^* gives us the aforementioned efficient frontier.

Another common objective is the Sharpe Ratio (Sharpe 1998; Chen, He, and Zhang 2011), which measures the return per unit risk. Formally, for portfolio p, the Sharpe Ratio is defined as:

$$\text{Sharpe Ratio}_p = \frac{E[R_p - R_f]}{\sigma_p}$$

where R_p are the returns of the portfolio, σ_p is the standard deviation of these returns, and R_f is a constant risk-free rate (e.g., US Treasuries, approximated by 0.0% in recent history). Although tricky to solve in its direct form –

$$\max_w \frac{\mu^T w - R_f}{(w^T\Sigma w)^{1/2}}$$

– it can be framed as a convex optimization problem through the use of a variable substitution (Cornuejols and Tütüncü 2006). We choose the Sharpe Ratio as our desired objective function for this study as we can optimize for risk-adjusted returns without having to specify explicit figures for minimum expected returns or maximum risk tolerance.

Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning that refers to a class of techniques that involve learning by optimizing long-term reward sequences obtained by interactions with an environment (Sutton and Barto 2018). An environment is typically formalized by means of a Markov Decision Process (MDP). An MDP consists of a a 5-tuple (S, A, P_a, R_a, γ), where:

- S is a set of states
- A is a set of actions
- $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$
- $R_a(s, s')$ is the immediate reward received after transitioning from state s to state s', due to action a
- γ is a discount factor between $[0, 1]$ that represents the difference in importance between present and future rewards

A solution to an MDP is a policy π that specifies the action $\pi(s)$ that the decision maker will choose when in state s. The objective is to choose a policy π that will maximize the expected discounted sum of rewards over a potentially infinite horizon:

$$E\left[\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})\right]$$

The field of Deep Reinforcement Learning (DRL) leverages the advancements in Deep Learning by using Neural Networks as function approximators to estimate state-action value functions, or to learn policy mappings π. These techniques have seen tremendous success in game-playing, robotics, continuous control, and finance (Mnih et al. 2013; Berner et al. 2019; Nguyen and La 2019; Hambly, Xu, and Yang 2021; Charpentier, Elie, and Remlinger 2021).

RL for Portfolio Allocation Given its success in stochastic control problems, RL extends nicely to the problem of portfolio optimization. Therefore, it is not surprising that the use of DRL to perform tasks such as trading and portfolio optimization has received a lot of attention lately. Recent methods focus on learning deep features and state representations, for example, through the use of embedding features derived from deep neural networks such as autoencoders and LSTM models. These embeddings capture price related features which can range from technical indicators (Wang et al. 2019; Soleymani and Paquet 2020; Wang et al. 2021), to information extracted from news in order to account for price fluctuations (Ye et al. 2020). Other proposed features use attention networks or graph structures (Wang et al. 2021, 2019) to perform cross-asset interrelationship feature extraction.

Problem Setup

We frame the portfolio optimization problem in the RL setting. As described in the Background section, RL entails learning in a framework with interactions between an agent and an environment. For the portfolio optimization setting, we create an environment that simulates the US Equities market (using market data replay), and create observation states derived from the observed asset prices. The agent's actions output a set of a portfolio weights, which are used to rebalance the portfolio at each timestep.

Actions

For portfolio allocation over N assets, an agent selects portfolio weights $\boldsymbol{w} = [w_1, \ldots, w_n]$ such that $\sum_{i=1}^{N} w_i = 1$, where $0 \leq w_i \leq 1$. An asset weight of 0 indicates zero holdings of a particular asset in a portfolio, whereas a weight of 1 means the entire portfolio is concentrated in said asset. In extensions of this framework, $w_i < 0$ would allow for shorting an asset, whereas $w_i > 1$ indicates a leveraged position. However, for our case, we restrict actions to non-leveraged long-only positions. These constraints can be enforced by applying the softmax function to an agent's continuous actions.

States

An asset's price at time t is denoted by P_t. The one-period simple return is defined as $R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$. Consequently, the one-period gross return can be defined as $\frac{P_t}{P_{t-1}} = R_t + 1$. Further, we can define the one-period log return as $r_t = log(\frac{P_t}{P_{t-1}}) = log(R_t + 1)$. For our setting, we choose the time period to be daily, and therefore calculate daily log returns using end-of-day close prices. An asset's log returns over a lookback period T can then be captured as $\boldsymbol{r_t} = [r_{t-1}, r_{t-2}, \ldots, r_{t-T+1}]$. In our case, the lookback period is $T = 60$ days.

For a selection of $n + 1$ assets - n securities and cash (denoted by c) - we form the agent's observation state at time t, $\boldsymbol{S_t}$ as a $[(n + 1) \times T]$ matrix:

$$S_t = \begin{bmatrix} w_1 & r_{1,t-1} & \cdots & r_{1,t-T+1} \\ w_2 & r_{2,t-1} & \cdots & r_{2,t-T+1} \\ \vdots & & \ddots & \vdots \\ w_n & r_{n,t-1} & \cdots & r_{n,t-T+1} \\ w_c & vol_{20} & \frac{vol_{20}}{vol_{60}} & VIX_t \ldots \end{bmatrix}$$

The first column is the agent's portfolio allocation vector $\boldsymbol{w}$ as it enters timestep t. This might differ slightly from the portfolio weights it chooses at the timestep before, as we convert the continuous weights into an actual allocation (whole shares only), and rebalance the allocation such that it sums to 1.

For each non-cash asset, we include the log returns over T. These are represented by the vector $[r_{n,t-1}, \ldots, r_{n,t-T+1}]$ for asset n in the state matrix above. Additionally, in the last row, we include three market volatility indicators at time t: vol_{20}, $\frac{vol_{20}}{vol_{60}}$, *VIX*, which we describe in detail in the Experiments section.

Reward

Rather than maximizing returns, most modern portfolio managers attempt to maximize risk-adjusted returns. Since we wish to utilize DRL for portfolio allocation, we want a reward function that helps optimize for risk-adjust returns. The Sharpe ratio is the most widely-used measure for this, however, it is inappropriate for online learning settings as it is defined over a period of time T. To combat this, we use the Differential Sharpe Ratio D_t (Moody et al. 1998) which represents the risk-adjusted returns at each timestep t and has been found yield more consistent returns than maximizing profit (Moody and Saffell 2001; Dempster and Leemans 2006). Therefore, an agent that aims to maximize its future Differential Sharpe rewards learns how to optimize for risk adjusted returns.

We can define the Sharpe Ratio over a period of t returns R_t, in terms of estimates of the first and second moments of the returns' distributions:

$$S_t = \frac{A_t}{K_t(B_t - A_t^2)^{1/2}}$$

with

$$A_t = \frac{1}{t}\sum_{i=1}^{t} R_i \text{ and } B_t = \frac{1}{t}\sum_{i=1}^{t} R_i^2\ ,\ K_t = (\frac{t}{t-1})^{1/2}$$

where K_t is a normalizing factor.

A and B can be recursively estimated as exponential moving averages of the returns and standard deviation of returns on time scale η^{-1}. We can obtain a *differential* Sharpe ratio D_t by expanding S_t to first order in η:

$$S_t \approx S_{t-1} + \eta D_t|_{\eta=0} + O(\eta^2)$$

Where Differential Sharpe Ratio D_t:

$$D_t \equiv \frac{\partial S_t}{\partial \eta} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{3/2}}$$

with

$$A_t = A_{t-1} + \eta \Delta A_t$$

$$B_t = B_{t-1} + \eta \Delta B_t$$

$$\Delta A_t = R_t - A_{t-1}$$

$$\Delta B_t = R_t^2 - B_{t-1}$$

initialized with $A_0 = B_0 = 0$. We pick $\eta \approx \frac{1}{252}$ (a year has approximately 252 trading days).

Learning Algorithm

RL algorithms can be mainly divided into two categories, model-based and model-free, depending whether the agent has access to or has to learn a model of the environment. Model-free algorithms seek to learn the outcomes of their actions through collecting experience via algorithms such as Policy Gradient, Q-Learning, etc. Such an algorithm will try an action multiple times and adjust its policy (its strategy) based on the outcomes of its action in order optimize rewards.

Policy Optimization Policy optimization methods are centered around the policy $\pi_\theta(a|s)$ which is the function that maps the agent's state s to the distribution of its next action a. These methods optimize the parameters θ either by gradient ascent on the performance objective $J(\pi_\theta)$ or by maximizing local approximations of $J(\pi_\theta)$. This optimization is almost always performed on-policy since the experiences are collected using the latest learned policy, and then using that experience to improve the policy. Some examples of popular policy optimization methods are A2C/A3C (Mnih et al. 2016) and PPO (Schulman et al. 2017). For our experiments we use PPO.

RL Environment Specifics

The environment serves as a wrapper for the market, sliding over historical data in an approach called market replay. It also serves as a broker and exchange; at every timestep, it processes the agents' actions and rebalances the portfolio using the latest prices and the given allocation. As the day shifts and new prices are received, it communicates these to the agent as observations, along with the Differential Sharpe reward. For the purposes of this study, we assume that there are no transaction costs in the environment, and we allow for immediate rebalancing of the portfolio.

At the beginning of each timestep t, the environment calculates the current portfolio value:

$$\text{port\_val}_t = \sum P_{i,t} * \text{shares}_{i,t-1} + c_{t-1}$$

In the above expression, P_i is the price of index i at day t, $\text{shares}_{i,t-1}$ are the index shares at $t-1$, and c_{t-1} is the amount of cash at $t-1$.

In order to calculate $\text{shares}_{i,t}$ and c_t, the environment allocates port_val_t to the indices and cash according to the new weights w_i. Next, it rebalances the portfolio weights w_i to w_{i_reb} by multiplying w_i with the current portfolio value, rounding down the number of shares and converting the remaining shares into cash.

After rebalancing, the environment creates the next state S_{t+1} and proceeds to the next timestep $t+1$. It calculates the new portfolio value based on P_{t+1} and computes the reward $R_t = D_t$ which it returns to the agent.

Experiments

Data & Features

For our experiments, we use daily adjusted close price data of the S&P500 sector indices as shown in Figure 1, the VIX index and the S&P500 index between 2006 and 2021 (inclusive), extracted from Yahoo Finance. The price data is used to compute log returns, as described in a previous section.

To capture market regime, we compute three volatility metrics from the S&P500 index. The first one, vol_{20}, is the 20-day rolling window standard deviation of the daily S&P500 index returns, the second, vol_{60}, is the 60-day rolling window standard deviation of the daily S&P500 index returns and the third is the ratio of these two $\frac{vol_{20}}{vol_{60}}$. This ratio indicates the short-term versus the long-term volatility trend. If $\frac{vol_{20}}{vol_{60}} > 1$, that indicates that the past 20-day

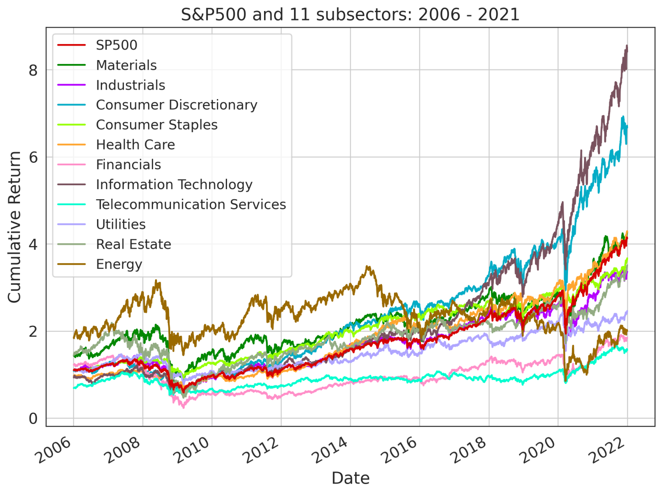

Figure 1: S&P500 and its 11 sector indices between 2006 and 2021.

daily returns of the S&P500 have been more volatile than the past 60-day daily returns, which might indicate a movement from lower volatility to a higher volatility regime (and vice versa). We use the first and third metrics in the observation matrix, along with the value of the VIX index. These values are standardized by subtracting the mean and dividing by the standard deviation, where the mean and standard deviation are estimated using an expanding lookback window to prevent information leakage.

