30th International Conference on
Automated Planning and Scheduling
October 19-30, 2020, Nancy, France (virtual)

FinPlan 2020

Preprints of the 1st Workshop on
Planning for
Financial Services (FinPlan)

Edited by:

Daniel Borrajo, Daniele Magazzeni and Sameena Shah

Organization

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

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

Sameena Shah
J.P.Morgan AI Research

Program Committee

Amedeo Cesta (CNR)
Giuseppe De Giacomo (Sapienza Università of Rome)
Sarah Keren (Harvard University)
Shawn Liu (S&P Global)
Fabio Mercorio (University of Milan-Bicocca)
Sebastian Sardina (RMIT University)

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, fraud detection and customer journeys.

The FinPlan workshop is the first workshop whose aims 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 three invited talks and short paper presentations. The invited talks were given by Tanveer Faruquie (Capital One), Sarah Keren (Harvard University) and Shirin Sohrabi (IBM).

Daniel Borrajo, Daniele Magazzeni and Sameena Shah
October 2020

Contents

Time your hedge with Deep Reinforcement Learning

Eric Benhamou [1,2], **David Saltiel** [1,3], **Sandrine Ungari** [4], **Abhishek Mukhopadhyay** [5]
[1] AI Square Connect, France, {eric.benhamou,david.saltiel}@aisquareconnect.com
[2]MILES, LAMSADE, Dauphine university, France
[3] LISIC, ULCO, France
[4] Societe Generale, Cross Asset Quantitative Research, UK,
[5] Societe Generale, Cross Asset Quantitative Research, France,
{sandrine.ungari,abhishek.mukhopadhyay}@sgcib.com

Abstract

Can an asset manager plan the optimal timing for her/his hedging strategies given market conditions? The standard approach based on Markowitz or other more or less sophisticated financial rules aims to find the best portfolio allocation thanks to forecasted expected returns and risk but fails to fully relate market conditions to hedging strategies decision. In contrast, Deep Reinforcement Learning (DRL) can tackle this challenge by creating a dynamic dependency between market information and hedging strategies allocation decisions. In this paper, we present a realistic and augmented DRL framework that: (i) uses additional contextual information to decide an action, (ii) has a one period lag between observations and actions to account for one day lag turnover of common asset managers to rebalance their hedge, (iii) is fully tested in terms of stability and robustness thanks to a repetitive train test method called anchored walk forward training, similar in spirit to k fold cross validation for time series and (iv) allows managing leverage of our hedging strategy. Our experiment for an augmented asset manager interested in sizing and timing his hedges shows that our approach achieves superior returns and lower risk.

Introduction

From an external point of view, the asset management (buy side) industry is a well-suited industry to apply machine learning as large amount of data are available thanks to the revolution of electronic trading and the methodical collection of data by asset managers or their acquisition from data providers. In addition, machine based decision can help reducing emotional bias and taking rational and systematic investment choices (Kahneman 2011). However, to date, the buy side industry is still largely relying on old and traditional methods to make investment decisions and in particular to choose portfolio allocation and hedging strategies. It is hardly using machine learning in investment decisions.

This is in sharp contrast with the ubiquitous usage of deep reinforcement learning (DRL) in other industries and in particular its use for solving challenging tasks like autonomous driving (Wang, Jia, and Weng 2018), learning advanced locomotion and manipulation skills from raw sensory inputs (Levine et al. 2015; 2016; Schulman et al. 2015; 2017; Lillicrap et al. 2015) or on a more conceptual side for reaching supra human level in popular games like Atari (Mnih et al. 2013), Go (Silver et al. 2016; 2017), StarCraft II (Vinyals et al. 2019), etc ...

It therefore makes sense to investigate if DRL can help help in financial planning and in particular in creating augmented asset managers. To narrow down our problem, we are specifically interested in finding hedging strategies for a risky asset. To make things concrete and more illustrative, we represent this risky asset in our experiment with the MSCI World index that captures large and mid cap securities across 23 developed financial markets. The targeted hedging strategies are on purpose different in nature and spirit. They are appropriate under distinctive market conditions. Financial planning is therefore critical for deciding the appropriate timing when to add and remove these hedging strategies.

Related works

At first, reinforcement learning was not used in portfolio allocation. Initial works focused on trying to make decisions using deep networks to forecast next period prices, (Freitas, De Souza, and Almeida 2009; Niaki and Hoseinzade 2013; Heaton, Polson, and Witte 2017). Armed with the forecast, an augmented asset manager could solve its financial planning problem to decide the optimal portfolio allocations. However, this initial usage of machine learning contains multiple caveats. First, there is no guarantee that the forecast is reliable in the near future. On the contrary, it is a stylized fact that financial markets are non stationary and exhibit regime changes (Salhi et al. 2015; Dias, Vermunt, and Ramos 2015; Zheng, Li, and Xu 2019), making the prediction exercise quite difficult and unreliable. Second, it does not target specifically the financial planning question of finding the optimal portfolio based on some reward metrics. Third, there is no consideration of online learning to adapt to changing environment as well as the incorporation of transaction costs.

A second stream of research around deep reinforcement learning has emerged to address these points (Jiang and

Liang 2016; Jiang, Xu, and Liang 2017; Liang et al. 2018; Yu et al. 2019; Wang and Zhou 2019; Liu et al. 2020; Ye et al. 2020; Li et al. 2019; Xiong et al. 2019; Benhamou et al. 2020a; 2020b). The dynamic nature of reinforcement learning makes it an obvious candidate for changing environment (Jiang and Liang 2016; Jiang, Xu, and Liang 2017; Liang et al. 2018). Transaction costs can be easily included in rules (Liang et al. 2018; Yu et al. 2019; Wang and Zhou 2019; Liu et al. 2020; Ye et al. 2020; Yu et al. 2019). However, these works, except (Ye et al. 2020) and (Benhamou et al. 2020a) rely only on time series of open high low close prices, which are known to be very noisy. Secondly, they all assume an immediate action after observing prices which is quite different from reality. Most asset managers need a one day turnaround to manage their new portfolio positions. Thirdly, except (Benhamou et al. 2020a), they rely on a single reward function and do not measure the impact of the reward function. Last but not least, they only do one train and test period, never testing for model stability.

Contributions

Our contributions are fourfold:

- **The addition of contextual information.** Using only past information is not sufficient to learn in a noisy and fast changing environment. The addition of contextual information improves results significantly. Technically, we create two sub-networks: one fed with direct observations (past prices and standard deviation) and another one with contextual information (level of risk aversion in financial markets, early warning indicators for future recession, corporate earnings etc...).
- **One day lag between price observation and action.** We assume that prices are observed at time t but action only occurs at time $t+1$, to be consistent with reality. This one day lag makes the RL problem more realistic but also more challenging.
- **The walk-forward procedure.** Because of the non stationarity nature of time dependent data and especially financial data, it is crucial to test DRL models stability. We present a new methodology in DRL model evaluation referred to as walk forward analysis that iteratively trains and tests the model on extending data-set. This can be seen as the analogy of cross validation for time series. This allows validating that selected hyper parameters work well over time and that the resulting model is stable over time.
- **Model leverage**. Not only do we do a multi inputs network, we also do a multi outputs network to compute at the same time the percentage in each hedging strategy and the overall leverage. This is a nice feature of this DRL model as it incorporates by design a leverage mechanism. To make sure the leverage is in line with the asset manager objective, we cap the leverage to the maximum authorized leverage, which is in our case 3. This byproduct of the method is another key difference with standard financial methods like Markwitz that do not care about leverage and only give a percentage for the hedging portfolio allocation.

Background and mathematical formulation

In standard reinforcement learning, models are based on Markov Decision Process (MDP) (Sutton and Barto 2018). A Markov decision process is defined as a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, p, r)$ where:

- $\mathcal{X}$ is the state space,
- $\mathcal{A}$ is the action space,
- $p(y|x,a)$ is the transition probability such that $p(y|x,a) = \mathbb{P}(x_{t+1} = y | x_t = x, a_t = a)$,
- $r(x,a,y)$ is the reward of transition (x, a, y).

MDP assumes that the we know all the states of the environment and have all the information to make the optimal decision in every state. The Markov property in addition implies that knowing the current state is sufficient.
From a practical standpoint, the general RL setting is modified by taking a pseudo state formed with a set of past observations $(o_{t-n}, o_{t-n-1}, \ldots, o_{t-1}, o_t)$. In practice to avoid large dimension and the curse of dimension, it is useful to reduce this set and take only a subset of these past observations with $j < n$ past observations, such that $0 < i_1 < \ldots < i_j$ and $i_k \in \mathbb{N}$ is an integer. The set $\delta_1 = (0, i_1, \ldots, i_j)$ is called the observation lags. In our experiment we typically use lag periods like (0, 1, 2, 3, 4, 20, 60) for daily data, where $(0,1,2,3,4)$ provides last week observation, 20 is for the one-month ago observation (as there is approximately 20 business days in a month) and 60 the three-month ago observation.

