31st International Conference on
Automated Planning and Scheduling
August 2-13, 2021, Guangzhou, China (virtual)

FinPlan 2021

Preprints of the 2nd Workshop on
Planning for
Financial Services (FinPlan)

Edited by:

Shirin Sohrabi, Sameena Shah, Daniele Magazzeni and Daniel Borrajo

Organization

Shirin Sohrabi
IBM Research

Sameena Shah
J.P.Morgan AI Research

Daniele Magazzeni
J.P.Morgan AI Research & King's College London, UK

Daniel Borrajo
J.P.Morgan AI Research & Universidad Carlos III de Madrid, Spain

Program Committee

Daniel Borrajo (J.P. Morgan AI Research, consultant, USA)
Amedeo Cesta (ISTC-CNR, Italy)
Giuseppe De Giacomo (Sapienza Università di Roma, Italy)
Mark Feblowitz (IBM, USA)
Fernando Fernández (Universidad Carlos III de Madrid, Spain)
Sarah Keren (Harvard University, USA)
Daniele Magazzeni (J.P. Morgan AI Research, UK)
Fabio Mercorio (Università di Milan-Bicocca, Italy)
Sameena Shah (J.P. Morgan AI Research, USA)
Shirin Sohrabi (IBM, USA)
Biplav Srivastava (University of South Carolina, USA)

Foreword

Planning is becoming a mature field in terms of base techniques and algorithms to solve goal- oriented tasks. It has been successfully applied to many domains including classical domains such as logistics or mars rovers, or more recently in oil and gas, as well as mining industry. However, very little work has been done in relation to financial institutions problems. Recently, some big financial corporations have started AI research labs and researchers at those teams have found there are plenty of open planning problems to be tackled by the planning community. For example, these include, trading markets, workflow learning, generation and execution, transactions flow understanding, risk management, fraud detection and customer journeys.

FinPlan'21 is the second workshop on Planning for Financial Services held in conjunction with ICAPS, whose aim is to bring together researchers and practitioners to discuss challenges for Planning in Financial Services, and the opportunities such challenges represent to the planning research community. The workshop consisted of paper presentations and a panel that discussed the current state of planning applied to finance, as well as open problems.

Shirin Sohrabi, Sameena Shah, Daniele Magazzeni and Daniel Borrajo, August 2021

Contents

A Planning Approach to Agile Project Management. The JIRA Planner

Salwa Alamir, Parisa Zehtabi Rui Silva, Alberto Pozanco,

Daniele Magazzeni, Daniel Borrajo, Sameena Shah, Manuela Veloso
J.P. Morgan Chase
salwa.alamir, parisa.zehtabi, rui.silva, alberto.pozancolancho,
daniele.magazzeni, daniel.borrajo, sameena.shah, manuela.veloso@jpmorganchase.com

Abstract

Currently, planning software development is mostly performed by humans. This task requires reasoning over multiple factors and constraints. Thus, it takes humans time to generate those plans. Also, the resulting plans often do not estimate well how good specific developers will address a task or how long it will take. In this paper, we present an automated approach to generate project plans, assigning human resources to tasks according to previous projects, and taking into account a variety of real constraints. We combine machine learning, to acquire a person's key skills based on previous development tasks performed, with planning technology in order to provide a unified end-to-end project management tool. This planning application has been developed within a large corporation utilizing historical data from an internal system that stores project tasks. We validate our approach by comparing the plans originally proposed by humans, against those generated by the planning tool. For this comparison, we contribute a set of metrics that assess different properties of the plans, such as the quality of task assignments.

Introduction

Software Project Scheduling (SPS) is an optimization problem within the broader field of Project Management that amounts to decide *who* does *what* and *when* on a software project Chicano et al. (2011). This problem is usually solved manually by project managers, who try to accommodate all the resources and constraints to create a plan that maximizes/minimizes a set of objective functions. In recent years, there have been some attempts to automate this process by using different Artificial Intelligence (AI) techniques that range from genetic algorithms Chang, Christensen, and Zhang (2001); Rodríguez et al. (2011) to ant colony optimization Xiao, Gao, and Huang (2015). Most current AI based approaches assume a static view of the project, not considering dynamic events such as the leave of an employee or the arrival of new tasks Rezende et al. (2019). Moreover, given the lack of publicly available data, they usually test their models in synthetically generated benchmarks, which might undermine the credibility of the computed solutions Vega-Velázquez, García-Nájera, and Cervantes (2018).

In this paper, we propose to combine planning and learning to solve the SPS problem. It is important to note that the type of planning we consider in this paper concerns about *resource allocation* and the terminology has been used interchangeably. For the learning side of our approach, we obtain task data from projects within J.P. Morgan Chase. Values such as developer skills are learned from the historical data using an unsupervised (clustering) approach. The values are then provided to the planner as inputs for the resource allocation. Empirical results show how our model can produce solutions fast. Also, they resemble the ones generated by project managers.The approach got positive feedback in a user study stating that the plans returned by our approach could be used within J.P. Morgan Chase.

Project development

The software development process at many companies follows the Agile methodology. In this section, we first introduce the Agile methodology and JIRA, a collaborative tool for tracking and managing projects.

In the Agile methodology for software development, a developers team $\mathcal{D} = \{d_1, \ldots, d_D\}$ and a manager m work on a project. At any given time there exists a pool of tasks $\mathcal{T} = \{t_1, \ldots, t_T\}$ that eventually need to be completed. The timeline of the project is divided into *sprints* of $\mathcal{L} = \{1, \ldots, N\}$ days. Typically the sprints last 2 weeks, *i.e.* $N = 14$, but this varies across teams.

Before a new sprint starts, the team meets for a *Sprint Planning Meeting*, where they: add new tasks to the pool $\mathcal{T}$; order the tasks in the pool by *priority* (Low, Medium, High, Major, Show Stopper); break down larger tasks into smaller more manageable subtasks; add dependencies between the tasks (*i.e.*, define task precedence); estimate the complexity of the tasks using a relative scale of effort dubbed *story points*;[1] and, finally, commit on the subset of tasks that should be completed by the end of the sprint, given the team bandwidth.

[1]Story points reward team members for solving problems based on complexity, and not on time spent.

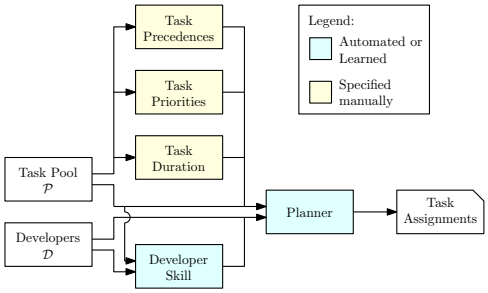


Figure 1: Architecture of the planning system.

The subset of tasks to be completed in the sprint is then assigned to the developers. This process is constrained by the skills of the developers and their personal preferences. The tasks follow a *workflow*. The default workflow is composed of three states: Open, In Progress and Closed. Additionally, we also assume that each task is assigned to a single developer, *i.e.*, it is not reassigned between the Open and Close states.

At J.P. Morgan Chase, the Agile methodology is supported by JIRA—an online collaborative tool for teams to perform sprint planning activities. In this work, the relevant features extracted are either industry standard (story points) or mandatory fields (description and priority).

Approach

We propose a planning approach for solving the aforementioned SPS problem, combining planning and scheduling, with learning techniques. Given a task pool $\mathcal{T}$ and a set of developers $\mathcal{D}$ as inputs, the goal of the planning system is to output a schedule of the tasks to be completed by each developer throughout the sprint. This scheduling process assumes a daily temporal resolution, and is constrained by different properties of the tasks and the developers, which are also provided as inputs. We delve into these constraints in this section.

Our overall approach is depicted in Figure 1, where some of the aforementioned inputs are manually specified by the users, and others are automatically estimated through models learned from previous project data. In this paper, we propose a novel learning approach for estimating the expertise of a developer on a given task.

Developer Skill Estimator

The developer skill estimator learns a model $S(d,t) : \mathcal{D} \times \mathcal{T} \to [0,1]$ that estimates the expertise level of a developer d at a task t based on the tasks the developer has previously completed in the project. The more similar tasks the developer has completed, the higher its expertise. In practice, we cluster tasks based on their description, and assess the skills of the developer in terms of the space of clusters.

Learning clusters of tasks The first step aims to learn how to cluster project tasks into a set of *task classes* $\mathcal{C} = \{c_1, \ldots, c_k\}$. For this purpose, we assume that tasks can be clustered based on their *descriptions*, which contain text paragraphs that provide technical details on the task. We first build a dictionary $\mathcal{D}$ of all the words observed in the descriptions of the tasks within the project. We use this to create a vector representation of the tasks based on the TF-IDF method. Each task is then represented by a $|\mathcal{D}|$-dimensional vector where the vector task t_j has i-th entry $w_j(i) = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$, where $tf_{i,j}$ is the frequency of word i in the task description t_j, df_i is the frequency of i in all task descriptions, and N is the number of previous tasks in the project.

Given the vector representation of all tasks, we adopt the Latent Dirichlet Allocation (LDA) topic modelling method in order to extract tasks clusters Blei, Ng, and Jordan (2003). Like most unsupervised clustering methods, LDA depends on a parameter k that specifies the number of clusters. For each project, we select a value k that leads to the highest cross-validated *coherence score* Röder, Both, and Hinneburg (2015). After trained, the LDA model allows us to cluster a new task. Specifically, for a new task t, $LDA : \mathcal{T} \to [0,1]^k$ outputs a distribution over the set of k tasks classes $\mathcal{C}$. In practice, we assume that each task t is classified into the highest probability cluster outputted by the LDA model—for convenience, we denote with $C : \mathcal{T} \to \mathcal{C}$.

Estimating the expertise of each developer for every task We assume the expertise of a developer d on a task t is captured by his/her expertise on the corresponding task class $c = C(t)$. Specifically, the expertise of d in task class c is defined as the normalized frequency of this class in the history of tasks the developer has completed, and can be found on JIRA. Formally,

$$S(d,t) = \frac{\sum_{t' \in \mathcal{H}(d)} \mathbb{I}\left(C(t') = C(t)\right)}{|\mathcal{H}(d)|},$$

where $\mathcal{H}(d)$ denotes the history of tasks completed by developer d, and where $\mathbb{I}(x = y)$ is the indicator function taking value 1 if $x = y$, and 0 otherwise. Consequently, the expertise of a developer d at a task t grows with the number of tasks of similar class he/she has completed.

Manually Specified Inputs

We assume some properties of the tasks and the developers are to be estimated manually by the team, and provided as input to the planner.

Task Duration Estimator The planner is provided a map $D : \mathcal{T} \to [1, N]$, where $D(t)$ denotes the duration of task t in days. Moreover, we assume that the duration of a given task ranges between a minimum of 1 day and a maximum of N days—the entire duration of the sprint.

Task Dependencies The planner is provided a map $R(t)$ denoting the set of tasks that need to be completed before task t can start—*i.e.,* its dependencies. Formally, $R : \mathcal{T} \to \mathcal{P}(\mathcal{T})$, where $\mathcal{P}(\mathcal{T})$ is the power set of all tasks. If task t has no dependencies, we have $R(t) = \emptyset$.

Task Priorities The task priorities—LOW, MEDIUM, HIGH, MAJOR, SHOW STOPPER—are provided as inputs. We assign each priority a value in $\{1, \ldots, 5\}$. Formally, we let $P(t)$ denote the priority of task t.

Planner

We adopted an Integer Linear Programming (ILP) approach to model the SPS problem as an optimization problem. Our approach starts with the following binary variables that take value 1, when

- $x_{i,j}$, if we assign task $j \in \mathcal{T}$ to developer $i \in \mathcal{D}$;
- $y_{i,j,k}$, if developer $i \in \mathcal{D}$ starts task $j \in \mathcal{T}$ on day $k \in \mathcal{L}$;
- $z_{i,j,k}$, if developer $i \in \mathcal{D}$ works on task $j \in \mathcal{T}$ on day $k \in \mathcal{L}$,

yielding a total of $(|\mathcal{T}| \times |\mathcal{D}|) + 2(|\mathcal{T}| \times |\mathcal{D}| \times N)$ variables.

The objective function is defined as

$$\max_{x,y,z} \sum_{\substack{i \in \mathcal{D} \\ j \in \mathcal{T}}} S(i,j)\ P(j)\ x_{i,j}\ -\ \alpha \sum_{\substack{i \in \mathcal{D},\ j \in \mathcal{T} \\ k \in \mathcal{L}}} k\ y_{i,j,k},$$

where α is a normalization constant. The first term of the objective function aims to maximise the number of tasks completed, while trying to assign each task to developers with higher corresponding expertise, and also considering task priority. The second term of the objective function aims at scheduling tasks as early as possible in the sprint. This second term is weighted by a small scalar α as it is a secondary objective. Our model considers multiple constraints.

Number of developers per task We enforce that each task may only start once, by letting

$$\sum_{i \in \mathcal{D}, k \in \mathcal{L}} y_{i,j,k} \leq 1, \quad \forall j \in \mathcal{T}. \tag{1}$$

Moreover, each task may only be assigned to one developer:

$$\sum_{i \in \mathcal{D}} x_{i,j} \leq 1, \quad \forall j \in \mathcal{T}. \tag{2}$$

Task dependencies The dependencies are enforced by two sets of constraints. The first ensures any task j is only assigned if all dependencies $R(j)$ are also assigned:

$$\sum_{i \in \mathcal{D}} x_{i,j} \leq \sum_{i \in \mathcal{D}} x_{i,r}, \quad \forall j \in \mathcal{T},\ r \in R(j). \tag{3}$$

The second set of constraints ensures that a task j may only start after its dependencies $R(i)$ finish:

$$\sum_{i \in \mathcal{D}} \sum_{k'=1}^{k+D(r)} y_{i,j,k'} \leq \left(1 - \sum_{i \in \mathcal{D}} y_{i,r,k}\right) N, \quad \forall j \in \mathcal{T},\ r \in R(j),\ k \in \{1, \ldots, N - D(r)\}. \tag{4}$$

This constraint enforces that, if a dependency $r \in R(j)$ starts on a day k by some developer i (*i.e.*, $y_{i,r,k} = 1$), then task j may only start after r finishes at $k + D(r)$.

Multi-tasking We enforce that, on any given day, a developer i may work on at most $t_{\max}$ tasks in parallel. This is enforced by the set of constraints:

$$\sum_{j \in \mathcal{T}} z_{i,j,k} \leq t_{\max}, \qquad \forall i \in \mathcal{D},\ k \in \mathcal{L}. \tag{5}$$

Makespan We enforce that a task j may only start on a day k if it can be completed within the sprint

$$y_{i,j,k} = 0,\ \forall i \in \mathcal{D},\ j \in \mathcal{T},\ k \in \{N - D(t), \ldots, N\}. \tag{6}$$

Connections Between Variables The final set of constraints establishes the connections between the different decision variables in our model. We start by enforcing that a task j may only start on some day k if it is assigned to some developer i:

$$\sum_{k \in \mathcal{L}} y_{i,j,k} = x_{i,j}, \quad \forall i \in \mathcal{D},\ j \in \mathcal{T}. \tag{7}$$

Finally, if a developer i is assigned to task j, then i must be busy with j for the duration of the task:

$$\sum_{k \in \mathcal{L}} z_{i,j,k} = D(j)\ x_{i,j}, \quad \forall i \in \mathcal{D},\ j \in \mathcal{T}. \tag{8}$$

These constraints indirectly connect variables y and z, rendering it redundant to specify their direct connection.

Task Assignment

The task assignments generated by the planner associate tasks to developers and days of the sprint. Formally, for a given developer $i \in \mathcal{D}$ and day $j \in \mathcal{L}$, the assignment may map to, either a task t, or none if no tasks are to be performed by the developer on that day.

Experimental Evaluation

We empirically evaluate our approach and show that the solutions computed resemble those generated in real projects developed at J.P. Morgan Chase, and could be used within the company. Our evaluation is from a quantitative perspective—through a set of metrics comparing the resulting plans against those of the real projects—and from a qualitative perspective—through a user study.

Data Collection

We start by describing the process followed for collecting the data used for training and testing our approach. We used JIRA's REST API to fetch all projects, and corresponding historical tasks, stored in a specific server. All projects with less than 100 CLOSED tasks were discarded. We then generated a training and test datasets as follows. For each project, we first fetch the most recent CLOSED task, and then pull all tasks leading up to 1 month before that. This entire month of CLOSED tasks corresponds to the test set, whereas the training set corresponds to all the tasks before that. A second round of filtering occurs at this stage, by discarding all projects with less than 15 tasks in the test set. This yielded a grand total of 88 projects used in our experiments.

Table 1: Mean and standard deviation of the different metrics values from planning multiple sprint intervals (S). Test/Plan are the test tasks and proportion planned. Train tasks are used to train solvable projects. Devs is number of developers, and Runtime is time to solve the problem in seconds. Solved/Total is the proportion of projects that had at least 1 optimal solution.

S	m_1 (s.d)	m_2 (s.d)	m_3 (s.d)	Test/Plan Task	Train Task	Devs.	Runtime (sec)	Solved/Total
1	0.83 ± 0.22	0.42 ± 0.35	0.92 ± 0.13	$10 \pm 09/08 \pm 07$	231 ± 159	4 ± 3	08.18 ± 53.57	88/88
2	0.84 ± 0.21	0.40 ± 0.31	0.93 ± 0.11	$12 \pm 10/09 \pm 08$	235 ± 161	4 ± 3	38.91 ± 122.9	61/88
3	0.79 ± 0.25	0.45 ± 0.33	0.92 ± 0.12	$13 \pm 11/11 \pm 11$	223 ± 159	4 ± 3	86.38 ± 184.3	37/88

Quantitative Evaluation

We evaluate our approach by comparing the plans computed against sprints of real projects. For a given project and sprint, we assume the task pool $\mathcal{T}$ corresponds to the set of tasks that were performed by the available developers $\mathcal{D}$ during that sprint. The duration estimator of the tasks D was defined to match the duration of the tasks in the real project. For the first round of experiments we made two assumptions. First, task dependencies are provided as a manual input; if not provided, no dependencies exist. Second, we assumed that the developers could work on at most $t_{\max} = 1$ simultaneous tasks. We then impose a 15 minutes timeout for the planner. Thus, if a project plan exceeds this limit, we deem it *Unsolvable*. Given this setup, we assess if the solutions generated by the planner are *similar* to the actual plans using three metrics.

Jaccard Similarity This metric assesses if the planner schedules a similar number of tasks to be completed during the sprint. Formally, m_1 is the Jaccard similarity $m_1(\mathcal{T}, \hat{\mathcal{T}}) = \frac{|\mathcal{T} \cap \hat{\mathcal{T}}|}{|\mathcal{T} \cup \hat{\mathcal{T}}|}$, where $\mathcal{T}$ and $\hat{\mathcal{T}}$ are the set of tasks planned by the developers and by the planner, respectively. m_1 takes value 1 when the tasks planned are the same, and 0 when the planner outputs no tasks.

Task Matching This metric assesses if the task assignments computed by the planner are similar to those in the real plans. Let A denote the set of assignments (t_i, d_j) in the real plan, and $\hat{A}$ the set of assignments proposed by the planner. Formally, we define the metric as $m_2(A, \hat{A}) = \frac{\sum_{(t_i,d_j) \in \hat{A}} \mathbb{I}((t_i,d_j) \in A)}{|\hat{A}|}$. The metric takes value 1 when all the task assignments proposed by the planner match those followed by the developers. The metric takes value 0 when no task assignments match.

Expertise-Aware Task Matching Metric m_3 extends m_2 by assessing if the planner assigns tasks to developers with similar, or higher, estimated expertise levels. Formally, $m_3(A, \hat{A}) = \frac{\sum_{(t_i,d_j) \in \hat{A}} \mathbb{I}(S(t_i,\hat{d}_j) \geq S(t_i,d_j))}{|\hat{A}|}$. The metric takes value 1 when all the task assignments computed by the planner assign a developer with similar or higher expertise when compared to the assignment in the real plan. This metric is inspired by the observation that, on a given project, there may exist multiple developers equally capable of performing the same task.

We experimented on a project where a team of developers manually specified the dependencies existing between the tasks of the one month long test set. The plans were generated for 14 tasks and 6 programmers over a 2-week sprint. The metrics with and without precedences were the same (M1: 0.93, M2: 0.31, M3:0.92). However, the new plan showed a different order where the precedences were respected as per the user inputs. We ran the experiment on a 3-week sprint and observed the same behaviours between the plans with and without precedence defined. We then investigate the impact of varying sprint size.

Table 1 summarizes the results achieved under varying sprint sizes. The test set is divided by the number weeks in a sprint and is run accordingly (i.e: for a 1 week sprint, 4 runs per project, 2 weeks twice, and 3 weeks once). We observe that the number of tasks to plan for increases with respect to the size of the sprint; as expected, this affects runtime exponentially. Metric 1 reduces as the number of tasks increase due to overlapping tasks whilst the planner assigns only 1 task per person at a time. We observe that metric 2 is also reduced when developers have overlapping skillsets. The planner will assign tasks to any qualified developer. This is validated by Metric 3 which remains close to 0.92 regardless of sprint size.

Qualitative Evaluation

We obtained feedback from 3 project teams in the form of a survey comprised of 2 questions, where the responses are on a scale of *"Strongly Agree"* to *"Strongly Disagree"*:

- Q1: *Would you find an automated planning tool useful for planning your software developer sprints?*
- Q2: *Does the automatically generated plan make sense for your project and team? If not, why?*

All users responded 'Strongly Agree' to Q1, whereas a 33.3% 'Agree' and 66.7% 'Neutral' response for Q2. Enquiry into how to improve the generated plans yielded a request for learning task precedence. This is expected as the current planner is equipped to handle this feature as a manual input.

Conclusions

In this paper we contributed an approach to solve the Software Project Scheduling problem, combining planning and scheduling, with learning techniques. The introduction of the learning component successfully allows our approach to tailor the planner for different software engineering projects. We empirically evaluate our planning approach on a large number of real software engineering projects, showing that the plans computed resemble those generated by the humans themselves. In fact, we evaluate the quality of the plans produced both quantitatively—through a set of metrics that measure plan similarity—and qualitatively—through a user study. The results show that the plans returned by our approach could be used within our company. In the future, we plan to learn models for the remaining inputs to the planner. We also intend to correlate the implementation of the learning approach with developer competence and planner performance.

References

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3(Jan): 993–1022.

Chang, C. K.; Christensen, M. J.; and Zhang, T. 2001. Genetic algorithms for project management. *Annals of Software Engineering* 11(1): 107–139.

Chicano, F.; Luna, F.; Nebro, A. J.; and Alba, E. 2011. Using multi-objective metaheuristics to solve the software project scheduling problem. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011*, 1915–1922.

Rezende, A. V.; Silva, L.; Britto, A.; and Amaral, R. 2019. Software project scheduling problem in the context of search-based software engineering: A systematic review. *Journal of Systems and Software* 155: 43–56.

Röder, M.; Both, A.; and Hinneburg, A. 2015. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM International Conference on Web Search and Data Mining*, 399–408.

Rodríguez, D.; Carreira, M. R.; Riquelme, J. C.; and Harrison, R. 2011. Multiobjective simulation optimisation in software project management. In *Proceedings of the 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011*, 1883–1890.

Vega-Velázquez, M. Á.; García-Nájera, A.; and Cervantes, H. 2018. A survey on the software project scheduling problem. *International Journal of Production Economics* 202: 145–161.

Xiao, J.; Gao, M.; and Huang, M. 2015. Empirical Study of Multi-objective Ant Colony Optimization to Software Project Scheduling Problems. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015*, 759–766.

TD3-Based Ensemble Reinforcement Learning for Financial Portfolio Optimisation

Nigel Cuschieri and **Vince Vella** and **Josef Bajada**
Department of Artificial Intelligence,
Faculty of Information & Communication Technology
University of Malta

Abstract

Portfolio Selection (PS) is a perennial financial engineering problem that requires determining a strategy for dynamically allocating wealth among a set of portfolio assets to maximise the long-term return. We investigate state-of-the-art Deep Reinforcement Learning (DRL) algorithms that have proven to be ideal for continuous action spaces, mainly Deep Deterministic Policy Gradient (DDPG) and Twin Delayed Deep Deterministic Policy Gradient (TD3), for the PS problem. Furthermore, we investigate the effect of including stock movement prediction indicators in the state representation, and the potential of using an ensemble framework that combines multiple DRL models. Our experiments show that TD3-based models generally perform better than DDPG-based ones when used on real stock trading data. Furthermore, the introduction of additional financial indicators in the state representation was found to have a positive effect over all. Lastly, an ensemble model also showed promising results, consistently beating the baselines used, albeit not all other DRL models.

Introduction

Portfolio Selection (PS) has now been studied extensively for almost 70 years (Markowitz 1952; Pogue 1970; Rubinstein 2002; Zhou and Yin 2003; Fagiuoli, Stella, and Ventura 2007). Throughout literature, two major schools for investigating the PS problem are identified. These are the *Mean Variance Theory* (Markowitz 1952) originating from the finance community, and the *Capital Growth Theory* (Kelly 1956; Cover 1996) originating from information theory (Li and Hoi 2014). The Mean Variance Theory focuses on a single-period or batch portfolio selection. It aims to trade off expected return with risk (variance), to create an optimal portfolio subject to the investor's risk-return profile (Li and Hoi 2014). In contrast, the Capital Growth Theory focuses on multiple-period or sequential portfolio selection. Additionally, it aims to maximise the portfolio's expected growth rate, or expected log return (Li and Hoi 2014). Both theories pose a solution to the portfolio selection task. However, only the Capital Growth Theory incorporates the online machine learning perspective (Li and Hoi 2014). A variety of Online Portfolio Selection (OLPS) strategies and approaches, such as Online Newton Step (ONS) and Passive Aggressive Mean Reversion (PAMR) have been proposed, which are popularly used as baselines to evaluate stock trading performance (Islam et al. 2017; Jiang and Liang 2017).

Deep Reinforcement Learning (DRL) combines Reinforcement Learning (RL) with Deep Learning (DL) to address tasks with high-dimensional input and action spaces (Khadka et al. 2019). DL has become very popular in speech recognition (Noda et al. 2015) and image identification (Liu et al. 2020), and has been shown to work well with complex non-linear patterns (Liang et al. 2018). DRL algorithms such as Deep Deterministic Policy Gradient (DDPG) (Silver et al. 2014; Hunt et al. 2016), and Twin Delayed Deep Deterministic Policy Gradient (TD3) (Fujimoto, Van Hoof, and Meger 2018) provide greater support for continuous action spaces such as the one encountered in PS, and for this reason, DDPG and its variants have been proposed for this problem domain (Hegde, Kumar, and Singh 2018; Zhang et al. 2020; Kanwar 2019; Gran, Holm, and Søgård 2019).