Deep RL Approach

Training Process Although financial data is notoriously scarce (atleast on the daily scale), we want to test the DRL framework across multiple years (backtests). Additionally, financial timeseries exhibit non-stationarity (Cont 2001); this can be tackled by retraining or fine-tuning models by utilizing the most recently available data. In light of these stylized facts, we devise our experiment framework as follows:

The data is split into 10 sliding window groups (shifted by 1-year). Each group contains 7 years worth of data, the first 5 years are used for training, the next 1 year is a burn year used for training validation, and the last year is kept out-of-sample for backtesting.

During the first round of training, we initialize 5 agents (different seeds) with the hyperparameters described in the following section. All five agents start training on data from $[2006-2011)$ and their performance is periodically evaluated using the validation period 2011. At the end of the first round of training, we save the best performing agent (based on highest mean episode validation reward). The final year (2012) is kept held-out for backtesting.

This agent is used as a seed policy for the next group of 5 agents in the following training window $[2007-2012)$, validation year 2012 and testing year 2013, where this experiment is repeated. This process continues till we reach the final validation period of 2020, generating a total of 50 agents (10 periods x 5 agents), and 10 corresponding backtests (described in a following section).

training_timesteps	7.5M
n_envs	10
n_steps	756
batch_size	1260
n_epochs	16
gamma	0.9
gae_lambda	0.9
clip_range	0.25
learning_rate	3e-4 annealed to 1e-5

Table 1: Hyperparameters used for PPO.

PPO Implementation & Hyperparameters We use the StableBaselines3 (Raffin et al. 2021) implementation of PPO, and report the hyperparameters used in Table 1. These were picked based on empirical studies (Henderson et al. 2018; Engstrom et al. 2019; Rao et al. 2020), as well as a coarse grid search over held-out validation data.

Additionally, we make use of the Vectorized SubProcVecEnv environment wrappers provided by StableBaselines3 to collect experience rollouts through multiprocessing across independent instances of our environment. Therefore, instead of training the DRL agent on one environment per step, we trained our model on $n\_envs = 10$ environments per step in order to gain more diverse experience and speed up training.

Each round of training lasted a total 7.5M timesteps so as to have approximately 600 episodes per round per environment: (252 trading days per yr $\times$ 5 yrs per round) $\times$ (10 environments) $\times$ (600 episodes) $\approx$ 7.5M timesteps. The rollout buffer size was set to $n\_steps = 252 \times 3 \times n\_envs$ so as to collect sufficient experiences across environments. We set up the learning rate as a decaying function of the current progress remaining, starting from $3e-4$, annealed to a final value of $1e-5$. We used a $batch\_size$ of $1260 = (252 \times 5)$, set the number of epochs when optimizing the surrogate loss to $n\_epochs = 16$, picked the discount factor $\gamma = 0.9$, set the bias-variance trade-off factor for Generalized Advantage Estimator $gae\_lambda = 0.9$ and $clip\_range = 0.25$. Additionally, we use a $[64, 64]$ fully-connected architecture with $tanh$ activations, and intiailize the policy with a log standard deviation $log\_std\_init = -1$.

Mean-Variance Optimization Approach

As we wish to compare the model-free DRL approach with MVO, we equalize the training and operational conditions. For training, the MVO approach uses a 60-day lookback period (same as DRL) to estimate the means and covariances of assets. Asset means are simply the sample means over the lookback period. However, we do not directly use the sample covariance, as this has been shown to be subject to estimation error that is incompatible with MVO. To tackle this, we make use of the Ledoit-Wolf Shrinkage operator (Ledoit and Wolf 2004). Additionally, we enforce non-singular and positive-semi-definite conditions on the covariance matri-

Metric	DRL	MVO
Annual return	0.1211	0.0653
Cumulative returns	0.1195	0.0650
Annual volatility	0.1249	0.1460
Sharpe ratio	1.1662	0.6776
Calmar ratio	2.3133	1.1608
Stability	0.6234	0.4841
Max drawdown	-0.3296	-0.3303
Omega ratio	1.2360	1.1315
Sortino ratio	1.7208	1.0060
Skew	-0.4063	-0.3328
Kurtosis	2.7054	2.6801
Tail ratio	1.0423	0.9448
Daily value at risk	-0.0152	-0.0181

Table 2: Statistics for the DRL and MVO approaches. All metrics are averaged across 10 backtests (backtesting period: $[2012 - 2021]$), except Max Drawdown which is reported as the maximum seen in any period.

ces, setting negative eigenvalues to 0, and then rebuilding the non-compliant matrices.

Given the estimated means and covariances for a lookback period, we then optimize for the Sharpe Maximization problem and obtain the weights at every timestep. We use the implementation in PyPortfolioOpt (Martin 2021) to aid us with this process.

Evaluation & Backtesting

We evaluate the performance of both techniques through 10 independent backtests $[2012 - 2021]$. Both strategies start each backtest period with an all cash portfolio allocation of $\$100,000$. Then, the strategies trade daily using the portfolio weights obtained by each method, enforcing for weight constraints $\sum w = 1, 0 \leq w_i \leq 1$, and ensuring only whole number of shares are purchased. By doing so, we can obtain daily portfolio values (and returns), which we subsequently use to compute the statistics we will discuss in the Results section. These are computed with the aid of the Python library Pyfolio.

DRL Agent: We evaluate the trained PPO agents in deterministic mode. For each backtest, the agent used has a gap burn year between the last day seen in training and the backtest period. For example, a DRL backtest carried out in 2012 would use an agent trained in $[2006-2011)$, with 2011 being the burn year.

MVO: As the MVO approach does not require any training, it simply uses the past 60-day lookback period before any given day to calculate portfolio weights. For example, a MVO backtest starting January 2012 will use data starting October 2011 (this 60-day window shifts with each day).

Results

Figure 2 illustrates the performance metrics obtained by applying the aforementioned backtest process on all testing periods [2012-2021]. The DRL agent outperforms the MVO portfolio by exhibiting higher Sharpe and lower yearly maximum drawdowns in virtually every year throughout the backtest period (see Figure 1). It also outperforms the MVO portfolio in terms of having marginally lower maximum drawdown.

To compare overall performance on the entire backtest period between the two methods, we compute the average performance across all 10 backtest periods. For DRL, we average the performance across the 5 agents (each trained on a different seed) for each year and then average performance across all backtest periods. Similarly, for MVO, we average its performance across all 10 years. By looking at Table 2 we observe that DRL annual returns and Sharpe ratio are 1.85x higher than those of the MVO portfolio. The DRL strategy's Sharpe throughout the whole backtest period is 1.16x compared to 0.66x for MVO.

Figure 3a) and Figure 4a) plot the monthly returns over all backtest periods for the two methods. It is evident that DRL is experiencing more steady returns month-to-month than MVO. On the other hand, MVO swings between periods of high returns to periods of low returns a lot more frequently without a steady positive return trajectory. Similarly, in Figure 3b) and Figure 3b), we plot the annual returns for the two methods. The vertical dashed line indicates the average annual return across the 10 backtests. For DRL we observe positive returns for almost all backtest years which is a lot more consistent than the behavior of MVO's annual returns. Figure 3c) and Figure 3c) plot the distribution of monthly returns averaged across all months. The DRL monthly returns distribution has a lower standard deviation and and spread than MVO and a positive mean.

Further, we compute the daily portfolio change for each strategy by measuring the change in its portfolio weights. Δp_w is the absolute value of the element-wise difference between two allocations (ignoring the cash component). As buying and selling are treated as individual transactions, $\Delta p_w \in [0.0, 2.0]$. For example, take a case where the portfolio at time $t-1$ is concentrated in non-cash asset A, and at time t is entirely concentrated in non-cash asset B. This requires selling all holdings of A, and acquiring the equivalent shares in B, leading to $\Delta p_w = 2.0$.

Using metric Δp_w, we observe that the Reinforcement Learning strategy has less frequent changes to its portfolio. In practice, this would result in lower average transaction costs. In particular, the average change in portfolio composition is nearly double for Mean-Variance portfolio compared to the DRL strategy during market downturn in March 2020, as shown in Figure 2, when trading conditions were particularly challenging (i.e. significantly lower market liquidity and elevated bid/ask spreads). Finally, the DRL strategy's performance is derived from the average of five individual agents initialized with different seeds, providing additional regularization which is likely to result in a more stable out-of-sample strategy compared to the MVO strategy.

Conclusion

We highlight our key contributions as follows:

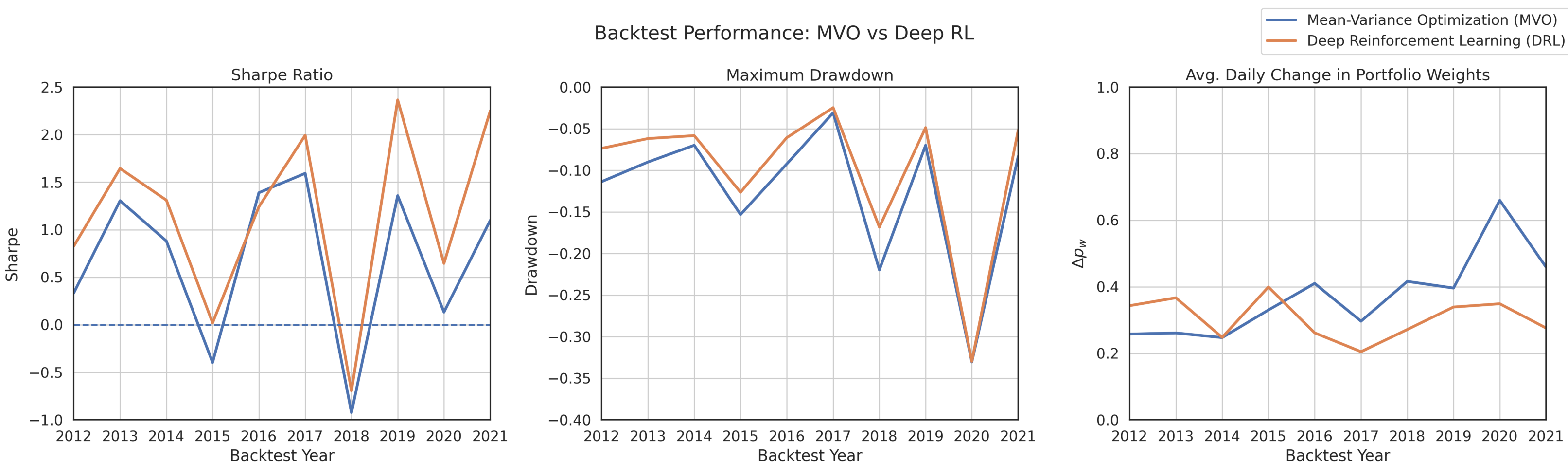

Figure 2: Backtest Results: MVO vs DRL Portfolio Allocation.