Observations

Regular observations There are two types of observations: regular and contextual information. Regular observations are data directly linked to the problem to solve. In the case of a trading framework, regular observations are past prices observed over a lag period $\delta = (0 < i_1 < \ldots < i_j)$. To renormalize data, we rather use past returns computed as $r_t = \frac{p_t^k}{p_{t-1}^k} - 1$ where p_t^k is the price at time t of the asset k. To give information about regime changes, our trading agent receives also empirical standard deviation computed over a sliding estimation window denoted by d as follows $\sigma_t^k = \sqrt{\frac{1}{d}\sum_{u=t-d+1}^{t}(r_u - \mu)^2}$, where the empirical mean μ is computed as $\mu = \frac{1}{d}\sum_{u=t-d+1}^{t} r_u$. Hence our regular observations is a three dimensional tensor $A_t = \left[A_t^1, A_t^2\right]$

$$\text{with} \quad A_t^1 = \begin{pmatrix} r_{t-i_j}^1 \ \ldots \ r_t^1 \\ \ldots \ \ldots \ \ldots \\ r_{t-i_j}^m \ \ldots \ r_t^m \end{pmatrix}, \ A_t^2 = \begin{pmatrix} \sigma_{t-i_j}^1 \ \ldots \ \sigma_t^1 \\ \ldots \ \ldots \ \ldots \\ \sigma_{t-i_j}^m \ \ldots \ \sigma_t^m \end{pmatrix}$$

This setting with two layers (past returns and past volatilities) is quite different from the one presented in (Jiang and Liang 2016; Jiang, Xu, and Liang 2017; Liang et al. 2018) that uses different layers representing closing, open high low prices. There are various remarks to be made. First, high

low information does not make sense for portfolio strategies that are only evaluated daily, which is the case of all the funds. Secondly, open high low prices tend to be highly correlated creating some noise in the inputs. Third, the concept of volatility is crucial to detect regime change and is surprisingly absent from these works as well as from other works like (Yu et al. 2019; Wang and Zhou 2019; Liu et al. 2020; Ye et al. 2020; Li et al. 2019; Xiong et al. 2019).

Context observation Contextual observations are additional information that provide intuition about current context. For our asset manager, they are other financial data not directly linked to its portfolio assumed to have some predictive power for portfolio assets. Contextual observations are stored in a 2D matrix denoted by C_t with stacked past p individual contextual observations. Among these observations, we have the maximum and minimum portfolio strategies return and the maximum portfolio strategies volatility. The latter information is like for regular observations motivated by the stylized fact that standard deviations are useful features to detect crisis. The contextual state writes as $C^t = \begin{pmatrix} c_t^1 \ ... \ c_{t-i_k}^1 \\ ... \ ... \ ... \\ c_t^p \ .... \ c_{t-i_k}^p \end{pmatrix}$. The matrix nature of contextual states C_t implies in particular that we will use 1D convolutions should we use convolutional layers. All in all, observations that are augmented observations, write as $O_t = [A_t, C_t]$, with $A_t = [A_t^1, A_t^2]$ that will feed the two sub-networks of our global network.

Action

In our deep reinforcement learning the augmented asset manager trading agent needs to decide at each period in which hedging strategy it invests. The augmented asset manager can invest in l strategies that can be simple strategies or strategies that are also done by asset management agent. To cope with reality, the agent will only be able to act after one period. This is because asset managers have a one day turn around to change their position. We will see on experiments that this one day turnaround lag makes a big difference in results. As it has access to l potential hedging strategies, the output is a l dimension vector that provides how much it invest in each hedging strategy. For our deep network, this means that the last layer is a softmax layer to ensure that portfolio weights are between 0 and 100% and sum to 1, denoted by $(p_t^1, ..., p_t^l)$. In addition, to include leverage, our deep network has a second output which is the overall leverage that is between 0 and a maximum leverage value (in our experiment 3), denoted by lvg_t. Hence the final allocation is given by $lvg_t \times (p_t^1, ..., p_t^l)$.

Reward

There are multiple choices for our reward and it's a key point for the asset manager to decide the reward corresponding to his her risk profile.

- A straightforward reward function is to compute the final net performance of the combination of our portfolio computed as the value of our portfolio at the last train date t_T over the initial value of the portfolio t_0 minus one: $\frac{P_{t_T}}{P_{t_0}} - 1$.
- Another natural reward function is to compute the Sharpe ratio. There are various ways to compute Sharpe ratio and we take explicitly the annualized Sharpe ratio. This annualized Sharpe ratio computed from daily data is defined as the ratio of the annualized return over the annualized volatility μ/σ. The intuition of the Sharpe ratio is to account for risk when comparing returns with risk is represented by volatility.
- The last reward we are interested in is the Sortino ratio. This metric is a variation of the Sharpe ratio where the risk is computed by the downside standard deviation whose definition is to compute the standard deviation only on negative daily returns $(\tilde{r}_t)_{t=0..T}$. Hence the downside standard deviation is computed by $\sqrt{250} \times \text{StdDev}[(\tilde{r}_t)_{t=0..T}]$.

Convolutional network

The similarities with image recognition (where pixels are stored in 3 different matrices representing red, green and blue image) enable us using convolution networks for our deep neural network. The analogy goes even further as it is well known in image recognition that convolutional networks achieve strong performances thanks to their capacity to extract meaningful features and to have very limited parameters hence avoiding over-fitting. Indeed, convolution allows us to extract features; blindly weighting locally the variables over the tensor. There is however something to notice. We use a convolution layer with a convolution window or kernel with a single row and a resulting vertical stride of 1. This particularity enables us to avoid mixing data from different strategies. We only mix data of the same strategies but for different observation dates. Recall that in convolution network, the stride parameter controls how the filter convolves around our input. Likewise the size of the window also referred to as the kernel size controls how the filter applies to data. Thus, a kernel with a row of 1 and a stride with a row of 1 allows us to detect the vertical (temporal) relation for each strategy by shifting one unit at a time, without mixing any data from different strategies. This concept is illustrated in figure 1. Because of this peculiarity, we can interpret our 2-D convolution as an iteration over a 1-D convolution network for each variable.

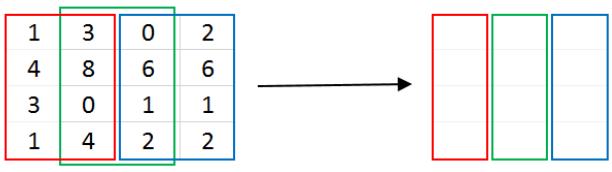

Figure 1: 2-D Convolution with stride of 1

Multi inputs and outputs

We display in figure 2 the architecture of our network. Because we feed our network with both data from the strategies to select but also contextual information, our network is a multiple inputs network.

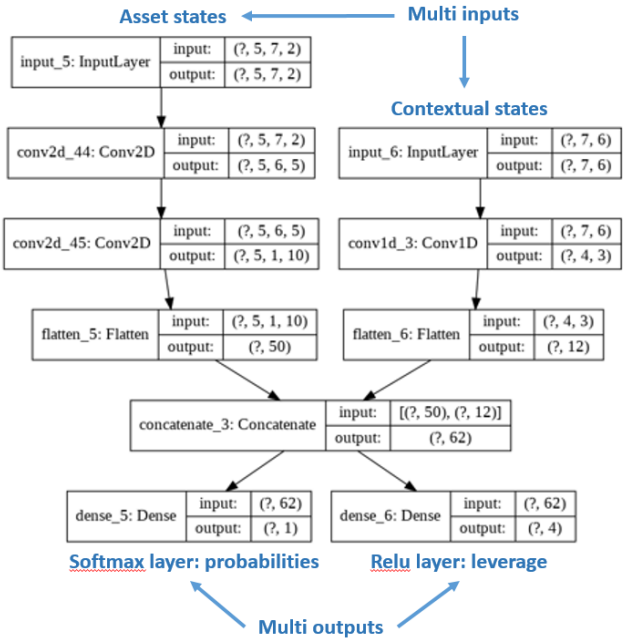

Figure 2: network architecture obtained via tensorflow plot-model function. Our network is very different from standard DRL networks that have single inputs and outputs. Contextual information introduces a second input while the leverage adds a second output

Additionally, as we want from these inputs to provide not only percentage in the different hedging strategies (with a softmax activation of a dense layer) but also the overall leverage (with a dense layer with one single ouput neurons), we also have a multi outputs network. Additional hyperparameters that are used in the network as L2 regularization with a coefficient of 1e-8.

Adversarial Policy Gradient

To learn the parameters of our network depicted in 2, we use a modified policy gradient algorithm called adversarial as we introduce noise in the data as suggested in (Liang et al. 2018).. The idea of introducing noise in the data is to have some randomness in each training to make it more robust. This is somehow similar to drop out in deep networks where we randomly pertubate the network by randomly removing some neurons to make it more robust and less prone to overfitting. A policy is a mapping from the observation space to the action space, $\pi : \mathcal{O} \rightarrow \mathcal{A}$. To achieve this, a policy is specified by a deep network with a set of parameters $\vec{\theta}$. The action is a vector function of the observation given the parameters: $\vec{a}_t = \pi_{\vec{\theta}}(\boldsymbol{o}_t)$. The performance metric of $\pi_{\vec{\theta}}$ for time interval $[0, t]$ is defined as the corresponding total reward function of the interval $J_{[0,t]}(\pi_{\vec{\theta}}) = R\left(\vec{o}_1, \pi_{\vec{\theta}}(o_1), \cdots, \vec{o}_t, \pi_{\vec{\theta}}(o_t), \vec{o}_{t+1}\right)$. After random initialization, the parameters are continuously updated along the gradient direction with a learning rate λ: $\vec{\theta} \longrightarrow \vec{\theta} + \lambda \nabla_{\vec{\theta}} J_{[0,t]}(\pi_{\vec{\theta}})$. The gradient ascent optimization is done with standard Adam (short for Adaptive Moment Estimation) optimizer to have the benefit of adaptive gradient descent with root mean square propagation (Kingma and Ba 2014). The whole process is summarized in algorithm 1.

Algorithm 1 Adversarial Policy Gradient

Input: initial policy parameters θ, empty replay buffer $\mathcal{D}$
repeat
  reset replay buffer
  while not terminal **do**
    Observe observation o and select action $a = \pi_\theta(o)$ with probability p and random action with probability $1 - p$,
    Execute a in the environment
    Observe next observation o', reward r, and done signal d to indicate whether o' is terminal
    apply noise to next observation o'
    store (o, a, o') in replay buffer $\mathcal{D}$
    if Terminal **then**
      for however many updates in $\mathcal{D}$ **do**
        compute final reward R
      end for
      update network parameter with Adam gradient ascent $\vec{\theta} \longrightarrow \vec{\theta} + \lambda \nabla_{\vec{\theta}} J_{[0,t]}(\pi_{\vec{\theta}})$
    end if
  end while
until convergence

In our gradient ascent, we use a learning rate of 0.01, an adversarial Gaussian noise with a standard deviation of 0.002. We do up to 500 maximum iterations with an early stop condition if on the train set, there is no improvement over the last 50 iterations.