At the time of writing, TD3 has been scarcely used in PS. The findings presented by Fujimoto, Van Hoof, and Meger (2018), together with the use of TD3 in other domains (Li and Yu 2020; MacHalek, Quah, and Powell 2020), suggest that TD3 also has the potential to perform well in financial planning for PS.

In this work we present the results of different experiments that use TD3 for portfolio selection, mainly:

- TD3 using only normalised asset logarithmic returns in the state representation, with different time window sizes.
- TD3 using normalised asset logarithmic returns together with Robust Median Reversion (RMR) predictions in the state representation, also using different time window sizes.
- An ensemble model that combines the best four models together and switches between them according to their performance.

These models were compared to their DDPG counterparts and also the state-of-the-art OLPS baselines. The performance of each model was measured in terms of Average Daily Yield, Sharpe Ratio (Sharpe 1966), Sortino Ratio (Sortino and Price 1994), and Maximum Drawdown. Throughout the study, we follow the below two assumptions:

- **Zero Slippage:** All market assets are liquid enough to make every trading at the last price immediately possible when an order is placed.
- **Zero impact on market:** The investments done by our trading agents are so small that they have no effect on the market.

Due to this, the implemented models are mainly directed at optimising portfolios for retail consumers. All the software developed for this work has been open-sourced and made publicly available on GitHub [1].

Background

Problem Setting

OLPS algorithms typically model the PS problem as a financial market with m assets and a series of n trading steps, each consisting of wealth distributed over all the assets. The *price relative vector*, consisting of m dimensions, represents the portfolio price change for each trading step. This is defined as $x_t \in \mathbb{R}_+^m, t = 1, ..., n$, where the t^{th} price relative vector consists of the ratio of t^{th} closing prices to the previous $(t-1)^{th}$. Therefore, the investment in an asset i during step t is increased (multiplied) by a factor of $x_{t,i}$ (Li and Hoi 2014). Naturally, the investment for the asset would decrease if the factor is less than 1. The *portfolio vector* represents the allocation of investment wealth throughout the trading steps, and is denoted as b_t, for the t^{th} portfolio. Therefore, $b_{t,i}$ represents the ratio of wealth assigned to the i^{th} asset. No negative entries are allowed in the capital investment, as it is assumed that a portfolio is self-financed (Li and Hoi 2014). A portfolio strategy for n periods can be denoted as: $b_1^n = b_1, ..., b_n$. At any period t, the capital is adjusted according to portfolio b_t at the opening time. Then, the position is held until the closing time is met. Therefore the portfolio value, when excluding transaction costs, increases by a factor of $b_t^T x_t = \sum_{i=1}^{m} b_{t,i} x_{t,i}$. The goal of a portfolio manager is to produce a portfolio strategy b_1^n, that is computed in a sequential fashion in order to achieve certain targets, such as maximising the portfolio cumulative wealth S_n (Li and Hoi 2014).

Reinforcement Learning

Reinforcement Learning is typically modelled as a Markov Decision Process (MDP). An MDP is a directed graph with state nodes, S, action nodes A, and directed edges connecting states to actions and actions to states. An action can have multiple outcomes if its effects are stochastic, with a probability associated with each outcome. State transitions can also carry rewards. An optimal policy, π^*, maps states to actions, such that the sum of all rewards, R, for each transition between a state, s_t, executed at time step t, and its successor, s_{t+1}, with a chosen action, a, is maximised. A policy could also be in itself stochastic, selecting which action, a, to take in a specific state, s, out of a probability distribution, $\pi(a|s)$. A discount factor, $0 \leq \gamma \leq 1$, makes the policy prefer solutions that achieve rewards sooner. Equation 1 shows the expected reward of an optimal policy, where R_t is the reward obtained in step t, and n is the total number of actions executed ($t = 0$ corresponds to the initial state).

$$\max \mathbb{E}\left[\sum_{t=0}^{n} \gamma^t R_t\right] \tag{1}$$

The value of any state can thus be computed as the expected return over the possible actions and outcomes, each weighted by their respective probabilities. The *Bellman Equation* (Bellman 1954) shown in Equation 2, defines the value, $v(s_t)$, of a state, s_t, by computing the expected discounted reward that can be obtained from each of its possible successors, s_{t+1}.

$$v(s_t) = \mathbb{E}\left[R_{t+1} + \gamma v(s_{t+1}) | s_t\right] \tag{2}$$

Similarly, the *state-action value function* or *Q-Function*, measures how good it is for an action to be taken in a specific state. Equation 3 defines the Q-Function of an action, a_t, to be taken in a state, s_t, given a policy, π. This is expressed in terms of the immediate expected rewards, R_{t+1}, from the possible successors of s_t after executing a_t, and the discounted value of each possible action, a_{t+1}, that can be taken at step $t+1$.

$$q_\pi(s_t, a_t) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1}) \big| s_t, a_t\right] \tag{3}$$

Finding an optimal policy is very hard (Papadimitriou and Tsitsiklis 1987), and the goal of RL is to learn a policy, π, that approximates π^* as closely as possible. At each step, the learning agent can sense the *state* of its environment, decide which *action* to take, perceive the new state, and collect a numeric *reward* signal associated with the state transition.

RL is also effective when the probabilities of the MDP are not known, since the RL *agent* can also infer these probabilities by interacting with the *environment* repeatedly (Sutton and Barto 1999) and sampling the outcomes. This approach is known as model-free RL and is ideal for problem domains such as stock trading or PS, where the outcome of investing in a specific asset is hard to predict (Liang et al. 2018; Jiang and Liang 2017).

Model-free RL algorithms are typically either *off-Policy*, where the agent improves a different policy than the one being followed, or *on-Policy*, where the agent optimises and follows the same policy (Sutton and Barto 1999). Q-Learning is one example of an off-policy algorithm, while SARSA is on-policy (Sutton and Barto 1999). In order to balance the trade-off between both approaches (Kanwar 2019), more sophisticated algorithms such as DDPG (Silver et al. 2014) and TD3 (Fujimoto, Van Hoof, and Meger 2018), have been proposed, which also incorporate an Artificial Neural Network (ANN) to learn the Q-Function and the policy.

Various attempts were made to use RL in finance, where the objective was to output discrete trading signals on singular assets (Cumming 2015; Deng et al. 2017). More recent studies have set new benchmarks for portfolio optimisation by using Deep Reinforcement Learning (DRL) (Jiang, Xu, and Liang 2017; Jiang and Liang 2017; Liang et al. 2018).

[1] https://github.com/NigelCusc/DDPG_TD3_Portfolio Optimization_tensorflow-1.15.4

These models operate over continuous action spaces, and are typically based on the DDPG framework (Silver et al. 2014; Hunt et al. 2016). These techniques were also extended to: work with more risk-averse strategies (Hegde, Kumar, and Singh 2018; Zhang et al. 2020), focus on asset correlation (Zhang et al. 2020), and also incorporate genetic algorithms and sentiment analysis (Gran, Holm, and Søgård 2019). Additionally, an ensemble strategy for automated stock trading with three different trading agents: Proximal Policy Optimisation (PPO), Advantage Actor Critic (A2C), and DDPG, has also been used (Yang et al. 2020). The aim of this ensemble strategy was to benefit from the best features of the three algorithms, thereby robustly adjusting to different market situations.

Deep Deterministic Policy Gradient

DDPG and its variants have been used in finance in various prior works (Hegde, Kumar, and Singh 2018; Zhang et al. 2020; Kanwar 2019; Gran, Holm, and Søgård 2019). DDPG combines Policy Gradient and Q-learning frameworks in an off-policy actor-critic framework, and provides greater support for continuous action spaces such as the one encountered in PS (Hunt et al. 2016). DDPG evolved from Deep Q-network (DQN) methodologies (Mnih et al. 2015), where deep learning was introduced to allow for policies to be learnt from highly dimensional state spaces. As its name implies, DDPG learns a deterministic policy that maps the state vector to an action vector.

If the optimal action-value function, $Q^*(s,a)$, is known, the best action for any given state, s, is defined as $a^*(s) = \arg\max_a Q^*(s,a)$. In a continuous action space, the function $Q^*(s,a)$ is presumed to be differentiable with respect to the action argument. This way an efficient, gradient-based learning rule for a policy $\mu(s)$ can be set up, making use of the approximation $\max_a Q(s,a) \approx Q(s,\mu(s))$ (Kanwar 2019). $Q^*(s,a)$ is thus defined as:

$$Q^*(s,a) = \underset{s' \sim P}{E}[r(s,a) + \gamma \max_{a'} Q^*(s',a')] \qquad (4)$$

where $s' \sim P$ is the next state s' sampled from some distribution $P(\cdot|s,a)$ and $r(s,a)$ is any immediate reward obtained from executing action a in state s. In order to approximate $Q^*(s,a)$, a neural network based predictor is used, $Q_\phi(s,a)$, where ϕ corresponds to its parameters and the collected set of transitions D consisting of (s,a,r,s',d), with $d \in \{0,1\}$ indicating whether s' is a terminal state.

DDPG makes use of a *Replay Buffer* in which experiences are accumulated by the algorithm, and a mini-batch is used to compute the gradient in every iteration (Hunt et al. 2016). The neural network is trained to minimise an error function such as the Mean Squared Bellman Error (MSBE) (Zhang, Boehmer, and Whiteson 2020). In order to improve the convergence stability, updates to the parameters, ϕ, towards the target network are performed using Polyak Averaging, $\phi' \leftarrow p\phi' + (1-p)\phi$, with $p \in [0,1]$ (Polyak and Juditsky 1992). In order to encourage exploration, noise is also added to the actions during training, typically using the Ornstein-Uhlenbeck method (Uhlenbeck and Ornstein 1930; Hunt et al. 2016; Lillicrap et al. 2015).

One of the weaknesses of DDPG is its susceptibility to overestimation due to taking the maximum of the estimated value, which ends up propagating error that builds up over time, resulting in an inferior policy.

Twin Delayed Deep Deterministic Policy Gradient

The TD3 algorithm was introduced to address the issue of overestimation bias in DDPG (Fujimoto, Van Hoof, and Meger 2018). It uses *Clipped Double Q-learning* (Van Hasselt 2010) to replace the critic in the actor-critic framework, together with *Delayed Policy Updates* and *Target Policy Smoothing Regularisation* (Fujimoto, Van Hoof, and Meger 2018). This approach was found to significantly decrease the overestimation bias and achieve better stability.

Dankwa and Zheng (2019) compare this framework to DDPG, PPO, Trust region policy Optimisation (TRPO), Actor-Critic using Kronecker-factored Trust Region (ACKTR) and Soft Actor-Critic (SAC) on the MuJoCo pybullet continuous control environment. The TD3 model achieved a higher average reward when compared with the other state-of-the-art models.

Clipped Double Q-Learning for Actor-Critic In *Double Q-learning* (Van Hasselt, Guez, and Silver 2016), two separate value estimates are maintained. Each of these is used to update the other. With independent value estimates, unbiased estimates of the actions selected using the opposite value estimate can be made (Fujimoto, Van Hoof, and Meger 2018). Additionally, in *Double DQN*, the target network is used as one of the value estimates, and a policy by greedy maximisation of the current value network is obtained rather than the target network (Van Hasselt, Guez, and Silver 2016). Double Q-learning is found to be more effective than Double DQN but does not eliminate the overestimation bias (Fujimoto, Van Hoof, and Meger 2018). The critics are not entirely independent, as the learning targets use the opposite critic and the same replay buffer. This in turn may cause the overestimation to be significant in certain areas of the state space.

Clipped Double Q-Learning addresses this problem by taking the minimum of the two estimates:

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi_1}(s')) \qquad (5)$$

where r is the reward received, s' is the new state of the environment, γ is a discount factor, π_ϕ is the policy with parameters ϕ, and $Q_{\theta'}$ is the function approximator with parameters θ updated at each time step also using Polyak averaging for smoother network updates (Fujimoto, Van Hoof, and Meger 2018). This way, the value target cannot introduce any additional overestimation over the standard Q-learning target. This can still introduce an element of underestimation bias due to underestimated action values not explicitly propagated through the policy update, but this is generally more preferable to overestimation.

Target Networks and Delayed Policy Updates The work done by Fujimoto, Van Hoof, and Meger (2018) shows how error accumulation can be reduced by maintaining a stable target. The learning behaviour with and without the target networks were examined on both the critic and the actor, and

results suggest that failure may occur in the interplay of the updates done by the actor and critic. A value estimate would deviate because of overestimation with a poor policy, and in turn, the policy would become poor because of inaccurate value estimates (Fujimoto, Van Hoof, and Meger 2018). Due to this, they suggest that the policy network should be updated at a lower frequency than that of the value network to minimise the error before introducing a policy update. This is done by updating the policy and target networks after a fixed number of updates, d, to the critic, while still updating the target networks slowly $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$, with $\tau \in [0, 1]$. Delaying the policy updates also avoids repeated updates when the critic has not changed.

Target Policy Smoothing Regularisation TD3 also makes use of Target Policy Smoothing, which mimics the learning updates of SARSA (Sutton and Barto 1999). This is used to reduce the target variance induced by the approximation error. This approach is based on the assumption that similar actions should have a similar value (Fujimoto, Van Hoof, and Meger 2018). Function approximation should do this implicitly, but the relationship between similar actions can be forced explicitly by modifying the training procedure.

Fujimoto, Van Hoof, and Meger (2018) state that fitting the value of a small area around the target action should help smooth the value estimate with similar state-action value estimates. This is done by adding clipped noise to keep the target within a small range and averaging over mini-batches, as shown in Equation 6:

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon) \tag{6}$$

where $\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$ is sampled from a Gaussian distribution with standard deviation σ clipped at $\pm c$.

Ensemble Deep Reinforcement Learning Strategy

The study done by Yang et al. (2020) employed an ensemble strategy for automated stock trading with three different trading agents. Each of these trading agents consisted of a distinct actor-critic based algorithm. These are PPO, Advantage Actor-Critic (A2C), and DDPG. The ensemble strategy aimed to inherit and integrate the best features of the three algorithms, thereby robustly adjusting to different market situations (Yang et al. 2020). The proposed ensemble strategy first trains the three agents, then validates all agents via Sharpe Ratio (Sharpe 1994) to find the best model, which is then used for trading. The results obtained show that the ensemble strategy outperformed the three individual RL algorithms (Yang et al. 2020).

Experiments

Throughout our experiments, we evaluate and compare models with the DDPG and TD3 DRL frameworks against the baselines described in Table 2. Additionally we evaluate the performance of our models with an enhancement in the state representation, making use of financial indicators based on the RMR OLPS algorithm. Finally we evaluate the performance of an ensemble strategy, making use of our models.

Datasets

Throughout our experiments we make use of two datasets consisting of real stock trading data. These are:

- **NYSE(N)** Part of an OLPS benchmark dataset [2] consisting of actual daily closing prices of a variety of stocks from the New York Stock Exchange.
- **SP500** Dataset gathered from yahoo finance [3]. The assets chosen for the SP500 dataset were selected according to market capitalisation and their liquidity.

Since the OLPS benchmark dataset only has daily close prices, and most benchmarks also use close prices, we also make use of daily close prices and process them into logarithmic returns.

Environment

A portfolio consists of a number of assets. Our models are allowed to change portfolio weights after each time step t denoting a market open day. The close prices for day t are stored into a *price vector* $\mathbf{v}_t = \{v_{t,0}, v_{t,1}, ..., v_{t,n}\}$. The *price relative vector*, $\mathbf{y}_t$, is then calculated from the price movements of two consecutive days, $\mathbf{v}_{t-1}$ and $\mathbf{v}_t$ as follows:

$$\mathbf{y}_t = \left(1, \frac{v_{t,1}}{v_{t-1,1}}, \frac{v_{t,2}}{v_{t-1,2}}, ..., \frac{v_{t,n}}{v_{t-1,n}}\right) \tag{7}$$

where $v_{t,1}$ is the closing price of the first asset in the vector for day t, and n is the number of assets. The first element in the price vector, $v_{t,0}$, and the price relative vector $y_{t,0}$ are always equal to 1, in order to provide an option to keep some capital liquid without being held in any asset position (Jiang, Xu, and Liang 2017). The initial portfolio weight vector $\mathbf{w}_0$ is set to $(1, 0, ..., 0)$, and the sum of all the elements in the portfolio weight vector $\mathbf{w}_t$ at any step t, is always 1: $\forall t \sum_{i=0}^{n} \mathbf{w}_{t,i} = 1$.

A *state*, s_t for a time step, t, consists of an $m \times n$ matrix where m is the *window length* and n is the number of assets. The window length is a configurable parameter that denotes the number of past time steps considered relevant for each state. We experiment using the window sizes 3, 7, 11 and 14. Each value in the state corresponds to the normalised *log return*, $R_{t,i}$ of an asset, i, on a specific day, t is defined as:

$$R_{t,i} = \log\left(\frac{c_{t,i}}{c_{t-1,i}}\right) \tag{8}$$

where $c_{t,i}$ is the asset's close price for day t, and $c_{t-1,i}$ is its close price of the previous day, $t-1$. The log return data within the state is normalised using z-score normalisation $\frac{value-\mu}{\sigma}$ across all assets, where $value$ is the log return of an asset on day t. μ and σ are the mean and standard deviation of the log return of all assets on day t, respectively.

An *action*, α_t, is the weight vector $\mathbf{w}_t = (w_{t,0}, ..., w_{t,n})$ that distributes the allocation of capital across n assets at time step t (Hegde, Kumar, and Singh 2018; Zhang et al. 2020). Adjusting the portfolio's asset allocation is usually not free, but incurs *transaction fees*. These are either a

[2] http://www.mysmu.edu.sg/faculty/chhoi/olps/datasets.html

[3] https://finance.yahoo.com/

fixed percentage, or hidden inside the *spread* (the difference between the selling and buying price of an asset), and are incurred whenever the weights allocated to the assets change. The portfolio vector at the beginning of day t is w_{t-1}. At the end of the day, the portfolio weight vector needs to be adjusted in relation to any price movements to: $w'_t = \frac{y_t \odot w_{t-1}}{y_t \cdot w_{t-1}}$, where $\odot$ is the Hadamard product (element-wise multiplication). Before the next period starts, w'_t is then adjusted into w_t by reallocating the portfolio weights. The *transaction remainder factor*, $\mu_t \in (0, 1]$, is the factor by which the portfolio value shrinks due to this reallocation procedure. In our case, $\mu_t = c\sum_{i=1}^{m} |w'_{t,i} - w_{t,i}|$, where c is the transaction cost rate (Jiang, Xu, and Liang 2017). Throughout all our experiments, c is set as 0.25%.

The goal of the agent is to maximise the final portfolio value p_f at the end of the $t_f + 1$ period. Due to the agent not having control over the choice of the initial portfolio weights p_0, and the number of total time steps, t_f, this is equivalent to maximising the average logarithmic accumulated return R, from immediate rewards r_t as shown in Equation 9 (Jiang and Liang 2017):

$$R := \frac{1}{t_f} ln \frac{p_f}{p_0} = \frac{1}{t_f} \sum_{t=1}^{t_f+1} ln(\mu_t \mathbf{y}_t \cdot \mathbf{w}_{t-1}) = \frac{1}{t_f} \sum_{t=1}^{t_f+1} r_t \quad (9)$$

In order to extract patterns from a portfolio series, we use the same approach used in prior works, based on a Long Sort-Term Memory (LSTM) predictor (Hegde, Kumar, and Singh 2018; Zhang et al. 2020; Patel 2018). Both the actor and critic networks, for both our DDPG and TD3 frameworks, utilise the same configuration.

Training

The datasets were split in a 6:1 ratio for training and testing, respectively. Training was done over 400 episodes, each consisting of 1000 steps. At the start of each episode, the agent is placed at a random point within the training subset. This starting step is to allow for the training steps to be completed. When the networks are to be trained, the training is done with a mini-batch of 64, sampled uniformly from a replay buffer consisting of the agents' history.

In order to avoid over-fitting, we include a *value function threshold* parameter which terminates the training phase when a reward value threshold is exceeded for a number of consecutive episodes. For example, if the rewards exceed the threshold value in 10 successive episodes, training will stop before reaching the defined 400 episodes. The reward value limit is selected for each dataset based on the best possible baseline performance.

Table 1 shows the training configuration and hyperparameters used in our experiments.

Evaluation Criteria

We evaluate our portfolio optimisation models with the following criteria:

- **Average Daily Yield:** The mean of all the returns obtained. A higher value is better.
- **Sharpe Ratio:** Measures the return of an investment when compared to a risk-free asset while adjusting for its risk (Sharpe 1966; 1994); $\frac{R_p - r_f}{\alpha}$, where R_p is the portfolio return, r_f is the risk free rate, and α is the standard deviation. A higher value is better.
- **Sortino Ratio:** Very similar to the Sharpe ratio, but instead penalises only the downside deviation α_d, the risk of losing value (Sortino and Price 1994). A higher value is better; $\frac{R_p - r_f}{\alpha_d}$
- **Maximum Drawdown (MDD):** The maximum loss from a peak to a trough, before a new peak is attained. In this case a lower value is better; $\frac{ThroughValue - PeakValue}{PeakValue}$
- **Final Portfolio Value:** The total wealth an agent has after the last step has been completed. Expressed as a ratio to the initial portfolio value of 1.

The baselines to which we compare the performance of our DRL models are shown in Table 2.

Experiment 1: Comparison of TD3 with DDPG

DDPG and TD3 were trained on the *NYSE(N)* and *SP500* datasets using different window lengths (3, 7, 11, 14). The results of the best performing DRL models with their respective time windows are shown in Tables 3 and 4. We observed that individually, TD3 with a window length of 14 performed best overall on most criteria on the *NYSE(N)* dataset. Similarly, TD3 with a window length of 11 performed best overall on the *SP500* dataset.

Experiment 2: Inclusion of stock movement prediction functions within state format

One of the best performing OLPS models that include a prediction phase is RMR, which is classified as a 'Follow the Loser' algorithm. The Mean Reversion strategy used in RMR makes use of a windowed moving average prediction function based on OLMAR, with reduced estimation errors caused by noise and outliers in the data (Huang et al. 2016).

To assess whether RMR can help our models, we included the RMR prediction function inside the state of our TD3 model, using the same four window sizes. For each model, the RMR function was executed with the window parameter, w, set to the window length of the corresponding model. The best performing model in-sample was then evaluated out-of sample along with the baselines and the previous models. For both the datasets, the inclusion of the RMR was found to lead to better portfolio value results on most window lengths as shown in Table 3 and 4, except for TD3 with window length 11, where it was found to have an adverse effect.

Experiment 3: Ensemble DRL model

The objective of this experiment is to determine whether we can benefit from the features of the four best models found in the previous experiments, which may have non-correlating periods during testing due to differences in the state representation and RL framework. This lack of correlation suggests that the models behave differently in different market situations. The main intuition behind this approach is to evaluate

Hyperparameter	Value	Description
Dataset	”nyse_n” or ”SP500”	The dataset selected to perform training on
Episodes	400	Maximum number of training episodes
Window length	3, 7, 11, and 14	Window or observation size
Value function threshold	3 for NYSE(N), 2 for SP500	Reward threshold used to stop training
Max Step	1000	Number of steps completed in episode
Buffer size	10^5	Size of replay buffer
Batch size	64	Mini-batch size during training
τ (tau)	0.001	Target network update ratio
γ (gamma)	0.99	Reward discounting factor
Actor learning rate (α)	$1x10^{-4}$	Actor learning rate
Critic learning rate (β)	$1x10^{-3}$	Critic learning rate
Seed	1338	Random seed number
(DDPG parameters)		
σ (sigma)	0.2	Ornstein-Uhlenbeck parameter
θ (theta)	0.15	Ornstein-Uhlenbeck parameter
dt	0.002	Ornstein-Uhlenbeck parameter
(TD3 parameters)		
Policy noise	0.2	Exploration noise
Noise clip	0.5	Maximum value of the Gaussian noise
Policy frequency	2	Number of iterations to wait before the policy network updates
(Ensemble parameters)		
Window length	21	Window or observation size
Action length	1	Steps taken with action

Table 1: Training Hyperparameters

Classification	Strategy	References
Benchmark	Constant Rebalanced Portfolios (CRP/UCRP)	(Kelly 1956)
Benchmark	Best Constant Rebalanced Portfolios (BCRP)	(Cover 1996)
Follow-the-Winner	Exponential Gradient (EG)	(Helmbold et al. 1996; 1998)
Follow the Winner	Universal Portfolios (UP)	(Cover 1996)
Follow-the-Loser	Passive Aggressive Mean Reversion (PAMR)	(Li et al. 2012)
Follow-the-Loser	Online Moving Average Reversion (OLMAR)	(Li and Hoi 2012)
Follow-the-Loser	Weighted Moving Average Mean Reversion (WMAMR)	(Gao and Zhang 2013)
Follow-the-Loser	Robust Median Reversion (RMR)	(Huang et al. 2016)
Meta-Learning	Online Newton Step (ONS)	(Agarwal et al. 2006)

Table 2: Baselines along with references

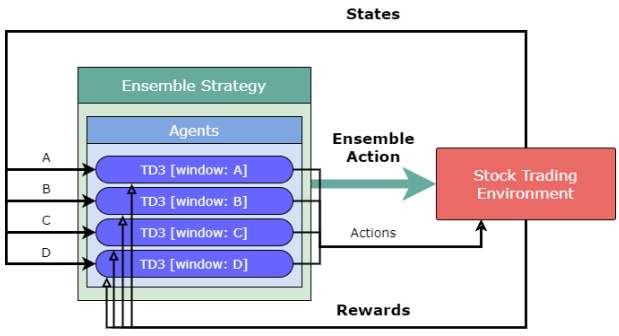


Figure 1: Ensemble TD3 Strategy.

the performance of each model over the most recent time window, and identify which model is exhibiting the most profitable behaviour, as shown in Figure 1.