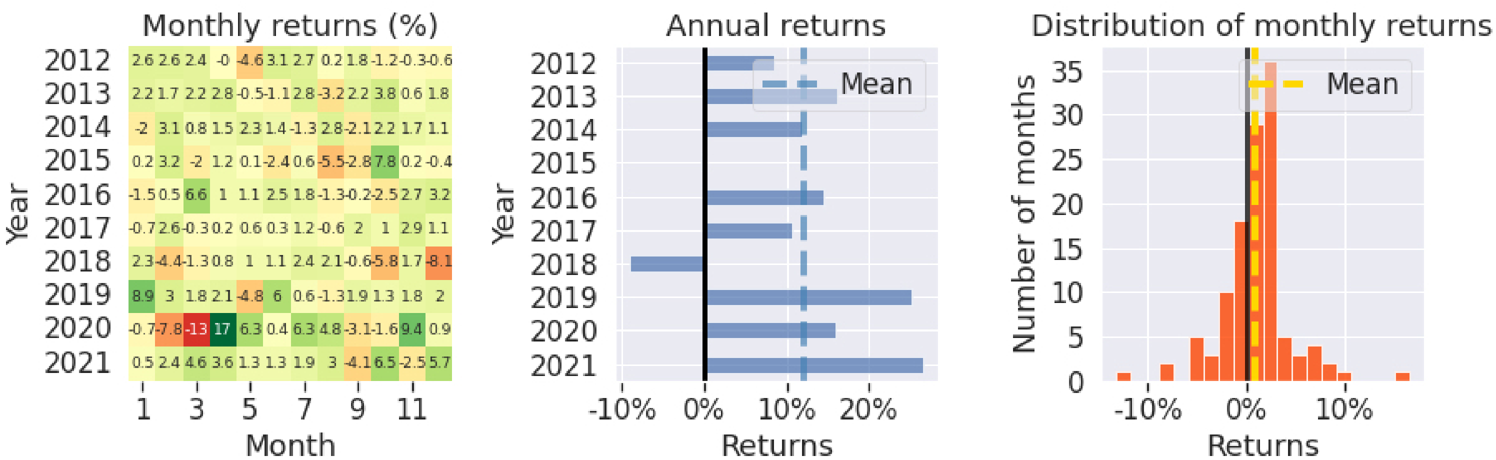

Figure 3: a) DRL Monthly Returns b) DRL Annual Returns c) DRL Monthly Distribution of Returns.

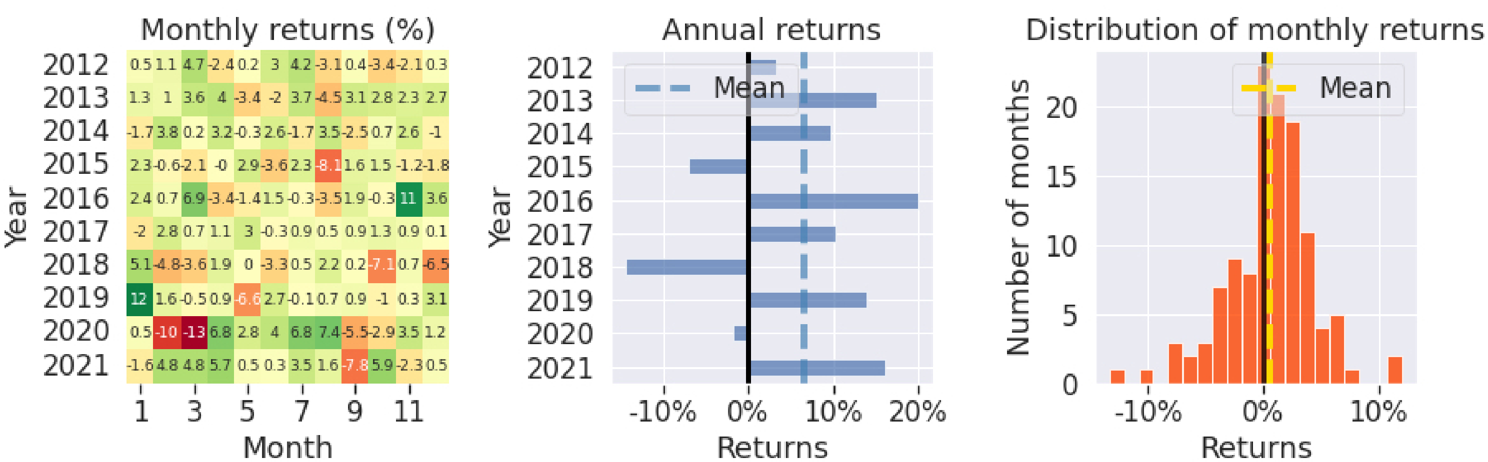

Figure 4: a) MVO Monthly Returns b) MVO Annual Returns c) MVO Monthly Distribution of Returns.

- We have designed a simple environment that serves as a wrapper for the market, sliding over historical data using market replay. The environment can allocate multiple assets and can be easily modified to reflect transaction costs.
- We compare our framework's performance during ten backtest experiments over different periods for the US Equities Market using S&P500 Sector indices. Our experiments demonstrate the improved performance of the deep reinforcement learning framework over Mean-Variance portfolio optimization.
- The profitability of the framework surpasses MVO in terms of Annual Returns, Sharpe ratio and Maximum Drawdown. Additionally, we observe that DRL strategy leads to more consistent returns and more stable portfolios with decreased turnover. This has implications for live-deployment, where transaction costs and slippage affect P&L.

Future Work

In our future work, we would like to model transaction costs and slippage either by explicitly calculating them during asset reallocation or as a penalty term to our reward. Moreover, we would like to explore adding a drawdown minimization component to our reward that will potentially help the agent learn a more consistent trading strategy.

Another area of exploration is training a regime switching model which will balance its funds amongst two agents depending on market volatility (low vs high). One of them will be a low-volatility trained agent and the other a high volatility trained agent. We would like to compare performance between our current implicit regime parametrization and an explicit one. Further exploration of these research directions may produce significant insights into practical trading behavior.

References

Benhamou, E.; Saltiel, D.; Ungari, S.; and Mukhopadhyay, A. 2020a. Bridging the gap between Markowitz planning and deep reinforcement learning. *arXiv preprint arXiv:2010.09108*.

Benhamou, E.; Saltiel, D.; Ungari, S.; and Mukhopadhyay, A. 2020b. Time your hedge with deep reinforcement learning. *arXiv preprint arXiv:2009.14136*.

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Buehler, H.; Gonon, L.; Teichmann, J.; and Wood, B. 2019. Deep hedging. *Quantitative Finance*, 19(8): 1271–1291.

Cao, J.; Chen, J.; Hull, J.; and Poulos, Z. 2021. Deep hedging of derivatives using reinforcement learning. *The Journal of Financial Data Science*, 3(1): 10–27.

Charpentier, A.; Elie, R.; and Remlinger, C. 2021. Reinforcement learning in economics and finance. *Computational Economics*, 1–38.

Chen, L.; He, S.; and Zhang, S. 2011. When all risk-adjusted performance measures are the same: In praise of the Sharpe ratio. *Quantitative Finance*, 11(10): 1439–1447.

Cong, L. W.; Tang, K.; Wang, J.; and Zhang, Y. 2021. AlphaPortfolio: Direct construction through deep reinforcement learning and interpretable AI. *Available at SSRN*, 3554486.

Cont, R. 2001. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative finance*, 1(2): 223.

Cornuejols, G.; and Tütüncü, R. 2006. *Optimization methods in finance*, volume 5. Cambridge University Press.

Dempster, M. A.; and Leemans, V. 2006. An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30(3): 543–552.

Deng, Y.; Bao, F.; Kong, Y.; Ren, Z.; and Dai, Q. 2016. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3): 653–664.

Du, J.; Jin, M.; Kolm, P. N.; Ritter, G.; Wang, Y.; and Zhang, B. 2020. Deep reinforcement learning for option replication and hedging. *The Journal of Financial Data Science*, 2(4): 44–57.

Du, X.; and Tanaka-Ishii, K. 2020. Stock embeddings acquired from news articles and price history, and an application to portfolio optimization. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, 3353–3363.

Engstrom, L.; Ilyas, A.; Santurkar, S.; Tsipras, D.; Janoos, F.; Rudolph, L.; and Madry, A. 2019. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*.

Ghahtarani, A.; Saif, A.; and Ghasemi, A. 2022. Robust portfolio selection problems: a comprehensive review. *Operational Research*, 1–62.

Hambly, B.; Xu, R.; and Yang, H. 2021. Recent advances in reinforcement learning in finance. *arXiv preprint arXiv:2112.04553*.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2018. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

i Alonso, M. N.; Srivastava, S.; et al. 2020. Deep reinforcement learning for asset allocation in us equities. Technical report.

Jiang, Z.; and Liang, J. 2017. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*, 905–913. IEEE.

Kalayci, C.; Ertenlice, O.; Akyer, H.; and Aygören, H. 2017. A review on the current applications of genetic algorithms in mean-variance portfolio optimization. *Pamukkale University Journal of Engineering Sciences*, 23: 470–476.

Koratamaddi, P.; Wadhwani, K.; Gupta, M.; and Sanjeevi, S. G. 2021. Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation. *Engineering Science and Technology, an International Journal*, 24(4): 848–859.

Ledoit, O.; and Wolf, M. 2004. Honey, I shrunk the sample covariance matrix. *The Journal of Portfolio Management*, 30(4): 110–119.

Li, B.; and Hoi, S. C. 2014. Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, 46(3): 1–36.

Li, X.; Li, Y.; Zhan, Y.; and Liu, X.-Y. 2019. Optimistic bull or pessimistic bear: Adaptive deep reinforcement learning for stock portfolio allocation. *arXiv preprint arXiv:1907.01503*.

Liang, Z.; Chen, H.; Zhu, J.; Jiang, K.; and Li, Y. 2018. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*.

Lima Paiva, F. C.; Felizardo, L. K.; Bianchi, R. A. d. C.; and Costa, A. H. R. 2021. Intelligent trading systems: a sentiment-aware reinforcement learning approach. In *Proceedings of the Second ACM International Conference on AI in Finance*, 1–9.

Liu, X.-Y.; Yang, H.; Chen, Q.; Zhang, R.; Yang, L.; Xiao, B.; and Wang, C. D. 2020. FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. *arXiv preprint arXiv:2011.09607*.

Lu, D. W. 2017. Agent inspired trading using recurrent reinforcement learning and lstm neural networks. *arXiv preprint arXiv:1707.07338*.

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

Martin, R. A. 2021. PyPortfolioOpt: portfolio optimization in Python. *Journal of Open Source Software*, 6(61): 3066.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Moody, J.; and Saffell, M. 2001. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4): 875–889.

Moody, J.; Wu, L.; Liao, Y.; and Saffell, M. 1998. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5-6): 441–470.

Nguyen, H.; and La, H. 2019. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 590–595. IEEE.

Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; and Dormann, N. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268): 1–8.

Rao, N.; Aljalbout, E.; Sauer, A.; and Haddadin, S. 2020. How to make deep RL work in practice. *arXiv preprint arXiv:2010.13083*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sharpe, W. F. 1998. The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 169–185.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Soleymani, F.; and Paquet, E. 2020. Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—DeepBreath. *Expert Systems with Applications*, 156: 113456.

Su, P.-H.; Gasic, M.; Mrksic, N.; Rojas-Barahona, L.; Ultes, S.; Vandyke, D.; Wen, T.-H.; and Young, S. 2016. On-line active reward learning for policy optimisation in spoken dialogue systems. *arXiv preprint arXiv:1605.07669*.