Walk forward analysis

In machine learning, the standard approach is to do k-fold cross validation as shown in figure 3. This approach breaks the chronology of data and potentially uses past data in the test set. Rather, we can take sliding test set and take past data as training data as show in the two sub-figures on the right of figure 4. To ensure some stability, we favor to add incrementally new data in the training set, at each new step. This method is sometimes referred to as anchored walk forward as we have anchored training data. The negative effect of using extending training data set is to adapt slowly to new information. To our experience, because we do not have so much data to train our DRL model, we use anchored walk forward to make sure we have enough training data. Last but not least, as the test set is always after the train set, walk forward analysis gives less steps compared to cross validation. In practice for our data set, we train our models from 2000 to end of 2006 (to have at least seven years of data) and use a repetitive test period of one year.

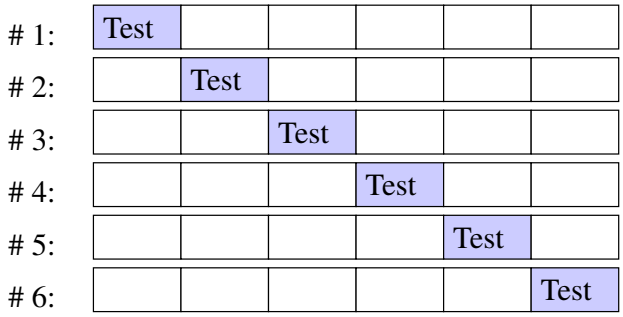

Figure 3: k-fold cross validation

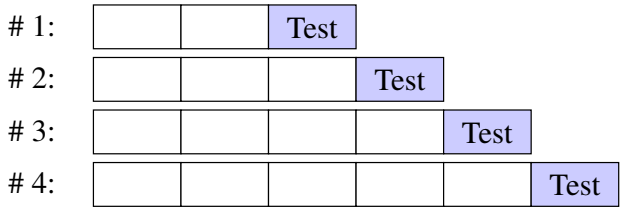

Figure 4: anchored walk forward

Experiments

Goal of the experiment

We are interested in planing a hedging strategy for a risky asset. The experiment is using daily data from 01/05/2000 to 19/06/2020. The risky asset is the MSCI world index. We choose this index because it is a good proxy for a wide range of asset manager portfolios. The hedging strategies are 4 SG-CIB proprietary systematic strategies further described below .

Data-set description

Systematic strategies are similar to asset managers that invest in financial markets according to an adaptive, predefined trading rule. Here, we use 4 SG CIB proprietary 'hedging strategies', that tend to perform when stock markets are down:

- Directional hedges - react to small negative return in equities,
- Gap risk hedges - perform well in sudden market crashes,
- Proxy hedges - tend to perform in some market configurations, like for example when highly indebted stocks under-perform other stocks,
- Duration hedges - invest in bond market, a classical diversifier to equity risk in finance.

The underlying financial instruments vary from put options, listed futures, single stocks, to government bonds. Some of those strategies are akin to an insurance contract and bear a negative cost over the long run. The challenge consists in balancing cost versus benefits.

In practice, asset managers have to decide how much of these hedging strategies are needed on top of an existing portfolio to achieve a better risk reward. The decision making process is often based on contextual information, such as the economic and geopolitical environment, the level of risk aversion among investors and other correlation regimes. The contextual information is modeled by a large range of features :

- the level of risk aversion in financial markets, or market sentiment, measured as an indicator varying between 0 for maximum risk aversion and 1 for maximum risk appetite,
- the bond/equity historical correlation, a classical ex-post measure of the diversification benefits of a duration hedge, measured on a 1-month, 3-month and 1-year rolling window,
- The credit spreads of global corporate - investment grade, high yield, in Europe and in the US - known to be an early indicator of potential economic tensions,
- The equity implied volatility, a measure of the 'fear factor' in financial market,
- The spread between the yield of Italian government bonds and the German government bond, a measure of potential tensions in the European Union,
- The US Treasury slope, a classical early indicator for US recession,
- And some more financial variables, often used as a gauge for global trade and activity: the dollar, the level of rates in the US, the estimated earnings per shares (EPS).

A cross validation step selects the most relevant features. In the present case, the first three features are selected. The rebalancing of strategies in the portfolio comes with transaction costs, that can be quite high since hedges use options. Transactions costs are like frictions in physical systems. They are taken into account dynamically to penalise solutions with a high turnover rate.

Evaluation metrics

Asset managers use a wide range of metrics to evaluate the success of their investment decision. For a thorough review of those metrics, see for example (Cogneau and Hübner 2009). The metrics we are interested in for our hedging problem are listed below:

- annualized return defined as the average annualized compounded return,
- annualized daily based Sharpe ratio defined as the ratio of the annualized return over the annualized daily based volatility μ/σ,
- Sortino ratio computed as the ratio of the annualized return over the downside standard deviation,
- maximum drawdown (max DD) computed as the maximum of all daily drawdowns. The daily drawdown is computed as the ratio of the difference between the running maximum of the portfolio value ($RM_T = \max_{t=0..T}(P_t)$) and the portfolio value over the running maximum of the portfolio value. Hence $DD_T = (RM_T - P_T)/RM_T$ and $MDD_T = \max_{t=0..T}(DD_t)$. It is the maximum loss in return that an investor will incur if she/he invested at the worst time (at peak).

Baseline

Pure risky asset This first evaluation is to compare our portfolio composed only of the risky asset (in our case, the MSCI world index) with the one augmented by the trading agent and composed of the risky asset and the hedging overlay. If our agent is successful in identifying good hedging strategies, it should improve the overall portfolio and have a better performance than the risky asset.

Markowitz In Markowitz theory (Markowitz 1952), risk is represented by the variance of the portfolio. Hence the Markowitz portfolio consists in maximizing the expected return for a given level of risk, represented by a given variance. Using dual optimization, this is also equivalent to minimize variance for a given expected return, which is solved by standard quadratic programming optimization. Recall that we have l possible strategies and we want to find the best allocation according to the Sharpe ratio. Let $w = (w_1, ..., w_l)$ be the allocation weights with $1 \geq w_i \geq 0$ for $i = 0...l$, which is summarized by $1 \geq w \geq 0$, with the additional constraints that these weights sum to 1: $\sum_{i=1}^{l} w_i = 1$.
Let $\mu = (\mu_1, ..., \mu_l)^T$ be the expected returns for our l strategies and Σ the matrix of variance covariances of the l strategies' returns. Let r_{min} be the minimum expected return. The Markowitz optimization problem to solve that is done by standard quadratic programming is the following:

$$\begin{aligned} &\text{Minimize} \quad w^T \Sigma w \\ &\text{subject to} \quad \mu^T w \geq r_{min}, \sum_{i=1...l} w_i = 1, w \geq 0 \end{aligned}$$

The Markowitz portfolio is a good benchmark very often used in portfolio theory as it allows investors to construct more efficient portfolios by controlling the variance of their strategies. One of the famous critic of this theory is that it controls the variance (and then the standard deviation) of the portfolio but it doesn't allow controlling a better risk indicator which is the downside standard deviation (representing the potential loss that may arise from risk compared to a minimum acceptable return). Another limitation of this theory relies on the fact that it works under the assumption that investors are risk-averse. In other words, an investor prefers a portfolio with less risk for a given level of return and will only take on high-risk investments if he can expect a larger reward.

Follow the winner This is a simple strategy that consists in selecting the hedging strategy that was the best performer in the past year. If there is some persistence over time of the hedging strategies' performance, this simple methodology works well. It replicates standard investors behavior that tends to select strategies that performed well in the past.

Follow the loser As it name stands for, follow the loser is exactly the opposite of follow the winner. It assumes that there is some mean reversion in strategies' performance, meaning that strategies tend to perform equally well on long term and mean revert around their trend. Hence if a strategy did not perform well in the past, and if there is mean reversion, there is a lot of chance that this strategy will recover with its pairs.

Results and discussion

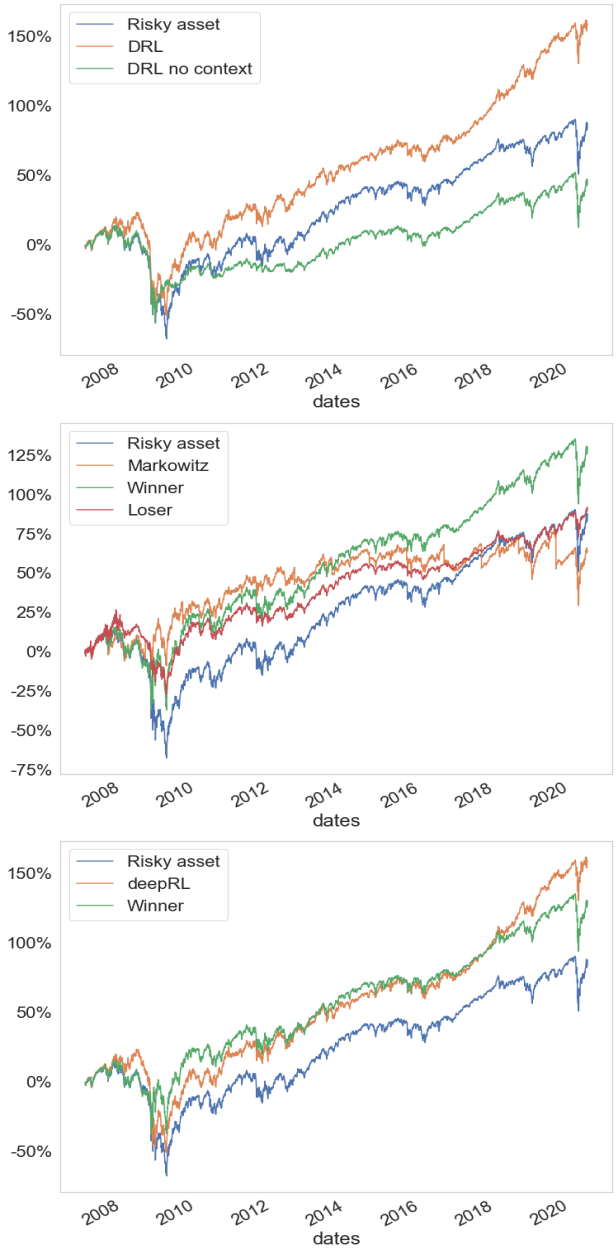

Figure 5: performance of all models

We compare the performance of the following 5 models: DRL model based on convolutional networks with contextual states (Sentiment indicator, 6 month correlation between equity and bonds and credit main index), same DRL model without contextual states, follow the winner, follow the loser and Markowitz portfolio. The resulting graphics are displayed in figure 5 with the risky asset position alone