The actions chosen by each agent are all passed to the environment, and the corresponding returns are recorded. Our ensemble model requires two additional parameters: the window length, w, and the number of actions, a. The models are evaluated using a window, w, of recorded returns at each step, and the best agent is selected according to the best *average return*. After the best agent is picked, it is used to trade for the next a steps. We set w to 21, approximately a month of trading days, and a is set to 1. This means that the ensemble model can switch agent at every step. To create our optimised portfolio, we make use of the four agents trained in our previous experiment (TD3-3-ltsm-rmr, TD3-7-ltsm-rmr, TD3-11-ltsm-rmr, TD3-14-ltsm-rmr). For the first w days in the test dataset, the model with the greatest performance

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
Ensemble	**0.276**	**7.387**	**12.703**	47.108	**6.810**
TD3-3-lstm-rmr	0.195	6.877	11.145	**33.125**	4.174
TD3-7-lstm-rmr	0.103	3.233	4.448	62.606	1.670
TD3-11-lstm-rmr	-0.066	-1.611	-2.035	92.460	0.250
TD3-14-lstm-rmr	0.151	3.584	5.264	79.801	1.838
TD3-14-lstm	0.059	2.382	3.301	52.122	1.305
DDPG-7-lstm	0.071	2.062	2.760	69.351	1.151
Market Value (UCRP)	0.015	0.768	0.982	63.279	0.970
BCRP	0.114	3.737	5.803	73.208	1.876
OLMAR	0.220	3.414	4.678	91.443	1.206
PAMR	0.182	3.666	5.072	79.034	1.797
RMR	0.210	3.260	4.527	91.612	1.104
WMAMR	0.024	0.377	0.478	94.642	0.206
EG	0.013	0.711	0.907	63.122	0.962
ONS	0.006	0.076	0.085	96.181	0.072
UP	0.013	0.696	0.892	63.490	0.958

Table 3: Results or the *NYSE(N)* test dataset. (The best value for each criteria is in boldface.)

Model	Average Daily Yield (%)	Sharpe Ratio (%)	Sortino Ratio (%)	Maximum Drawdown (%)	Final Portfolio Value
Ensemble	0.268	19.775	29.750	15.724	10.917
TD3-3-lstm-rmr	**0.379**	**30.306**	**49.204**	**10.848**	**30.843**
TD3-7-lstm-rmr	0.346	22.460	35.208	18.581	21.913
TD3-11-lstm-rmr	0.211	16.246	25.876	18.537	6.513
TD3-14-lstm-rmr	0.069	6.401	8.730	25.494	1.785
TD3-11-lstm	0.331	24.852	43.255	12.265	19.412
DDPG-11-lstm	0.31	21.21	35	21.945	15.985
Market Value (UCRP)	0.017	2.312	2.815	17.072	1.140
BCRP	0.070	4.211	5.625	33.206	1.671
OLMAR	0.145	9.702	13.834	32.493	3.481
PAMR	0.146	10.508	15.173	24.744	3.506
RMR	0.144	9.725	14.117	24.196	3.451
WMAMR	0.086	5.803	8.109	26.658	2.023
EG	0.017	2.346	2.852	17.111	1.143
ONS	-0.044	-4.256	-5.281	41.920	0.633
UP	0.018	2.506	3.046	16.63	1.154

Table 4: Results for the *SP500* dataset. (The best value for each criteria is boldface.)

throughout the in-sample dataset is automatically selected. After day w is reached, the agents are re-evaluated and one is selected accordingly for each day. Therefore, when applied to a real world scenario, the portfolio weights for the next trading day are selected by firstly evaluating the performance of the trained models on the past w trading days and then selecting the next action suggested by the model with the greatest performance.

Despite the ensemble model being the most successful one on the *NYSE(N)* dataset, as seen in Table 3 and Figure 2, this behaviour was not entirely replicated on the *SP500* dataset. On the *SP500* test dataset, the ensemble model achieves greater portfolio results than all the baselines but fails to excel over models created in previous experiments, namely *TD3-11-lstm* and *TD3-3-lstm-rmr*, as shown in Table 4.

Conclusions and Future work

In this work, we have presented portfolio optimisation models implemented using the TD3 DRL framework. The performance of these models was compared with its predecessor, DDPG, using different time window parameters, on the *NYSE(N)* and *SP500* datasets. These agents made use of a state representation that included normalised logarithmic returns for the specified time window. This state representation was subsequently enhanced further, by including the RMR prediction function within the state. These models were also compared to baseline OLPS algorithms found in literature, and TD3 was found to perform best over all, with the RMR variant outperforming all the others on almost all metrics, on both datasets.

We also implemented an ensemble strategy for portfolio

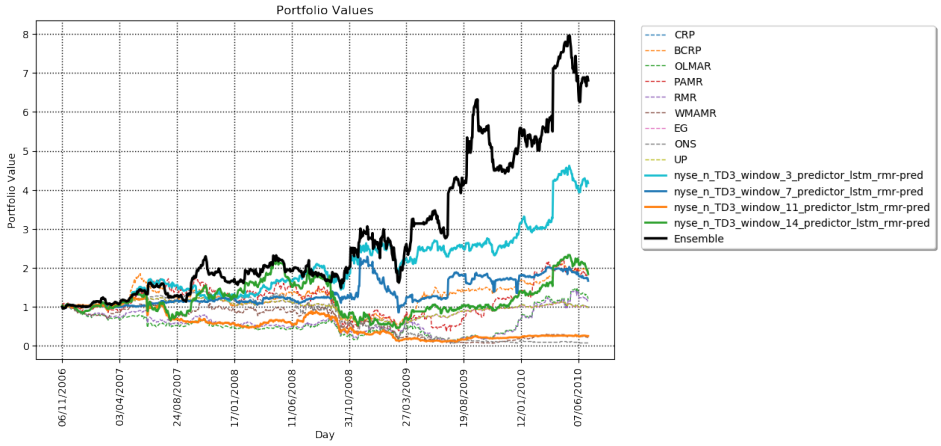


Figure 2: Portfolio values for our best DRL models along with baselines on the NYSE(N) dataset.

optimisation, using a combination of DRL agents. This strategy was found to enhance performance significantly in terms of Average Daily Yield, Sharpe Ratio and Sortino Ratio on the *NYSE(N)* dataset, but fell short on the *SP500* dataset, whilst still beating all the baseline OLPS algorithms.

We have shown that even with a simple state representation consisting of normalised log returns, a DRL framework such as TD3 can converge to a good policy that performs well on financial metrics. This approach can be enhanced further by including additional features in the state representation, such as making use of other financial indicators that capture risk and momentum. One could also explore the possibility of using Evolutionary Reinforcement Learning algorithms, where a population of actors are evaluated after each episode and the best performing ones survive to the next episode, after which the population is perturbed to create a new population. Such further research has the potential to improve the performance and robustness of the resultant policies.

References

Agarwal, A.; Hazan, E.; Kale, S.; and Schapire, R. E. 2006. Algorithms for portfolio management based on the Newton method. *ACM International Conference Proceeding Series* 148(January 2006):9–16.

Bellman, R. 1954. The theory of dynamic programming. Technical report, Rand corp santa monica ca.

Cover, T. M. 1996. Universal Portfolios. *Department of Statistics and Electrical Engineering, Stanford University*.

Cumming, J. 2015. An Investigation into the Use of Reinforcement Learning Techniques within the Algorithmic Trading Domain. *Imperial College London: London, UK*.

Dankwa, S., and Zheng, W. 2019. Modeling a Continuous Locomotion Behavior of an Intelligent Agent Using Deep Reinforcement Technique. *2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology, CCET 2019* 172–175.

Deng, Y.; Bao, F.; Kong, Y.; Ren, Z.; and Dai, Q. 2017. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*.

Fagiuoli, E.; Stella, F.; and Ventura, A. 2007. Constant rebalanced portfolios and side-information. *Quantitative Finance* 7(2):161–173.

Fujimoto, S.; Van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)* 4:2587–2601.

Gao, L., and Zhang, W. 2013. Weighted Moving Average Passive Aggressive Algorithm for Online Portfolio Selection. *5th International Conference on Intelligent Human-Machine Systems and Cybernetics* 327–330.

Gran, P. K.; Holm, A. J. K.; and Søgård, S. G. 2019. A deep reinforcement learning approach to stock trading. Master's thesis, NTNU.

Hegde, S.; Kumar, V.; and Singh, A. 2018. Risk aware portfolio construction using deep deterministic policy gradients. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1861–1867. IEEE.

Helmbold, D. P.; Schapire, R. E.; Singer, Y.; and Warmuth, M. K. 1996. On-line portfolio selection using multiplicative updates. *In Proceedings of the International Conference on Machine Learning* 243–251.

Helmbold, D. P.; Schapire, R. E.; Singer, Y.; and Warmuth, M. K. 1998. On-line portfolio selection using multiplicative updates. *Mathematical Finance 8* 4:325–347.

Huang, D. J.; Zhou, J.; Li, B.; Hoi, S. C.; and Zhou, S. 2016. Robust Median Reversion Strategy for Online Portfolio Selection. *IEEE Transactions on Knowledge and Data Engineering* 28(9):2480–2493.

Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous learning control with deep reinforcement. In *International Conference on Learning Representations (ICLR)*.

Islam, R.; Henderson, P.; Gomrokchi, M.; and Precup, D. 2017. Reproducibility of Benchmarked Deep Reinforce-

ment Learning Tasks for Continuous Control. *Transportation Quarterly.*

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

Jiang, Z.; Xu, D.; and Liang, J. 2017. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*.

Kanwar, N. 2019. *Deep Reinforcement Learning-Based Portfolio Management*. Ph.D. Dissertation, UTA.

Kelly, J. L. 1956. A new interpretation of information rate. *Bell Systems Technical Journal* 35:917–926.

Khadka, S.; Majumdar, S.; Nassar, T.; Dwiel, Z.; Tumer, E.; Miret, S.; Liu, Y.; and Tumer, K. 2019. Collaborative evolutionary reinforcement learning. In *International Conference on Machine Learning*, 3341–3350. PMLR.

Li, B., and Hoi, S. C. 2012. On-line portfolio selection with moving average reversion. *Proceedings of the 29th International Conference on Machine Learning, ICML 2012* 1:273–280.

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

Li, J., and Yu, T. 2020. Deep Reinforcement Learning based Multi-Objective Integrated Automatic Generation Control for Multiple Continuous Power Disturbances. *National Natural Science Foundation of China*.

Li, B.; Zhao, P.; Hoi, S. C.; and Gopalkrishnan, V. 2012. PAMR: Passive aggressive mean reversion strategy for portfolio selection. *Machine Learning* 87(2):221–258.

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

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; and Pietikäinen, M. 2020. Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision* 128(2):261–318.

MacHalek, D.; Quah, T.; and Powell, K. M. 2020. Dynamic Economic Optimization of a Continuously Stirred Tank Reactor Using Reinforcement Learning. *Proceedings of the American Control Conference* 2020-July:2955–2960.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Noda, K.; Yamaguchi, Y.; Nakadai, K.; Okuno, H. G.; and Ogata, T. 2015. Audio-visual speech recognition using deep learning. *Applied Intelligence* 42(4):722–737.

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of markov decision processes. *Mathematics of operations research* 12(3):441–450.

Patel, S. S. 2018. A Deep Reinforcement Learning Approach to the Portfolio Management Problem [m]. *ProQuest Dissertations and Theses* 141.

Pogue, G. A. 1970. An extension of the markowitz portfolio selection model to include variable transactions' costs, short sales, leverage policies and taxes. *The Journal of Finance* 25(5):1005–1027.

Polyak, B. T., and Juditsky, A. 1992. Acceleration of Stochastic Approximation by Averaging. *SIAM Journal on Control and Optimization* 30(4):838–855.

Rubinstein, M. 2002. Markowitz's" portfolio selection": A fifty-year retrospective. *The Journal of finance* 57(3):1041–1045.

Sharpe, W. F. 1966. Mutual fund performance. *The Journal of business* 39(1):119–138.

Sharpe, W. F. 1994. The sharpe ratio. *The Journal of Portfolio Management* 21(1):49–58.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. *31st International Conference on Machine Learning, ICML 2014* 1:605–619.

Sortino, F. A., and Price, L. N. 1994. Performance Measurement in a Downside Risk Framework. *The Journal of Investing* 3(3):59–64.

Sutton, R. S., and Barto, A. G. 1999. Reinforcement Learning: An Introduction. *Trends in Cognitive Sciences* 3(9):360.

Uhlenbeck, G. E., and Ornstein, L. S. 1930. On the theory of the brownian motion. *Physical review* 36(5):823.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, 2094–2100.

Van Hasselt, H. 2010. Double Q-learning. *Advances in Neural Information Processing Systems* 2613–2621.

Yang, H.; Liu, X.-Y.; Zhong, S.; and Walid, A. 2020. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. *SSRN Electronic Journal*.

Zhang, Y.; Zhao, P.; Li, B.; Wu, Q.; Huang, J.; and Tan, M. 2020. Cost-Sensitive Portfolio Selection via Deep Reinforcement Learning. *IEEE Transactions on Knowledge and Data Engineering* 4347(c):1–1.

Zhang, S.; Boehmer, W.; and Whiteson, S. 2020. Deep residual reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 1611–1619.

Zhou, X. Y., and Yin, G. 2003. Markowitz's mean-variance portfolio selection with regime switching: A continuous-time model. *SIAM Journal on Control and Optimization* 42(4):1466–1482.

Scenario Planning In The Wild: A Neuro-Symbolic Approach

Michael Katz[†], Kavitha Srinivas[†], Shirin Sohrabi[†], Mark Feblowitz[†], Octavian Udrea [⋆*], Oktie Hassanzadeh[†]

[†] IBM Research, [⋆] Dataminr

michael.katz1@ibm.com, kavitha.srinivas@ibm.com, ssohrab@us.ibm.com, mfeb@us.ibm.com, oudrea@dataminr.com, hassanzadeh@us.ibm.com

Abstract

Scenario Planning is a commonly used technique to better prepare organizations for the future. Its key idea is to generate a variety of alternative futures to help strategic decision makers with developing long-term plans. While most prior work deals with manual scenario creation and exploration, some suggested modeling scenario planning problem for enterprise risk management using Mind Maps, allowing automated exploration of scenarios with AI Planning tools. Mind Map creation, however, remains a manual process, requiring an extensive effort of multiple experts. In this work, we suggest further automating the entire process, eliminating the need for Mind Map creation, assuming instead, access to a collection of documents. We employ neural causal extraction techniques to derive hidden causal relations from these documents. Replacing Mind Maps with these causal models allows to revisit the definition of scenario planning. We suggest an alternative planning model, to capture the general scenario planning problem, showing that scenarios can be derived from sets of plans for the corresponding planning problem. To evaluate our approach, we apply these neural techniques to existing collections of documents and produce a planning domain and problems collection. We then compare existing top-k planners on this collection, to discover the more suitable tools to be used for scenario planning in the real world.

1 Introduction

Scenario Planning is a commonly used by many organizations technique to be better prepared for the future (Schoemaker 1995; Oliver and Parrett 2017). Its key idea is to generate a variety of alternative futures called "scenarios" for strategic decision makers to help them develop their long-term plans. Scenario planning can be used in many application settings such as oil and gas, military, healthcare, and education (Peterson, Cumming, and Carpenter 2003; Cardoso and Emes 2014; Edgar, Abouzeedan, and Hedner 2011). Scenario planning involves uncovering a set of forces and their influence, trends, and effects amongst them as well as selecting a small subset of the forces (often two), called "critical uncertainties", using their value range as the axes of the scenario space. Major categories of forces include social, economics, technology, politics, healthcare, and environment. Scenarios are results of analyzing the relationship between the forces together with the interaction of the selected critical uncertainties (Garvin and Levesque 2006). Most existing approaches or methodologies are manual, often carried out in a workshop with group of experts in a potential domain of interest. While scenario planning involves subjective analysis and creative thinking, given that most approaches are manual they fail to generate large number of scenarios, consider large number of *selected forces* or "critical uncertainties", and may include bias. To exemplify, previous work considers 2 critical uncertainties, which results in 4 scenarios(Garvin and Levesque 2006).

The closest and, to the best of our knowledge, the only related work that introduces automation to address the scenario planning problem is by Sohrabi et. al., 2018; 2019, where AI Planning is used to address the scenario planning problem for enterprise risk management. While this helps create many scenarios quickly, it heavily relies on capturing the domain knowledge through so called *Mind Maps*. In fact, Mind Maps are integral to the very definition of the scenario planning problem tackled by Sohrabi et al. (2018). Thus, the domain experts still have to go through the manual process of creating Mind Maps that describe the causal relation between all the *forces*. Additionally, information on the impact or likelihood of the pairs are to be captured manually.

In this paper, we present a general definition of the scenario planning problem. Our definition generalizes the previous one, eliminating the need for Mind Maps, assuming instead access to a collection of documents, and thus allowing to further automate scenario planning. Our neuro-symbolic approach consists of employing neural causal extraction techniques to derive causal relations between the forces from the input documents and a proposed alternative symbolic AI Planning model, derived from the causal relations, to capture the solutions to the general scenario planning problem. We show that scenarios can be derived from sets of plans for the corresponding planning problem. To that end, we employ neural techniques to existing collections of documents to produce a planning domain and problems collection. We then employ planning techniques that produce multiple plans of top quality to the planning problem (Katz et al. 2018; Speck, Geißer, and Mattmüller 2018) and compare the performance of various *top-k* planners in order to discover the more suitable among the existing tools to be used for scenario planning in the real world.

[*] Work done while at IBM Research

2 Background

We present here the necessary background on scenario planning, classical planning, and planning with soft goals.

2.1 Scenario Planning

As a basis for our work, we consider the *Scenario Planning* problem (Sohrabi et al. 2018). In this work, we adapt the notation and concept names to better fit an existing literature (Garvin and Levesque 2006). A scenario planning task $SP = \langle M_F, I_F, O_F \rangle$ is a tuple of *Forces Model* M_F, *Forces Impact* I_F, and a set of forces O_F called *Key Forces* or *Selected Forces*, describing the current situation. The *Forces Model* is often implemented as a set of structured Mind Maps $\mathcal{M}$ (Definition 1 in (Sohrabi et al. 2018)), while forces impact is a mapping from a pair of forces that appear together in some structured Mind Map $M \in \mathcal{M}$ to a vector of values that represent the *impact* and *likelihood* of the pair. Further, each Mind Map is associated with a number representing its *importance*. Solutions to scenario planning are sets of trajectories through the structured Mind Maps. These sets are partitioned into scenarios (sets of similar trajectories), often using clustering techniques (Sohrabi et al. 2018).

2.2 Classical Planning

We consider STRIPS planning tasks $\Pi = \langle P, A, I, G, cost \rangle$, extended with action costs and negative preconditions. P is a set of Boolean *propositions*. Each subset $s \subseteq P$ is a *state*, and $S = 2^P$ is the *state space* of Π. The state I is the *initial state* of Π and the goal $G \subseteq P$ is a set of propositions, where a state s is a *goal state* if $G \subseteq s$. The set A is a finite set of *actions*. Each action $a \in A$ has an associated set of *preconditions* $pre(a) \subseteq P \cup \{\neg f \mid f \in P\}$, *add* effects $add(a) \subseteq P$ and *delete* effects $del(a) \subseteq P$, and $cost : A \to \mathbb{R}^{0+}$ is a non-negative *action cost* function.

The semantics of STRIPS planning is as follows. An action a is *applicable* in the state s if $pre(a) \cap P \subseteq s$ and for all $f \in s$, we have $\neg f \notin pre(a)$. Applying a in s results in the state $s[\![a]\!] := (s \setminus del(a)) \cup add(a)$. A sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ is *applicable* in s if there exists a sequence of states $s = s_1 \ldots s_{n+1}$ such that a_i is applicable in s_i and $s_{i+1} = s_i[\![a_i]\!]$. We denote its end state by $s[\![\pi]\!]$. An applicable action sequence is a *plan* for s if $s[\![\pi]\!]$ is a goal state. Its *cost* is the cumulative cost of actions in the sequence: $cost(\pi) = \sum_{i=1}^{k} cost(a_i)$. A plan for s with minimal cost is called *optimal*. The objective of (optimal) planning is to find an (optimal) plan for I, and the objective of top-k planning is to find k plans of top quality (Katz et al. 2018).

2.3 Planning with Soft Goals

We further consider the extension of STRIPS to soft goals (Keyder and Geffner 2009). The task $\Pi_{sg} = \langle P, A, I, G, cost, u \rangle$ is a STRIPS planning task with soft goals if $\Pi = \langle P, A, I, G, cost \rangle$ is a STRIPS planning task and $u : P \mapsto \mathbb{R}^{0+}$ is a mapping from propositions P to non-negative reals, with the propositions that are mapped to strictly positive values being called *soft goals*. A sequence of actions π is a plan for Π_{sg} if it is a plan for Π, and the utility of a plan is the difference between the summed utility of the end state and the cost of the plan: $u(\pi) = \sum_{f \in I[\![\pi]\!]} u(f) - cost(\pi)$.

While dedicated solvers for STRIPS with soft goals exist, the prevailing method of solving such tasks is by compiling soft goals away (Keyder and Geffner 2009). The compilation adds two types of actions, *collect* and *forgo*. These actions are applied after all original actions, and either collect the achieved soft goals or pay a penalty equivalent to the utility of the not achieved soft goal.

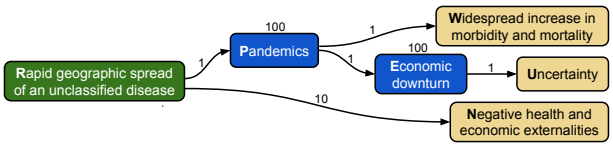


Figure 1: A running example of forces, their causal connections, costs and penalties.

3 General Scenario Planning

We start by defining the *Forces Causal Model*, which encodes the causal relation between forces.

Definition 1 (Forces Causal Model) *A Forces Causal Model (FCM) is a pair of $\langle F, \mathcal{I} \rangle$, where F is a set of possible forces and $\mathcal{I} : F \times F \mapsto (\mathbb{R}^{0+})^n$ is a mapping from a pair of forces to a vector of values $\mathcal{I}(a, b)$ that describe the properties of a causing b.*

Some example properties are *likelihood* of a causing b, an *impact* of a causing b, or a *confidence* provided by a particular tool that a is causing b.

Throughout the paper we are referring to a running example, extracted from a significantly larger model.[1] The example is depicted in Figure 1, with the forces being the nodes of the graph. While we refer to the edges later on, the example includes 6 forces, referred in text by their first letter:

- Pandemics,
- Widespread increases in morbidity and mortality,
- Economic downturn,
- Uncertainty,
- Rapid geographic spread of an unclassified disease, and
- Negative health and economic externalities.

Note that while the likelihood of R $\to$ P (Rapid geographic spread of an unclassified disease causing Pandemics) is somewhat lower than of E $\to$ U, the impact of R $\to$ P is certainly higher than of E $\to$ U.

Given Definition 1, we can now define the General Scenario Planning (GSP) problem.

Definition 2 (General Scenario Planning) *A General Scenario Planning (GSP) problem is a tuple $\mathcal{G} = \langle M, F_0, F_*; F_o, F_s \rangle$, where M is a* Forces Causal Model*, $F_0 \subseteq F$ is a set of* Initial Forces*, $F_* \subseteq F$ is a set of* Implications*, $F_o \subseteq F$ is a set of* Selected Forces*, and $F_s \subseteq F_0$ is a set of* Indicators*.*

[1] Model's semi-automated generation is explained in Section 4.

Semantically, F_0 are the forces to start an exploration from, F_* are the implications, the forces the exploration leads to. The forces in F_o describe the current situation and F_s describe the forces of possible initial interest. Looking at our running example, the node marked with a green background is an indicator, and the nodes marked with yellow and blue backgrounds are implications and selected forces, respectively. Note that given Forces Model and Forces Impact, one can define a corresponding FCM. Therefore, our definition of General Scenario Planning generalizes the previous definition of Scenario Planning over *structured Mind Maps* (Sohrabi et al. 2018).

We can now define a solution for General Scenario Planning, similar in spirit to the way it was defined for Scenario Planning by Sohrabi et al. (2018), aiming at automating the manual process of generating scenarios (Garvin and Levesque 2006).

Definition 3 *A solution Φ to a GSP over forces F is a set of* valid *trajectories ϕ. We call a trajectory valid if it starts in F_0 and ends in F_*. By solution* size *we refer to the number of trajectories $|\Phi|$. Each trajectory $\phi \in \Phi$ traverses a (possibly empty) subset of selected forces, denoted by F_o^ϕ, as well as a (possibly empty) subset of indicators, denoted by F_s^ϕ.*

The solution is often partitioned into one or more scenarios. One possibility of defining such a partitioning is based on the traversed indicators (Oliver and Parrett 2017), another possibility is based on the sets of all forces traversed (Sohrabi et al. 2018). The actual partitioning of the solution into scenarios is outside of the scope of this work.

We sometimes slightly abuse the notation and refer to ϕ as a sequence of pairs of forces. Additionally, we assume a mapping $p_o : F_o \mapsto \mathbb{R}^+$ of penalties for not traversing a selected force and $c_0 : F_0 \mapsto \mathbb{R}^+$ of costs of starting a traversal in a particular initial force, and for a valid trajectory ϕ, we denote that cost by $c_0(\phi)$.

While any such collection of trajectories is a solution, we are particularly interested in solutions of bounded size that minimize $\sum_{o \in F_o \setminus \phi(F_o)} p_o(o)$ for each trajectory. Additionally, $\mathcal{I}$ can be extended from pairs to trajectories (sequences of pairs) and to sets of trajectories, and GSP can be viewed as a multi-objective optimization.

In this work, we restrict our attention to finding trajectories under a single objective function. We assume a mapping $c : F \times F \mapsto \mathbb{R}^{0+} \cup \{\infty\}$ that represents a *cost* of traversing an edge and define the cost of a traversal by $c(\phi) = \sum_{(a,b)\in\phi} c(a,b) + c_0(\phi)$. We define traversal *net-cost* as the sum of (i) the penalty for not traversing selected forces, and (ii) the cost of a traversal, and aim at minimizing traversal net-cost.

Looking at our running example, there are 5 pairs of forces for which the cost of traversing the edge is finite, depicted with edges in Figure 1. The costs are shown on these edges. Further, for each selected force (blue), a number above the corresponding node depicts the penalty for not traversing it. A valid trajectory corresponds to a path from the initial force (green) to some implication (yellow). The cost of starting a traversal in the initial force is 0. The following are example trajectories.