Sun, S.; Wang, R.; He, X.; Zhu, J.; Li, J.; and An, B. 2021. Deepscalper: A risk-aware deep reinforcement learning framework for intraday trading with micro-level market embedding. *arXiv preprint arXiv:2201.09058*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Théate, T.; and Ernst, D. 2021. An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*, 173: 114632.

Wang, J.; Zhang, Y.; Tang, K.; Wu, J.; and Xiong, Z. 2019. Alphastock: A buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 1900–1908.

Wang, Z.; Huang, B.; Tu, S.; Zhang, K.; and Xu, L. 2021. DeepTrader: A Deep Reinforcement Learning Approach for Risk-Return Balanced Portfolio Management with Market Conditions Embedding. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(1): 643–650.

Wu, X.; Chen, H.; Wang, J.; Troiano, L.; Loia, V.; and Fujita, H. 2020. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, 538: 142–158.

Yang, H.; Liu, X.-Y.; Zhong, S.; and Walid, A. 2020. Deep reinforcement learning for automated stock trading: An ensemble strategy. In *Proceedings of the First ACM International Conference on AI in Finance*, 1–8.

Ye, Y.; Pei, H.; Wang, B.; Chen, P.-Y.; Zhu, Y.; Xiao, J.; and Li, B. 2020. Reinforcement-learning based portfolio management with augmented asset movement prediction states. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 1112–1119.

Zhang, Z.; Zohren, S.; and Roberts, S. 2020. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2): 25–40.

Surrogate Assisted Monte Carlo Tree Search in Combinatorial Optimization

Saeid Amiri[1], Parisa Zehtabi[2], Danial Dervovic[3], Michael Cashmore[3]
[1]JP Morgan AI Research. New York City, NY, USA
[2]JP Morgan AI Research. London, UK
[3]JP Morgan AI Research. Edinburgh, UK
saeid.amiri@jpmchase.com, parisa.zehtabi@jpmorgan.com, danial.dervovic@jpmchase.com, michael.cashmore@jpmorgan.com

Abstract

Industries frequently adjust their facilities network by opening new branches in promising areas and closing branches in areas where they expect low profits. In this paper, we examine a particular class of facility location problems. Our objective is to minimize the loss of sales resulting from the removal of several retail stores. However, estimating sales accurately is expensive and time-consuming. To overcome this challenge, we leverage Monte Carlo Tree Search (MCTS) assisted by a surrogate model that computes evaluations faster. Results suggest that MCTS supported by a fast surrogate function can generate solutions faster while maintaining a consistent solution compared to MCTS that does not benefit from the surrogate function.

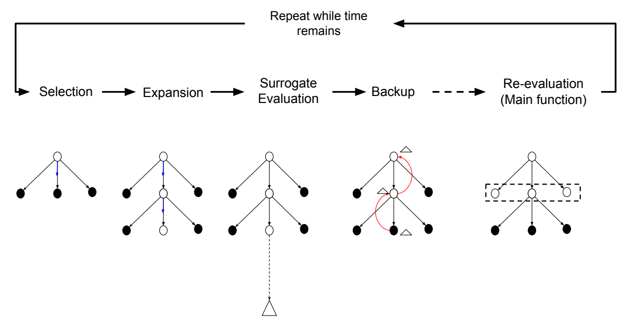

Figure 1: Surrogate assisted Monte Carlo Tree Search (SMCTS) where an occasional reevaluation step refines the node values.

Introduction

As populations shift, market trends change, and customer demands evolve, many service industries and retail stores are faced with the decision of adding, removing, relocating, or consolidating their facility locations. An example is a 2018 survey that showed the market trends resulting in some liquor stores becoming obsolete [1]. In this paper, we focus on a particular class of facility location problem that involves closing a fixed number of retail stores in which computing the features of the evaluation function is expensive. This problem is a Combinatorial Optimization (CO). COs are often NP-hard and computationally intractable due to the large state-spaces. Consequently, solving CO problems often requires designing heuristics or approximation algorithms (Williamson and Shmoys 2011). Furthermore, real-world optimization problems are often complex, nonlinear, and may have multiple objectives and constraints that can be computationally expensive to evaluate. The solutions to CO often involves the design of heuristics or approximation algorithms.

Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári 2006; Coulom 2006) is a popular technique for solving search problems in large spaces, particularly in the domain of games. It involves building a search tree of possible actions and their corresponding outcomes, and using evaluations (simulations) to estimate the value of each action. MCTS has been applied to a wide range of problems such as games (Silver et al. 2017; Rubin and Watson 2011), robotics (Kim et al. 2020), finance (Vittori, Likmeta, and Restelli 2021) and music (Liebman et al. 2017). Recent works have used MCTS in CO. One work used Graph Neural Networks and Reinforcement Learning in order to compute heuristics for the MCTS-based action selection in scheduling and vehicle routing problems (Oren et al. 2021). MCTS has also been applied to capacity expansion in a residency matching problem to find an optimal policy for matching medical doctors to hospital vacancies (Abe, Komiyama, and Iwasaki 2022).

Inspired by these successes, we leverage MCTS in the facility location problem. We propose Surrogate-assisted MCTS (SMCTS) to solve a combinatorial search problem where we use a fast surrogate evaluation function in concert with the slow default evaluation function. The main evaluation function is a regression model that evaluates the current network profitability but is computationally expensive due to the varying network-dependent features. The surrogate is fast to compute but is less accurate. The choice of an efficient surrogate function is its own research problem, and we simply assume that a surrogate function is available. In this

[1]https://www.forbes.com/sites/taranurin/2019/11/22/independent-liquor-stores-will-become-obsolete-believe-nearly-one-out-of-two-owners/?sh=5eb971f57b9b

paper, we focus on how to use the surrogate function jointly with the main evaluation function, aiming at faster solution computation. Figure 1 depicts the SMCTS steps where the selection, expansion, evaluation (by surrogate function), and backup is complemented by an occasional re-evaluation step that takes place in order to refine possible inaccurate surrogate model evaluations.

We apply this approach to the problem of store closure in a network of liquor stores with the goal of minimizing the overall sales loss. Our empirical results show that by using MCTS with a surrogate function, the overall computation time is reduced.

Related Work

Classes of facility location problems are among the fundamental problems in Operations Research. Traditionally, they have been framed using Operation Research techniques such as set covering (Namazian and Roghanian 2021; Murray 2016; Miliotis, Dimopoulou, and Giannikos 2002), maximal covering (Church and ReVelle 1974; Berman and Krass 2002), or p-median problems where the goal is minimizing the travel distance from customers (Kariv and Hakimi 1979). Facilities could be static such as branches and warehouses (Zaikin, Ignatiev, and Marques-Silva 2020) or dynamic such as charging stations (Andrenacci, Ragona, and Valenti 2016; Drezner and Wesolowsky 1991; Wesolowsky 1973).

Most works formulate the problem either as integer programming or clustering methods. In the former, various heuristic techniques such as tabu search, Lagrangian relaxation (Santos 2009), greedy interchange (Kuehn and Hamburger 1963), branch and bound (Järvinen, Rajala, and Sinervo 1972; Dupont 2008), primal and dual (Captivo 1991), binomic approach (Maniezzo, Mingozzi, and Baldacci 1998) and gamma heuristics (Rosing, Revelle, and Schilling 1999) have been used. In the latter, some works leveraged unsupervised clustering methods to find the demand centroids for charging stations and assign customers based on distance to the closest centroid (Andrenacci, Ragona, and Valenti 2016; Ip, Fong, and Liu 2010). In some papers, Analytic Hierarchy Process have been proposed for banking branch location based on customer demographic and economic features (Gorener, Dinçer, and Hacioglu 2013; Sharmin and Solaiman 2019).

Most works have used the travel distance and customer demand as the only features in their objective and constraints formulation. Zaikin, Ignatiev, and Marques-Silva (2020) set the customer dissatisfaction minimization as the goal in the branch closure problem using Max-SAT methods.

Almost all works have considered the solution to the facility location problem as a one-shot static solution where the location of multiple facilities are selected all at once. In practice, firms and industries would decide to alter the exisiting network of facilities. One work considered removing or addition of existing facilities using integer programming and approximation techniques (Wang et al. 2003).

In comparison, this paper focuses on a data-driven evaluation function that estimates the overall sales profit based on numerous features such as the number and the distance of nearby facilities. In addition, we leverage surrogate evaluations as a fast evaluation for the facility location problems.

Surrogate-assisted optimization Leveraging surrogates for optimization has already been explored by researchers in other contexts (Namazi et al. 2020; Gu et al. 2021; Liu, Zhang, and Gielen 2013). For instance, (Liu, Zhang, and Gielen 2013) proposed a Gaussian Process-assisted evolutionary algorithm to solve computationally expensive problems. A surrogate model has been used to prune the solution search space in the Travelling Thief problem (Namazi et al. 2020).

To the best of our knowledge, this is the first work of leveraging surrogates in MCTS in CO problems.

Problem Statement

This problem is a class of facility location problem in which a fixed number of retail stores are going to be closed. There is a city network of $N \in \mathbb{N}$ stores. We seek to remove M stores ($M < N$), that result in minimum forgone sales of the network. Our decision variable is the vector X ($|X| = N$) such that:

$$X_j = \begin{cases} 1 & \text{if store } j \text{ remains open} \\ 0 & \text{If store } j \text{ remains closed} \end{cases}$$

and the objective is expressed as:

$$\text{Minimize} \sum_{j=1}^{N} F_m(\mathbb{1}_N, j) - \sum_{j=1}^{N} F_m(X, j)$$

$$\text{Subject to } ||X||_2 = M$$

where $\mathbb{1}_N$ is a vector of 1s and size N. The objective as mentioned above is to minimize the total loss of sales as a result of store closures and the constraint states that exactly M stores will be closed. F_m is an evaluation function that estimates the sales of store j. It is important to note that the sales estimated per store depend not only upon the features of that store, but also on other stores, including whether or not they are closed.

Next, we describe how we find the solutions to this optimization problem.

Framework

In this section, we explain the surrogate assisted MCTS framework.

Node representation: In our search tree, a node is identified by the set of candidate stores for removal according to the path from the root. The root node represents no store removal and the tree depth is M where M is the total number of stores to be removed from the network. In other words, the nodes at depth M are terminal nodes. Each node keeps duplicate attributes V_s' and N_s' for its value and the number of visits in case it goes through the re-evaluation step. SMCTS has five components:

- **Selection:** Starting at the root node, the tree policy, in our case UCB1 (Kocsis and Szepesvári 2006), is used to select the next node based on its value, as described below:
$$\text{argmax}\left\{v_s + C\sqrt{N_p/N_s}\right\}$$
where v_s is the value of the node s, N_s the number of times node s is visited, N_p is the number of times the parent node has been visited. The choice of C affect the ratio of exploration versus exploitation in the search.[2]
- **Evaluate:** A node can be evaluated using functions F_m and F_s. F_m is the main evaluation function that is costly to compute. F_s is an approximate surrogate function that is faster to compute but less accurate compared to F_m.
- **Backup:** The return generated by the main or surrogate evaluation function is backed up to update the values.
- **Expand:** A node is expanded to its children by removing any of the remaining stores from the network. The number of children expanded is equal to the remaining number of stores in the network. Removing a store can be denoted as taking action $a_i \in A$ meaning the removal of the ith store.
- **Re-evaluation:** A node's children are re-evaluated if their values are within the estimation error of the neighboring nodes in the same tier.

Next, we briefly explain the SMCTS algorithm.