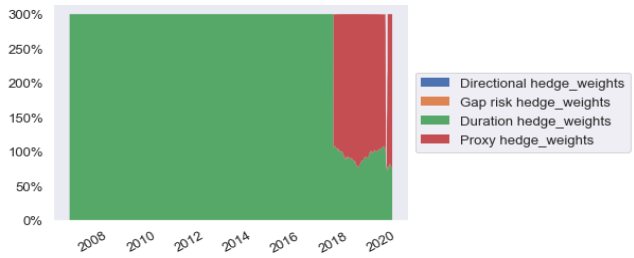

Figure 6: DRL weights

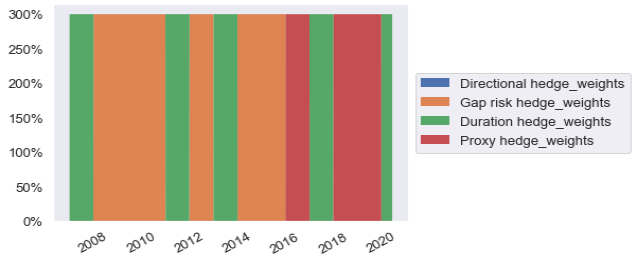

Figure 7: Follow the winner weights

Figure 8: Markowitz weights

Table 1: Models comparison over 3 and 5 years

	3 Years			
	return	Sortino	Sharpe	max DD
Risky asset	10.27%	0.34	0.38	- 0.34
DRL	**22.45%**	**1.18**	**1.17**	-0.27
Winner	13.19%	0.66	0.72	-0.35
Loser	9.30%	0.89	0.89	**-0.15**
DRL no context	8.11%	0.42	0.47	-0.34
Markowitz	-0.31%	-0.01	-0.01	-0.41
	5 Years			
	return	Sortino	Sharpe	max DD
Risky asset	9.16%	0.54	0.57	- 0.34
DRL	**16.42%**	**0.98**	**0.96**	-0.27
Winner	10.84%	0.65	0.68	-0.35
Loser	7.04%	0.78	0.76	**-0.15**
DRL no context	6.87%	0.44	0.47	-0.34
Markowitz	-0.07%	-0.00	-0.00	-0.41

in blue and the other models in orange, green and red. To make figures readable, we first show the two DRL models, with the risky asset and clearly see the impact of contextual information as the DRL model (in orange) is well above the green curve (the same model without contextual information) and is also well above the risky asset position alone (the blue curve). We then plot more traditional models like Markowitz, follow the Winner (entitled for space reason Winner) and follow the Loser (entitled for the same reason Loser). We finally plot the two best performers: the DRL and the Follow the Winner model, emphasizing that the difference between DRL and Follow the Winner is mostly in years 2018 to 2020 that exhibit regime changes, with in particular the recent Covid crisis.

Out of these 5 models, only DRL and Follow the winner are able to provide significant net performance increase compared to the risky asset alone thanks to an efficient hedging strategy over the 2007 to 2020 period. The DRL model is in addition able to better adapt to the Covid crisis and to have better efficiency in net return but also Sharpe and Sortino ratios over 3 and 5 years as shown in table 1. In addition, on the last graphic of figure 5, we can remark that the DRL model has a tendency to move away from the blue curve (the risky asset) continuously and increasingly whereas the follow the winner model has moved away from the blue curve in 2015 and 2016 and tends to remain in parallel after this period, indicating that there is no continuous improvement of the model. The growing divergence of the DRL from the blue curve is a positive sign of its regular performance whci is illustrated in numbers in table 1.

Moreover, when comparing the weights obtained by the different models (figures 6, 7, and 8), we see that the bad performance of Markowitz can be a consequence of its diversification as it takes each year a non null position in the four hedging strategies and tends to change this allocation quite frequently. The rapid change of allocation is a sign of unstability of this method (which is a well known drawback of Markowitz).

In contrast, DRL and Follow the winner models tends to choose only one or two strategies, in a stock picking manner. DRL model tends to choose mostly duration hedge and is able to dynamically adapt its behavior over the last 3 years and to better manage the Covid crisis with a mix allocation between duration and proxy hedge.

In terms of the smallest maximum drawdown, the follow the loser model is able to significantly reduce maximum drawdown but at the price of a lower return, Sharpe and Sortino ratios. Removing contextual information deteriorates model performances significantly and is illustrated by the difference in term of return, Sharpe, Sortino ratio and maximum drawdown between the DRL and the DRL no context model. Last but not least, Markowitz model is not able to adapt to the new regime change of 2015 onwards despite its good performance from 2007 to 2015. It is the worst performer over the last 3 and 5 years because of this lack of adaptation.

For all models, we use the walk forward analysis as described earlier. Hence, we start training the models from 2000 to end of 2006 and use the best model on the test set

in 2007. We then train the model from 2000 to end of 2007 and use the best model on the test set in 2008 and etc ... In total, we do 14 training (from 2007 to 2020). This process ensures that we detect models that are unstable overtime and is similar in spirit to delayed online training. We also provide in table 2 different configurations (adversarial training, use of context, and use of day lag), which leads to a total of 16 models. The frist 8 models are the ones with a daylag sorted in order of decreasing performance. The best model is the one with a reward in net profit, adversarial training, use of context information with a total performance of 81.8 %. We also provide the corresponding same models but with no day lag (model 9 to 16). These models are theoreticall and not considered as they do not cope with reality.

Table 2: Model comparison based on reward function, adversarial training (noise in data) and use of contextual state

#	reward	adversarial?	context?	day lag	performance
1	**Net_Profit**	**Yes**	**Yes**	**Yes**	**81.8%**
2	net profit	No	Yes	Yes	75.2%
3	Sortino	No	Yes	Yes	26.5%
4	Sortino	Yes	Yes	Yes	26.3%
5	Sortino	Yes	No	Yes	-16.7%
6	net profit	Yes	No	Yes	-29.5%
7	Sortino	No	No	Yes	-45.0%
8	net profit	No	No	Yes	-47.7%
9	net profit	Yes	Yes	No	193.8%
10	net profit	No	Yes	No	152.3%
11	Sortino	No	Yes	No	45.3%
12	Sortino	Yes	Yes	No	29.3%
13	Sortino	Yes	No	No	16.9%
14	net profit	Yes	No	No	13.9%
15	Sortino	No	No	No	10.6%
16	net profit	No	No	No	8.6%

Impact of context

In table 2, we provide a list of 16 models based on the following choices: the choice of the reward function (net profit or Sortino), the use of adversarial training with noise in data or not, the use of contextual states, and the use of day lag between observations and actions.

We see that the best DRL model with the day-lag turnover constraint is the one using convolutional networks, adversarial training, contextual states and net profit reward function. These 4 parameters are meaningful for our DRL model and change model performance substantially as illustrated by the table with a difference between model 1 (the best model) and model 8 (the worst model) of 129.5 % (=81.8 % - (-47.7 %)).

To measure the impact of the contextual information for our best model, we can measure it simply by doing the difference between model 1 and model 6 (as there are the same model except the presence or absence of contextual information). We find a significant impact as it accounts for 111.4 % (=81.8 % - (-29.5 %)). It is quite intuitive that adding a context should improve the model as we provide more meaningful information to the model.

Impact of one day lag

Our model accounts for the fact that asset managers cannot immediately change their position at the close of the financial markets. It is easy to measure the impact of the one day lag as we simply need to take the difference of performance between model 9 and model 1. We find an impact of the one day lag of 112 % (= 193.8 % - 81.8%). This is like for contextual information substantial. It is not surprising that a delayed action (with one period lag) after observation makes the learning process more challenging for the DRL agent as influence of variables tends to decrease with time. Surprisingly, this salient modeling characteristic is ignored in existing literature (Jiang, Xu, and Liang 2017; Liang et al. 2018; Yu et al. 2019; Wang and Zhou 2019; Liu et al. 2020; Ye et al. 2020; Li et al. 2019).

Future work

As nice as this work is, there is room for improvement as we have only tested a few possible hyper-parameters for our convolutional networks and could play with more layers, other design choice like combination of max pooling layers (like in image recognition) and ways to create more predictive contextual information.

Conclusion

In this paper, we address the challenging task of financial planning in a noisy and self adapting environment with sequential, non-stationary and non-homogeneous observations. Our approach is based on deep reinforcement learning using contextual information thanks to a second subnetwork. We also show that the additional constraint of a delayed action following observations has a substantial impact that should not be overlooked. We introduce the novel concept of walk forward analysis to test the robustness of the deep RL model. This is very important for regime changing environments that cannot be evaluated with a simple train validation test procedure, neither a k-fold cross validation as it ignores the strong chronological feature of observations.