- R $\rightarrow$ P $\rightarrow$ E $\rightarrow$ U with cost 3 and net-cost 3,
- R $\rightarrow$ P $\rightarrow$ W, with cost 2 and net-cost 102, and
- R $\rightarrow$ N, with cost 10 and net-cost 210, which includes the penalty costs of not traversing the forces P,E.

4 Forces Causal Model

As mentioned earlier, most existing approaches to creating the causal models (mostly modeled as Mind Maps with the help of visual tools) are manual, and often created by establishing consensus with a group of experts, which in turn is quite tedious and expensive. There has been a great deal of progress made recently in natural language understanding that can be leveraged to largely automate the process of causal model extraction. Automating the process also allows to switch from visual oriented Mind Maps to Forces Causal Models (FCMs).[2] Here we outline one mechanism that we use to derive neural FCMs. There are numerous other natural language processing techniques that can be applied to the problem of automating FCM generation, but we focus on one to demonstrate feasibility of adopting neural methods to derive FCMs. We mention some of the other methods in Section 4.2.

4.1 Seeded FCM extraction based on QA

One semi-automated technique to generate FCMs is to extract causal relations from a user-provided seed set of forces $F_{\text{seed}} \subset F$, and a domain specific corpus C of a set of documents D. The rationale behind this approach is the observation that users can often provide some key factors important to their domain for scenario planning, and they have access to the source documents relevant to those factors, specifying causal risk connectivity in natural language. As an example, a user interested in corporate risk may have a starting set of SEC 10-K filings[3] where risks are usually outlined explicitly, often indicated in bold as subheadings, and a set of seed forces that they already know will affect their business, $F_{\text{seed}} \subset F$, such as *Sudden drop in oil prices*, or *hurricanes*. Note that there is no requirement to outline every single force, only the seed set. Hence, we expect very minimal effort required from humans both in terms of identifying the initial set of forces, as well as the identification of the specific corpus (i.e., the set of documents).

FCMs can then be generated by distilling the problem into causal relation extraction over that corpus C, using the seed forces F_{seed} as starting points. We use neural question answering models to extract causal relations by automatically generating questions of a corpus as follows. First, we partition the corpus C of D documents into a set of paragraphs P. For each $f \in F_{\text{seed}}$, we determine the subset of paragraphs $p \subset P$ that contain concepts similar to f. This phase of paragraph selection is implemented as a semantic search task, with a set of search results for each F_{seed}. Text search is inadequate for this step because it suffers from both poor recall and poor precision due to the query length of typical forces. To perform the semantic search, we embed each

[2] We further discuss the two modeling approaches in Section 7.

[3] https://en.wikipedia.org/wiki/Form_10-K

paragraph p in each document in the corpus using two neural models for sentence embeddings to improve recall. We use sentence embeddings from the Google USE (Cer et al. 2018) and the best BERT based model for encoding sentence similarity (e.g., (Reimers and Gurevych 2019)), which was BERT trained on the GLUE tasks (Wang et al. 2018) of natural language inference (NLI) and sentence similarity (STS). We computed the vector embeddings of all paragraphs p, and inserted them into the FAISS approximate nearest neighbors index, which can scale to a billion vectors (Johnson, Douze, and Jégou 2017). Each force f was used as a query against the index, and we took the top k paragraphs (where k was set to 3 in our experiments) for each model as the relevant set. The final relevant set of paragraphs was the union of the results from BERT and USE.

For each force $f \in F_{\text{seed}}$, we processed the following two questions over relevant paragraphs in the corpus, using a neural model:

- what does each force $f \in F_{\text{seed}}$ cause?
- what causes each force $f \in F_{\text{seed}}$?

We used ALBERT-xlarge trained on SQUAD 2.0 as our model because it achieves state of the art performance in question answering over the SQUAD 2.0 benchmark (Lan et al. 2019).

Processing of these two questions for each force $f \in F_{\text{seed}}$ against each paragraph P results in a (possibly empty) set of answers $F_{\text{a}}(f)$. The subset of forces $f \in F_{\text{seed}}$ for which $F_{\text{a}}(f) = \emptyset$ is referred to as F_{u}. Because this approach asks open-ended questions, it provides the benefit of discovering new forces $F_{\text{new}} = \bigcup_{f \in F_{\text{seed}}} F_{\text{a}}(f) \setminus F_{\text{seed}}$, that can be incorporated into the seed set for the next iteration. The final set F_{final} of an iteration is $(F_{\text{seed}} \setminus F_{\text{u}}) \cup F_{\text{new}}$. F_{final} is in turn provided as the seed set for the next iteration, until a fixed point is reached. This iterative approach helps us find many new forces, as well as causal chains of length greater than 1. Note that, the generated FCM can contain loops as there is nothing to prevent a bi-directional cause-effect relation.

As an example of the types of causal pairs that can be extracted, the relation depicted in Figure 1 between "Economic downturn" and "Pandemics" is automatically determined by our model with a confidence score of 84%, based on the following context: *"Economists estimate that, in the coming decades, pandemics will cause average annual economic losses of 0.7% of global GDP..."*. "Economic downturn" was provided as a seed force but "Pandemics" is a new force identified by this approach. Also note that *"economic losses"* in the paragraph are a semantic match to *"Economic downturn"*, the designated force. We show how one might use the approach outlined here to provide two auto-generated FCMs for two domains in Section 8.

4.2 Issues with auto-generation of FCMs

There are two primary issues related to the auto-generation of FCMs. Auto-generation of forces means that there must be a step to perhaps curate the FCM that has been generated. We do rely on high quality neural models for question answering; the performance of neural models built on ALBERT have reached accuracies of 92.77% (Lan et al. 2019) for question answering benchmarks. However, there is no guarantee that the technique works well in all cases (e.g., across new domains), so human curation of the generated FCM is still an important optional step. Often this is best accomplished after scenario generation when a domain expert can look at a scenario that has been generated and potentially alter causal relations in the FCM[4]. Alternatively, other causal extraction techniques such as using neural models for relation extraction (Lin et al. 2016; dos Santos, Xiang, and Zhou 2015; Zeng et al. 2014; Dunietz, Carbonell, and Levin 2018; Li and Mao 2019) or pattern based causal extraction (Li et al. 2020; Hassanzadeh et al. 2020, 2019; Bhandari et al. 2021) can also be applied to vet the extracted causal relations. We leave this line of research of vetting causal relations using multiple natural language processing techniques for future work.

Candidates for inclusion in F_{new} might be semantically equivalent to some force in F_{seed}. We automatically apply semantic equivalence matching between F_{new} and F_{seed} using neural embeddings at the phrase or sentence level. Specifically, if two forces f_1 and f_2 have an embedding distance that is above some high threshold in terms of vector distance, we could assume/recommend equivalence, and regenerate the FCM. In the current experiments, we used neural sentence models from BERT (Reimers and Gurevych 2019) to determine force equivalence[5]. These equivalent forces can be merged automatically or optionally vetted for merging by domain experts.

5 From Forces Causal Model to General Scenario Planning

Once an FCM $M = \langle F, \mathcal{I} \rangle$ is obtained, we need to specify the following sets and functions to construct a GSP.

Initial forces and implications. The set of Initial Forces F_0 and the set of Implications F_* are the forces that start and end valid trajectories. In most practical situations we only need user input in specifying one of these sets and automatically compute the other. For example, in most scenario planning applications, the implications F_* are forces of special relevance to the end users, and as such should be specified by them to the extent possible. However, if end users do not select enough forces out of F, this may leave forces that cannot belong to any valid trajectory. In such cases, we can either: (i) warn that such nodes exist or (ii) suggest additional nodes to add to F_* such that every force can belong to at least one valid trajectory. Note that the latter option, at its extreme, can compute and suggest the entire set F_*.

The Initial Forces F_0 are useful in diagnosis oriented applications, when the trajectories leading to the set of selected forces F_o is of more interest than anticipating what may happen after the forces in F_o. In such cases, they can be specified at least partially by a user. However, in scenario planning applications, F_0 can be automatically computed as a

[4] Alternatively, domain experts can directly curate the FCM itself. However, it is much more tedious for a human to do, since that tends to be a large model, but this curation can be crowd-sourced.

[5] Additionally, the Google universal sentence encoder (Cer et al. 2018) can augment equivalence detection.

set of forces such that every force in F is reachable from at least one force in F_0. If minimality of F_0 is not desired, such a set can be computed by a greedy algorithm where we sort forces by their outdegree in descending order, iteratively add them to F_0, eliminating from consideration all forces reachable from the current partial set. We can further prioritize forces in F_0 by assigning smaller penalties via c_0 for preferred Initial Forces.

Selected forces and indicators. The Indicators F_s are a way of giving priority to trajectories, and therefore scenarios containing forces the user wants to emphasize. We can also safely leave $F_s = \emptyset$. The Selected Forces F_o typically indicate a current situation of interest to the user, and therefore should be specified by them either directly, or after being suggested by analysis of news or social media. Such an awareness piece can be built using state of the art textual search methods combined with neural search techniques such as Mitra and Craswell (2017) to find the forces in F_o. Additionally, the function p_o can assign penalties for trajectories that do not include a particular force, with higher penalties for preferred forces in F_o. In most cases, each observation can have an equal penalty, its magnitude dependant on which trajectories we would like to prioritize. For example, if we would always like to see trajectories that contain as many forces in F_o as possible *before* any trajectory that skips a force from F_o, we would set p_o to be greater than $max_\phi(c(\phi))$. If instead we would like to see shorter trajectories first, prioritizing those that contain more forces from F_o, we would set p_o to a multiple of $max_{x,y \in F}(c(x, y))$.

Cost function. Generation of the cost function c can be problematic for large FCMs. One way to create it is to simply crowdsource filling its values among users, with a default value for those $x, y \in F$ that cannot be specified. A better approach is to base the c function values in this process on something the users can select from a discrete scale, such as the *likelihood* or *impact* of x causing y. The function $c : F \times F$ can also be automatically generated for auto-generated FCMs, based on the number of *support* statements that were used to derive the specific edge. We observe that support distributions tend to be highly skewed, with a predominance of small numbers of supporting statements for a given causal assertion. Our proposal is to use a function such as sigmoid (possibly with rounding) to map the frequency distribution into c.

6 Planning Model

Previous work has shown the NP-hardness of the Scenario Planning problem and suggested a planning-based approach to solving Scenario Planning (Sohrabi et al. 2018). The idea was to treat the problem of generating multiple trajectories as plan recognition, treating Mind Maps traversal as the planning problem and selected forces traversed as possible goals. The plan recognition problem was then compiled into a planning problem, and a top-k planner was applied to the compiled problem (Sohrabi, Riabov, and Udrea 2016). Here, the hardness result still stands. To solve General Scenario Planning, we follow a similar approach, adapting the planning model to traverse graphs that are defined by forces causal models instead of Mind Maps. As a result, the planning model becomes significantly simpler[6] and can be also viewed as a variant of the soft goals compilation (Keyder and Geffner 2009). In what follows, we present the revised planning model and show that it can be used to solve the general scenario planning problem.

Definition 4 *Given a General Scenario Planning problem $\mathcal{G} = \langle M, F_0, F_*; F_o, F_s \rangle$, the corresponding planning task $\Pi_\mathcal{G} = \langle P, A, I, G, cost \rangle$ is defined as follows:*

- *the set of fluents* $P = \{(\textit{not-started}), (\textit{goalachieved})\} \cup \{(\textit{at } x), (\textit{considered } y) \mid x \in F, y \in F_o\}$,
- *A is the set of actions, union of the following sets:*
  - $\{(\textit{enter } x) \mid x \in F_0\}$ *with precondition* $\{(\textit{not-started})\}$, *delete effect* $\{(\textit{not-started})\}$, *add effect* $\{(\textit{at } x)\}$, *and cost* $c_0(x)$,
  - $\{(\textit{traverse } x\ y) \mid x, y \in F, c(x, y) < \infty\}$ *with precondition* $\{(\textit{at } x)\}$, *delete effect* $\{(\textit{at } x)\}$, *add effect* $\{(\textit{at } y)\}$, *and cost* $c(x, y)$,
  - $\{(\textit{achievegoal } x) \mid x \in F_*\}$ *with precondition* $\{(\textit{at } x)\}$, *delete effect* $\{(\textit{at } x)\}$, *add effect* $\{(\textit{goalachieved})\}$, *and cost* 0,
  - $\{(\textit{explain } x) \mid x \in F_o\}$ *with precondition* $\{\neg(\textit{considered } x), (\textit{at } x)\}$, *add effect* $\{(\textit{considered } x)\}$, *and cost* 1, *and*
  - $\{(\textit{discard } x) \mid x \in F_o\}$ *with precondition* $\{\neg(\textit{considered } x), (\textit{goalachieved})\}$, *add effect* $\{(\textit{considered } x)\}$, *and cost* $p_o(x)$.
- *I is the initial state, consisting only of* $(\textit{not-started})$, *and*
- *the goal* $G = \{(\textit{goalachieved})\} \cup \{(\textit{considered } x) \mid x \in F_o\}$.

Note that while *explain* actions are applied during traversal, *discard* actions are applied only after *achievegoal*. These discard actions can be applied in any order. It is possible therefore to perform a simple modification of the planning task to impose one chosen order. For simplicity of presentation, we do not show the modification here but we assume that the discard actions can be applied only in a particular order. While our definition describes a STRIPS task with negative preconditions, we have created a corresponding lifted PDDL model, exemplified in Figure 2.

Also note that the indicators $F_s \subseteq F_0$ are not directly reflected in the planning model, except for being part of initial forces F_0, and are sometimes used for partitioning of the solution to GSP into scenarios, as mentioned in Section 3.

We now show the correspondence between solutions for Π and solutions for the General Scenario Planning.

Theorem 1 *Given a General Scenario Planning problem $\mathcal{G} = \langle M, F_0, F_*; F_o, F_s \rangle$ and its corresponding planning problem $\Pi_\mathcal{G}$, for each valid trajectory ϕ of $\mathcal{G}$ with net-cost c, there exists a plan of $\Pi_\mathcal{G}$ that traverses that trajectory, with the cost c. Further, each plan π of $\Pi_\mathcal{G}$ induces a valid trajectory in $\mathcal{G}$.*

[6] We discuss the differences between the previous and the new planning model in Section 7.

```
(:action enter
  :parameters (?x)
  :precondition (and (source ?x) (not (started)))
  :effect (and (started) (at ?x)
    (increase (total-cost) (starting-cost ?x))))

(:action traverse
  :parameters (?f ?t)
  :precondition (and (connected ?f ?t) (at ?f))
  :effect (and (at ?t) (not (at ?f))
    (increase (total-cost) (connect-cost ?f ?t))))

(:action explain
  :parameters (?x)
  :precondition (and (selected-force ?x)
    (at ?x) (not (considered ?x)))
  :effect (and (considered ?x)
    (increase (total-cost) 1)))
```

Figure 2: Partial corresponding lifted PDDL domain.

Proof: Let $\phi = s_0, s_1, \ldots, s_n$ be a valid trajectory of net-cost c. Then, we have $s_0 \in F_0$ and $s_n \in F_*$. Further, we have $c = \sum_{i=1}^{n} cost(s_{i-1}, s_i) + \sum_{o \in F_o \setminus \phi(F_o)} p_o(o) + c_0(s_0)$. Let π be a sequence of actions that traverses that trajectory, as follows. (enter s_0) is the first action, then the sequence (traverse s_0 s_1), ..., (traverse s_{n-1} s_n) is interchanged with actions (explain s_i) for $s_i \in o$. Next, there is an (achievegoal s_n) action, followed by (discard s_i) actions for $s_i \in F_o \setminus \phi(F_o)$. Note that π is applicable in I and $I[\![\pi]\!]$ includes (goalachieved), as well as (considered o) for all $o \in \phi(F_o)$, and therefore is a plan. Note also that the cost of the plan is exactly c.

Let π be a plan for $\Pi_{\mathcal{G}}$. Let $s_0, s_1, \ldots, s_n$ be the sequence of forces for which (at s_i) appear in the add effects of the actions of π, in the order of appearance of these actions in the plan π. Since π is a plan, it must hold that:

- π has to have (enter s_0) as a first action, and therefore we have $s_0 \in F_0$, and
- π must include (achievegoal s) for some $s \in F_*$ and can only have discard after it. Thus, we must have $s = s_n$.

Since $s_0 \in F_0$ and $s_n \in F_*$, we have $s_0, s_1, \ldots, s_n$ being a valid trajectory in $\mathcal{G}$. □

Note that each plan π for $\Pi_{\mathcal{G}}$ must include exactly one of (explain o) and (discard o) actions for each $o \in F_o$. When costs are not taken into account, nothing prevents (discard o) to be used for achieving the goal fact (considered o), even when (explain o) could be used for that, that is, when the plan traverses the force o. However, the cheapest among the plans that traverse a particular trajectory will have (discard o) actions only for $o \in F_o$ that are not traversed by the plan. Therefore, when using planning tools for valid trajectory generation, plan cost must be taken into account.

7 Comparison to Prior Work

As mentioned earlier, the closest and, to the best of our knowledge, the only related work that introduces automation to address the scenario planning problem is by Sohrabi et al. (2018) and Sohrabi et al. (2019). It is worth noting that their approach relies heavily on the use of *Mind Maps*. These Mind Maps are created by human experts and require a significant manual effort. Each such Mind Map often captures a single force and its related ones, with each Mind Map being small enough to allow human experts to handle. Together, a collection of many Mind Maps capture the expert belief about causal connections between various forces.

If instead the causal relations are captured by an algorithm, there is no need for Mind Maps. While in principle it is possible to automatically translate an FCM to a collection of Mind Maps, such a translation has limited practical value – existing editing visual tools can focus on particular regions of a large FCM (Feblowitz et al. 2021). Note also that there are many ways to automatically build Mind Maps from an FCM. One approach is to create a Mind Map per force, capturing incoming and outgoing causal connections. Such Mind Maps are known as *bow ties*. Another approach is to think of Mind Maps as a spanning forest over the FCM structure. In all those cases, most causal pairs will be captured multiple times across various Mind Maps.

Comparing now the planning model based on Mind Maps that was suggested in the previous work (Sohrabi et al. 2018) to ours, there are several differences. First, the new model is fully lifted, while the previous one was partially grounded. Second, the actions that start the traversal in the previous model were dependent on the Mind Map, and therefore the number of (ground) instances of such actions is now significantly smaller. Third, the previous model separated the plan recognition part, while the new model has integrated the discard/explain actions. That integration allows us to significantly reduce the number of discard actions, restricting applicability to trajectory ends. We note that it is possible to perform a similar integration on the previous model as well.

8 Experiments

To evaluate the feasibility of our approach, we conducted an empirical evaluation in two steps. As a first step, we explore neural techniques to derive FCMs. Then, we create multiple GSPs from each FCM and the corresponding planning tasks, according to Definition 4. As a second step, we compare existing top-k planners on the derived planning tasks, to investigate which of them performs better on these tasks.

8.1 FCM generation

Here we report experiments on the use of seeded FCM extraction based on question answering to demonstrate the feasibility of the approach for semi-automated FCM generation. Table 1 shows the results of our experiments. We built models for the COVID and energy (power and utilities) domains, each with a corpus of authoritative documents D and seed forces F_{seed}, derived from experts[7]. Column $|F_{\text{seed}}|$ represents the size of the seed set for each iteration, starting with 279 forces for covid-large and producing an FCM with 916 forces and 1936 edges in 4.55 hours. A seed set for the next iteration is constructed from the union of the seed

[7] F_{seed} are often accumulated over time by risk practitioners.

	I	$\lvert D\rvert$	$\lvert F_{\text{seed}}\rvert$	$\lvert F\rvert$	$\lvert E\rvert$	Time	$\lvert F_{\text{seed}} \cup F\rvert$	Inc
covid-small	0	2825	33	60	49	0.30	93	182%
	1	2825	93	144	246	0.47	177	90%
	2	2825	177	243	558	0.68	276	56%
	3	2825	276	309	857	0.93	342	24%
	4	2825	342	360	1107	1.08	393	15%
	5	2825	393	399	1238	1.22	432	10%
	6	2825	432	412	1312	1.33	445	3%
	7	2825	445	419	1339	1.35	452	2%
covid-large	0	12618	279	916	1936	4.55	964	246%
	1	12618	964	1714	7566	13.22	1762	83%
	2	12618	1762	2192	12661	23.32	2240	27%
	3	12618	2240	2434	15422	29.38	2482	11%
	4	12618	2482	2552	16819	32.38	2600	5%
energy-small	0	364	168	167	221	0.52	276	64%
	1	364	276	261	739	0.75	370	34%
	2	364	370	293	1188	0.95	402	9%
	3	364	402	301	1283	1.00	410	2%
energy-large	0	5087	488	1145	2560	2.85	1633	235%
	1	5087	1633	1837	9843	7.82	2325	42%
	2	5087	2325	2207	15164	11.33	2695	16%
	3	5087	2695	2324	17574	13.20	2812	4%
	4	5087	2812	2362	18493	13.78	2850	1%

Table 1: Seeded FCM extraction across iterations (I) with number of paragraphs in the documents $|D|$, seed forces F_{seed}, new extracted forces F and edges E, time per iteration (in hours), the seed set for the next iteration $F_{\text{seed}} \cup F$ and its size increase.

set and the produced forces and includes 964 forces, an increase of 246%. We iterate until convergence, that is until the increase in seed set size is smaller than 2%. For pragmatic purposes, we put an additional time constraint of 2 days, some datasets (covid-large 5) went above that. Both the seed forces and the corpus documents were provided by authoritative sources knowledgeable in the domain. The table depicts *small* and *large* datasets, where the large datasets depict realistically sized F_{seed} and sets of corpus documents, and the small datasets are constructed by arbitrarily choosing a subset of documents and seed forces. We provide statistics in the table reflecting the number of initial seed forces, and the size of the FCMs. Because both forces and edges are automatically derived, the quality of the causal pair inference can vary, with NLP models providing a score on how well the text matches the question at hand. We used a threshold of 0.55 as a minimal score to cut off causal pairs below that threshold.[8] We then used sentence embedding based similarity to de-duplicate forces using neural sentence models from Reimers and Gurevych (2019) and Cer et al. (2018). This filtering step on edges eliminates forces when the force is not part of any causal pair, as well as duplicate edges. As shown in the table, the approach does derive fairly large models, even after aggressive filtering.

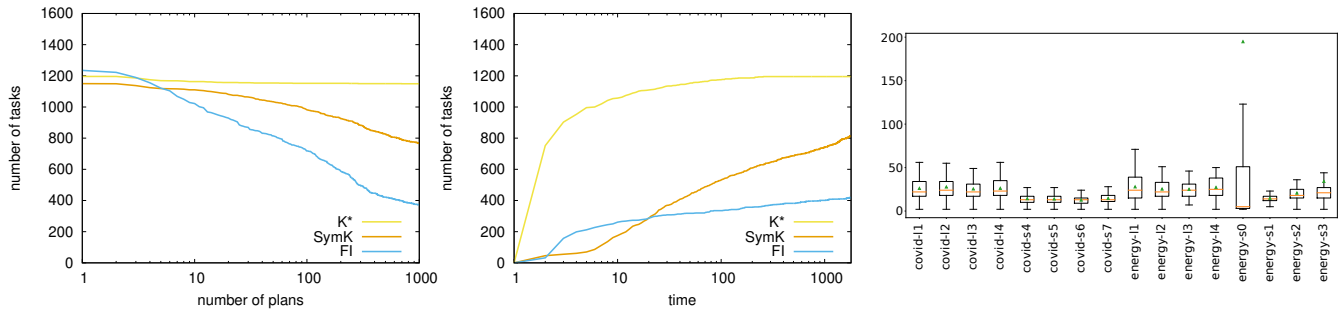
[8]The threshold was determined from past experience and was applied across domains without change.

8.2 Planning for GSP

In order to investigate how well the existing tools for deriving multiple quality-aware solutions to classical planning tasks handle real world size scenario planning problems, we have created 16 collections of planning tasks, corresponding to the last four iterations for each of the models described in Section 8.1 and presented in Table 1. Each of the collections consists of 100 tasks of varying number of indicators and selected forces. The indicators are selected based on force in-degree in the model, using the edges determined for that model (see Section 8.1). Then, 2, 3, 5, or 10 indicators are chosen uniformly, giving preference to forces with 0 in-degree. The selected forces are randomly chosen out of the set of all forces, choosing 3, 5, 10, 25, or 50 forces uniformly out of all forces. The process is repeated 5 times for each combination of number of indicators and number of selected forces, which results in 100 instances generated for each of the 16 models. The resulting 16 PDDL domains with 100 tasks each will be available upon acceptance. We test the performance of three existing top-k planners on the benchmark set, ForbidIterative (FI) and K* (Katz et al. 2018), as well as SymK (Speck, Mattmüller, and Nebel 2020). We choose top-k planners over top-quality planners (Katz, Sohrabi, and Udrea 2020) for two reasons. The first one is historical, since previous work focused on solutions of certain size (Sohrabi et al. 2018). The second one is pragmatic, as our domain was carefully crafted to remove ambiguity in unimportant ordering between actions, the only difference between unordered top-quality planning and top-k planning comes down to a stopping criteria: whether the predefined number of plans was found or a plan with the cost higher that the quality bound was generated. The experiments were performed on Intel(R) Xeon(R) Intel(R) Xeon(R) Platinum 8260 CPU @ 2.40GHz machines, with the time and memory limit of 30min and 2GB, respectively.

Figure 3 (a) plots the number of tasks solved by each of the approaches for each value of $k \in [1, 1000]$, under the entire time bound of 30 minutes. While FI has a small advantage over K* for small values of k, it quickly disappears, and for $k \geq 4$, K* is the dominating approach. For $k \geq 6$, SymK takes the second place, with the difference in coverage between K* and SymK growing from 51 task for $k = 6$ to 382 tasks for $k = 1000$.

When solving scenario planning tasks in an interactive users mode, 30 minutes can be an infeasible time bound. Therefore, we compare the approaches in terms of the overall time until a top-k solution for $k = 1000$ was obtained. Figure 3 (b) shows an anytime performance of the three approaches. It shows the number of tasks solved (1000 or all existing plans found) by a given time bound, for each value $t \in [1, 1800]$. Clearly, K* performs best among the three approaches. For 752 out of the 1600 tasks, it was able to produce a solution in under 2 seconds. Further, 1158 of the tasks are solved in under 60 seconds, and it reaches its maximal obtained coverage of 1195 in 257 seconds. Other approaches perform significantly worse under small time bounds. For FI, it is understandable, as each iteration requires some startup time, and therefore the approach is not well suited for small time bounds. For SymK, to the best of



(a) (b) (c)

Figure 3: (a) Any-k and (b) any-time plots of the number of tasks solved; (c) traversal length data per domain for K*.

our knowledge, its default parameters are tuned for the 30 minutes time bound, and there might be a way to improve its performance for shorter time bounds. This, however, is not the focus of current work.