Algorithm: Algorithm 1 requires a surrogate function F_s with the error bound σ_s, (in our case, σ_s is the difference between the Root Mean Squared Error (RMSE) of the F_s and F_m) and a main evaluation function is F_m. Node s is initialized with the root node s_0. The selection is done using the UCB1 algorithm where it suggests the best action a denoting the next best store for removal (Lines 4-5). Once the next node is selected, it expands into new children. (Lines 6 -7). The value of the node is estimated by the surrogate function F_s and backed up to the parent nodes recursively (Lines 9-10). The novelty of SMCTS is in the **re-evaluate** step where an occasional refinement of node values is done in order to reduce value errors. The re-evaluation step is presented in Algorithm 2. This algorithm is called when all the children of node s are visited an equal number of times. In that case, Algorithm 2 sorts the values of all children in the subtree (sharing same parent node). We name the values of two adjacent sorted nodes V_{s_i} and $V_{s_{i+1}}$. These values may not be accurate as they have been evaluated using F_s, therefore if $V_{s_{i+1}} - \sigma_s$ is less than $V_{s_i} + \sigma_s$, then these node values need to be updated with F_m. The number of times that re-evaluate is called would depend on σ_s and the distribution of node values.

Experiments

This section details the conducted experimental evaluation, analysing the performance of SMCTS in different problem settings. Our evaluation aimed at studying the following hypothesis:

[2] We use v'_s, N'_p, N'_s instead, after the node is reevaluated.

Algorithm 1: Surrogate-assisted MCTS

Input: Surrogate function F_s, evaluation function F_m, action set A, root node s_0, error bound σ_s

1: **while** Computational budget **do**
2: $s \leftarrow s_0$
3: **while** $s.terminal$ is $False$ **do**
4: $a \leftarrow Select(s, A)$
5: $s \leftarrow s.children[a]$
6: **if** $s.leaf$ is $True$ **then**
7: $Expand(s)$
8: **end if**
9: $v \leftarrow Evaluate(s, F_s)$
10: $Backup(s, v)$
11: **if** $s.leaf$ is $False$ and $s.children$ equally visited **then**
12: $Re - evaluate(s, F_m, \sigma_s)$
13: **end if**
14: **end while**
15: **end while**
16: **return** Node with the highest value

Algorithm 2: Re-evaluate nodes

Input: Node s, F_m, σ_s error bound

Sort children of s based on value
for i in $[0, s.children.length - 1]$ **do**
$v_{s_i} \leftarrow s.children[i].value$
$v_{s_{i+1}} \leftarrow s.children[i+1].value$
if $v_{s_{i+1}} - \sigma_s < v_{s_i} + \sigma_s$ **then**
$v'_{s_{i+1}} \leftarrow Evaluate(s.children[i+1], F_m)$
$v'_{s_i} \leftarrow Evaluate(s.children[i], F_m)$
$Backup(s_i, v'_{s_i})$
$Backup(s_{i+1}, v'_{s_{i+1}})$
end if
end for

1. In scenarios with higher scale (large number of total stores or large number of removals), SMCTS tends to leverage surrogate function more than the evaluation function.
2. The number of surrogate evaluations depends on the surrogate quality. The higher error it has, the more re-evaluation steps are needed.
3. With an efficient choice of a surrogate function, SMCTS maintains a solution consistent with unassisted MCTS.

Next we explain the dataset and the evaluation functions.

Dataset: We use the Iowa Liquor Dataset [3] that contains the daily purchase information of various liquors in each store in the state of Iowa. The dataset has the information of the stores such as the store name, address, coordinates, zip code with 978 unique values, and the city name with 476

[3] https://data.iowa.gov/Sales-Distribution/Iowa-Liquor-Sales/m3tr-qhgy

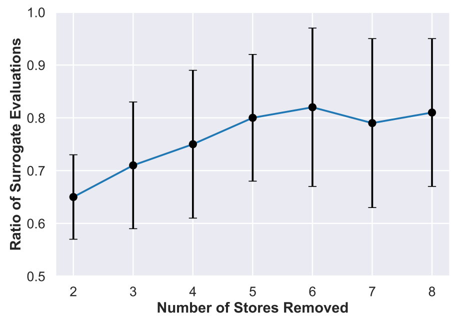

Figure 2: SMCTS where an occasional reevaluation step refines the node values. The horizontal axis represents the number of stores that need to be removed.

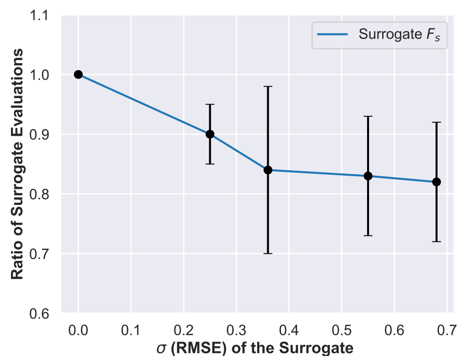

Figure 3: SMCTS with various surrogate errors. The vertical axis is the ratio of surrogate function F_s evaluations to the total evaluations. The horizontal axis represent surrogate functions with increasing normalized RMSEs.

cities and the type and amount of the liquor they have sold at each day. We preprocess the data by calculating the total sales of all liquors at each store in a year and by defining some new features such as the number of stores in 0.5 miles proximity. These additional features are network dependent and require to be recomputed for every node in the tree, resulting in F_m to be costly.

Evaluation Functions: Our main evaluation function F_m is an XGBoost regression model that estimates the sales amount for a store. Given a store removal, some features in the dataset need to be recalculated, resulting in a costly evaluation. To create a surrogate function F_s, we use a subset of the features of the dataset and train another XGBoost regression model. F_s is less accurate on sales estimation compared to F_m. In our case, F_s has a normalized RMSE of 0.27 and F_m has the RMSE of 0.16 (both on the test set). We use SMCTS for liquor store removals for a given county with varying counts of store to remove. Figure 2 shows the average ratio of the number of times F_s has been called versus F_m for ten counties (randomly sampled) where the number of stores in those counties ranged from 17 to 64, representing 70 store removal problems. We observe that with the increase in the number of removed branches, the relative number of times the surrogate function is called increases, facilitating reduction of the overall evaluation burden. Figure 3 shows the ratio of surrogate evaluation to the total evaluation using various surrogates with various error bound. With the increase in the error of the surrogate, we observe an increase in the re-evaluation step. Such increase is valuable as long as SMCTS is consistent with MCTS store selection. Figure 4 presents the consistency comparison of the two approaches for various store removals. We use the Sørensen–Dice coefficient to measure the similarity of the results of the two methods. The values are the average of ten counties, randomly sampled from the dataset. We observe that in most cases, SMCTS output is consistent with MCTS. There is a bit of inconsistency for 3 and 4 branch removals, such inconsistencies are due to the weaker estimations of F_S in outlier counties.

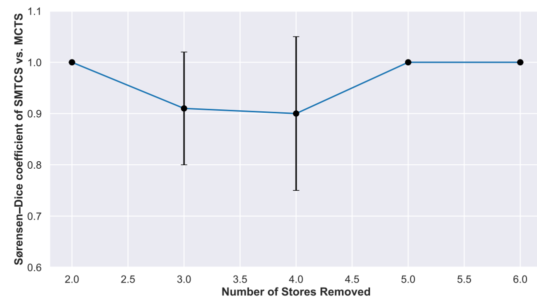

Figure 4: Evaluation of the consistency of stores selected by SMCTS vs. MCTS. The vertical axis shows the number of the selected stores by SMCTS being different from MCTS. The results are the average of 10 counties that are randomly selected.

Conclusion & Future Work

In this work, we proposed MCTS search with surrogate functions for combinatorial optimization. We demonstrated that by using less accurate but faster surrogate function, we can solve optimization problems more efficiently. We applied our approach to a store closure problem in which the goal is to minimize the total sales loss of a retail store.

In this paper, we assumed the surrogate function is provided while this is not the case in practice. For future work, we propose to investigate ways to implement and design the surrogate function and the criteria for it to improve the SMCTS. In addition, we will explore the applicability of SMCTS with other datasets and domains with stochasticity in the action space.

Disclaimer. This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co. and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP 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.

References

Abe, K.; Komiyama, J.; and Iwasaki, A. 2022. Anytime capacity expansion in medical residency match by monte carlo tree search. *arXiv preprint arXiv:2202.06570*.

Andrenacci, N.; Ragona, R.; and Valenti, G. 2016. A demand-side approach to the optimal deployment of electric vehicle charging stations in metropolitan areas. *Applied Energy*, 182: 39–46.

Berman, O.; and Krass, D. 2002. The generalized maximal covering location problem. *Computers & Operations Research*, 29(6): 563–581.

Captivo, M. E. 1991. Fast primal and dual heuristics for the p-median location problem. *European Journal of Operational Research*, 52(1): 65–74.

Church, R.; and ReVelle, C. 1974. The maximal covering location problem. In *Papers of the regional science association*, volume 32, 101–118. Springer-Verlag Berlin/Heidelberg.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, 72–83. Springer.

Drezner, Z.; and Wesolowsky, G. 1991. Facility location when demand is time dependent. *Naval Research Logistics (NRL)*, 38(5): 763–777.

Dupont, L. 2008. Branch and bound algorithm for a facility location problem with concave site dependent costs. *International journal of production economics*, 112(1): 245–254.

Gorener, A.; Dinçer, H.; and Hacioglu, U. 2013. Application of multi-objective optimization on the basis of ratio analysis (MOORA) method for bank branch location selection. *International Journal of Finance & Banking Studies (2147-4486)*, 2(2): 41–52.

Gu, Q.; Wang, Q.; Li, X.; and Li, X. 2021. A surrogate-assisted multi-objective particle swarm optimization of expensive constrained combinatorial optimization problems. *Knowledge-Based Systems*, 223: 107049.

Ip, A.; Fong, S.; and Liu, E. 2010. Optimization for allocating BEV recharging stations in urban areas by using hierarchical clustering. In *2010 6th International conference on advanced information management and service (IMS)*, 460–465. IEEE.

Järvinen, P.; Rajala, J.; and Sinervo, H. 1972. A branch-and-bound algorithm for seeking the p-median. *Operations Research*, 20(1).

Kariv, O.; and Hakimi, S. L. 1979. An algorithmic approach to network location problems. I: The p-centers. *SIAM journal on applied mathematics*, 37(3): 513–538.

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 *European conference on machine learning*. Springer.

Kuehn, A. A.; and Hamburger, M. J. 1963. A heuristic program for locating warehouses. *Management science*, 9(4): 643–666.

Liebman, E.; Khandelwal, P.; Saar-Tsechansky, M.; and Stone, P. 2017. Designing better playlists with monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 4715–4720.

Liu, B.; Zhang, Q.; and Gielen, G. G. 2013. A Gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(2): 180–192.

Maniezzo, V.; Mingozzi, A.; and Baldacci, R. 1998. A bionomic approach to the capacitated p-median problem. *Journal of Heuristics*, 4(3): 263.

Miliotis, P.; Dimopoulou, M.; and Giannikos, I. 2002. A hierarchical location model for locating bank branches in a competitive environment. *International transactions in operational research*, 9(5): 549–565.

Murray, A. T. 2016. Maximal coverage location problem: impacts, significance, and evolution. *International Regional Science Review*, 39(1): 5–27.

Namazi, M.; Sanderson, C.; Newton, M.; and Sattar, A. 2020. Surrogate assisted optimisation for travelling thief problems. In *Proceedings of the International Symposium on Combinatorial Search*, volume 11, 111–115.

Namazian, Z.; and Roghanian, E. 2021. A decision problem for bank branch site selection: a GIS mapping perspective with maximal covering location problem: a case study of Isfahan, Iran. *International Journal of Services and Operations Management*, 39(3): 337–361.