For our trading agent, we take not only past performances of portfolio strategies over different rolling period, but also standard deviations to provide predictive variables for regime changes. Augmented states with contextual information make a big difference in the model and help the agent learning more efficiently in a noisy environment. On experiment, contextual based approach over-performs baseline methods like Markowitz or naive follow the winner and follow the loser. Last but not least, it is quite important to fine tune the numerous hyper-parameters of the contextual based DRL model, namely the various lags (lags period for the sub network fed by portfolio strategies past returns, lags period for common contextual features referred to as the common features in the paper), standard deviation period, learning rate, etc...

Despite the efficiency of contextual based DRL models, there is room for improvement. Other information like news could be incorporated to continue increasing model performance. For large stocks, like tech stocks, sentiment information based on social media activity could also be relevant.

Acknowledgments. We would like to thank Beatrice Guez and Marc Pantic for meaningful remarks while working on this project. The views contained in this document are those of the authors and do not necessarily reflect the ones of SG CIB.

References

Benhamou, E.; Saltiel, D.; Ohana, J.-J.; and Atif, J. 2020a. Detecting and adapting to crisis pattern with context based deep reinforcement learning. *ICPR 2020*.

Benhamou, E.; Saltiel, D.; Ungari, S.; and Mukhopadhyay, A. 2020b. Bridging the gap between markowitz planning and deep reinforcement learning. *ICAPS PRL workshop*.

Cogneau, P., and Hübner, G. 2009. The 101 ways to measure portfolio performance. *SSRN Electronic Journal*.

Dias, J.; Vermunt, J.; and Ramos, S. 2015. Clustering financial time series: New insights from an extended hidden markov model. *European Journal of Operational Research* 243:852–864.

Freitas, F.; De Souza, A.; and Almeida, A. 2009. Prediction-based portfolio optimization model using neural networks. *Neurocomputing* 72:2155–2170.

Heaton, J. B.; Polson, N. G.; and Witte, J. H. 2017. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry* 33(1):3–12.

Jiang, Z., and Liang, J. 2016. Cryptocurrency Portfolio Management with Deep Reinforcement Learning. *arXiv e-prints*.

Jiang, Z.; Xu, D.; and Liang, J. 2017. Reinforcement learning framework for the financial portfolio management problem. *arXiv*.

Kahneman, D. 2011. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization.

Levine, S.; Finn, C.; Darrell, T.; and Abbeel, P. 2015. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17.

Levine, S.; Pastor, P.; Krizhevsky, A.; and Quillen, D. 2016. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*.

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

Liang et al. 2018. Adversarial deep reinforcement learning in portfolio management.

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

Liu, Y.; Liu, Q.; Zhao, H.; Pan, Z.; and Liu, C. 2020. Adaptive quantitative trading: an imitative deep reinforcement learning approach. In *AAAI*.

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

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*.

Niaki, S., and Hoseinzade, S. 2013. Forecasting s&p 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International* 9.

Salhi, K.; Deaconu, M.; Lejay, A.; Champagnat, N.; and Navet, N. 2015. Regime switching model for financial data: empirical risk analysis. *Physica A: Statistical Mechanics and its Applications* 461.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.; and Abbeel, P. 2015. Trust region policy optimization. In *ICML*.

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

Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529:484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of go without human knowledge. *Nature* 550:354–.

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

Vinyals, O.; Babuschkin, I.; Czarnecki, W.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.; Powell, R.; Ewalds, T.; Georgiev, P.; Oh, J.; Horgan, D.; Kroiss, M.; Danihelka, I.; Huang, A.; Sifre, L.; Cai, T.; Agapiou, J.; Jaderberg, M.; and Silver, D. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575.

Wang, H., and Zhou, X. Y. 2019. Continuous-Time Mean-Variance Portfolio Selection: A Reinforcement Learning Framework. *arXiv e-prints*.

Wang, S.; Jia, D.; and Weng, X. 2018. Deep reinforcement learning for autonomous driving. *ArXiv* abs/1811.11329.

Xiong, Z.; Liu, X.-Y.; Zhong, S.; Yang, H.; and Walid, A. 2019. Practical deep reinforcement learning approach for stock trading.

Ye, Y.; Pei, H.; Wang, B.; Chen, P.-Y.; Zhu, Y.; Xiao, J.; and Li, B. 2020. Reinforcement-learning based portfolio management with augmented asset movement prediction states. In *AAAI*.

Yu, P.; Lee, J. S.; Kulyatin, I.; Shi, Z.; and Dasgupta, S. 2019. Model-based deep reinforcement learning for financial portfolio optimization. *RWSDM Workshop, ICML 2019*.

Zheng, K.; Li, Y.; and Xu, W. 2019. Regime switching model estimation: spectral clustering hidden markov model. *Annals of Operations Research*.

Goal Recognition via Model-based and Model-free Techniques

Daniel Borrajo*, Sriram Gopalakrishnan† and Vamsi K. Potluru‡

Abstract

Goal recognition aims at predicting human intentions from a trace of observations. This ability allows people or organizations to anticipate future actions and intervene in a positive (collaborative) or negative (adversarial) way. Goal recognition has been successfully used in many domains, but it has been seldom been used by financial institutions. We claim the techniques are ripe for its wide use in finance-related tasks. The main two approaches to perform goal recognition are model-based (planning-based) and model-free (learning-based). In this paper, we adapt state-of-the-art learning techniques to goal recognition, and compare model-based and model-free approaches in different domains. We analyze the experimental data to understand the trade-offs of using both types of methods. The experiments show that planning-based approaches are ready for some goal-recognition finance tasks.

Introduction

Humans interact with the world based on their inner motivations (goals) by performing actions. Those actions might be observable by financial institutions. In turn, financial institutions might log all these observed actions for better understanding human behavior. Examples of such interactions are investment operations (buying or selling options), account-related activities (creating accounts, making transactions, withdrawing money), digital interactions (utilizing the bank's web or mobile app for configuring alerts, or applying for a new credit card), or even illicit operations (such as fraud or money laundering). Once human behavior can be better understood, financial institutions can improve their processes allowing them to deepen the relationship with clients, offering targeted services (marketing), handling complaints-related interactions (operations), or performing fraud or money laundering investigations (compliance) (Borrajo, Veloso, and Shah 2020).

*J.P.Morgan AI Research, New York, NY (USA). On leave from Universidad Carlos III de Madrid. The position at the lab is as a consultant. daniel.borrajo@jpmchase.com

†Arizona State University. sgopal28@asu.edu

‡J.P.Morgan AI Research, New York, NY (USA). vamsi.k.potluru@jpmchase.com

The field of goal recognition lies at the crux of understanding human behavior (Carberry 2001; Kautz and Allen 1986; Sukthankar et al. 2004). Given a trace of observations of some agent taking actions in the environment, goal recognition techniques try to infer the agents' goals. Similarly, plan recognition will try to infer the next actions (plan) that the agent will execute (Kautz and Allen 1986), and activity recognition will infer the current activity of the agent (Ortiz-Laguna, Ángel García-Olaya, and Borrajo 2013). These three tasks are highly related, though the objective of the recognition problem is different; agents come up with goals they would like to achieve for which they generate plans composed of sequences of actions (Ghallab, Nau, and Traverso 2004). However, these tasks are not exactly equivalent. For instance, there might be more than one plan to achieve a goal and more than one goal achievable by a plan, so even if one recognizes what plan an agent is pursuing, that might not uniquely identify the goals the agent is trying to achieve, and vice versa. In this paper we focus on goal recognition.

Goal recognition by an agent aims at inferring the goals of another agent from observation. A convenient way to model this process has been in terms of using Bayesian models that compute a posterior probability of some goals/plans based on prior probabilities and new observations (Bui, Venkatesh, and West 2002; Pynadath and Wellman 1995; Ramírez and Geffner 2010). Other alternatives are based on creating plan libraries and matching new observations with those libraries (Kautz and Allen 1986; Avrahami-Zilberbrand and Kaminka 2005). The latter kind of techniques require careful curation of those libraries, which can result in a time-consuming effort and is limited by the set of libraries used. Some works have automated the construction of those libraries (Mooney 1990). However, often the search in the space of plans can result in a huge computational effort due to the exponential number of potential plans when the size of the action space increases (Kautz 1987).

More recently, automated planning techniques have been used to infer an agent's goals (Höller et al. 2018; Ramírez and Geffner 2010; Pereira, Oren, and Meneguzzi 2020; E-Martín, R-Moreno, and Smith 2015;

Sohrabi, Riabov, and Udrea 2016; Vered and Kaminka 2017). They replace the plan libraries by using a domain model. The techniques based on automated planning provide good performance, are domain-independent and are provably sound. However, they require again to manually define the underlying planning model (domain and problem descriptions) and rely on the correctness of the model. The modeling effort of these approaches varies from STRIPS planning models (Ramírez and Geffner 2010; Pereira, Oren, and Meneguzzi 2020; E-Martín, R-Moreno, and Smith 2015) to more knowledge-intensive as Hierarchical Task Networks (HTNs) (Höller et al. 2018).

Another approach to perform goal recognition consists of using state of the art machine learning approaches to learn patterns of actions that predict goals. So, instead of requiring an action model, they require training instances.

These two approaches to goal recognition are based on the two major approaches to AI (Geffner 2018): model-based (e.g. planning, search) and model-free (learning). A key distinction between the two approaches in relation to goal recognition relates to the assumption that planning approaches make about agents' rationality: they will try to achieve their goals in the best possible way (optimal). On the contrary. model-free approaches do not have to assume that the agents' behavior is rational. Some authors have previously shown lack of rationality and noisy decisions in specific domains, as games (Min et al. 2016).