Finally, in cases when solution quality can be somewhat sacrificed for the sake of speed, diverse planners might be used to derive a set of plans even faster. Unfortunately, for problems with the soft-goals compilation like structure, where there exist actions that can easily achieve sub-goals with paying a penalty, diverse planners tend to prefer such plans. It is an open problem how to obtain a diverse set of plan in such a setting.

8.3 Causal Chains in FCMs

In order to evaluate the richness of the causal relations obtained by the causal extraction part, for each plan produced by a planner, we measure the traversal length, the number of forces traversed by the plan. In addition, we compute the number of unique forces traversed by a plan. Figure 3 (c) shows the traversal lengths for the plans found by K*. The data is aggregated per domain, and the figure shows mean (green triangle), median (red line), the range from the lower quartile to the upper quartile (the box), as well as the variability in the points that lie outside the upper and lower quartiles (the whiskers). As shown in the figure, the mean traversal length across multiple domains is quite high (e.g., 14.6 for covid-small-7, 26.2 for covid-large-4, 34.1 for energy-small-3, and 27.1 for energy-large-4). Note that these traversals are not necessarily simple and may traverse the same force multiple times. Looking at the number of unique forces traversed by each plan (e.g., 10.4 for covid-small-7, 24.4 for covid-large-4, 17.2 for energy-small-3, and 24.6 for energy-large-4) we see, however, that this is mostly not the case, giving some evidence for the richness of the created models.

9 User Feedback

Previous work has demonstrated a working system, built based on the theory presented in this work. Feblowitz et al. (2021) outlines the tool (UI, etc), as well as additional details of the actual deployment of the system by an organization with large risk practice. This organization has independently verified the significant reduction in both time and money by using the system, validating the practical usefulness of our proposed approach. Below we outline their key findings. Users have reported that the generated scenarios are of comparable quality to those created by human experts. In particular, a large percentage (90%) of generated information was considered acceptable or correct by a practitioner, giving a practical evidence for the quality of our generated FCM. Further, the users have reported that the system is "*30x faster in time to generate a dynamic risk model and the first scenario*" and is "*3,000x faster in time to generate the second scenario and each subsequent scenario*" compared to human experts. Note that the key advantage of our approach is the ability to generate and explore orders of magnitude more scenarios than is humanly possible without it.

10 Discussion and Future Work

We formalize the general scenario planning problem and describe a novel combination of neural techniques for generating forces causal models, and symbolic techniques that can solve the derived general scenario planning problems in practice. We propose multiple ways for semi-automated derivation of general scenario planning problems and automated exploration of the solution space of these problems with symbolic planning techniques. Further, we generate real-world sized general scenario planning instances, as well as the corresponding PDDL tasks and empirically compare the existing tools for top-k planning to identify the best tool for this planning domain. Our work constitutes an important step towards fully automating Scenario Planning.

Automating derivation of forces seed sets as well as automation of choice of indicators and implications is the focus of our future work. In particular, we have hinted at an automated awareness component that could be used to continuously monitor news and other document sources to extract the initial forces. We have already started experimenting with such a component based on the BERT fine-tuned with STSB to find public news articles relevant to the existing forces in a forces causal model. We envision multiple ways in which it can be used to drive further automation for the general scenario planning problem.

References

Bhandari, M.; Feblowitz, M.; Hassanzadeh, O.; Srinivas, K.; and Sohrabi, S. 2021. Unsupervised Causal Knowledge Extraction from Text using Natural Language Inference (Student Abstract). In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 15759–15760. AAAI Press.

Cardoso, J. F.; and Emes, M. R. 2014. The Use and Value of Scenario Planning. *Modern Management Science and Engineering* 2(1).

Cer, D.; Yang, Y.; Kong, S.-y.; Hua, N.; Limtiaco, N.; St. John, R.; Constant, N.; Guajardo-Cespedes, M.; Yuan, S.; Tar, C.; Strope, B.; and Kurzweil, R. 2018. Universal Sentence Encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 169–174.

dos Santos, C. N.; Xiang, B.; and Zhou, B. 2015. Classifying Relations by Ranking with Convolutional Neural Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015*, 626–634.

Dunietz, J.; Carbonell, J.; and Levin, L. 2018. DeepCx: A transition-based approach for shallow semantic parsing with complex constructional triggers. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 1691–1701. Association for Computational Linguistics.

Edgar, B.; Abouzeedan, A.; and Hedner, T. 2011. Scenario Planning as a Tool to Promote Innovation in Regional Development Context. Technical report, European Regional Science Association.

Feblowitz, M.; Hassanzadeh, O.; Katz, M.; Sohrabi, S.; Srinivas, K.; and Udrea, O. 2021. IBM Scenario Planning Advisor: A Neuro-Symbolic ERM Solution. In *Proceedings of the Demonstration Track at the 35th Conference on Artificial Intelligence (AAAI-21)*.

Garvin, D. A.; and Levesque, L. C. 2006. A Note on Scenario Planning. *Harvard Business School Background Note* 306–003.

Hassanzadeh, O.; Bhattacharjya, D.; Feblowitz, M.; Srinivas, K.; Perrone, M.; Sohrabi, S.; and Katz, M. 2019. Answering Binary Causal Questions Through Large-Scale Text Mining: An Evaluation Using Cause-Effect Pairs from Human Experts. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5003–5009. IJCAI.

Hassanzadeh, O.; Bhattacharjya, D.; Feblowitz, M.; Srinivas, K.; Perrone, M.; Sohrabi, S.; and Katz, M. 2020. Causal Knowledge Extraction through Large-Scale Text Mining. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*, 13610–13611. AAAI Press.

Johnson, J.; Douze, M.; and Jégou, H. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* .

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-Quality Planning: Finding Practically Useful Sets of Best Plans. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. AAAI Press.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A Novel Iterative Approach to Top-k Planning. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Keyder, E.; and Geffner, H. 2009. Soft Goals Can Be Compiled Away. *Journal of Artificial Intelligence Research* 36: 547–556.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.

Li, P.; and Mao, K. 2019. Knowledge-oriented convolutional neural network for causal relation extraction from natural language texts. *Expert Systems with Applications* 115: 512 – 523. ISSN 0957-4174.

Li, Z.; Ding, X.; Liu, T.; Hu, J. E.; and Durme, B. V. 2020. Guided Generation of Cause and Effect. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, 3629–3636. IJCAI.

Lin, Y.; Shen, S.; Liu, Z.; Luan, H.; and Sun, M. 2016. Neural Relation Extraction with Selective Attention over Instances. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL*.

Mitra, B.; and Craswell, N. 2017. Neural Text Embeddings for Information Retrieval. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, 813–814. Association for Computing Machinery. ISBN 9781450346757.

Oliver, J.; and Parrett, E. 2017. Managing future uncertainty: Re-evaluating the role of scenario planning. *Business Horizons* 61: 339–352. doi:10.1016/j.bushor.2017.11.013.

Peterson, G. D.; Cumming, G. S.; and Carpenter, S. R. 2003. Scenario planning: a tool for conservation in an uncertain world. *Conservation biology* 17(2): 358–366.

Reimers, N.; and Gurevych, I. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Schoemaker, P. J. 1995. Scenario planning: a tool for strategic thinking. *Sloan management review* 36(2): 25.

Sohrabi, S.; Katz, M.; Hassanzadeh, O.; Udrea, O.; Feblowitz, M. D.; and Riabov, A. 2019. IBM Scenario Planning Advisor: Plan recognition as AI planning in practice. *AI Communications* 32(1): 1–13.

Sohrabi, S.; Riabov, A. V.; Katz, M.; and Udrea, O. 2018. An AI Planning Solution to Scenario Generation for Enterprise Risk Management. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI 2018)*, 160–167. AAAI Press.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan Recognition as Planning Revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, 3258–3264. AAAI Press.

Speck, D.; Geißer, F.; and Mattmüller, R. 2018. Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, 250–258. AAAI Press.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic Top-k Planning. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2020)*. AAAI Press.

Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; and Bowman, S. R. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *CoRR* abs/1804.07461. URL http://arxiv.org/abs/1804.07461.

Zeng, D.; Liu, K.; Lai, S.; Zhou, G.; and Zhao, J. 2014. Relation Classification via Convolutional Deep Neural Network. In Hajic, J.; and Tsujii, J., eds., *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers*, 2335–2344. ACL.

GPT3-to-plan: Extracting plans from text using GPT-3

Alberto Olmo, Sarath Sreedharan, Subbarao Kambhampati
Arizona State University
{aolmoher, ssreedh3, rao}@asu.edu

Abstract

Operations in many essential industries including finance and banking are often characterized by the need to perform repetitive sequential tasks. Despite their criticality to the business, workflows are rarely fully automated or even formally specified, though there may exist a number of natural language documents describing these procedures for the employees of the company. Plan extraction methods provide us with the possibility of extracting structure plans from such natural language descriptions of the plans/workflows, which could then be leveraged by an automated system. In this paper, we investigate the utility of generalized language models in performing such extractions directly from such texts. Such models have already been shown to be quite effective in multiple translation tasks, and our initial results seem to point to their effectiveness also in the context of plan extractions. Particularly, we show that GPT-3 is able to generate plan extraction results that are comparable to many of the current state of the art plan extraction methods.

Introduction

Following sequential procedures and plans undergird many aspects of our everyday lives. As we look at many vital and consequential industries, including finance and banking, the ability to identify the correct procedures and adhere to them perfectly, becomes essential. So it is of no surprise that many enterprises invest heavily in accurately documenting these workflows in forms that are easy for their employees to follow. As we start automating many of these day-to-day activities, it becomes important that our automated systems are also able to pick up and execute them. Unfortunately, having these procedures documented is not the same as them being easy and readily available for an AI system to use. Additionally, in many of these high-risk domains, the agent cannot just try to figure out these procedures on their own through trial and error. Instead, we would want to develop ways wherein we can convert these procedures designed for human consumption to easier forms for agents to use. Within the planning community, there has been a lot of recent interest in developing *plan extraction* methods that are able to take natural language text describing a sequential plan. Some of the more recent works in this direction include, works like Feng, Zhuo, and Kambhampati (2018); Daniele, Bansal, and Walter (2017), which have proposed specialized frameworks for performing sequence-to-sequence translation that maps natural language sentences into structured plans.

On the other hand, the mainstream Natural Language Processing (NLP) has started shifting its focus from more specialized translation methodologies to developing general purpose models such as transformer networks (Radford et al. 2019; Brown, Mann, and et al. 2020). These networks have already shown very encouraging results in many tasks and proven their ability to generalize to unseen ones. These are task-agnostic language models trained on large general web corpora and have shown to be comparable (and in some cases better than) their state-of-art task-specific counterparts. Examples of some tasks these models have been tested on includes, question-answering, translation, on-the-fly reasoning and even generation of news articles that are arguably indistinguishable from human-written ones. In light of these advancements, we try to answer the following question: *to what extent can the current state-of-art in general natural language models compete against task-specific action sequences extractors?* These papers have generally looked at employing learning based methods that expect access to large amounts of pre-processed/task-specific data, including annotations that allow mapping of text to the required structured output. These characteristics make the methods fragile to changes in input and output format. Combining this with the need for extensive training data, we expect these systems to require heavy time and resource investment and expert oversight to set up.

In this paper, we want to investigate how GPT-3 (Brown, Mann, and et al. 2020), one of the most recent transformer-based language models, can be used to extract structured actions from natural language texts. We find that these models achieve comparable, and in some cases better scores than previous state-of-the-art task specific methods. We make use of natural language text from three domains and measure the performance of the model in terms of its F_1 score, a commonly used quantitative measure for the task. We then compare it to previously published results for task-specific action extractors which use a varied range of solutions, including, reinforcement learning, (Feng, Zhuo, and Kambhampati

2018), sequence-to-sequence models (Daniele, Bansal, and Walter 2017), Bi-directional LSTMs (Ma and Hovy 2016) or clustering of action templates (Lindsay et al. 2017).

The proliferation and effectiveness of such general language models even in specific tasks, open up new opportunities for planning researchers and practitioners. In particular, it empowers us to deploy planning techniques in real-world applications without worrying about the natural-language interaction aspects of the problem. Also, note that all results reported here are directly calculated from the best GPT-3 raw predictions, with no additional filtering or reasoning employed atop of it. We expect most of the results reported here to improve should we additionally exploit domain-level or task-level insights to filter the results from these models.

Background and Related Works

The Generative Pre-trained Transformer 3 (GPT-3) (Brown, Mann, and et al. 2020) is the latest version of the GPT models developed by OpenAI[1]. A 175 billion parameter autoregressive language model with 96 layers trained on a 560GB+ web corpora (Common Crawl[2] and WebText2 (Gokaslan and Cohen 2019)), internet-based book corpora and Wikipedia datasets each with different weightings in the training mix and billions of tokens or words. Tested on several unrelated natural language tasks, GPT-3 has proven successful in generalizing to them with just a few examples (zero in some cases). GPT-3 comes in 4 versions, Davinci, Curie, Babbage and Ada which differ in the amount of trainable parameters – 175, 13, 6.7 and 2.7 billion respectively (Brown, Mann, and et al. 2020). Previous work on action sequence extraction from descriptions has revolved around specific models for action extraction, some of them trained on largely task-specific preprocessed data. (Mei, Bansal, and Walter 2016; Daniele, Bansal, and Walter 2017) use sequence-to-sequence models and inverse reinforcement learning to generate instructions from natural language corpora. Similarly, Feng, Zhuo, and Kambhampati (2018) uses a reinforcement learning model to extract word actions directly from free text (i.e. the set of possible actions is not provided in advance) where, within the RL framework, actions select or eliminate words in the text and states represent the text associated with them. This allows them to learn the policy of extracting actions and plans from labeled text. In a same fashion, Branavan et al. (2009) also use Reinforcement Learning, a policy gradient algorithm and a log-linear model to predict, construct and ultimately learn the sequence of actions from text. Other works like Addis and Borrajo (2010) define a system of tools through which they crawl, extract and denoise data from plan-rich websites and parse their actions and respective arguments with statistical correlation tools to acquire domain knowledge.

However, to the best of our knowledge this paper is the first work to assess the performance of a general purpose NLP language model on action sequence extraction tasks compared to its current state-of-art task-specific counterpart.

[1]https://openai.com/

[2]https://commoncrawl.org/

	WHS	CT	WHG
Labeled texts	154	116	150
Input-output pairs	1.5K	134K	34M
Action name rate (%)	19.47	10.37	7.61
Action argument rate (%)	15.45	7.44	6.30
Unlabeled texts	0	0	80

Table 1: Characteristics of the datasets used.

Length	Temp.	Top P	Freq.	Pres.	Best of
100	0.0	1	0.0	0.0	1

Table 2: GPT-3 parameters used for all our experiments.

Experiments

Datasets and GPT-3 API We use the three most common datasets for action sequence extraction tasks used in evaluating many of the previous task-specific approaches, including Feng, Zhuo, and Kambhampati (2018) or Miglani and Yorke-Smith (2020). Namely, the "Microsoft Windows Help and Support" (WHS), the "WikiHow Home and Garden" (WHG) and the "CookingTutorial" (CT) datasets. The characteristics of these datasets are provided in Table 1.

The GPT-3 model is currently hosted online[3] and can be accessed via paid user queries with either their API or website in real time. Some example use cases of their service include keyword extraction from natural text, mood extraction from reviews, open-ended chat conversations and even text to SQL and JavaScript to Python converters amongst many others. In general, the service takes free natural language as input and the user is expected to encode the type of interaction/output desired in the input query. The system then generates output as a completion of the provided query. The API also allows the user to further tweak the output by manipulating the following parameters: `Max Tokens` sets the maximum number of words that the model will generate as a response, `Temperature` (between 0 and 1) allows the user to control the randomness (with 0 forcing the system to generate output with the highest probability consistently and rendering it effectively deterministic for a given input). `Top P` also controls diversity; closer to 1 ensures more determinism, `Frequency Penalty` and `Presence Penalty` penalize newly generated words based on their existing frequency so far, and `Best of` is the number of multiple completions to compute in parallel. It outputs only the best according to the model. In Table 2 we show the values that we used for all our experiments to ensure the most consistency in the model's responses.

Query generation Each query consists of a few shot training in natural language text and the corresponding structured representation of the plan. For each example, we annotate the beginning of the natural language text portion with the tag `TEXT` followed by the plan (annotated with the tag `ACTIONS`). In the structure representation, each

[3]More information at https://beta.openai.com/

action is represented in a functional notation of the form $a_0^j(arg_0^0, arg_1^0 \dots arg_k^0) \dots a_n^j(arg_0^n, arg_1^n \dots arg_k^n)$ where a_i^j represents action *i* in sentence j and arg_k^n is the *kth* argument from action a_n in the text. After the training pairs, we include the test sample in natural language text after another tag `TEXT` and then we add a final tag `ACTIONS`, with the expectation that GPT3 will generate the corresponding plan representation after that.

Evaluation and Metrics In order to directly compare the performance of GPT-3 to Miglani and Yorke-Smith (2020), the current state-of-art, we followed a translation scheme with three types of actions, namely, *essential* (essential action and its corresponding arguments should be included in the plan) *exclusive* (the plan must only contain one of the exclusive actions) and *optional* actions (the action may or may not be part of the plan). We use this scheme to generate both the example data points provided to the system and to calculate the final metrics.

In particular, we will use *precision*, *recall* and *F1*, similar to Feng, Zhuo, and Kambhampati (2018); Miglani and Yorke-Smith (2020) to measure the effectiveness of the method.

$$Precision = \frac{\#TotalRight}{\#TotalTagged},\ Recall = \frac{\#TotalRight}{\#TotalTruth}$$
$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \tag{1}$$

Note that the ground truth number and the number of true extracted actions depend on the type that each action in the text corresponds to. For example, a set of exclusive actions only contribute one action to *#TotalTruth* and we only count an extracted exclusive action in *#TotalRight*, if and only if, one of the exclusive actions is extracted. Both *essential* and *optional* actions only contribute once to *#TotalTruth* and *#TotalRight*.

Baselines In Table 3 we compare GPT-3 to several action sequence extractor models:

- EAD: Mei, Bansal, and Walter (2016) design an Encoder-Aligner-Decoder method that uses a neural sequence-to-sequence model to translate natural language instructions into action sequences.
- BLCC: The Bi-directional LSTM-CNN-CRF model from Ma and Hovy (2016) benefits from both word and character-level semantics and implement an end-to-end system that can be applied to action sequence extraction tasks with pre-trained word embeddings.
- Stanford CoreNLP: in Lindsay et al. (2017) they reduce Natural Language texts to action templates and based on their functional similarity, cluster them and induce their PDDL domain using a model acquisition tool.
- EASDRL and cEASDRL: Feng, Zhuo, and Kambhampati (2018) and Miglani and Yorke-Smith (2020) use similar reinforcement learning approaches; they define two Deep Q-Networks which perform the actions of *selecting* or *rejecting* a word. The first DQN handles the extraction of Essential, Exclusive and Optional actions while the second uses them to select and extract relevant arguments.

	Action names			*Action arguments*		
Model	WHS	CT	WHG	WHS	CT	WHG
EAD	86.25	64.74	53.49	57.71	51.77	37.70
CMLP	83.15	83.00	67.36	47.29	34.14	32.54
BLCC	90.16	80.50	69.46	93.30	**76.33**	70.32
STFC	62.66	67.39	62.75	38.79	43.31	42.75
EASDRL	93.46	84.18	75.40	**95.07**	74.80	75.02
cEASDRL	**97.32**	**89.18**	**82.59**	92.78	75.81	**76.99**
GPT-3						
Davinci	**86.32**	**58.14**	**43.36**	22.90	**29.63**	**22.25**
Curie	75.80	35.57	22.41	**31.75**	22.16	13.79
Babbage	62.59	20.62	14.95	22.91	12.59	7.33
Ada	60.68	14.68	8.90	17.91	4.13	2.27

Table 3: F_1 scores for all actions and their arguments across the WHS, CT and WHG datasets for the state-of-art sequence extraction models and GPT-3. State-of-art task-specific model F_1 scores are extracted from Miglani and Yorke-Smith (2020); Feng, Zhuo, and Kambhampati (2018) and represent their best possible recorded performance.

The corresponding *precision*, *recall* and *F1* scores for each method were picked directly from their respective papers.

Results Given that GPT-3 is a few-shot learner we want to know how it performs given different amounts of training samples. To measure this, we query the language model with increasing numbers of examples (with a maximum of four examples) for all domains and report their F1 scores. We stop at the four-shot mark as the total amount of *tokens* or words that the request can contain is 2048. Additionally for the CookingTutorial and Wikihow Garden and Home datasets, 4-shot training examples already exceed this threshold, so we limit the length of input text to 10 sentences per training example. Specifically, we select the training examples as 1-shot (one datapoint is selected at random from the dataset), 2-shot (the two datapoints with the largest proportion of *optional* and *exclusive* actions from the dataset are selected), 3-shot (the three datapoints with the largest proportion of *optional*, *exclusive* and *essential* actions) and 4-shot (an additional random datapoint is added to 3-shot).

In Figure 1 we show how the F_1 score changes given 1, 2, 3 and 4-shot training samples when tested on the whole Windows Help and Support dataset and actions alone. Order and action arguments are not considered. Unsurprisingly, Davinci, the model with the most amount of trainable parameters, performs best with over 80% F_1 score for each category. Both Davinci and Curie show the tendency to perform better the more examples they are given peaking at 3 and 4-shots respectively. Similarly, Babbage and Ada show their peaks given 2 and 4 examples while underperforming at one-shot training. This is unsurprising, given the fact that these models are simplified versions of GPT-3 which have also been trained on a smaller corpus of data for higher speed. Hence, they need more than one example to grasp the task.

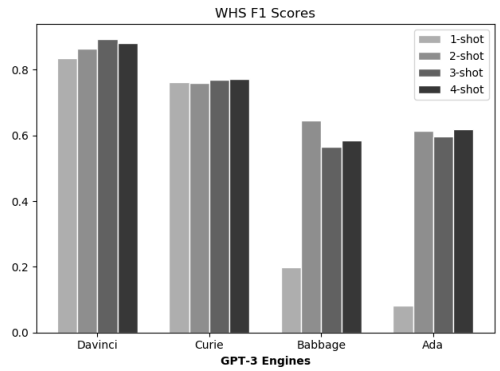


Figure 1: F_1 action scores of the model on the Windows Help and Support dataset for 1 to 4 few-shot training.

In table 3 we compare the F_1 scores for action name and their argument extractions as reported by previous and current state of the art task-specific action sequence extractors against all GPT-3 engines: Davinci, Curie, Babbage and Ada, ordered from most to least powerful The scores are calculated based on 1 and account for *essential*, *exclusive* and *optional* actions and their respective arguments. All GPT-3 models are trained with two-shot examples. As expected, Davinci overall performs the best compared to the rest of engines. We can see that Davinci also outperforms the EAD, CMLP and STFC task-specific models for the Windows Help and Support domain on extracting actions. Even though it underperforms on the argument extraction task compared to the state of art, it's worth nothing that still obtains better than random extraction scoring.

Ordering We want to assess whether GPT-3 is capable of inferring plan order from text. This is a feature which is mostly missing in previous task-specific state of the art like Feng, Zhuo, and Kambhampati (2018) or Miglani and Yorke-Smith (2020). As a preliminary evaluation, we create three examples (one for each dataset, shown in Figure 2), where order of the plans does not match how actions are listed in the text. In the WHS domain, we state on the second and third sentences that action *click(advanced)* must be performed eventually but only after *click(internet, options)*, and, even though the corresponding sentences appear in the opposite order, GPT-3 places them as expected. Similarly, in the CT example, we state that *first* we need to *measure the quantity of oats* and *cook them* only later and once again, it generates the actions in correct ordering. For the last example, GPT-3 shows to understand that action *paint(walls)* has to be done before *remove(furniture)* and, interestingly, even though *decorate(floor)* is stated on the first sentence, the model seems to understand that it can be performed *anytime* and places the action last. Note that these are just anecdotal evidences and we would need to perform studies over larger test sets to further evaluate GPT-3's ability to identify the ordering of plans. Our current evaluation along this dimension is limited by the lack of ordering annotation and one of our future works would be to create/identify such ordered and annotated text-to-plan datasets.

TEXT: Windows Help and Support
Start internet explorer.
You need to eventually click on the advanced tab
But before that, click on the internet options on the tools menu.
After the click on the advanced tab, click to clear or select the enable personalized favorites menu check box.
ACTIONS:
click(internet,explorer), click(tools), click(internet,options), click(advanced), click(personalized,favorites), click(check,box)

TEXT: CookingTutorial
For this recipe you need to let the beef simmer for 30 minutes.
After that, make sure that you don't start cooking oats right away.
You have to first measure their quantity which needs to be 150g.
Then cook them and pour the vegetables with the meat.
ACTIONS:
let(beef,simmer), measure(quantity,oats), cook(oats), pour(vegetables,meat)

TEXT: WikiHow Home and Garden
You can decorate your floor anytime.
But before decorating your floor it's important that you paint your walls first.
After painting your walls, remove all furniture in the room.
ACTIONS:
paint(walls), remove(furniture), decorate(floor)

Figure 2: Query examples on WHS, CT and WHG. Each query was input to Davinci along with two preceding training instances containing the largest proportion of *optional* and *exclusive* actions. The output is shown in regular text while the input is displayed in bold.

Discussion and Conclusion

In this paper we have shown that GPT-3, the state-of-art general purpose Natural Language model, can compete against task-specific approaches in the action sequence extraction domain, getting closer than ever to surpassing their performance. From the user's perspective, these transformer models pose the advantage of needing almost negligible computational resources from the user side by being readily available at just one query away and seem like a possible solution in the future to many natural language tasks should they keep up with their rate of improvement. However, some limitations are still prevalent on GPT-3. It is still far from being accurate for the more action-diverse natural text datasets. This becomes all the more apparent during argument extraction where, as shown, it generally fails to obtain competitive scores even on its most powerful Davinci version. Thus, this hinders the possibility of using GPT-3 directly for general extraction tasks other than the most simple. For less diverse plans, it does show competing performance and we posit that it could be used as an intermediate step in a hybrid system.