Oren, J.; Ross, C.; Lefarov, M.; Richter, F.; Taitler, A.; Feldman, Z.; Di Castro, D.; and Daniel, C. 2021. SOLO: search online, learn offline for combinatorial optimization problems. In *Proceedings of the International Symposium on Combinatorial Search*, volume 12, 97–105.

Rosing, K. E.; Revelle, C. S.; and Schilling, D. A. 1999. A gamma heuristic for the p-median problem. *European Journal of Operational Research*, 117(3): 522–532.

Rubin, J.; and Watson, I. 2011. Computer poker: A review. *Artificial intelligence*, 175(5-6): 958–987.

Santos, A. C. 2009. *Solving large p-median problems using a Lagrangean heuristic*. Ph.D. thesis, Université Blaise Pascal (Clermont Ferrand 2).

Sharmin, S.; and Solaiman, K. 2019. BigBank: A GIS integrated AHP-TOPSIS based expansion model for banks. In *2019 SITIS*, 311–318. IEEE.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359.

Vittori, E.; Likmeta, A.; and Restelli, M. 2021. Monte carlo tree search for trading and hedging. In *Proceedings of the Second ACM International Conference on AI in Finance*, 1–9.

Wang, Q.; Batta, R.; Bhadury, J.; and Rump, C. M. 2003. Budget constrained location problem with opening and closing of facilities. *Computers & Operations Research*, 30(13): 2047–2069.

Wesolowsky, G. O. 1973. Dynamic facility location. *Management Science*, 19(11): 1241–1248.

Williamson, D. P.; and Shmoys, D. B. 2011. *The design of approximation algorithms*. Cambridge university press.

Zaikin, O.; Ignatiev, A.; and Marques-Silva, J. 2020. Branch location problems with maximum satisfiability. In *ECAI 2020*, 379–386. IOS Press.

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 tf-plan (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:

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

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:

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

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

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

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:

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

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,

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

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 consideration 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,

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

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.

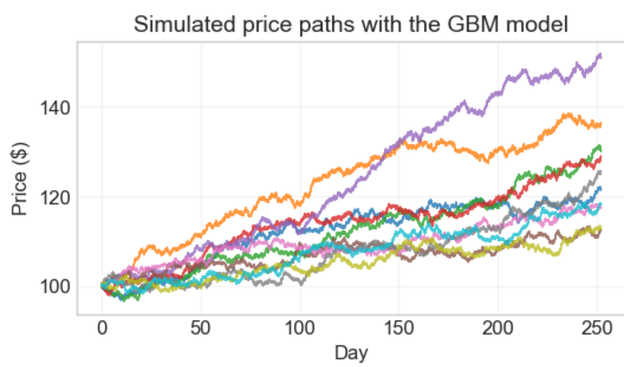

Figure 1: Different asset price paths following the Geometric Brownian Motion (GBM) model with a starting price of $100, drift ($\mu$) of 0.001 and volatility (σ) of 0.005.

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).

$$w(\textit{asset})(\text{real}) : \text{current weight of } \textit{asset} \tag{3.3}$$

$$p(\textit{asset})(\text{real}) : \text{current price of } \textit{asset} \tag{3.4}$$

$$\text{day(int)} : \text{current day} \tag{3.5}$$

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:

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

$$= w(?a), \quad \text{otherwise}, \tag{3.7}$$

where $\textit{period}$ 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:

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

$$- \alpha_\delta \, |w'(?a) - w(?a)|\} \tag{3.11}$$

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

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

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:

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

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

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

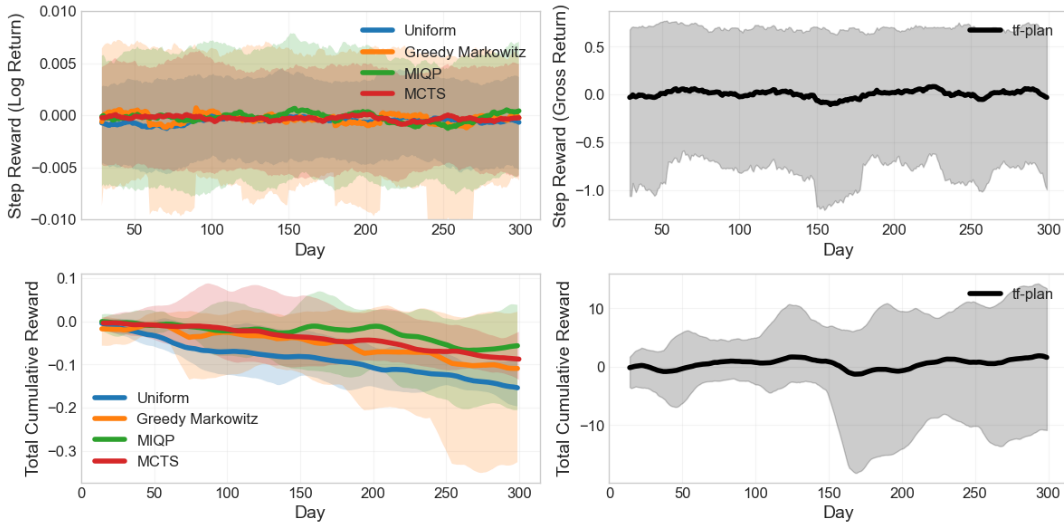

Figure 2: The top plots show the observed daily step reward for AI actors with different planning strategies. The bottom plot shows the total cumulative reward upto that day. Higher rewards indicate better performance. The plots on the right show the step and cumulative rewards for the tf-plan actor. 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.

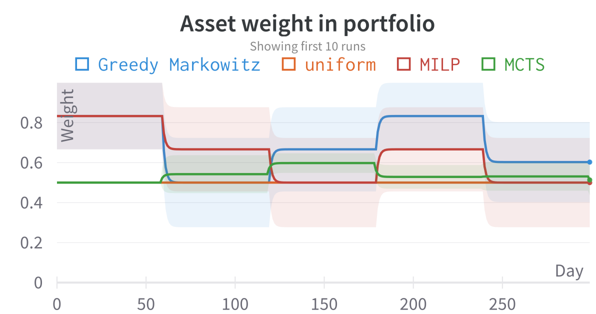

Figure 3: This plot shows how the weight (ratio of the total wealth) of a single portfolio asset changes with time. Please note that the transitions in weights are sharp jumps but they are smoothed out for improved readability.

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:

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

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.

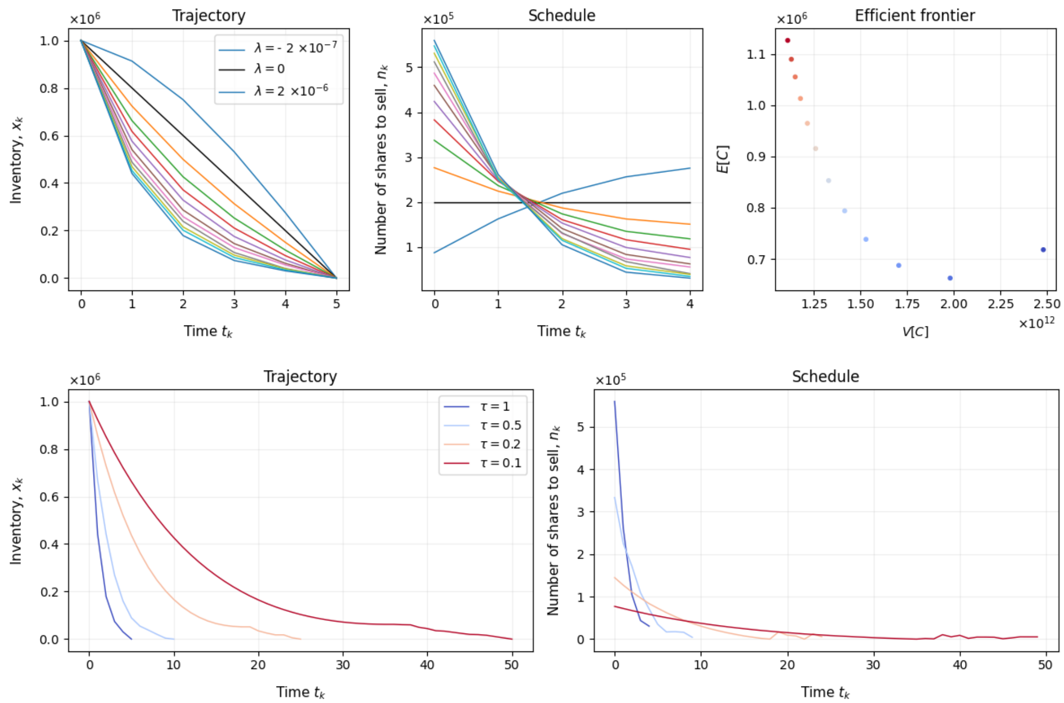

Figure 4: Trade trajectories and schedules with different risk-aversion values λ (top) and trading speed τ (bottom). Our RDDL domain which can simulate this behavior is provided in listing 3 in the appendix.