In this paper, we provide a comparison of state of the art domain-independent model-based approaches (Pereira, Oren, and Meneguzzi 2020) and the adaptation to goal recognition of two state of the art model-free approaches: long short-term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and XGBoost (Chen and Guestrin 2016). Previous work on model-based goal recognition showed impressive results in terms of accuracy of recognized goals, close to 100%, with very few observations. We claim this is due to the use of simple benchmark cases. Therefore, we have created a few harder goal recognition tasks. In particular, we have created a benchmark focusing on a finance-related application to show its potential impact on this area. In this new domain, bank clients can open accounts, receive their payrolls and make payments to buy items or pay for their kids college. Financial institutions can partially observe all these actions (traces of human behavior), as described in (Borrajo, Veloso, and Shah 2020). In this domain, the task of a goal recognition system is to determine for which products or services humans will be making payments by analyzing previous observations.

In the next sections, we will present related work, introduce the learning-based approaches, show and discuss the experiments and draw some conclusions.

Related Work

We have covered in the introduction the closest model-based works related to goal recognition (Höller et al. 2018; Ramírez and Geffner 2010; Pereira, Oren, and Meneguzzi 2020; E-Martín, R-Moreno, and Smith 2015; Sohrabi, Riabov, and Udrea 2016; Vered and Kaminka 2017). In the case of model-free (learning) approaches, some authors have started recently using machine learning techniques based on Convolutional Neural Networks (CNNs) combined with plan libraries (Granada et al. 2020) or LSTM (Amado et al. 2019) to infer goals from a sequence of observations. In these works, observations are images that represent states when some task is being solved. Our observations are not required to take the format of an image and they are actions performed by the agent instead of states; states contain more information than just the actions. We argue that action traces (sequences) match more closely financial transaction traces, and they are thus a better fit for goal recognition in finance (Borrajo, Veloso, and Shah 2020; Borrajo and Veloso 2020).

There has also been increasing interest in several domains to devise goal recognition systems, such as the work on story understanding (Charniak and Goldman 1993), network security (Geib and Goldman 2009), or computer games (Hooshyar, Yousefi, and Lim 2018). The latter domain can be considered similar to some financial tasks, the adversarial ones, such as fraud, money laundering or market based. In games, some examples used a combination of model-based (using manually generated Markov Logic Networks) with model-free approaches that learned the associated probabilities (Ha et al. 2011). Other authors used LSTMs to predict user's goals in game playing (Min et al. 2016). The input to the LSTM includes the previously achieved subgoals, as well as the executed action. They also assumed agents do not interleave working on several goals, since the goals they handle do not share anything in common. However, in many situations (e.g. all domains considered in planning-based approaches), goals are sets of propositions and goals can have some shared propositions. Additionally plans/action traces can approach multiple goals before settling on one or the other. So the assumption of non-interleaving plans is very constraining.

As an example, suppose that there are three propositions in a financial application : $P1$=`(paid Client Car)`, $P2$=`(paid Client House)` and $P3$=`(has-credit-card Client)`. Then, one goal could be G1=$\{P1, P3\}$ and another one could be G2=$\{P1, P2\}$. While both goals are different, they do share some common parts. Therefore, the assumption that agents cannot interleave work on them is too restrictive, since the client could be trying to reach both by pursuing $P1$. Instead, we allow goals to share common components, so agents can potentially be working on two goals at the same time, until they commit finally to one of them.

Also, as work in other goal recognition domains, their

work is domain dependent. Adapting it to a new domain would require some feature modeling effort as they mention in the paper. Our use of machine learning techniques is domain-independent and thus is a better comparison to model-based methods. Our LSTM approach returns the predicted goal after taking as input a sequence of one-hot encoding of actions (which are the observations). This approach makes it independent of the domain.

Background and Problem Definition

We define the goal recognition framework as a tuple $GR = \langle F, A, \mathcal{G} \rangle$, where $F = \{f_1, \ldots, f_n\}$ is a set of propositions, $A = \{a_1, \ldots, a_m\}$ is a set of instantiated actions, and $\mathcal{G} = \{G_1, \ldots, G_p\}$ is a set of goals. In automated planning, a proposition is an instantiated literal. For instance, `account-owner-C1-A1` would represent the fact that a client `C1` is the owner an account `A1` in a bank in a financial domain. Similarly, an instantiated action is traditionally composed of an action name and constants as parameters. For instance, `open-account-C1-A1` would be an instantiated action where the client `C1` opens the account `A1` in the same domain. Each goal $G_i \in \mathcal{G}$ is a set of propositions in F; that is, $\forall g_j \in G_i, g_j \in F$. In the example domain, a potential goal G_i could be: $G_i = \{$`paid-C1-House1,paid-C1-College1`$\}$ representing the goals of buying a house and paying for the children college.

The learning algorithms to be defined in the next section receive as input a set of training instances and return a goal-recognition classifier. Each training instance is a tuple $i = \langle O, G \rangle$. $O = \{o_1, \ldots, o_o\}$ is a sequence of observations of instantiated actions ($\forall o_i, o_i \in A$), and $G \in \mathcal{G}$. A goal-recognition classifier is an algorithm that takes as input an observation sequence O' and returns a label that is a goal G in $\mathcal{G}$. Note that the learning technique does not take as input a domain model (actions and predicates), nor any prior information, as other planning-based goal-recognition approaches (Ramírez and Geffner 2010). So, each action is just a label without any semantics. In fact, in order to properly work with the proposed learning techniques, we used numerical values for those labels, as well as for the goals.

As an example of an observation sequence in the hypothetical financial domain, we could have observed:

$\langle$`create-account-C1-A1,work-C1,...,work-C1, pay-C1-House1`$\rangle$

The action `work` increases the balance of the account. And the `pay` action achieves the goal (paying House1). The training instance could be something like:

$$\langle \langle 11, 23, 23, \ldots, 23, \rangle, \{35\} \rangle$$

where 11 represents `account-owner-C1-A1` and similarly for the other literals, while 35 represents `pay-C1-House1`. These integer values are replaced by a one-hot encoding.

Learning Models for Goal Recognition

We apply two state of the art model-free methods for learning goals, namely recurrent neural networks (RNNs) and gradient boosted trees (GBTs). RNNs have been applied to a wide-range of problems including neural machine translation (Cho et al. 2014) and sequence to sequence learning (Sutskever, Vinyals, and Le 2014). They apply the same set of weights for each action of the sequence which makes them memory efficient. It also provides them translational symmetry. LSTMs are a type of RNN with memory and gating that can handle longer sequences of data (Hochreiter and Schmidhuber 1997). A recent version has been widely adopted to handle even longer input sequences (Cho et al. 2014).

Gradient boosting is used to learn an ensemble model combining weak learners to create a good classifier. The weak learners are typically decision trees and hence the term GBT. At a high-level, gradient boosting fits a new tree at each step to the residual of the current pseudo-residuals. A popular implementation of GBT with additional tuning, such as proportional shrinking of leaves and Newton boosting, is provided by XGBoost (Chen and Guestrin 2016).

Experiments

In this section, we present the experimental setting, the results obtained and an in-depth analysis.

Settings

Our experiments were conducted on domains used in previous works on goal recognition, such as three International Planning Competition (IPC[1]) domains: Block-words, Logistics, and a Simple Grid domain. We also ran experiments on a new domain inspired on a finance task, which we simply call Buy-domain. We will now describe each of the domains in more detail.

Block-words is equivalent to the known Blocksworld domain, but involves blocks whose names correspond to letters. Blocks start in a specific configuration on a table. A robotic arm can unstack/stack blocks from/to others or pick-up/put-down blocks from/on the table, in order to reach a specified goal configuration that represents a specific English word. As an example, given the blocks named R, A, E, D, we could create problems whose goals are to create towers of blocks with the words READ, DEAR, or RED. Figure 1 shows an example of an initial state (left) and two possible goals (right): READ and DEAR. It also shows a partial observation (two actions). The goal would be to predict which of the two words the agent is trying to create given those observations.

We generated two sets of instances in this domain. In the first one, there are two goals composed of 24 propositions each (blocks forming a big tower). The only difference between the two goals is the block on top. The

[1] `ipc.icaps-conference.org`

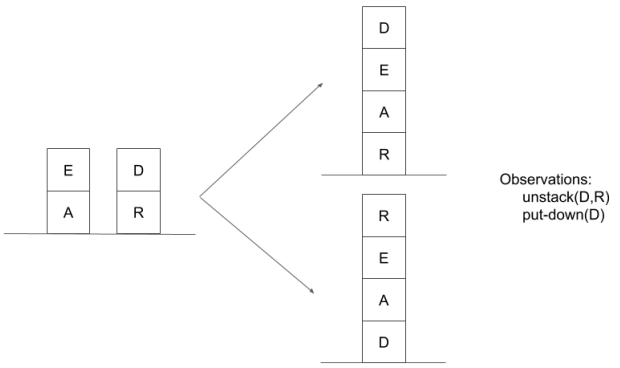

Figure 1: Example of an initial state in the Blocks-words domain (left) and two potential goals (middle). It also shows some observations (actions) on the right.

initial state was randomly generated. This is an example of a type of task that is hard for goal recognition, since it is not until the end that the observer can really decide which goal the other agent is pursuing (which block will be on top). Also, in order to test the ability of learning some prior probability distributions, the probability of selecting one of the words was 80% and the probability of selecting the other one 20%. We would expect that techniques that can handle this prior probabilities or learn them from data would have a higher accuracy from the start of the observation sequence. The second set of instances was composed of five goals (words) and for each generated problem we picked randomly which one should be the problem's goal. The initial state was also randomly generated. This task was designed to test the generalization capability of the goal recognition systems (especially the learning-based ones). In order to train the learning-based systems, we randomly generated 10,000 problems that were solved by a sub-optimal planner (the first iteration of the LAMA planner (Richter and Westphal 2010a)). The generated plans were used as training traces together with their corresponding goals.