On the other hand, GPT-3 seems to show some ability to identify the underlying sequentiality of the plan by recognizing words like *before, after, first, anytime* or *eventually* and rearranging the plans accordingly. This is a capability generally missing from most state of the art plan extractors as they assume the ordering of the plan to be same as that of the sentences corresponding to each action in the text. Hence, ordering speaks for yet another potential advantage of using general models, as in they are usually not limited by specific assumptions made by system designers. Finally, note that the aforementioned strengths of the model could be further augmented should OpenAI allow for more finetuning in the future.

Acknowledgements

Dr. Kambhampati's research is supported by the J.P. Morgan Faculty Research Award, ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, N00014-9-1-2119, AFOSR grant FA9550-18-1-0067 and DARPA SAIL-ON grant W911NF19-2-0006. We also want to thank OpenAI and Miles Brundage for letting us get research access to the GPT-3 API.

References

Addis, A.; and Borrajo, D. 2010. From unstructured web knowledge to plan descriptions. In *Information Retrieval and Mining in Distributed Environments*, 41–59. Springer.

Branavan, S.; Chen, H.; Zettlemoyer, L.; and Barzilay, R. 2009. Reinforcement Learning for Mapping Instructions to Actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 82–90. Suntec, Singapore: Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P09-1010.

Brown, T.; Mann, B.; and et al., R. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Daniele, A. F.; Bansal, M.; and Walter, M. R. 2017. Navigational instruction generation as inverse reinforcement learning with neural machine translation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, 109–118. IEEE.

Feng, W.; Zhuo, H. H.; and Kambhampati, S. 2018. Extracting action sequences from texts based on deep reinforcement learning. *arXiv preprint arXiv:1803.02632* .

Gokaslan, A.; and Cohen, V. 2019. OpenWebText Corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Lindsay, A.; Read, J.; Ferreira, J.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning models from natural language action descriptions. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27.

Ma, X.; and Hovy, E. H. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics. doi:10.18653/v1/p16-1101. URL https://doi.org/10.18653/v1/p16-1101.

Mei, H.; Bansal, M.; and Walter, M. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Miglani, S.; and Yorke-Smith, N. 2020. NLtoPDDL: One-Shot Learning of PDDL Models from Natural Language Process Manuals. In *ICAPS'20 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS'20)*. ICAPS.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1(8): 9.

Similarity Metrics for Transfer Learning in Financial Markets

Diego Pino[1], **Javier García**[1], **Fernando Fernández**[1], **Svitlana S Vyetrenko**[2]
[1]Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. 28911 Leganés (Madrid). Spain
[2]J. P. Morgan AI Research, New York, NY, USA
{dpino,fjgpolo,ffernand}@inf.uc3m.es, svitlana.s.vyetrenko@jpmchase.com

Abstract

Markov Decision Processes (MDPs) are an effective way to formally describe many Machine Learning problems. In fact, recently MDPs have also emerged as a powerful framework to model financial trading tasks. For example, financial MDPs can model different market scenarios. However, the learning of a (near-)optimal policy for each of these financial MDPs can be a very time-consuming process, especially when nothing is known about the policy to begin with. An alternative approach is to find a similar financial MDP for which we have already learned its policy, and then reuse such policy in the learning of a new policy for a new financial MDP. Such a knowledge transfer between market scenarios raises several issues. On the one hand, how to measure the similarity between financial MDPs. On the other hand, how to use this similarity measurement to effectively transfer the knowledge between financial MDPs. This paper addresses both of these issues. Regarding the first one, this paper analyzes the use of three similarity metrics based on *conceptual*, *structural* and *performance* aspects of the financial MDPs. Regarding the second one, this paper uses Probabilistic Policy Reuse to balance the exploitation/exploration in the learning of a new financial MDP according to the similarity of the previous financial MDPs whose knowledge is reused.

Introduction

Markov decision processes (MDPs) are a common way of formulating decision making problems in reinforcement learning (RL) tasks (Sutton and Barto 2018). An MDP provides a standard formalism for multi-stage decision processes, whilst at the same time being able to capture the stochastic nature of realistic situations. This is the reason why MDPs have also emerged as a powerful framework for modeling real financial trading problems (Chakraborty 2019; Huang 2018; Bäuerle and Rieder 2011). In these financial MDPs, the objective is to learn a trading strategy or policy able to maximize some measure of performance over time, typically the profit. Interestingly, these financial MDPs can be configured to model different market scenarios, so that different trading policies can be tested and analyzed. However, finding these policies from scratch is often a hard task. On the one hand, the learning process is based on a computationally intensive trial-and-error process guided by reward signals from the environment. On the other hand, such a trial-and-error process is unfeasible in a real trading scenario where a single bad decision can lead to large losses. Therefore, when it is possible to obtain these trading policies, it would be good to take advantage of them as much as possible. Specifically, rather than learning a new trading policy for every financial MDP, a policy could be learned on one financial MDP, then transferred to another, similar financial MDP, and either used as is, or treated as a starting point from which to learn a new trading policy.

Such a knowledge transfer is particularly interesting in a *Sim-to-Real* scenario (Tan et al. 2018). Due to the reality gap, trading strategies learned in simulation usually do not perform well in real environments. Since the reality gap is caused by model discrepancies between the simulation and the real dynamics, similarity metrics can be used to effectively quantify that gap, and as a direct way to measure simulation fidelity. In this way, the hedge funds, investors, or banks can test their trading strategies in simulation before risking their funds in a real trading problem (Vyetrenko et al. 2020). In this context, similarity metrics can be used to select the best simulated financial MDP to transfer from, avoiding real losses.

Therefore, it is a critical step to decide when two MDPs are similar. This paper investigates three different strategies to compute the similarity between MDPs in a financial context. These strategies allows to measure the similarity from three different perspectives considering *conceptual*, *structural* and *performance* aspects of the MDPs. The *conceptual* similarity is based on comparing the statistical properties (also known as *stylized facts* (Vyetrenko et al. 2020)) of the asset returns, order volumes, order arrival times, order cancellations, etc. Typically, such stylized facts are used by human experts to analyze the similarities and differences between simulated and real financial data (Fabretti 2013). In contrast, the *structural* similarity is based on the comparison of the experience tuples generated by two different MDPs: two MDPs are considered to be similar whenever the experience tuples they generate are similar as well. In this paper, such a structural comparison is performed by using restricted Bolztmann machines (RBMs) (Ammar et al. 2014). Finally, this paper uses the well-known exploitation/exploitation strategy, π-reuse, to measure the *perfor-*

mance similarity between MDPs. In particular, it measures the reuse gain (i.e., the "advantage") of using one task to aid the learning of another (Fernández and Veloso 2013). The higher the reuse gain, the greater the similarity between the MDPs. Finally, such a reuse gains are used by a Policy Reuse algorithm to bias the learning process of a new MDP in a transfer learning context.

This paper is organized as follows. Section *Related Work* describes some previous related work. Section *Background* presents key concept on RL, transfer learning and similarity metrics required to better understand the rest of the paper. Section *Abides* introduces the abides simulator used to recreate different financial markets. Section *Similarity Metrics* presents the similarity metrics based on the conceptual, structural and performance relationships between the different financial markets. Finally, Section *Experimentation* shows the evaluation performed, and Section *Conclusions* summarizes the conclusions and some future work.

Related Work

RL has been used in several applications in finance and trading, including portfolio optimization and optimal trade execution. Given that actions taken in finance and trading may have long-term effects not immediately measurable, some financial problems can be viewed as sequential decision problems. Furthermore, the environments in which these areas work may be very large or continuous, so RL can well-suited to solving finance problems. RL was first introduced and implemented in the financial market in 1997 (Moody et al. 1998).

Some examples where RL has been successfully applied to financial problems include portfolio optimization, optimized trade execution, and market-making. In portfolio optimization the goal is to create an optimum portfolio given the specific factors that should be maximized or minimized (e.g. expected return, financial risk) and taking into account any constraints (Kanwar et al. 2019). Instead, the goal of optimized trade execution is to sell or buy a specific number of shares of a stock in a fixed time period, such that the revenue received (in the case of selling) is maximized or the capital spent (in the case of buying) is minimized (Nevmyvaka, Feng, and Kearns 2006). In market-making, the market maker buys and sells stocks with the goal of maximizing the profit from buying and selling them and minimizing the inventory risk. In this context, RL has been used successfully to come up with price setting strategies to maximize profit and minimize inventory risk (Ganesh et al. 2019). However, in contrast to these approaches, this work focuses on a different objective: to measure the similarity of financial markets, modeled in the multi-agent simulator Abides, in order to determine which scenarios are most similar to the real world and transfer the knowledge learned to the real world to maximize profit. Transfer RL has also been used in a financial context (Martínez, García, and Fernández 2020) using PPR to identify the similarity among different market scenarios in the context of pricing. Identifying these similarities permit to measure how a market scenario is similar to another, and thus how the pricing policy learned in a market scenario is useful to learn a pricing policy in a related, but different, market scenario.

Background

This section introduces key concepts required to better understand the rest of the paper. First, some background in RL is introduced, then the main concepts of transfer RL are visited, and finally the concepts of similarity and distance are described.

Reinforcement Learning

RL (Kaelbling, Littman, and Moore 1996) is an area of machine learning where agents learn what actions to take in an environment to maximize an accumulated reward. The learning process is based on trial and error guided by reinforcement signals from the environment that reward agents for performing actions that bring them closer to solving the problem. A combination of exploration (trying the unknown) and exploitation (using knowledge the agent already has) can be used to make improvements in the performance of RL algorithms.

In particular, RL tasks are described as Markov Decision Processes (MDPs) represented by tuples in the form $\mathcal{M} = \langle S, A, T, R \rangle$, where S is the state space, A is the action space, $T : S \times A \to S$ is the transition function between states, and $R : S \times A \to \mathbb{R}$ is the reward function. At each step, the agent is able to observe the current state, and choose an action according to its policy $\pi : S \to A$. The goal of the RL agent is to learn an optimal policy π^* that maximizes the return $J(\pi)$:

$$J(\pi) = \sum_{k=0}^{K} \gamma^k r_k \tag{1}$$

where r_k is the immediate reward obtained by the agent on step k, and γ is the discount factor, which determines how relevant the future is (with $0 \leq \gamma \leq 1$). The interaction between the agent and the environment tends to be broken into episodes, that end when reaching a terminal state, or when a fixed amount of time has passed. With the goal of learning the policy π, Temporal Differences methods (Sutton and Barto 2018) estimate the sum of rewards represented in Equation 1. The function that estimates the sum of rewards, i.e., the return for each state s given the policy π is called the value-function $V^\pi(s) = E[J(\pi)|s_0 = s]$. Similarly, the action-value function $Q^\pi(s, a) = E[J(\pi)|s_0 = s, a_0 = a]$ is the estimation of the value of performing a given action a at a state s being π the policy followed. The Q-learning algorithm (Watkins 1989) is one of the most widely used for computing the action-value function.

Transfer Learning

The learning process in RL can sometimes be too costly, thus the concept of transfer learning was born. In the transfer learning scenario we assume there is an agent who previously has addressed a set of source tasks represented as a sequence of MDPs, $\mathcal{M}_1, \ldots, \mathcal{M}_n$. If these tasks are somehow "similar" to a new task $\mathcal{M}_{n+1}$, then it seems reasonable the agent uses the acquired knowledge solving $\mathcal{M}_1, \ldots, \mathcal{M}_n$ to

solve the new task $\mathcal{M}_{n+1}$ faster than it would be able to from scratch. Transfer learning is the problem of how to obtain, represent and, ultimately, use the previous knowledge of an agent (Torrey and Shavlik 2010; Taylor and Stone 2009).

However, transferring knowledge is not an easy endeavour. On the one hand, we can distinguish different transfer settings depending on whether the source and target tasks share or not the state and action spaces, the transition probabilities and the reward functions. It is common to assume that the tasks share the state space and the action set, but differing the transition probabilities and/or reward functions. However, in case the tasks do not share the state and/or the action spaces, it is required to build mapping functions, $\mathcal{X}_S(s_t) = s_s$, $\mathcal{X}_A(a_t) = a_s$, able to map a state s_t or action a_t in the target task to a state s_s or action a_s in the source task. Such mapping functions require not only knowing if two tasks are related, but *how* they are related, which means an added difficulty. On the other hand, it is required to select what type of information is going to be transferred. Different types of information have been transferred so far ranging from instance transfer (a set of samples collected in the source task) to policy transfer (i.e., the policy π learned in the source task). Nor is this a simple task, because depending on how much and how the source and target tasks are related, it could be transferred one type of information or another. For example, some of them transfer the whole Q-value function (Mahmud et al. 2013), others transfer the learned policy (Mehta et al. 2008), or they could transfer a model of the state transition function and/or the reward function (Sunmola and Wyatt 2006). But this paper focuses on reusing the policy by π-reuse.

Finally, the most "similar" task among $\mathcal{M}_1, \ldots, \mathcal{M}_n$ to solve $\mathcal{M}_{n+1}$ should be selected in the hope that it produces the most positive transfer. For this purpose, similarity metrics could be used, which translate into a measurable quantity of how related two tasks are.

Similarity Metrics

Similarity metrics are a very important part of transfer learning, as they provide a measure of distance between tasks. Similarity functions, or their complementary distance functions, are mathematical functions that assign a numerical value to each pair of concepts or objects in a given domain. This value measures how similar these two concepts or objects are: if they are very similar, it is assigned a very low distance, and if they are very dissimilar, it is assigned a larger distance (Ontañón 2020). Therefore, through similarity metrics it is possible to determine how good the transfer of knowledge will be.

Simulated Environment

Limit Order Books The limit order book represents a snapshot of the supply and demand for an asset at a given point in time. It is an electronic record of all the outstanding buy and sell limit orders organized by price levels. A matching engine, such as first-in-first-out (FIFO), is used to pair incoming buy and sell order interest (Bouchaud et al. 2018). The limit order book is split into two sides–the ask and bid sides containing all the sell and buy limit orders, respectively. At time t, let b_t be the best bid price, and let a_t be the best ask price. We define mid-price as $m_t = \frac{a_t + b_t}{2}$. Choose time scale Δt. Given a time scale Δt, which can range from milliseconds to months, the log return (or simply return) at scale Δt is defined as $r_{t,\Delta t} = \ln m_{t+\Delta t} - \ln m_t$.

Order types are further distinguished between limit orders and market orders. A limit order specifies a price that should not be exceeded in the case of a buy order, or should not be gone below in the case of a sell order. Hence, a limit order queues a resting order in the order book at the side of the market participant. A market order indicates that the trader is willing to accept the best price available immediately. A diagram illustrating the limit order book structure is provided in Figure 1.

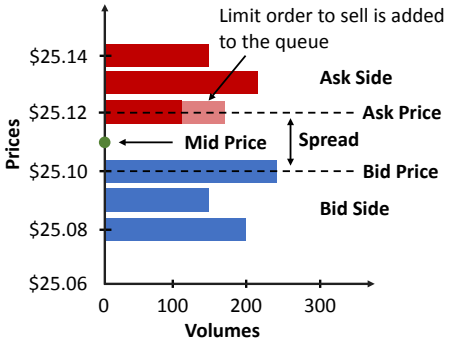


Figure 1: Visualization of the limit order book structure.

ABIDES Simulator ABIDES (Byrd, Hybinette, and Balch 2019) – which stands for Agent Based Interactive Discrete Event Simulator – was designed from the ground up to support AI agent research in market applications, and currently enables the simulation of tens of thousands of trading agents interacting with an exchange agent to facilitate transactions. ABIDES supports configurable pairwise network latencies between each individual agent as well as the exchange. Similar to the real exchange, ABIDES allows to simulate market activity as a result of interaction of multiple background agents with diverse objectives.

We use ABIDES to simulate market scenarios which are distinguished by different number of background agents. Each scenario has a unique agent configuration making each learning process a different task. In particular, we use configurations with **non-learning** exchange, zero intelligence, momentum and noise traders as well a **learning** Q-Learner agent.

The Q-learner agent is a basic agent that will try to learn to beat the market by using reinforcement learning. To do this, at each instant of time, it will have to perform one of the following actions: buy an asset, sell an asset or do nothing. The agent will not be able to buy another asset if he already has one, just as he will not be able to sell one if he already owes one. The agent's state is represented by the difference between the buying and selling volume, and the information whether he currently owes, owns or doesn't own an asset.

A brief description of non-learning background agents is given below.

Exchange agent: NASDAQ-like agent which lists any number of securities for trade against a LOB with price-then-FIFO matching rules, and a simulation kernel which manages the flow of time and handles all inter-agent communication.

Zero intelligence agents: In our experiment, we use the implementation of zero intelligence agents described in (Byrd 2019) to represent institutional investors who follow mean reverting fundamental price series to provide liquidity to the market via limit orders.

Momentum agents: The momentum agents base their trading decision on observed price trends. Our implementation compares the 20 past mid-price observations with the 50 past observations and places a buy order of random size, if the former exceeds the latter and a sell order otherwise.

Noise agents: Noise agents are designed to emulate the actions of consumer agents who trade on demand without any other considerations (e.g., (Kyle 1985)). Each retail agent trades once a day by placing a market order. The direction and the size of the trade are chosen randomly.

Similarity Metrics

This paper uses three different concept of metrics which are based on *conceptual*, *structural* and *performance* aspects of the MDPs to be compared. The first one, called stylized facts, is based on the features of the financial markets themselves. The second one uses Restricted Boltzmann Machines (RBM) to determine the structural similarity between tasks. Finally, the last one is based in the reuse gain obtained applying π-reuse.

Stylized Facts

The conceptual metric is based on the use of what it is known as stylized facts (Vyetrenko et al. 2020; R.Cont 2001). The result of more than half a century of empirical studies on financial time series indicates that although different assets are not necessarily influenced by the same events or information sets, if one examines their properties from a statistical point of view, the seemingly random variations of asset prices share some quite nontrivial statistical properties. Such properties, common across a wide range of instruments, markets and time periods are called stylized empirical facts. The stylized facts are not a metric per se, but are actually based on an analysis of the graphs by studying the shape of the graphs. Vyetrenko et al. (2020) describe several groups of stylized facts, but this paper uses only those stylized facts related to asset return distributions. In particular, we focus on:

- **Mid Price Returns**: Statistical similarity of asset mid price returns $r_{t,\Delta t}$ for $\Delta t = 1$ and $\Delta t = 10$ minutes.
- **Auto-correlation**: Statistical similarity of linear autocorrelations of asset mid price returns $\text{corr}(r_{t+\tau,\Delta t}, r_{t,\Delta t})$ over $\Delta t = 30$ minutes.

Restricted Boltzmann Machines

RBMs (Ammar et al. 2014) are energy-based models for unsupervised learning. They use a generative model of the distribution of training data for prediction. These models employ stochastic nodes and layers, making them less vulnerable to local minima (Salakhutdinov, Mnih, and Hinton 2007). RBMs are stochastic neural networks with bidirectional connections between the visible and hidden layers. This allows RBMs to posses the capability of regenerating visible layer values, given a hidden layer configuration. The visible layer represents input data, while the hidden layer discovers more informative spaces to describe input instances. Therefore, RBMs could also be seen as density estimators, where the hidden layer approximates a (factored) density representation of the input units. Formally, an RBM consists of two binary layers: one visible and one hidden. The visible layer models the data while the hidden layer enlarges the class of distributions that can be represented to an arbitrary complexity.

The main idea is that we can use an RBM to describe different MDPs in a common representation, providing a similarity measure. The RBMs are good as similarity metrics since they can automatically discover the dynamic phases of MDPs, and predict the transfer performance between source and target tasks. Moreover, they have been used in many domains for this purpose with good results (Ammar et al. 2014).

We first train an RBM to model data collected in the source task, yielding a set of relevant and informative features that characterize the source MDP. These features can then be tested on the target task to assess MDP similarity. To do this, we collect the tuples from the source task and pass them as input to the RBM, the expected output being the input tuples, the weights of the neurons are adjusted by contrastive divergence. Then, the tuples of the target task are collected and provided to the RBM, already trained, and we calculate the mean square error obtained between the input and the output, so that the lower the error the higher the similarity.

π-reuse

The π-reuse strategy is an exploration strategy able to bias a new learning process with a past policy (Fernández and Veloso 2013). Let Π_{past} be the past policy to reuse and Π_{new} the new policy to be learned. We assume that we are using a RL algorithm to learn the action policy, so we are learning the related Q function. Any RL algorithm can be used to learn the Q function, with the only requirement that it can learn off-policy, i.e., it can learn a policy while executing a different one, as Q-Learning does.

The goal of π-reuse is to balance random exploration, exploitation of the past policy, and exploitation of the new policy, as represented in Equation 2.

$$a = \begin{cases} \Pi_{past}(s) & \text{w.p. } \psi \\ \epsilon - greedy(\Pi_{new}(s)) & \text{w.p. } (1-\psi) \end{cases} \quad (2)$$

The π-reuse strategy follows the past policy with probability ψ, and it exploits the new policy with probability of

$1 - \psi$. As random exploration is always required, it exploits the new policy with a ϵ-greedy strategy.

Given a policy Π_{past} that solves a task Ω_{past}, and a new task Ω, the reuse Gain of the policy Π_{past} on the task Ω, the Reuse Gain of the policy Π_{past} on the task Ω, is the gain obtained when applying the π-reuse exploration strategy with the policy Π_{past} to learn the policy Π_{new}. The Reuse Gain is used in this work to measure the distance between the scenarios, taking into account that the higher the reuse gain, the lower the distance.

PRQ-Learning

PRQ-Learning is an algorithm based on π-reuse and its goal is to solve a new task reusing the knowledge in a library of pasts tasks (Fernández and Veloso 2013). Therefore, PRQ-Learning is equipped with a policy library composed of n past optimal policies that solve n different tasks, respectively, plus the ongoing learned policy. PRQ-Learning is able to select which policy should be reused and what exploration/exploitation strategy follow.

Let W_i be the Reuse Gain of the policy Π_i on the task Ω. Also, let W_Ω be the average reward that is received when following the policy Π_Ω greedily. The solution we introduce consists of following a softmax strategy using the values W_Ω and W_i, with a temperature parameter τ, as shown in Equation 3. This value is also computed for Π_0, which we assume to be Π_Ω.

$$P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}} \quad (3)$$

In the first episode, all the policies have the same probability to be chosen. Once a policy is chosen, the algorithm reuse it to solve the task, updating its reuse gain with the reward obtained in the episode, and therefore, updating the probability to follow each policy. The policy being learned can also be chosen, although in the initial steps it behaves as a random policy, given that the Q values are initialized to 0. While new updates are performed over the Q-function, it becomes more accurate, and receives higher rewards when executed. After executing several episodes, it is expected that the new on-going policy obtains higher gains than reusing the past policies, so it will be chosen most of the time.

If the policy being learned is chosen, the algorithm follows a completely greedy strategy. However, if the policy chosen is one of the past policies, the π-reuse strategy, is followed instead. In this way, the reuse gain of each of the past policies can be estimated online with the learning of the new policy.

Experimentation

This section evaluates the similarity metrics described in Section *Similarity Metrics*. For this purpose, we will use 7 different market scenarios, each one recreating different market conditions. Finally, PRQ-Learning is used to learn in a target market scenario, reusing the knowledge learned in previous source markets. However, the experimental setting is described first.

Experimental setting

This section first describes each of the 7 market scenarios used for the experimentation, and then it introduces the parameter setting used for each of the approaches.

Market Scenarios There are 7 scenarios in this paper. The scenario 1 is the main scenario, i.e., we consider it as the target market scenario for the experiments with π-reuse and PRQ-Learning. Each scenario is composed of several agents that will determine the behavior of the system. The agents represents the traders who can place a buy or sell order. There are several types of agents in Abides, but in this paper we have used five, explained in the Section *Abides*.

Furthermore, each of the market scenarios require a fundamental value, and at the end, the behaviour of all scenarios is determined by this value and the different agents that trade in the environment. Each market scenario is configured as described in Table 1, where the columns represent the scenario and the rows represent the number of agents of each type.

	1	2	3	4	5	6	7
Zero Intelligent	100	100	100	100	100	100	100
Exchange	1	1	1	1	1	1	1
Q-Learner	1	1	1	1	1	1	1
Noise	0	10	5	10	5	10	0
Momentum	0	5	10	10	0	0	10

Table 1: Agent Configuration

These agent configurations were chosen randomly by adding noise and momentum agents. The idea is to analyze which type of agents perturbs more the initial configuration and whether the similarity metrics determine that scenarios with less new agents are more similar to the initial configuration.

Parameter Setting The parameter setting for Q-learning is as follows. The parameter γ is set to 0.98, α to 0.99, and it is decremented every step as $\alpha \times 0.999$. It uses ϵ-greedy for the exploration/exploitation of the state and action space, where ϵ is set initially at 0.999, and it is decremented every step as $\epsilon \times 0.9995$. Regarding π-reuse, ψ is set to 1, and then it is reduced every step as $\psi \times 0.99$. Finally, in PRQ-Learning the temperature parameter τ is initially set to 0 and it increases following a sigmoid function until 0.002.

Evaluation of the Similarity Metrics

This section presents the evaluation of the similarity between the different market scenarios by using the proposed styled facts, the RBMs and π-reuse.

Stylized Facts We first compare distributional similarity between stylized facts. In the below analysis, scenarios that generate similar stylized fact distributions will be interpreted as having less distance between them.

Table 2 shows the analysis of the stylized facts considered in this paper. The first row corresponds to autocorrelation

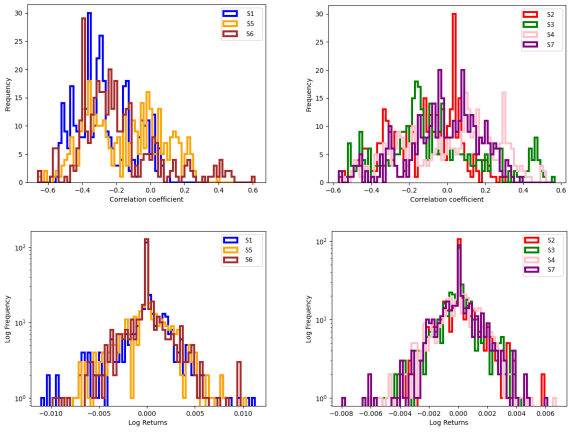


Table 2: Stylized Facts: Comparison between scenarios

and the second row corresponds to the minutely returns. Furthermore, the analysis of the stylized facts of the 7 proposed scenarios allows us to distinguish between two clusters: the first is composed of scenarios 1, 5, and 6, and the other of scenarios 2, 3, 4, and 7. For a better understanding, in Table 2 the scenarios are already grouped according to the similarity in the shape of the stylized facts considered. Therefore, the graphs in the first column of Table 2 show the value of the corresponding stylized fact to the markets 1, 5, and 6, whilst the graphs in the second column shows the stylized facts for the markets 2, 3, 4, and 7. The autocorrelation shows that in the cluster 1 the plot is skewed to the left, however in cluster 2 it is centered. Finally, the minutely returns in both clusters is similar but in the cluster 1 the graph is more wider.