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;
}
```

Can LLMs be Good Financial Advisors?: An Initial Study in Personal Decision Making for Optimized Outcomes

Kausik Lakkaraju, Sai Krishna Revanth Vuruma, Vishal Pallagani, Bharath Muppasani, Biplav Srivastava

University of South Carolina

Abstract

Increasingly powerful Large Language Model (LLM) based chatbots, like ChatGPT and Bard, are becoming available to users that have the potential to revolutionize the quality of decision-making achieved by the public. In this context, we set out to investigate how such systems perform in the personal finance domain, where financial inclusion has been an overarching stated aim of banks for decades. We asked 13 questions representing banking products in personal finance: bank account, credit card and certificate of deposits and their inter-product interactions, and decisions related to high-value purchases, payment of bank dues, and investment advice, and in different dialects and languages (English, African American Vernacular English, and Telugu). We find that although the outputs of the chatbots are fluent and plausible, there are still critical gaps in providing accurate and reliable financial information using LLM-based chatbots.

Introduction

Consider a freshman that has just started making personal financial decisions. They open a bank account to save up money and get their first credit card. They are given some seed money by their family and they also start earning by working on campus. The student is encouraged by their support system to start thinking about saving into products like Certificate of Deposits (CDs) that earn higher interest. As the student makes a series of decisions in their academic and subsequent professional life, they need to make sound financial decisions and may look for resources online to assist them. An optimal decision needs to consider how the banking products interact with each other along with the changing needs of the student.

For users like this student, increasingly powerful LLM-based chatbots that have the potential to revolutionize the quality of decision for personal finance are becoming available. LLMs have demonstrated tremendous potential across diverse domains (Zhao et al. 2023), such as natural language processing (Min et al. 2021) and protein structure (Hu et al. 2022), and have been claimed to show sparks of artificial general intelligence (Bubeck et al. 2023). These models have been implemented in several applications, ranging from mental health assistants (Xiao et al. 2023) to financial advisement (Yue and Au 2023). In the finance domain, LLMs have been used to develop applications such as fraud detection, risk management, and financial forecasting (Alberto Pozanco 2022). They have been used to analyze financial data, predict stock prices, and generate automated reports. However, with the advent of recent models such as OpenAI's ChatGPT, Google's Bard, and BloombergGPT (Wu et al. 2023), a comparative chatbot study is needed to evaluate their ability to be financial advisors. In this paper, we present an initial study of ChatGPT and Bard in providing personal decision-making for optimized outcomes.

It is widely known that LLMs based systems have unique limitations. For example, they may struggle with commonsense reasoning tasks (Li et al. 2022), encounter challenges when handling symbols (Frieder et al. 2023), and are susceptible to hallucinations (Bang et al. 2023).

With this work, we make the following contributions:

- identify a personal financial planning scenario involving a series of tasks (plans) and optimization of decisions.
- show how leading LLM-based chatbots perform in them and analyze their behavior.
- lay out challenges that future chatbots in this area should overcome to provide trusted financial recommendations.

We thus highlight the potential and limitations of current LLM-based systems - ChatGPT and Bard - in their role as financial advisors. We included all the queries posed and responses from both ChatGPT and Bard in our GitHub repository[1] along with a few snapshots of the actual conversations.

[1]https://github.com/ai4society/LLM-CaseStudies/tree/main/Finance

Product Interactions	Query Identifier	Queries	Variables with their values	Constraints
CC	Q1	I am making a **purchase of $1000** using my credit card. My **billing cycle is from March 25th to April 24th**. Today is March 31st, and I have a **due of $2000** on my account. My total **credit line is $2,800**. Would you recommend I make the purchase now or later in the future?	x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 2800	$x_{DA} + x_{PA} < x_{CL}$ (1)
	Q2	I am making a **purchase of $1000** using my credit card. My **billing cycle is from March 25th to April 24th**. Today is March 31st, and I have a **due of $2000** on my account. My total **credit line is $3,800**. Would you recommend I make the purchase now or later in the future?	x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 3800	
	Q3	I get **5% cashback** if I buy furniture using my credit card. I am **buying furniture worth $1000** using my credit card. My **billing cycle is from March 25th to April 24th**. Today is March 31st, and I have a **due of $2000** on my account. My total **credit line is $2,800**. Would you recommend I make the purchase now or later in the future?	x_{CP} = 5%, x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 2800	
	Q4	I get **5% cashback** if I buy furniture using my credit card. I am **buying furniture worth $1000** using my credit card. My **billing cycle is from March 25th to April 24th**. Today is March 31st, and I have a **due of $2000** on my account. My total **credit line is $3,800**. Would you recommend I make the purchase now or later in the future?	x_{CP} = 5%, x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 3800	
CC (AAVE)	Q5	I be makin' a **purchase of $1000** usin' i's credit card. I's **billin' cycle be from march 25th to april 24th**. Today be march 31ts, and i done a **due of $2000** on i's account. I's total **credit line be $2,800**. Would you recommend i make de purchase now o lateh in de future?	x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 2800	
CC (Telugu)	Q6	నేను నా క్రెడిట్ కార్డ్‌ని ఉపయోగించి **$1000 కొనుగోలు** చేస్తున్నాను. నా **బిల్లింగ్ సైకిల్ మార్చి 25 నుండి ఏప్రిల్ 24 వరకు** ఉంది. ఈరోజు మార్చి 31, మరియు నా ఖాతాలో **$2000 బకాయి** ఉంది. నా మొత్తం **క్రెడిట్ లైన్ $2,800**. నేను ఇప్పుడు లేదా భవిష్యత్తులో కొనుగోలు చేయాలని మీరు సిఫార్సు చేస్తారా?	x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 2800	
	Q7	నేను నా క్రెడిట్ కార్డ్‌ని ఉపయోగించి **$1000 కొనుగోలు** చేస్తున్నాను. నా **బిల్లింగ్ సైకిల్ మార్చి 25 నుండి ఏప్రిల్ 24 వరకు** ఉంది. ఈరోజు మార్చి 31, మరియు నా ఖాతాలో **$2000 బకాయి** ఉంది. నా మొత్తం **క్రెడిట్ లైన్ $3,800**. నేను ఇప్పుడు లేదా భవిష్యత్తులో కొనుగోలు చేయాలని మీరు సిఫార్సు చేస్తారా?	x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 3800	
CC and AB	Q8	I am making a **purchase of $1000** using my credit card. My **billing cycle is from March 25th to April 24th**. Today is March 31st, and I have a **due of $2000** on my account. My total **credit line is $3,800**. I have **$10,000 in my bank** which I can use to pay my credit card balance any time. Would you recommend I make the purchase now or later in the future?	x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 3800, x_{AB} = 10000	Constraint (1) must be satisfied. In addition, if the user chooses to pay the due immediately, the following constraints must also hold true. $x_{DA} < x_{AB}$ (2) , $x_{PA} < x_{CL}$ (3)
	Q9	I get **5% cashback** if I buy furniture using my credit card. I am **buying furniture worth $1000** using my credit card. My **billing cycle is from March 25th to April 24th**. Today is March 31st, and I have a **due of $2000** on my account. My total **credit line is $3,800**. I have **$10,000 in my bank** which I can use to pay my credit card balance any time. Would you recommend I make the purchase now or later in the future?	x_{CP} = 5%, x_{PA} = 1000, x_{BC} = (March 25th - April 24th), x_{DA} = 2000, x_{CL} = 3800, x_{AB} = 10000	
CC and CD	Q10	I have a credit card **due of $2800**. The total **credit line is $2800**. If I don't pay a **minimum of $100** by the end of billing cycle, my **APR would be 27%**. If I pay the minimum amount by the end of billing cycle, **APR will be 25%**. My **billing cycle is from March 25th to April 24th**. Today is March 31st. If I choose to deposit some amount as certificate of deposit (CD), I will get an **interest of 6% on the amount deposited**. Do you recommend I pay the full credit card due or do a certificate of deposit or pay my due and deposit the rest?	x_{APR} = 27% (with late fee) and 25% without late fee, x_{MD} = 100, x_{BC} = (March 25th - April 24th), x_{DA} = 2800, x_{CL} = 2800, x_{CDP} = 6%	$x_{DA} < x_{CL}$ (4) AB was not provided in this query. So we cannot specify any additional constraints in this case from the given data.
	Q11	I have a credit card **due of $2800**. The total **credit line is $3800**. If I don't pay a **minimum of $100** by the end of billing cycle, my **APR would be 27%**. If I pay the minimum amount by the end of billing cycle, **APR will be 25%**. My **billing cycle is from March 25th to April 24th**. Today is March 31st. If I choose to deposit some amount as certificate of deposit (CD), I will get an **interest of 6% on the amount deposited**. Do you recommend I pay the full credit card due or do a certificate of deposit or pay my due and deposit the rest?	x_{APR} = 27% (with late fee) and 25% without late fee, x_{MD} = 100, x_{BC} = (March 25th - April 24th), x_{DA} = 2800, x_{CL} = 3800, x_{CDP} = 6%	
CC, CD and AB	Q12	I have a credit card **due of $2800**. The total **credit line is $2800**. If I don't pay a **minimum of $100** by the end of billing cycle, my **APR would be 27%**. If I pay the minimum amount by the end of billing cycle, **APR will be 25%**. My **billing cycle is from March 25th to April 24th**. Today is March 31st. I currently have **$2,800 in my personal checking account**. If I choose to deposit some amount as certificate of deposit (CD), I will get an interest of **6% on the amount deposited**. Do you recommend I pay the full credit card due with my personal account balance or do a certificate of deposit or pay my due and deposit the rest?	x_{APR} = 27% (with late fee) and 25% without late fee, x_{MD} = 100, x_{BC} = (March 25th - April 24th), x_{DA} = 2800, x_{CL} = 2800, x_{CDP} = 6%, x_{AB} = 2800	$[(x_{DA} - x_{MD}) * x_{APR} \leq (x_{AB} - x_{MD}) * x_{CDP}]$ (5) , $[(x_{AB} - x_{DA}) > 0]$ (6)
	Q13	I have a credit card **due of $2800**. The total **credit line is $2800**. If I don't pay a **minimum of $100** by the end of billing cycle, my **APR would be 27%**. If I pay the minimum amount by the end of billing cycle, **APR will be 25%**. My **billing cycle is from March 25th to April 24th**. Today is March 31st. I currently have **$3,800 in my personal checking account**. If I choose to deposit some amount as certificate of deposit (CD), I will get an **interest of 6% on the amount deposited**. Do you recommend I pay the full credit card due with my personal account balance or do a certificate deposit or pay my due and deposit the rest?	x_{APR} = 27% (with late fee) and 25% without late fee, x_{MD} = 100, x_{BC} = (March 25th - April 24th), x_{DA} = 2800, x_{CL} = 2800, x_{CDP} = 6%, x_{AB} = 3800	

Table 1: Table showing different product interaction categories considered, query identifiers, the queries posed to the chatbots under each category, variables used in each query with their corresponding chosen values and constraints the chatbots need to consider while answering the user queries.

Personal Finance Use Case

Setup: Tools and Procedure

Chatbots Tested

1. **ChatGPT:** ChatGPT (OpenAI 2023) is an LLM-based chatbot created by OpenAI that was trained on large amount of text data from the internet, including books and articles. ChatGPT is capable of answering questions, generating text and converse with users in a natural way. It can also learn from users and adapt to new information.
2. **Bard:** Bard (Google 2023) is an LLM-based chatbot created by Google that was trained on large amount of text data and is capable of generating human-like text in response to user prompts and queries. Like ChatGPT, it is also capable of conversing with users about wide variety of topics in a natural way and adapt to new information.

Product Interaction Categories Product interaction refers to interaction between different products like Credit Card (CC), Certificate of Deposit (CD) and Account Balance (AB). Each product has different quantitative properties. For example, credit card due, limit line and billing cycle are some of the properties that would provide credit card information (not private information) of the user. Different properties pertaining to these products are:

- **Purchase Amount (PA)**: It is the amount spent by the user on purchase of a product.
- **Billing Cycle (BC)**: It is the billing cycle of user's credit card.
- **Due Amount (DA)**: The amount that is due on the user's credit card for the specified billing cycle.
- **Credit Line (CL)**: The maximum amount that user could spend using their credit card. If the amount spent exceeds this value, the credit card company could charge additional interest.
- **Cashback Percentage (CP)**: The % of amount which will be returned to the user in the form of cashback on buying furniture using their credit card.
- **Account Balance (AB)**: The amount of cash present in user's personal bank account.
- **Annual Percentage Rate (APR)**: The APR is charged if there is due on the credit card after the due date. Some financial institutions choose to charge a late fee if the **minimum due (MD)** is not paid. It is calculated by the formula, Daily Period Rate (DPR) x Billing Cycle (in days) x Average Daily Balance (ADB).
- **Certificate of Deposit Percentage (CDP)**: The % of interest accumulated on the cash deposited by the user in the form of CD.