Logistics involves transporting a set of packages from their initial locations to specified goal locations for each package. Packages can be transported within a city via a truck, and across cities via airplanes; this is illustrated in Figure 2. In our experiments there were 10 cities, with 4 locations within each city. Each city had 1 truck, and there was 1 airplane to move packages within cities. There were 10 packages in total, which had to be moved from their initial location to their respective goal locations. We used two settings in this domain. One setting involved two goals and a fixed initial state; the likelihood of one goal was 80% and the other was 20%. Since the plans started with the packages in the same initial location, the prefixes of the plans reaching the goal state had many shared actions. The other setting of the logistics domain involved 10 goals, and random initial states. For the first setting, we randomly generated 2,000 problems and for the second setting we randomly generated 10,000 problems. Again, these problems were solved by the same planner as before to generate the training traces.

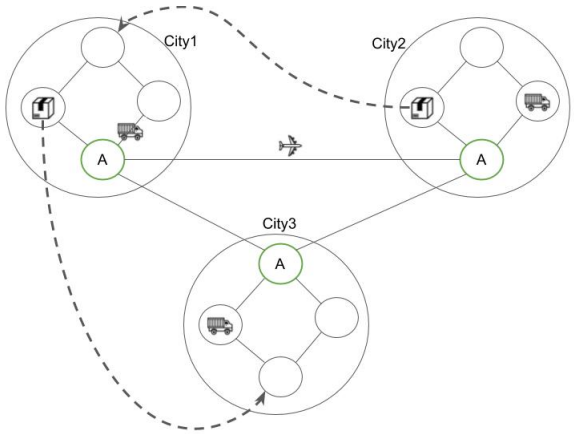

Figure 2: Example of an initial state in the logistics domain that has 3 cities, with 4 locations each.

Grid involves a robot that can move in a grid. Some tiles are locked and the robot has to use keys that are randomly distributed in the grid to open those tiles. The goal is to move the robot from its initial position to a final tile. Figure 3 shows an example of an initial state of the robot (bottom left marked with R), the grid, a set of keys (K) and locked positions (L). It also shows two possible goals (upper right) marked with G. At the right of the figure, there is a sequence of four observations (actions).

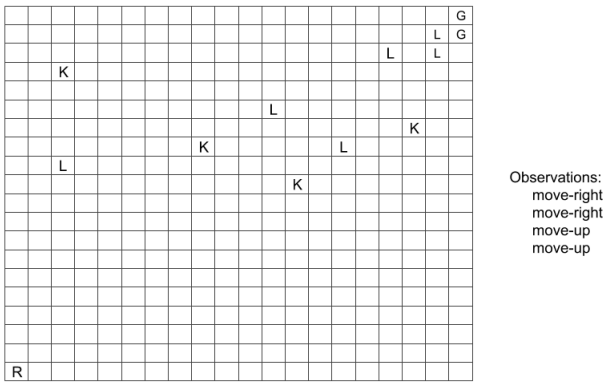

Figure 3: Example of an initial state in the Grid domain. R represents the initial state (position of the robot), K are keys, L are locked positions, and G are potential goals. It also shows some observations (actions) on the right.

In this domain we generated two sets of examples. In the first one, we selected the initial state and goals as in Figure 3 and some locked positions and keys randomly distributed. This task was designed again with the objective of making the goal recognition task difficult since the prefix of plans (first actions of the plan) will be similar to plans achieving the two goals. Again, we biased the goals with a 80/20% probability distribution to test the ability to learn the priors. In the second test, we generated random initial states and goals (from a set of 20 pre-defined goals). We also used 10,000 traces for each setting to train the learning approaches.

Buy domain is a simplified financial domain. Clients of a bank can open different kinds of accounts, work, receive their payrolls, move funds from one account to another and buy products (e.g. house) or pay for their kids education. In the initial state, the clients open an account with zero balance and when they have worked for some time and they have enough money in the balance, they can either buy products or pay for the college. This domain cannot be handled by model-based approaches easily since it includes numeric variables that are not handled by most goal recognition systems currently. Figure 4 shows an observation trace of this domain.

create-account Client1 Account1
payroll Client1
payroll Client1
...
payroll Client1
buy Client1 Product1
payroll Client1
...
payroll Client1
payroll Client1
pay-college Client1

Figure 4: Example of an observation trace in the Buy domain.

In this domain we also generated two sets of tasks. In the first task, we generated two potential goals (either buy a house or pay for the kids college) with a prior probability of 80/20%. In those problems where the goal was to buy a house, we added an artificial first action to the observations of opening a savings account. This action does not achieve any of the goals, so the planning-based approaches cannot guess the connection to the goals. This kind of task shows that when there is a relation between arbitrary irrelevant actions, planning-based approaches will not be able to correctly detect the right goal from the start. These actions are common to all real world domains where agents are humans, given that often they do not behave rationally. Instead, learning approaches will be able to detect those correlations and boost the accuracy in those tasks. The second task has 10 goals (buying different products or attending different colleges) and random initial states.

Training and Test Instances

The training data was generated using the first iteration of the LAMA sub-optimal planner (Richter and Westphal 2010b). For the techniques that involve learning, we used a 80/20 split of the data (traces) into training and validation. In order to compare with the landmark goal recognition technique, we kept a dedicated set of 100 test traces for each setting separately (not in the 80/20 data set). Table 1 shows a summary of some key metrics with respect to the training instances.

In our experiments, we tried to intentionally generate problems that are hard for goal recognition. We did this by having some goals close to each other, so plans would share a large prefix, and shared actions would make goal recognition difficult. Additionally, having some settings with random and diverse initial conditions would make it hard to learn patterns of actions (subsets or subsequences of actions) that correlate to goals.

Domain	Set	Max length	Num. actions	Num. goals
Blocks-words	Set1	434	424	2
Blocks-words	Set2	16	24	5
Logistics	Set1	67	79	2
Logistics	Set2	111	846	10
Grid	Set1	424	424	2
Grid	Set2	33	1212	10
Buy	Set1	43	6	2
Buy	Set2	541	25	10

Table 1: Summary of key characteristics of the training instances.

Model-Based and Learning Methods

To evaluate the performance of goal recognition with LSTMs and XGBoost, we compared the accuracy with two other baseline techniques based on Landmark-based goal recognition using planning (Pereira, Oren, and Meneguzzi 2020) (which we will call *LGR*). In the first version, we used their code. In that version, they do not contemplate prior probabilities of some goals, as early work on goal recognition by planning did (Ramírez and Geffner 2010). So, in the second version we modified their work to factor in the prior probability distribution of goals when making the prediction. We selected to compare against LGR given that their results are better or similar to the rest of current state of the art algorithms based on planning. We used the code made publicly available by the authors.[2] To compare with the baseline *LGR* method, we used the same method reported by Pereira et al.: reporting the accuracy results after some percentage of observations were made. We used the same proportions as in their work, viz $\{0.1, 0.3, 0.5, 0.7\}$. We set the threshold parameter to 0.1 for their *LGR* method.

[2] `https://github.com/ramonpereira/Landmark-Based-GoalRecognition`

For LSTMs, we set the embedding dimension of actions to 512 to cover the cases where we had a large number of actions. The learning rate was 0.01 and both batch sizes for training and hidden dimension were set to 32. The number of epochs for training was 10. These parameters were not tuned to the datasets. We would expect better performance if tuned to the problem. For GBT, we learned 100 trees with a maximum depth of 3.

Discussion

Table 2 shows the results in terms of accuracy and time to make predictions for the 100 test plan traces. In all domains except Logistics, learning-based methods outperform the model-based LGR method. Factoring the prior into the prediction of LGR only made sense when there were two goals, because those were the settings when the goal distribution was skewed (80/20). Otherwise, they were uniform and the priors should not make a difference. Factoring the prior into LGR only seemed to hurt the prediction since the LGR prediction mechanism had nothing to do with distribution and learning; rather LGR used the landmarks observed to predict. The only case when the prior distribution helped the accuracy was in the Grid domain, where LGR was performing abysmally (there are no landmarks or they are very uninformative in that domain). We now analyze the results domain-by-domain.

In both settings of the Grid domain, the LGR method did not perform well since there are no landmarks in this domain. Other methods for goal recognition such as computing optimal plans that satisfy the observations could be used instead. However, computing optimal plans on partial observations to all possible goals can be prohibitively expensive. Additionally, human behavior may not be optimal, and so assuming rationality in human goal recognition maybe misleading. As for the LSTM's performance in this domain, it is worse than XGBoost. We hypothesize that since XGBoost sees the data in sets, rather than sequences (as LSTMs), it works in it's favor; the variety of paths in the Grid domain makes learning sequential patterns more difficult. The lack of clear landmarks hurts the LSTM method too. This issue should be explored in the future.

In the Block-words domain, specifically for the setting in which there were 2 goals, the LSTM settled on predictions using the data distribution, i.e. predicting the most likely goal. XGBoost did better than LSTM. We think this maybe for a similar reason as in the Grid domain; the LSTM could not easily learn any clear action sub-sequences to use as predictive features, especially since the 2 goals were very close. XGBoost, on the other hand, is looking at traces as sets of actions rather than as sequences, so it does not suffer from long common prefix sequences. XGBoost also did better than LGR, possibly because the goals are very close to each other. In Block-words, LGR could not easily find landmarks for very similar goals with fewer actions in the traces. As the number of observations increases, LGR does better than LSTM, but still worse than XGBoost. However, in the setting where there were 5 goals, and the initialization was to random states, the LSTM did worse with fewer observations, but increased its performance with more observations. We hypothesize that the large number of prefix plans arising from random initial conditions made pattern detection difficult with fewer observations. Both XGBoost and LGR still fared better than LSTM with few observations. A key requirement for using XGBoost is that the maximum number of actions must be known and fixed, whereas the LSTMs allow for a variable number of actions.