Restricted Boltzmann Machines The second metric analyzed in this paper is that provided by the RBMs. For this purpose, a set of transitions has been gathered in each of the proposed scenarios, and the distance between them has been computed as described in the section *Restrictred Boltzmann Machines*. Figure 2 is a heat map that represents the distance calculated using RBMs from each pair of scenarios. In particular, the heat map shows as many rows and columns as there are market scenarios, and each cell represents the similarity of the X-axis market to the Y-axis market scenario. The distances range from 0 to 1, and a cell will be darker the greater the distance between the compared scenarios. The order of the scenarios has been changed to facilitate the readability of the data.

In this case, two clusters are observed again. The first one indicates that the scenario 1, 5 and 6 are similar to each others, the second one indicates that the scenario 2, 3, 4 and 7 are also similar to each others. It is important to note that these results are the same as with the stylized facts and it is so interesting because scenario 1, 5 and 6 do not have momentum agents, and on the other hand, the scenarios 2, 3, 4 and 7 all have momentum agents. Somehow momentum agents have a critical influence in the functioning of the scenarios which allows us to separate the scenarios according whether they have momentum agents or not.

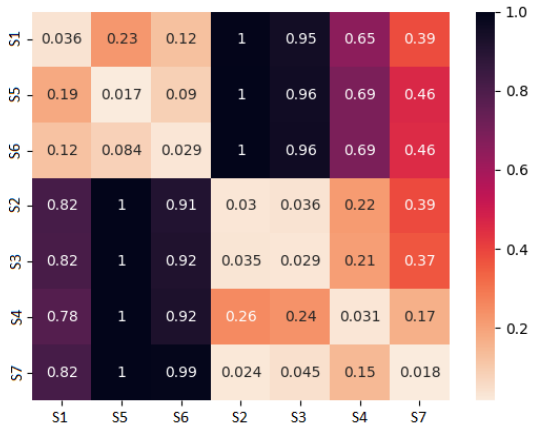


Figure 2: Heat map with the distance between each pair of scenarios using Restricted Boltzmann Machines

π-reuse. Finally, we use the π-reuse algorithm to compute the reuse gain, hence, the performance similarity between the market scenarios. Figure 3 shows six learning processes, each one corresponding to the reuse of the policy learned in a source market (2-7) to learn in the target market 1 by applying π-reuse. Additionally, the blue line represents the average accumulated reward obtained by Q-learning.

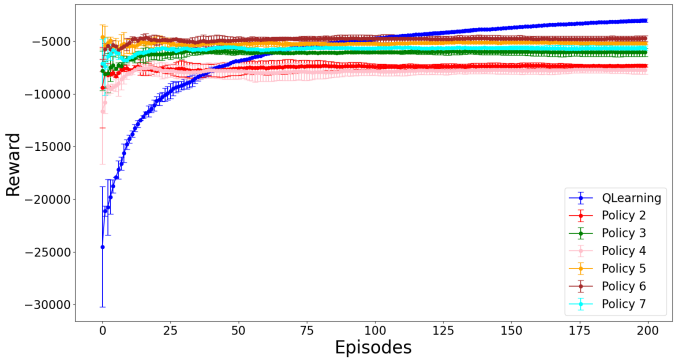


Figure 3: Reuse Gain to learn market scenario 1 by reusing the policies learned in the other scenarios by π-reuse

From Figure 3, it can be seen that π-reuse produces a jump-start at the initial episodes, although the final performance is lower than that obtained by Q-learning given that π-reuse always maintain active exploration. All policies achieve a similar reward, although the best seem to be policies 5 and 6. In Figure 4, the blue line represents the same as before, but the other lines represent the reuse gain obtained with the policies learned previously by π-reuse in a fully greedy setting.

It can be observed that when the policies are followed fully greedy, the policy 5, 6 and 7, show a performance equal or superior to Q-learning, precisely because the exploration has been deactivated. Besides, it can be observed that poli-

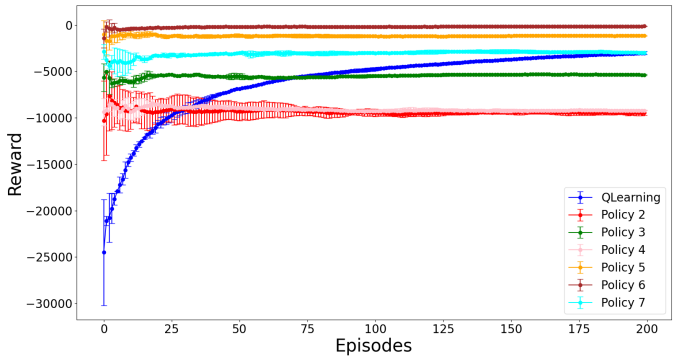


Figure 4: Reuse Gain of the previously learned policies by π-reuse in a fully greedy setting.

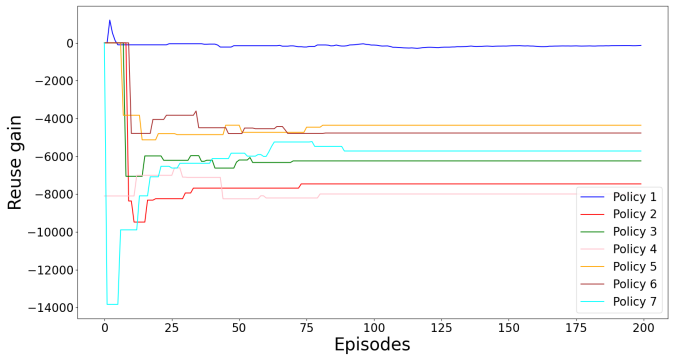


Figure 6: Reuse gain evolution

cies 5 and 6, who are in the same cluster, are the best performers.

Figures 3 and 4 show that there are two policies which obtain better results than the others. These policies are the the corresponding to the scenarios 5 and 6. Therefore, if we consider that more similar means higher final performance, then policies 5 and 6 are the most similar to scenario 1, the same conclusion we reached with the previous metrics.

Transfer learning through PRQ-Learning

Finally, the Q-Learner agent has been trained with PRQ-Learning in the scenario 1 reusing the policies learned in the scenarios 2-7 as a library. In Figure 5 we compare the accumulated reward obtained with PRQ-Learning against Q-Learning.

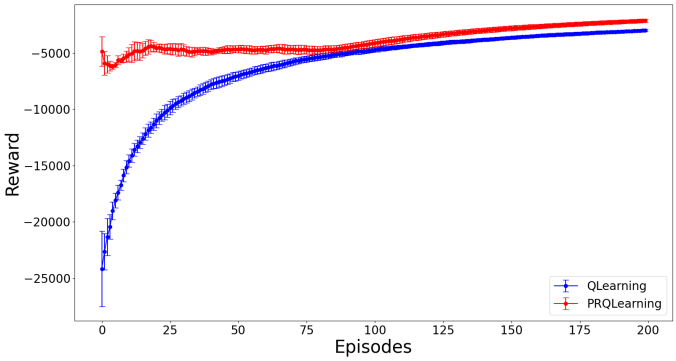


Figure 5: Accumulated reward obtained by Q-learning vs PRQ-Learning

Figure 5 shows that PRQ-Learning achieves better results than Q-Learning in all episodes, also in the Figure 6 we can see that the two policies most similar to scenario 1 (policy 5 and 6), according the similarity metrics, are the ones which reach higher reuse gain.

In Figure 7 we can also see the probability of selecting each policy in each episode. It can be seen how at the beginning of the learning process all policies have the same probability of being selected. However, around episode 50 it can be seen how the probability of the current policy, together with policies 5 and 6 increases with respect to the others. Therefore, policies 5 and 6 have a higher probability of being selected because they are more similar with respect to scenario 1.

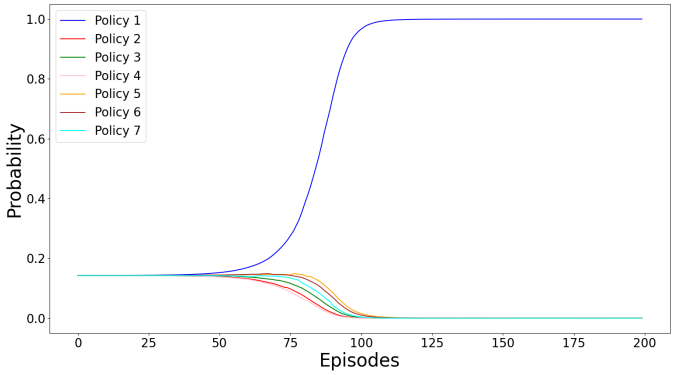


Figure 7: Evolution of the probability of selecting each policy

Conclusions

This paper has analyzed the use of three similarity metrics from a *conceptual*, *structural* and *performance* perspective to measure the similarity between market scenarios. Analyzing the similarity between markets from different perspectives allows us to obtain a more complete analysis of what is similar and what is not. In the particular case of the market scenarios considered in this paper, the three similarity metrics considered have been able to determine which scenarios are more similar to each others, reaching the same conclusion and determining that there are two clusters according to whether the scenarios have momentum agents or not. Furthermore, through PRQ-Learning it is confirmed that the scenarios of the same cluster as the target scenario are the most similar to this one, and also the ones that improve learning the most. Therefore, by using similarity measures to determine which configurations are most similar to the target task and by using PRQ-Learning to transfer the knowledge learned in a test environment, learning in the target

task could be improved. This is particular relevant in a *Sim-to-Real* context: similarity metrics can help to answer how similar simulations and the actual world are. They could be used to provide theoretical guaranties that ensure the learned policies transferred from simulation to the actual world will perform as required, or to define mechanisms to tune/modify the simulated environments, so the gap between the simulated world and the actual one decreases. The latter opens new lines of research that we are currently investigating.

Acknowledgements

This research was funded in part by JPMorgan Chase Co. Any views or opinions expressed herein are solely those of the authors listed, and may differ from the views and opinions expressed by JPMorgan Chase Co. or its affiliates. This material is not a product of the Research Department of J.P. Morgan Securities LLC. This material should not be construed as an individual recommendation for any particular client and is not intended as a recommendation of particular securities, financial instruments or strategies for a particular client. This material does not constitute a solicitation or offer in any jurisdiction. This work has also been supported by the Madrid Government (Comunidad de Madrid-Spain) under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17), and in the context of the V PRICIT(Regional Programme of Research and Technological Innovation). Finally, Javier García is partially supported by the Comunidad de Madrid funds under the project 2016-T2/TIC-1712.

References

Ammar, H.; Eaton, E.; Taylor, M.; Decebal, C.; Mocanu, D.; Driessens, K.; Weiss, G.; and Tuyls, K. 2014. An Automated Measure of MDP Similarity for Transfer in Reinforcement Learning.

Bäuerle, N.; and Rieder, U. 2011. *Markov decision processes with applications to finance*. Springer Science & Business Media.

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

Byrd, D. 2019. Explaining Agent-Based Financial Market Simulation. *arXiv preprint arXiv:1909.11650* .

Byrd, D.; Hybinette, M.; and Balch, T. H. 2019. ABIDES: Towards High-Fidelity Market Simulation for AI Research.

Chakraborty, S. 2019. Capturing financial markets to apply deep reinforcement learning. *arXiv preprint arXiv:1907.04373* .

Fabretti, A. 2013. On the problem of calibrating an agent based model for financial markets. *Journal of Economic Interaction and Coordination* 8(2): 277–293.

Fernández, F.; and Veloso, M. 2013. Learning domain structure through probabilistic policy reuse in reinforcement learning. *Progress in Artificial Intelligence* 2(1): 13–27. ISSN 21926360.

Ganesh, S.; Vadori, N.; Xu, M.; Zheng, H.; Reddy, P.; and Veloso, M. 2019. Reinforcement Learning for Market Making in a Multi-agent Dealer Market.

Huang, C. Y. 2018. Financial trading as a game: A deep reinforcement learning approach. *arXiv preprint arXiv:1807.02787* .

Kaelbling, L.; Littman, M.; and Moore, A. 1996. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.* 4: 237–285.

Kanwar, N.; et al. 2019. *Deep Reinforcement Learning-Based Portfolio Management*. Ph.D. thesis.

Kyle, A. S. 1985. Continuous auctions and insider trading. *Econometrica: Journal of the Econometric Society* 1315–1335.

Mahmud, M. M. H.; Hawasly, M.; Rosman, B.; and Ramamoorthy, S. 2013. Clustering Markov Decision Processes For Continual Transfer .

Martínez, E.; García, J.; and Fernández, F. 2020. Probabilistic Policy Reuse for Similarity Computation Among Market Scenarios. *FinPlan 2020* 28.

Mehta, N.; Natarajan, S.; Tadepalli, P.; and Fern, A. 2008. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning* 73(3): 289–312. ISSN 08856125.

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.

Nevmyvaka, Y.; Feng, Y.; and Kearns, M. 2006. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, 673–680.

Ontañón, S. 2020. An overview of distance and similarity functions for structured data. *Artificial Intelligence Review* 53(7): 5309–5351.

R.Cont. 2001. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance* 1(2): 223–236. doi:10.1080/713665670.

Salakhutdinov, R.; Mnih, A.; and Hinton, G. 2007. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, 791–798. New York, NY, USA: Association for Computing Machinery. ISBN 9781595937933. doi:10.1145/1273496.1273596. URL https://doi.org/10.1145/1273496.1273596.

Sunmola, F. T.; and Wyatt, J. L. 2006. Model transfer for Markov decision tasks via parameter matching. *In Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006)* .

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

Tan, J.; Zhang, T.; Coumans, E.; Iscen, A.; Bai, Y.; Hafner, D.; Bohez, S.; and Vanhoucke, V. 2018. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332* .

Taylor, M. E.; and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(7).

Torrey, L.; and Shavlik, J. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, 242–264. IGI global.

Vyetrenko, S.; Byrd, D.; Petosa, N.; Mahfouz, M.; Dervovic, D.; Veloso, M.; and Balch, T. H. 2020. Get Real: Realism Metrics for Robust Limit Order Book Market Simulations. *International Conference on AI in Finance.* .

Watkins, C. 1989. *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Cambridge, UK.

Proving Security of Cryptographic Protocols using Automated Planning

Alberto Pozanco, Antigoni Polychroniadou, Daniele Magazzeni and Daniel Borrajo*
AI Research
J.P. Morgan
{alberto.pozancolancho,antigoni.poly,daniele.magazzeni,daniel.borrajo}@jpmchase.com

Abstract

Lately, financial institutions have started implementing cryptographic protocols for various privacy-related use cases. One of the key aspects in applying these protocols consists of proving that they are secure. This paper presents ongoing work on using Automated Planning for that task. This analysis serves two purposes. On one side, it can help cryptographic practitioners to analyze some properties of their protocols. On the other hand, it can help automated planning work on unsolvability to enrich the type of domains they test their systems on.

Introduction

In a general definition, Automated Planning (AP) is the task of generating a sequence of actions, namely a *plan*, such that, when applied to a given initial state, it results in a state where some given goals are true (Ghallab, Nau, and Traverso 2004). There exist many real-world applications of AP including urban traffic control (Pozanco, Fernández, and Borrajo 2021), network security scanning (Hoffmann 2015), intermodal transportation (García et al. 2013), or robots operation (Cashmore et al. 2015), among others.

In all these cases the aim is to find a plan that solves the given task, assuming such a plan exists. However, there are other domains in which one might be interested in the opposite: proving that the given task does not have a solution, i.e., no plan exists to achieve the goals from the initial state. Proving planning tasks unsolvability is an active research area (Eriksson, Röger, and Helmert 2017; Sreedharan et al. 2019) as highlighted by the recent Unsolvability International Planning Competition[1]. The interest though, has lied mostly on the research side with few real-world applications needed of this planning capability. In this paper we propose yet another application of AP in which we need to demonstrate that a planning task is unsolvable: proving the security of cryptographic protocols.

Cryptographic protocols are communication protocols defined by a sequence of rules that describe how parties should communicate to preserve privacy or data integrity. An early example of a well-known cryptographic protocol is the Diffie-Hellman key exchange (Diffie and Hellman 1976). In order to ensure that such protocols are secure, they are designed under the assumption that an attacker might interfere at different stages of the protocol, for example having access to the secrets of some parties or altering the messages they exchange. All of this is formalized in the concept of secure Multi-Party Computation (MPC) which allows a set of mutually distrustful parties to compute a function jointly over their inputs while maintaining the privacy of the inputs and ensuring the correctness of the outputs. Secure computation is beneficial only in places where parties cannot trust each other, and cannot trust some external entity. In particular, in case such incorruptible trusted party can be found, computing the joint function can be performed easily: The parties send their inputs to the trusted party, which compute the function on the inputs and send the parties the output of the computation. However, in many realistic scenarios (such as decentralized consensus in block chain (Nakamoto 2008)) the parties cannot rely or trust such an external entity, and secure computation comes to eliminate the need for such trusted party. Seminal results established in the '80s (Yao 1982; Goldreich, Micali, and Wigderson 1987; Ben-Or, Goldwasser, and Wigderson 1988; Chaum, Crépeau, and Damgård 1987) show that *any* computation can be emulated by a secure protocol.

Cryptographic protocols security is usually verified either through simulators (computational approach) or using formal methods. In the former case, protocols are proved to be secure by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario (simulation), which is secure by definition. A protocol is secure if any attacker in the real model (where no trusted party exists) can do more harm than if it was involved in the ideal model (Goldreich 2001). In this paper we will focus on proving the security of cryptographic protocols using the latter approach. To do that, we need to succinctly formalize both the protocol and the actions that the attacker can execute to achieve its goals, i.e., interfere/extract information from the other parties. A protocol is secure if there is no plan the attacker can execute to achieve those goals (Dolev and Yao 1983). The impact in the financial institutions is clear given the recent approaches that use crytographic protocols for various applications (Sangers et al. 2019; Asharov et al. 2020; Cartlidge, Smart, and Alaoui 2020; Byrd and Polychroniadou 2020; Balch, Diamond, and Poly-

*On leave from Universidad Carlos III de Madrid (consultant)
[1] https://unsolve-ipc.eng.unimelb.edu.au/#

chroniadou 2020; Cozzo, Smart, and Alaoui 2021).

The rest of the paper is organized as follows. In the next section we review some basic notions of AP. Then, we introduce the privacy-preserving aggregation protocol we are focusing on. After that, we describe how we model cryptographic protocols in PDDL. Next, we show a proof of the privacy-preserving aggregation protocol for a fixed number of parties using AP. Finally, we put our contribution in the context of related work, discuss the results, and outline some future lines of research.

Background

Formally, a single-agent STRIPS planning task can be defined as a tuple $\Pi = \langle F, A, I, G \rangle$, where F is a set of propositions, A is a set of instantiated actions, $I \subseteq F$ is an initial state, and $G \subseteq F$ is a set of propositions that define a goal we want to reach.

A state consists of a set of propositions $s \subseteq F$ that are true at a given time. A state is totally specified if it assigns truth values to all the propositions in F, as the initial state I of a planning task. A state is partially specified (partial state) if it assigns truth values to only a subset of the propositions in F, as the conjunction of propositions G of a planning task.

Each action $a \in A$ is composed of a set of preconditions (pre(a)), which represent the literals that must be true in a state to execute an action a, and a set of effects (eff(a)), which represent the literals that become true (add(a) effects) or false (del(a) effects) in the state after the execution of action a. The definition of each action might also include a cost $c(a)$ (the default cost is one).

The execution of an action a in a state s is defined by a function γ such that $\gamma(s, a) = (s \setminus \text{del}(a)) \cup \text{add}(a)$ if pre(a) $\subseteq s$, and s otherwise (a cannot be applied). The output of a planning task is a sequence of actions, namely plan, $\pi = (a_1, \ldots, a_n)$. The execution of a plan π in a state s can be defined as:

$$\Gamma(s, \pi) = \begin{cases} \Gamma(\gamma(s, a_1), (a_2, \ldots, a_n)) & \text{if } \pi \neq \emptyset \\ s & \text{if } \pi = \emptyset \end{cases}$$

A plan π is valid for solving Π iff $G \subseteq \Gamma(I, \pi)$. Thus, $\Gamma(I, \pi)$ is a final state that fulfills the property of all propositions in the goal being true. The cost of a plan is commonly defined as $c(\pi) = \sum_{a_i \in \pi} c(a_i)$, where $c : A \to \mathbb{R}_{>0}$ is a non-negative action cost function. A plan with minimal cost is called optimal.

Planning tasks are compactly specified in a domain-independent fashion using a standard language: the Planning Domain Definition Language (PDDL) (McDermott et al. 1998). PDDL separates the definition of the planning task into two parts: domain and problem. The domain expresses the set of available actions, as well as the object types, predicates and functions. The problem contains the initial state and goals of the task.

Cryptographic Protocols

We will focus on the Privacy-preserving aggregation protocol as a running example of the use of planning for proving cryptographic protocols. This protocol is based on Additive secret-sharing, which we will introduce first.

Additive Secret-Sharing

An n-out-of-n (additive) secret-sharing scheme takes as input a secret s from some input domain and outputs n shares, with the property that it is possible to efficiently reconstruct s from n shares, but every subset of at most $n - 1$ shares reveals nothing about the secret s.

A secret-sharing scheme consists of two algorithms: the first algorithm, called the *sharing algorithm*, Share, takes as input the secret s and the parameters n, and outputs n shares. The second algorithm, called the *recovery algorithm*, Recover, takes as input n shares and outputs a value s. It is required that the reconstruction of shares generated from a value s produces the same value s.

In an additive secret-sharing scheme, n parties hold shares, the sum of which yields the desired secret. By setting all but a single share to be a random value, we ensure that any subset of $n - 1$ parties cannot recover the initial secret. Formally, consider the following.

Let $\mathbb{F}$ be a finite field and let $n \in \mathbb{N}$. Consider the secret-sharing scheme $\mathrm{A}^n = (\mathsf{Share}, \mathsf{Recover})$ defined below.

- The algorithm Share on input (s, n) performs the following:
  1. Generate $(s_1, \ldots, s_{n-1})$ uniformly at random from $\mathbb{F}$ and define $s_n = s - \sum_{i=1}^{n-1} s_i$.
  2. Output $(s_1, \ldots, s_n)$ where s_i is the share of the i-th party.
- The recovery algorithm Recover on input $(s_1, \cdots, s_n)$, outputs $\sum_{i=1}^{n} s_i$.

The distribution of any $n - 1$ of the shares is the uniform one on $\mathbb{F}^{n-1}$ and hence independent of s.

Privacy-preserving Aggregation Protocol

Secure aggregation is commonly used in Federated Learning (Konečnỳ et al. 2016; Kairouz et al. 2019), which enables a set of users working with a trusted server, to collaboratively learn a shared machine learning model while keeping each user's data within its own local systems. In particular, each user computes a local trained model based on his/her local data; the updates (e.g., the weights of the model) are sent to the server in an encrypted way; and the server aggregates these local updates to construct a global model. In this work we consider an aggregation protocol based on the additive secret-sharing property mentioned above where every subset of $n - 1$ shares of a secret does not reveal the secret (tailored to cross-silo federated learning). Suppose that each of n users U_i has a number x^i (such as the number of shares they own of a given company) and they all want to compute the addition of all those numbers without revealing each number to the rest of agents. Protocol 1 presents a high level description of the protocol. It is composed of three rounds. In the first one, each user U_i decomposes its secret number $\mathsf{Share}(x^i, n)$ in n numbers $x^i_j, j = 1..n$ such that their sum is equal to $\mathsf{Share}(x^i, n)$, $\mathsf{Share}(x^i, n) = \sum_{j=1}^{n} x^i_j$. Then, each user U_i sends x^i_j to user U_j.

In the second round, each user sends the sum of all received numbers from the rest of users, X_i, to the server. Finally, the server computes the sum of all received numbers from all users. A scheme of the protocol is shown in Figure 1.

Protocol 1 Privacy-Preserving Aggregation Protocol

The protocol runs with users $U_1, \ldots, U_n$ and a server S. It proceeds as follows:
Inputs: For $i \in [n]$, user U_i holds input x^i.
Round 1: Each user U_i proceeds as follows:

1. Secret shares its input x^i using additive secret sharing such that $(x_1^i, \ldots, x_n^i) \leftarrow \mathsf{Share}(x^i, n)$.
2. For all $j \in [n]$, user U_i sends x_j^i to user U_j.

Round 2: Each user U_i sends $X_i = \sum_{j \in [n]} x_i^j$ to S.
Round 3: The server S outputs $X = \sum_{i \in [n]} X_i$.

As we can easily see, the actual secret numbers $\mathsf{Share}(x^i, n)$ of each user cannot be known by the rest of users nor by the server. Also, in order to break this protocol, a malicious agent (attacker) would have to corrupt either all n users (to leak private inputs x^i) or $n-1$ users and the server (to leak the private input x^n of the incorruptible user U_n given the private inputs $(x^1, \ldots, x^{n-1})$ of the $n-1$ users and the final sum X). Note that if the malicious agent corrupts $n-1$ users, no private information is revealed about the private input of the incorruptible user U_n based on the properties of the additive secret sharing. That is, the malicious agent will not have enough shares x_i^n to reconstruct x^n. Same holds for corruption of $n-2$ users, $n-3$ users and so on. The goal now is to prove that this protocol is secure. In order to do so, we will model the task as a planning task and use a planner to construct the proof.

Modeling Cryptographic Protocols in PDDL

In this section we detail how we model Protocol 1 and its security proof in PDDL. Firstly, we formalize both the protocol and the actions the attacker can execute to extract information about the users in a PDDL domain. Then, we show how we specify the initial state and the goals in PDDL to prove the security of the protocol. What follows is intended to be a high level description of the domain and problems we are generating. For the sake of clarity, we will assume $n = 4$ when modeling the actions and problems. The complete version of the domains and problems we are using is available upon request.