Based on different combinations of these products, we classified the queries into 4 categories. These four categories along with the queries posed under each category, the variables used in each query and the constraints the chatbot has to take into consideration to make a sound recommendation are shown in Table 1. In the CC category, we considered a different dialect of English called African American Vernacular English (AAVE) and Telugu, one of the well-known languages from India, to observe how the chatbots handle queries in a different language or dialect.

S.No.	Bard	ChatGPT
1.	Bard gives accurate results if the question is asked directly (for ex., \$2,250 x 0.0006849 x 30 = \$46.23075.)	ChatGPT gives inaccurate results if the question is asked directly (\$2,250 x 0.0006849 x 30 = \$46.90 (rounded to the nearest cent))
2.	Bard does not utilize the information the user provides completely and calculates CUR less often than ChatGPT.	ChatGPT calculates CUR and reasons using the computed CUR more often than Bard
3.	Bard usually does not give personalized suggestions (especially, when the (Due + purchase amount) $>$ Credit line).	ChatGPT gives personalized suggestions more often than Bard.
4.	As a response to one of the queries, Bard gave a recommendation by making use of a table with different options that user could choose from as shown in Figure 1.	ChatGPT did not use any kind of visual aids.
4.	Bard gave biased recommendation i.e., biased towards recommending the user to make the purchase immediately (in one case, it gave only pros for buying the furniture immediately even though it has serious cons).	ChatGPT never gave biased recommendations (it never encourages the user to buy the furniture immediately unless there is no risk involved).
5.	Bard gives 3 different drafts (with some changes in the response) for the same query.	ChatGPT does not provide different drafts.
6.	With each query posed, the content (calculations) of Bard is not improving as much as ChatGPT. It is not learning from its mistakes immediately.	ChatGPT corrects its errors more often than Bard
7.	Bard understood African-American Vernacular English (AAVE) dialect and gave a reasonable response to the query	When query was posed in AAVE dialect, ChatGPT did not understand it immediately. When we posed the same query again in the same dialect, it understood the query and gave a reasonable recommendation .
8.	Bard was not trained to understand Telugu language.	Though ChatGPT can understand Telugu language and responds in Telugu if the user query is in Telugu, the response it generated was incomplete and had a lot of grammatical errors which made the response very hard to understand.

Table 2: Differences between the responses generated by Bard and ChatGPT when queries related to finance domain were posed.

Findings

In this subsection, we present the findings from the interesting (and sometimes insightful) conversations we had with Bard and ChatGPT.

Differences Between the Chatbots Table 2 shows the differences that were identified between Bard and ChatGPT when queries listed out in Table 1 were asked. We compare these models on various criteria related to their performance in answering queries. The criteria include accuracy, utilization of user information, personalized suggestions, use of visual aids, bias in recommendations, provision of multiple response drafts, learning from mistakes, and understanding of different dialects and languages.

Error Categories We identified some limitations / errors in the responses generated by both the chatbots and classified them into the following categories:

- **Lack of Personalized Recommendations:** When the agent makes a generalized recommendation without using all the information provided by the user, we consider this as lack of personalized recommendation.

I have a credit card due of $2800. The total credit line is $3800. If I don't pay a minimum of $100 by the end of billing cycle, my APR would be 27%. If I pay the minimum amount by the end of billing cycle, APR will be 25%. My billing cycle is from March 25th to April 24th. Today is March 31st. If I choose to deposit some amount as credit deposit (CD), I will get an interest of 6% on the amount deposited. Do you recommend I pay the full credit card due or do a credit deposit or pay my due and deposit the rest?

Given the information you have provided, I recommend that you pay the full credit card due of $2800. Here are a few reasons why:

- If you don't pay the full amount, you will be charged interest at a rate of 27%. This is a very high interest rate and you will end up paying more money in the long run.
- If you pay the minimum amount, your APR will be 25%. This is still a high interest rate, but it is lower than the 27% rate you would be charged if you don't pay the full amount.
- If you deposit some amount as a credit deposit, you will only earn interest on the amount deposited. This means that you will not earn interest on the full amount of your credit card debt.
- If you pay your due and deposit the rest, you will still be charged interest on the amount you don't pay. This means that you will end up paying more money in the long run.

The best way to save money on interest is to pay your credit card balance in full each month. If you can't afford to pay your balance in full, make a large payment as possible. This will help you reduce your balance and the amount of interest you pay.

If you are struggling to manage your credit card debt, consider talking to a credit counselor. They can help you develop a plan to get out of debt.

Here is a table that shows how much interest you would pay on your credit card debt if you paid the minimum amount, the full amount, or deposited some amount as a credit deposit:

Payment Option	Interest Charged	Total Amount Paid
Pay minimum amount	$784.40	$3584.40
Pay full amount	$0	$2800
Deposit some amount as a credit deposit	$168	$3062

As you can see, you would save the most money by paying the full amount of your credit card debt.

Additionally, if you deposited some amount as a credit deposit, you would still be charged interest on the amount you owe on your credit card. This means that you would end up paying more money in the long run.

Figure 1: Bard Response (Q11-AB11).

KA

I have a credit card due of $2800. The total credit line is $3800. If I don't pay a minimum of $100 by the end of billing cycle, my APR would be 27%. If I pay the minimum amount by the end of billing cycle, APR will be 25%. My billing cycle is from March 25th to April 24th. Today is March 31st. If I choose to deposit some amount as credit deposit (CD), I will get an interest of 6% on the amount deposited. Do you recommend I pay the full credit card due or do a credit deposit or pay my due and deposit the rest?

Based on the information you provided, I would recommend paying off the full credit card balance of $2800 before the end of the billing cycle to avoid paying any interest charges. Here's why:

1. High APR: If you don't pay the minimum payment of $100 by the end of the billing cycle, you will be charged a high APR of 27%. This can quickly add up and increase the amount you owe.
2. Lower APR with minimum payment: If you make the minimum payment of $100 by the end of the billing cycle, your APR will be 25%. While this is lower than the 27% APR, it is still a high interest rate.
3. Interest on credit deposit: While depositing some amount as a credit deposit may earn you a 6% interest, it may not be the best use of your funds as you will still be charged interest on the outstanding credit card balance.
4. Credit utilization: By paying off the credit card balance in full, your credit utilization ratio will be low, which is beneficial for your credit score. A low credit utilization ratio signals to lenders that you are a responsible borrower.

Overall, paying off the full credit card balance before the end of the billing cycle is the best option to avoid interest charges and maintain a healthy credit score.

Figure 2: ChatGPT Response (Q11-AC11).

- **Mathematical Errors:** We consider errors like rounding errors, calculation errors, etc. as mathematical errors.
- **Perceptual Errors:** When the agent misinterprets information given by the user or makes assumptions on un-

known data, we consider these as perceptual errors.

- **Grammatical Errors:** We consider typos, grammatical errors, etc. as grammatical errors (we encountered these errors only in Telugu text generated by ChatGPT).
- **Lack of Visual Aids:** When the agent doesn't use visual aids like tables, graphs, etc. in its response, we consider these as lack of visual aids.

Table 3 shows the percentage of queries for which the chatbots exhibited each of these errors. We also list out the individual query identifiers. Qi denotes the query identifier as previously defined (and also shown in Table 1). ABi and ACi refer to the corresponding Bard and ChatGPT responses respectively. 'i' denotes the identifier (number). Figures 1 and 2 show the response generated by Bard and ChatGPT chatbots respectively. For this one query, Bard made use of a table (though it misinterpreted user information) and ChatGPT did not.

Error Category	Queries	% of Bard Queries	% of ChatGPT Queries
Lack of Personalized Recommendations	Q1-AB1, Q3-AB3, Q3-AC3, Q4-AB4, Q5-AB5, Q6-AC6, Q7-AC7, Q8-AB8, Q9-AB9, Q10-AC10, Q11-AC11, Q12-AB12, Q12-AC12, Q13-AB13	53.84%	46.15%
Mathematical Errors	Q2-AB2, Q9-AC9, Q10-AB10	15.38%	7.69%
Perceptual Errors	Q8-AC8, Q10-AB10, Q11-AB11	15.38%	7.69%
Grammatical Errors	Q6-AC6, Q7-AC7	0%	15.38%*
Lack of Visual Aids	All except Q11-AB11	92.30%	100%

Table 3: Table showing % of queries for which the chatbots exhibited different errors along with individual query-response identifiers. 'Qi' denotes the query identifier, 'ABi' and 'ACi' represent the corresponding Bard and ChatGPT responses respectively where 'i' is the identifier.

Discussion and Conclusion

The application of language models in the finance industry has witnessed a surge in recent times due to their ability to process vast volumes of unstructured data and extract valuable insights. This paper delves into the performance of two prominent language models, Bard and ChatGPT, within the finance domain.

We also find the following challenges in evaluating LLM-based systems for finance domains:

- C1: Changing nature of answers for the same question. How does one create reference test cases since the answers change over time?
- C2: Inability of the chatbots to do numeric reasoning
- C3: Presenting results with easy to follow graphics.
- C4: Support for languages used by customers from different population groups. We considered AAVE - (African American Vernacular English) and Telugu, an Indian language spoken by nearly 100m people world-wide.
- C5: Evaluation the response of users from a diverse set of background. We only considered college students in this study.

C1 can be mitigated by carefully cataloging questions and system answers by identifiers that account for changing behavior over time. For C2, integration with numeric solvers like Wolfram may help (Wolfram 2023) although this makes the systems non-learnable over time. For C3, different data presentation strategies need to be tried. For C4, the LLM models or the chatbots need to be enhanced. For C5, more experiments are needed with inputs carefully modeling the characteristics of the different user groups. These are just preliminary challenges and we expect them to grow as more researchers will try LLM-based systems in complex and diverse application scenarios.

While our study only comprised thirteen queries, we meticulously selected them to cover various categories of credit card finance. However, there exists ample scope for more extensive testing of these chatbots by expanding the number of queries under each category or including additional categories like student loans and stock purchases. By doing so, we can gain a better understanding of the efficacy of language models in different financial domains and improve their functionality in real-world scenarios.

References

Alberto Pozanco, D. B., Kassiani Papasotiriou. 2022. PFPT: a Personal Finance Planning Tool by means of Heuristic Search and Automated Planning.

Bang, Y.; Cahyawijaya, S.; Lee, N.; Dai, W.; Su, D.; Wilie, B.; Lovenia, H.; Ji, Z.; Yu, T.; Chung, W.; et al. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity. *arXiv preprint arXiv:2302.04023*.

Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y. T.; Li, Y.; Lundberg, S.; et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Frieder, S.; Pinchetti, L.; Griffiths, R.-R.; Salvatori, T.; Lukasiewicz, T.; Petersen, P. C.; Chevalier, A.; and Berner, J. 2023. Mathematical capabilities of chatgpt. *arXiv preprint arXiv:2301.13867*.

Google, . 2023. Google BARD. In *https://bard.google.com/*.

Hu, B.; Xia, J.; Zheng, J.; Tan, C.; Huang, Y.; Xu, Y.; and Li, S. Z. 2022. Protein Language Models and Structure Prediction: Connection and Progression. *arXiv preprint arXiv:2211.16742*.

Li, X. L.; Kuncoro, A.; Hoffmann, J.; de Masson d'Autume, C.; Blunsom, P.; and Nematzadeh, A. 2022. A systematic investigation of commonsense knowledge in large language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 11838–11855.

Min, B.; Ross, H.; Sulem, E.; Veyseh, A. P. B.; Nguyen, T. H.; Sainz, O.; Agirre, E.; Heinz, I.; and Roth, D. 2021. Recent advances in natural language processing via large pre-trained language models: A survey. *arXiv preprint arXiv:2111.01243*.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Wolfram, S. 2023. ChatGPT Gets Its "Wolfram Superpowers"! *https://writings.stephenwolfram.com/2023/03/chatgpt-gets-its-wolfram-superpowers/*.

Wu, S.; Irsoy, O.; Lu, S.; Dabravolski, V.; Dredze, M.; Gehrmann, S.; Kambadur, P.; Rosenberg, D.; and Mann, G. 2023. BloombergGPT: A Large Language Model for Finance. arXiv:2303.17564.

Xiao, Z.; Liao, Q. V.; Zhou, M.; Grandison, T.; and Li, Y. 2023. Powering an AI Chatbot with Expert Sourcing to Support Credible Health Information Access. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, 2–18.

Yue, T.; and Au, C. C. 2023. GPTQuant's Conversational AI: Simplifying Investment Research for All. *Available at SSRN 4380516*.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; Du, Y.; Yang, C.; Chen, Y.; Chen, Z.; Jiang, J.; Ren, R.; Li, Y.; Tang, X.; Liu, Z.; Liu, P.; Nie, J.-Y.; and Wen, J.-R. 2023. A Survey of Large Language Models. arXiv:2303.18223.