Domain Model

We define two main types of objects: *weight*, which map to each of the x_j^i numbers that users exchange; and *party*, which refers to the different agents or parties acting in the protocol. Parties can be of type *malicious* if they can corrupt other parties, or *corruptible* if they can be corrupted. Finally, *corruptible* parties are divided into *user* and *server*.

We define the following set of predicates:

- *(known ?w - weight ?u - user ?p - party)*, which represents that a weight from one of the users is known by a party. This predicate implicitly model the weights of each user, x_j^i, adding also the party who has access to that information. In this case, j would be the *weight* parameter of the predicate, i would be the *user* parameter of the predicate, and *party* would represent which agent in the protocol has access to the given j weight from user i. At the beginning of the protocol, each user only knows its weights, i.e., we have *(known* W_1 U_1 U_1*)* but not *(known* W_2 U_1 U_2*)*, since U_1 has not shared its weights yet.
- *(will-sum ?u - user ?w - weight)*, which represents which user will sum each weight. This implicitly represents the mapping of users and weights taking place at the second step of Round 1.
- *(shared ?u - user)*, which represents that a user has shared all its weights with the rest of users.
- *(known-partial-sum ?w - weight ?p - party)*, which represents that the partial sum of a weight (X_i) is known by a party. This represents Round 2, where users are sending the sum of a given weight to the server.
- *(known-total-sum ?p - party)*, which represents that the aggregated sum of all the weights is known by a party (X). This represents Round 3, where the server knows the sum of all the weights from all the users and outputs it.
- *(can-corrupt ?m - malicious ?c - corruptible)*, which represents that a malicious agent can corrupt a corruptible party. This predicate is used in the initial state to define the set of parties the malicious agent can corrupt. If a party can be corrupted, the malicious agent will have access to the same information as the party.
- We also add three predicates that define the current round of the protocol: *(round1)*, *(round2)* and *(round3)*. During planning, the domain definition will force the planner to create the plans by following the order of these rounds, i.e., following the protocol.

We can divide the actions of the domain into two main categories: actions that represent the protocol execution, and actions that represent the possible steps in the protocol in which malicious agents could infer information from users. We define the following actions to represent the protocol execution:

- *share-weights*, which represents the first round of Protocol 1, where users exchange their weights with other users. The full action is shown in Figure 2.
- *next-round-1-2*, which advances the protocol from Round 1 to Round 2 once all users have exchanged their weights.
- *send-sum-to-server*, which represents the second round of Protocol 1 where a user sums the weights it is processing (defined by the *will-sum* predicate) and sends the results to the server. Assuming user U_1 has to sum all the weights W_1 from each user, *(will-sum* $U_1 W_1$*)*, the effect of executing this action is that now both U_1 and the server know the sum of that weight, i.e., *(known-partial-sum* W_1 U_1*)* and *(known-partial-sum* W_1 *Server)* are true.

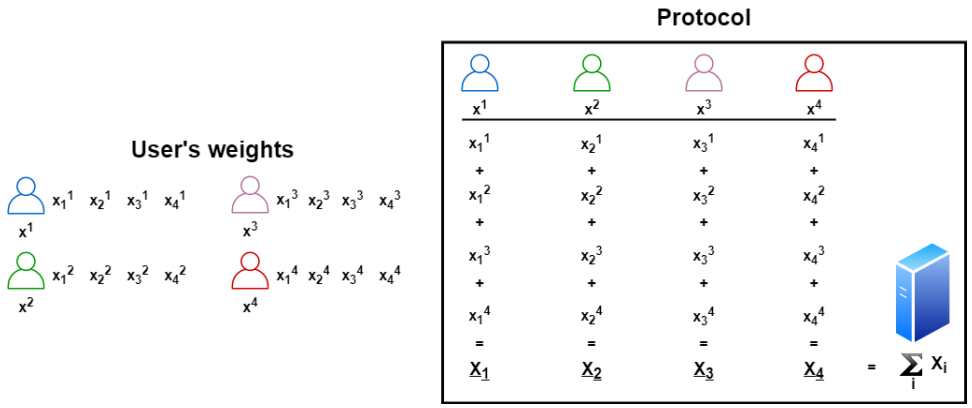


Figure 1: Scheme of the privacy-preserving aggregation protocol.

- *next-round-2-3*, which advances the protocol from Round 2 to Round 3 once each user has sent the sum of weights to the server.
- *compute-sum-server*, which represents the Round 3 of Protocol 1 where the server aggregates the results of the per-weight sums. The effect of executing this action is that *(known-total-sum Server)* becomes true.

We also define the following actions to represent the possible attacks; i.e., steps in the protocol in which the malicious agent could infer information from the users:

- *malicious-act-users*, represents a malicious agent corrupting one of the users. This can happen if the malicious agent can corrupt the user. The effect of this action is that now the malicious agent knows all the information that user has access to, i.e., its own weights plus the ones it has to process coming from other users.
- *malicious-act-servers*, which represents a malicious agent corrupting the server. This can happen if the malicious agent can corrupt the server. The effect of this action is that now the malicious agent knows all the information the server has access to, i.e., the partial sums of all the weights, and therefore the total sum.
- *malicious-user-weight-from-server*, which represents a malicious agent inferring one weight of a user by subtraction when it has corrupted other users and the server. The full action is shown in Figure 3.
- *malicious-server-sum-from-user-weights*, which represents a malicious agent inferring one partial sum of a weight by subtraction when it has other partial sums and the total sum.

Problem Description

In the problem file we specify the initial state and the goals. The initial state specifies: the information known by the parties at the beginning of the protocol; and the parties that the malicious agent can corrupt. The goals specify that the protocol must end, i.e., *(known-total-sum Server)* is true, and the information we want to preserve secure, i.e., the information that the malicious agent should not have access to.

An excerpt of a problem is shown in Figure 4. In this case, we are at the beginning of the protocol (only *(round1)* is true), users only know their own weights, and the malicious agent can corrupt three of the users plus the server. The goal state of the problem is that the malicious agent knows all the weights from every user plus *(known-total-sum server1)*, meaning that the protocol has finished.

This formulation allows us to easily define different scenarios. We can represent different malicious agent behaviors by modifying the information it has access to, i.e., the parties that can be corrupted. We can also prove different levels of security by changing the goals. For example, we could prove that the malicious agent will not be able to get the information of only a particular user, or that the protocol is secure up to a given round.

Proof Procedure

The idea behind this domain and problem formulation is that if a solution to such a planning task exists, there is a plan the malicious agent can follow to infer the information about all users, and therefore the protocol is not secure under that configuration. If the problem does not have a solution, the protocol is proved to be secure under that configuration. To guarantee this we need to make some assumptions about the model we are generating and the planner we are using to solve the tasks. Regarding the model, we assume that it precisely represents both the protocol and all the possible attacks a malicious agent can conduct. This assumption is common in all the techniques that formally verify the security of cryptographic protocols (Dolev and Yao 1983). For the planner, we assume it is complete, meaning that if there is a plan that solve the planning task, the planner will find it. In our scenario, the planner will always find a plan that allows the malicious agent to know all secrets in case such a plan exists.

This proves that the protocol is secure for a given configuration (e.g. number of users, number of corruptible users, ...) as specified in the problem. However, we are interested in proving the security of the protocol under multiple configurations. More specifically, proving the security of the protocol as the number of users that can be corrupted grows.

```
(:action share_weights
:parameters (?u1 ?u2 ?u3 ?u4 - user
             ?w1 ?w2 ?w3 ?w4 - weight)
:precondition (and (round1)
                   (not (= ?w1 ?w2))
                   (not (= ?w1 ?w4))
                   (not (= ?w1 ?w3))
                   (not (= ?w2 ?w3))
                   (not (= ?w2 ?w4))
                   (not (= ?w3 ?w4))
                   (not (= ?u1 ?u2))
                   (not (= ?u1 ?u4))
                   (not (= ?u1 ?u3))
                   (not (= ?u2 ?u3))
                   (not (= ?u2 ?u4))
                   (not (= ?u3 ?u4))
                   (known ?w1 ?u1 ?u1)
                   (known ?w2 ?u1 ?u1)
                   (known ?w3 ?u1 ?u1)
                   (known ?w4 ?u1 ?u1)
                   (will_sum ?u1 ?w1)
                   (will_sum ?u2 ?w2)
                   (will_sum ?u3 ?w3)
                   (will_sum ?u4 ?w4))
:effect (and (known ?w2 ?u1 ?u2)
             (known ?w3 ?u1 ?u3)
             (known ?w4 ?u1 ?u4)
             (shared ?u1)))
```

Figure 2: Action that represents the first round of Protocol 1, where users exchange their weights. The preconditions check that the users are different, user *?u1* knows its weights, and the mapping predicates that determine which user will sum which weight. The effects of the action is that user *?u1* has shared its weights, allowing other users to know some of them.

Algorithm 1 details the procedure we use to prove the security of the protocol up to a certain number of corrupted users.

Algorithm 1 Procedure to prove the security of Protocol 1.

Require: n, S
Ensure: Users, Plan
1: Corrupted-U $\leftarrow 0$
2: Plan $\leftarrow \emptyset$
3: **while** Corrupted-U $\leq n$ **and** Plan $= \emptyset$ **do**
4: Problem $\leftarrow$ GENERATEPROBLEM(Corrupted-U,S)
5: Plan $\leftarrow$ PLANNER(Problem)
6: Corrupted-U $\leftarrow$ Corrupted-U + 1
7: **return** Corrupted-U, Plan

It receives as inputs the number of users of the protocol, n, and a boolean variable S that determines if the malicious agent can corrupt the server or not. The algorithm consists of a loop that increases the number of corrupted users until n is reached or a plan is found. For each number of corrupted users, we generate a different planning problem (line 4) by adding or removing *can-corrupt* predicates. Then we call a planner and try to solve the generated problem. If the problem is solvable (Plan$\neq \emptyset$), the protocol is not secure for that number of corrupted users, and the algorithm finishes. In this case, it returns the number of corrupted users together with the plan that the malicious agent could follow to extract the information of the N users. The algorithm will always return the lowest number of users for which the protocol turns insecure.

```
(:action malicious_user_weight_from_server
:parameters (?u1 ?u2 ?u3 ?u4 - user
             ?w1 - weight
             ?m1 - malicious)
:precondition (and (round2)
                   (not (= ?u1 ?u2))
                   (not (= ?u1 ?u4))
                   (not (= ?u1 ?u3))
                   (not (= ?u2 ?u3))
                   (not (= ?u2 ?u4))
                   (not (= ?u3 ?u4))
                   (known ?w1 ?u1 ?m1)
                   (known ?w1 ?u2 ?m1)
                   (known ?w1 ?u3 ?m1)
                   (not (known ?w1 ?u4 ?m1))
                   (known_partial_sum ?w1 ?m1))
:effect (and (known ?w1 ?u4 ?m1)))
```

Figure 3: Action that allows the malicious agent to extract information from an honest user *?u4*.

Proof Example

As discussed, the Privacy-Preserving Aggregation Protocol is meant to be secure up to n corrupted users, or $n-1$ if the malicious agent can also corrupt the server. This section provides an example of a proof of the security of the protocol for a fixed numbers of users. We have fixed the number of users to four ($n = 4$), with one server *(server1)* and one malicious agent *(mal1)*.

We have used two complete planners in our evaluation. The first one is a version of FastDownward (Helmert 2006) that runs an A^* search algorithm using the LMCUT admissible heuristic (Helmert and Domshlak 2009). The second one is SymPA (Torralba 2016), a planner that combines symbolic bidirectional search and perimeter abstraction heuristics to prove the unsolvability of planning tasks.

For the first proof, we are assuming the malicious agent does not have access to the server so it cannot corrupt it (S =False). In this case, Algorithm 1 returns that the protocol is secure up to 4 users (n) being corrupted. This means that the only case in which the malicious agent can access the information of all the users is when it can corrupt all of them, being secure when 3 or less users are corrupted. The execution time of both planners is similar. Detecting the (un)solvability of each task takes less than a second, and completing the proof takes each planner less than 5 seconds. While SymPA only tells us if the problem is solvable or not, FastDownward returns a plan in case it exists, i.e., a sequence of actions that the malicious agent could execute along the protocol to get the information about the users.

Figure 5 shows the plan produced when 4 users can be

```
(define (problem p1) (:domain protocol1)
(:objects u1 u2 u3 u4 - user
          mal1 - malicious
          server1 - server
          w1 w2 w3 w4 - weight)
(:init (round1)
       (will_process u1 w1)
       (will_process u2 w2)
       (will_process u3 w3)
       (will_process u4 w4)
       ; User 1
       (known w1 u1 u1)
       (known w2 u1 u1)
       (known w3 u1 u1)
       (known w4 u1 u1)
       ; User 2
       (known w1 u2 u2)
       (known w2 u2 u2)
       (known w3 u2 u2)
       (known w4 u2 u2)
       ; User 3
       (known w1 u3 u3)
       (known w2 u3 u3)
       (known w3 u3 u3)
       (known w4 u3 u3)
       ; User 4
       (known w1 u4 u4)
       (known w2 u4 u4)
       (known w3 u4 u4)
       (known w4 u4 u4)
       ; Malicious agent capabilities
       (can_corrupt mal1 u1)
       (can_corrupt mal1 u2)
       (can_corrupt mal1 u3)
       (can_corrupt mal1 server1))
(:goal (and (known w1 u4 mal1)
            (known w2 u4 mal1)
            (known w3 u4 mal1)
            (known w4 u4 mal1)
            ...
            (known_total_sum server1)))
```

Figure 4: Excerpt of a problem used to prove the security of Protocol 1.

corrupted. As we can see, the plan interleaves protocol actions with malicious agent actions. In this case, the malicious agent has access to all the information about the users at time step 4, since it can corrupt all the users.

The second proof is more interesting, where the malicious agent can also corrupt the server. Algorithm 1 returns that the protocol is secure up to 3 users ($n-1$) being corrupted. The execution times are the same for both planners and similar to the previous proof. Figure 6 shows the plan produced when 3 users plus the server can be corrupted.

In this case, the malicious agent can corrupt $u1, u2$ and $u3$, thus inferring all the weights but $u4$'s $w4$. The malicious agent will know the rest of weights from $u4$ after the end of Round 1 when the users exchange their weights (steps 4-7 in the plan). However, it can infer $u4$'s $w4$ by doing a subtraction. At step 13, the malicious agent corrupts the server and therefore it has access to the partial sum of $w4$, *(known-partial-sum w4 mal1)* is true. With this information

```
1. malicious_act_users u1 u2 u3 u4 w1 w2 w3 w4 mal1
2. malicious_act_users u2 u1 u3 u4 w2 w1 w3 w4 mal1
3. malicious_act_users u4 u1 u3 u2 w1 w2 w3 w4 mal1
4. malicious_act_users u3 u1 u2 u4 w3 w1 w2 w4 mal1
5. share_weights u1 u2 u3 u4 w1 w2 w3 w4
6. share_weights u2 u1 u3 u4 w2 w1 w3 w4
7. share_weights u3 u1 u2 u4 w3 w1 w2 w4
8. share_weights u4 u1 u2 u3 w4 w1 w2 w3
9. next_round_1_2 u1 u2 u3 u4 mal1
10. send_sum_to_server u4 u1 u2 u3 w4 server1
11. send_sum_to_server u3 u1 u2 u4 w3 server1
12. send_sum_to_server u2 u1 u3 u4 w2 server1
13. send_sum_to_server u1 u2 u3 u4 w1 server1
14. next_round_2_3 u1 u2 u3 u4 mal1
15. compute_sum_server w1 w2 w3 w4 server1
```

Figure 5: Plan that allows the malicious agent to access the information of all users when 4 users can be corrupted.

```
1. malicious_act_users u1 u2 u3 u4 w1 w2 w3 w4 mal1
2. malicious_act_users u2 u1 u3 u4 w2 w1 w3 w4 mal1
3. malicious_act_users u3 u1 u2 u4 w3 w1 w2 w4 mal1
4. share_weights u1 u2 u3 u4 w1 w2 w3 w4
5. share_weights u2 u1 u3 u4 w2 w1 w3 w4
6. share_weights u3 u1 u2 u4 w3 w1 w2 w4
7. share_weights u4 u1 u2 u3 w4 w1 w2 w3
8. next_round_1_2 u1 u2 u3 u4 mal1
9. send_sum_to_server u4 u1 u2 u3 w4 server1
10. send_sum_to_server u3 u1 u2 u4 w3 server1
11. send_sum_to_server u2 u1 u3 u4 w2 server1
12. send_sum_to_server u1 u2 u3 u4 w1 server1
13. malicious_act_server w1 w2 w3 w4 server1 mal1
14. malicious_user_weight_from_server u1 u2 u3 u4 w4 mal1
15. next_round_2_3 u1 u2 u3 u4 mal1
16. compute_sum_server w1 w2 w3 w4 server1
```

Figure 6: Plan that allows the malicious agent to access all users' weights when 3 users plus the server can be corrupted.

and knowing $u4$'s $w1, w2$ and $w3$, the malicious agent can infer the value of $w4$ by applying the *malicious-user-weight-from-server* action (step 14), and therefore the protocol is not secure when 3 users plus the server are corrupted.

Related Work

There has been extensive research on formal methods to prove cryptographic protocols security. Formal methods combine a language which can be used to model a cryptographic protocol and its security properties, together with a procedure to determine whether a model does indeed satisfy those properties (Meadows 2003). Note that these methods have not considered complex cryptographic protocols based on secure multiparty computation.

Model checking (Clarke, Grumberg, and Peled 1999; Baier and Katoen 2008) is one of the most employed techniques to solve this problem (Basin, Cremers, and Meadows 2018), with many tools available that can be used to verify cryptographic protocols (Blanchet 2001; Armando et al.

2005). Model checkers employ different techniques to prove that a given property (often expressed as temporal logic formulas) holds in a system. The model checker either confirms that the property holds, or provide a counterexample of a trace that violates the property. One advantage of an AP formulation to solve this problem is that, while each model checker has its own language and requirements to define a cryptographic protocol and its security properties, we can leverage on the standard PDDL language to specify them. Once we have done this, we can use any existing planner to prove that the given task is unsolvable, and therefore the protocol is secure. This is known as *Planning-as-Model-Checking paradigm* (Giunchiglia and Traverso 1999; Pistore and Traverso 2001; Edelkamp 2003; Albarghouthi, Baier, and McIlraith 2009; Bogomolov et al. 2014).

Conclusions and Future Work

We have presented a new application of automated planning, that of proving the security of cryptographic protocols. We presented how to model a specific protocol in PDDL and an algorithm to generate a proof of whether the protocol is secure or not. The paper also presented an example of the proof by using the algorithm on the described protocol. The use of such a tool can be highly beneficial for cryptography experts in order to logically prove security-related properties.

Currently we are using a classical planning approach in which we make some assumptions. For example, we are considering a single-agent view of the process as a meta-agent that has full observability of the state and all the actions that each party can execute. A more realistic formulation that would allow us to consider more complex cryptographic protocols would be to move to a multi-agent setting in which now each party has belief states (Bonet and Geffner 2014) about the state of the world and the information other parties have. This setting is closely related to epistemic logic and planning (Kominis and Geffner 2015; Muise et al. 2015; Li and Wang 2021), and we would like to explore how to formulate and proof the security of cryptographic protocols using these approaches. We would also like to compare this work with related work using model checking.

Regarding the security of multiparty computation cryptographic protocols, we used as an example a secure aggregation protocol. We plan to explore and consider the security of other protocols.

Acknowledgements

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

Albarghouthi, A.; Baier, J. A.; and McIlraith, S. A. 2009. On the use of planning technology for verification. In *In VVPS'09. Proceedings of the ICAPS Workshop on Verification & Validation of Planning & Scheduling Systems*. Citeseer.

Armando, A.; Basin, D. A.; Boichut, Y.; Chevalier, Y.; Compagna, L.; Cuéllar, J.; Drielsma, P. H.; Héam, P.; Kouchnarenko, O.; Mantovani, J.; Mödersheim, S.; von Oheimb, D.; Rusinowitch, M.; Santiago, J.; Turuani, M.; Viganò, L.; and Vigneron, L. 2005. The AVISPA tool for the automated validation of internet security protocols and applications. In *Proceedings of CAV Edinburgh, Scotland, UK, July 6-10, 2005*, volume 3576 of *Lecture Notes in Computer Science*, 281–285. Springer.

Asharov, G.; Balch, T. H.; Polychroniadou, A.; and Veloso, M. 2020. Privacy-preserving dark pools. In Seghrouchni, A. E. F.; Sukthankar, G.; An, B.; and Yorke-Smith, N., eds., *Proceedings of AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, 1747–1749. International Foundation for Autonomous Agents and Multiagent Systems.

Baier, C., and Katoen, J.-P. 2008. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Balch, T.; Diamond, B.; and Polychroniadou, A. 2020. Secretmatch: Inventory matching from fully homomorphic encryption. In *Proceedings of the 2020 ACM International Conference on AI in Finance*, ACM ICAIF '20. New York, NY, USA: Association for Computing Machinery.

Basin, D. A.; Cremers, C.; and Meadows, C. A. 2018. Model checking security protocols. In Clarke, E. M.; Henzinger, T. A.; Veith, H.; and Bloem, R., eds., *Handbook of Model Checking*. Springer. 727–762.

Ben-Or, M.; Goldwasser, S.; and Wigderson, A. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, 1–10.

Blanchet, B. 2001. An efficient cryptographic protocol verifier based on prolog rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*, 82–96. IEEE.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In Brodley, C. E., and Stone, P., eds., *Proceedings of AAAI'14, July 27 -31, 2014, Québec City, Québec, Canada*, 2228–2234. AAAI Press.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *J. Artif. Intell. Res.* 50:923–970.

Byrd, D., and Polychroniadou, A. 2020. Differentially private secure multi-party computation for federated learning in financial applications. In *Proceedings of the 2020 ACM International Conference on AI in Finance*, ICAIF '20.

Cartlidge, J.; Smart, N. P.; and Alaoui, Y. T. 2020. Multiparty computation mechanism for anonymous equity block trading: A secure implementation of turquoise plato uncross. *IACR Cryptol. ePrint Arch.* 2020:662.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 333–341. AAAI Press.

Chaum, D.; Crépeau, C.; and Damgård, I. 1987. Multiparty unconditionally secure protocols (abstract). In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, 462.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model checking*. London, Cambridge: MIT Press.

Cozzo, D.; Smart, N. P.; and Alaoui, Y. T. 2021. Secure fast evaluation of iterative methods: With an application to secure pagerank. In *Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings*, volume 12704 of *Lecture Notes in Computer Science*, 1–25. Springer.

Diffie, W., and Hellman, M. E. 1976. New directions in cryptography. *IEEE Trans. Information Theory* 22(6):644–654.

Dolev, D., and Yao, A. C. 1983. On the security of public key protocols. *IEEE Trans. Inf. Theory* 29(2):198–207.

Edelkamp, S. 2003. Limits and possibilities of pddl for model checking software. *Edelkamp, & Hoffmann (Edelkamp & Hoffmann, 2003)*.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability certificates for classical planning. In *Proceedings of ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017*, 88–97. AAAI Press.

García, J.; Flórez, J. E.; de Reyna, Á. T. A.; Borrajo, D.; López, C. L.; Olaya, A. G.; and Sáenz, J. 2013. Combining linear programming and automated planning to solve intermodal transportation problems. *Eur. J. Oper. Res.* 227(1):216–226.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In Biundo, S., and Fox, M., eds., *Recent Advances in AI Planning, 5th European Conference on Planning, ECP'99, Durham, UK, September 8-10, 1999, Proceedings*, volume 1809 of *Lecture Notes in Computer Science*, 1–20. Springer.

Goldreich, O.; Micali, S.; and Wigderson, A. 1987. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 218–229.

Goldreich, O. 2001. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.* 26:191–246.

Hoffmann, J. 2015. Simulated penetration testing: From "dijkstra" to "turing test++". In *Proceedings of ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 364–372. AAAI Press.

Kairouz, P.; McMahan, H. B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A. N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*.

Kominis, F., and Geffner, H. 2015. Beliefs in multiagent planning: From one agent to many. In Brafman, R. I.; Domshlak, C.; Haslum, P.; and Zilberstein, S., eds., *Proceedings of ICAPS 2015, Jerusalem, Israel, June 7-11, 2015*, 147–155. AAAI Press.

Konečnỳ, J.; McMahan, H. B.; Yu, F. X.; Richtárik, P.; Suresh, A. T.; and Bacon, D. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Li, Y., and Wang, Y. 2021. Planning-based knowing how: A unified approach. *Artificial Intelligence* 296:103487.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language.

Meadows, C. A. 2003. Formal methods for cryptographic protocol analysis: emerging issues and trends. *IEEE J. Sel. Areas Commun.* 21(1):44–54.

Muise, C. J.; Belle, V.; Felli, P.; McIlraith, S. A.; Miller, T.; Pearce, A. R.; and Sonenberg, L. 2015. Planning over multi-agent epistemic states: A classical planning approach. In Bonet, B., and Koenig, S., eds., *Proceedings of AAAI'15, January 25-30, 2015, Austin, Texas, USA*, 3327–3334. AAAI Press.

Nakamoto, S. 2008. Bitcoin: A peer-to-peer electronic cash system.

Pistore, M., and Traverso, P. 2001. Planning as model checking for extended goals in non-deterministic domains. In *Proceedings of IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, 479–486. Morgan Kaufmann.

Pozanco, A.; Fernández, S.; and Borrajo, D. 2021. Online modelling and planning for urban traffic control. *Expert Systems* e12693.

Sangers, A.; van Heesch, M.; Attema, T.; Veugen, T.; Wiggerman, M.; Veldsink, J.; Bloemen, O.; and Worm, D. 2019. Secure multiparty pagerank algorithm for collaborative fraud detection. In Goldberg, I., and Moore, T., eds., *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, 605–623. Springer.

Sreedharan, S.; Srivastava, S.; Smith, D. E.; and Kambhampati, S. 2019. Why can't you do that hal? explaining unsolvability of planning tasks. In Kraus, S., ed., *Proceedings*

of IJCAI 2019, Macao, China, August 10-16, 2019, 1422–1430. ijcai.org.

Torralba, A. 2016. Sympa: Symbolic perimeter abstractions for proving unsolvability. *UIPC 2016 abstracts* 8–11.

Yao, A. C. 1982. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, 160–164.