The Buy domain was designed to be difficult, especially in the case of 10 goals as the results show. LGR could not solve any problem given that it cannot handle numeric variables. Again, XGBoost performs better than LSTMs, probably again for the same reasons as before. The prefix of the observation traces were very similar with many payroll actions before committing to a specific goal. This kind of domain is not appropriate for planning-based approaches given that the rational of which goal is closer to the plan does not depend on the observed actions.

With respect to the results for the Logistics domain, we noticed a stark difference in performance between the case with two goals and the case with 10 goals. When there were only two goals and the same fixed initial location of the packages, the LSTM and XGBoost methods were able to perform similarly to the LGR method. XGBoost was able to detect and adjust its predictions based on the discerning actions in the tail end of the traces. Since the plans to the two goals had a lot of shared actions in the prefix, XGBoost probably learned to ignore them, and paid attention to the tail end of the traces. This is seen empirically: XGBoost's accuracy increases when the number of actions increases from 50% to 70%. LSTM on the other hand seemed to converge to predicting based on the distribution of the goals, and stayed there. However, the performance for the Logistics domain setting with 10 goals was quite different. In this setting, the initial state was random, so there was a great diversity in the plan traces, and this made learning patterns very difficult. The actions in Logistics have a partial order that allows for a greater variety of plan traces. This makes finding and utilizing sequence-based patterns harder. So the LGR method performed much better than the LSTM-based approach for Logistics with 10 goals. So, contrary to our expectations, the LSTM learning-based approach did not always guarantee better results than a model-based approach. For some other domains, the order of actions is more strict and so learning-based methods would have an easier time finding patterns to use for goal prediction.

We did try to coax the LSTM method into better performance with fine tuning the hyper-parameters and increasing model capacity. Even with 50 epochs of training, the accuracy was only as high as about 47% and that too only after seeing over half of the entire plan

Domain	Observability	LGR		LGR w Prior		XGBoost		LSTM	
		Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time
Blockswords	10	69.0	160.81	77.0	126.87	79	0.03	79	0.02
2 goals	30	76.0	147.75	77.0	137.23	92	0.03	79	0.02
	50	86.0	155.22	77.0	151.40	97	0.03	79	0.02
	70	86.0	147.35	77.0	140.75	96	0.03	79	0.02
Blockswords	10	23.0	28.96	23.0	28.96	41	0.001	12	0.03
5 goals	30	23.0	29.32	23.0	29.32	41	0.001	24	0.03
	50	44.0	29.17	44.0	29.17	64	0.001	31	0.03
	70	64.0	29.93	64.0	29.93	64	0.001	78	0.03
Logistics	10	68.0	417.89	81.0	426.75	75.0	0.001	79.2	0.06
2 goals	30	81.0	461.64	81.0	415.65	75.0	0.001	79.2	0.06
	50	81.0	398.90	81.0	462.80	75.0	0.001	79.2	0.06
	70	81.0	405.61	81.0	429.85	87.0	0.001	79.2	0.06
Logistics	10	10.0	1657.51	10.0	1657.51	14.0	0.001	11.79	0.02
10 goals	30	24.0	1668.00	24.0	1668.00	44.0	0.001	21.30	0.02
	50	50.0	1283.90	50.0	1283.90	45.0	0.001	47.78	0.02
	70	69.0	1300.52	69.0	1300.52	47.0	0.001	49.12	0.02
Grid	10	1.0	115.07	77.0	78.99	77	0.001	78	0.03
2 goals	30	1.0	96.01	77.0	80.35	94	0.001	94	0.03
	50	1.0	104.15	77.0	94.98	94	0.001	96	0.03
	70	1.0	100.12	77.0	84.98	96	0.001	98	0.03
Grid	10	0.0	319.00	0.0	319.00	15	0.01	5	0.19
10 goals	30	0.0	427.90	0.0	427.90	29	0.01	11	0.19
	50	0.0	361.14	0.0	361.14	55	0.01	19	0.19
	70	0.0	312.50	0.0	312.50	59	0.01	21	0.19
Buy	10	-	-	-	-	100	0.0003	80	0.19
2 goals	30	-	-	-	-	100	0.0003	80	0.19
	50	-	-	-	-	100	0.0003	80	0.19
	70	-	-	-	-	100	0.0003	80	0.19
Buy	10	-	-	-	-	12	0.002	10	0.26
10 goals	30	-	-	-	-	34	0.002	9	0.26
	50	-	-	-	-	55	0.002	9	0.26
	70	-	-	-	-	74	0.002	9	0.26

Table 2: Comparison of LSTM, XGBoost, LGR and LGR+prior information with respect to accuracy and time to perform goal recognition in several domains under some observability ratios.

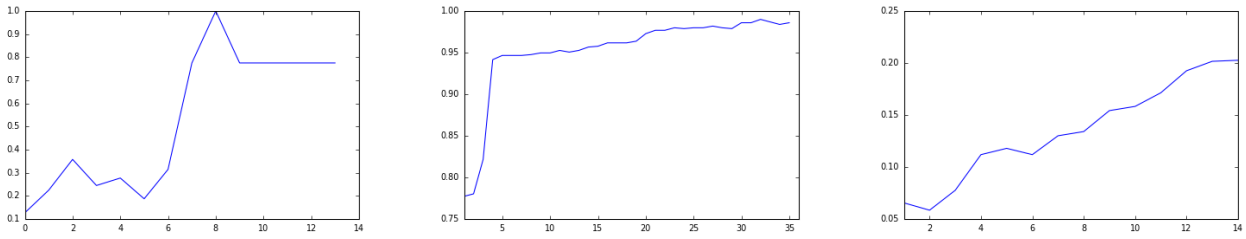

Figure 5: Convergence results for the LSTM model in three cases corresponding to the Block-words domain fixed initial state (left) as well as the Grid domain fixed initial state problems (middle) and the Grid domain problem with random initial states with 10 goals (right). Typically, the performance increases as the model observes more actions except for one of the cases where we see that towards the end it drops back to 80 percent. x-axis represent the number of traces used for training, and the y-axis represents the corresponding accuracy.

trace. Doubling the size of the hidden dimensional space of the LSTM from 32 to 64 increased the accuracy by less than 1%. So, when the data complexity is high, or rather the partial ordering of actions allows for a greater variety of plan traces, using a principled model-based approach seems to do better. The trade-off is that model-based approaches require more computation time during inference as seen in Table 2, while learning-based approaches need more computation time during training.

Another issue has to do with model complexity. Model-based methods are either prohibitively expensive for numeric domains or simply do not support them. In such cases, learning-based methods might be the only alternative, or at least provide better performance. This gap in model-based goal recognition is an opportunity for research, and will be needed for financial domains.

Conclusions and Future Work

Goal recognition on plan traces of actions can be used in finance to better understand customer goals from their actions, and then provide better services to them. Goal recognition can be done through model-based or model-free (learning) methods. In this paper, we have adapted state-of-the-art model-free techniques to work for goal-recognition and compared their performance with that of state-of-the-art model-based approaches. We have performed this comparison using different domains: three standard IPC domains; and one novel domain geared towards mimicking the use of goal recognition for financial tasks. We have analyzed the results and discussed the trade-offs between the two types of approaches.

The obtained results reinforce the knowledge on the differences between the two main AI paradigms. The two main known differences are: model-based need a model, which requires some knowledge engineering effort, while model-free require a potentially huge amount of examples and training effort. Also, model-based approaches typically assume agents' rationality, while model-free methods automatically adapt to the particularities of observed agents.

One of the new conclusions is that model-based approaches are better when there is a partial order of actions in plans. This feature allows for a great diversity of plan traces, which makes learning action-patterns for goal predictions very hard for learning-based methods, more specifically to sequence-based ones, as LSTMs. As expected, this comes at the cost of computation time during inference. On the other hand, if there is a relation between some actions (not directly associated to goal achievement) and the goals, learning techniques will be able to capture that relation, while model-free will fail to recognize it and present worse performance.

In future work, we would like to study hybrid approaches that combine model-based and model-free approaches to goal recognition, and how to combine them for different settings. We would also like to investigate the performance of different goal-recognition techniques under noisy observability, goal-recognition under plan obfuscation from an opponent planning-based system (Kulkarni, Srivastava, and Kambhampati 2020), and plan completion for subsequent goal recognition. We would also like to use attention-based models such as transformers (Vaswani et al. 2017) for goal recognition, which has hitherto not been done.

Acknowledgements

This paper was prepared for information 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

Amado, L. R.; Aires, J. P.; Fraga Pereira, R.; Magnaguagno, M. C.; Granada, R.; Licks, G. P.; and Meneguzzi, F. R. 2019. Latrec: Recognizing goals in latent space. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS)*.

Avrahami-Zilberbrand, D., and Kaminka, G. 2005. Fast and complete symbolic plan recognition. In *Proceedings of IJCAI-05*.

Borrajo, D., and Veloso, M. 2020. Domain-independent generation and classification of behavior traces. In *Preprints of the Workshop on Planning for Financial Services (ICAPS'20)*.

Borrajo, D.; Veloso, M.; and Shah, S. 2020. Simulating and classifying behavior in adversarial environments based on action-state traces: An application to money laundering. In Balch, T., ed., *Proceedings of the 2020 ACM International Conference on AI in Finance*. New York (EEUU): ACM.

Bui, H. H.; Venkatesh, S.; and West, G. 2002. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research* 17:451–499.

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11(1):31–48.

Charniak, E., and Goldman, R. P. 1993. A bayesian

model of plan recognition. *Artificial Intelligence* 64(1):53 – 79.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

E-Martín, Y.; R-Moreno, M. D.; and Smith, D. E. 2015. A fast goal recognition technique based on Interaction estimates. In *IJCAI*.

Geffner, H. 2018. Model-free, model-based, and general intelligence. *arXiv preprint arXiv:1806.02308*.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101 – 1132.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning. Theory & Practice*. Morgan Kaufmann.

Granada, R.; Pereira, R. F.; Monteiro, J.; Amado, L.; Barros, R. C.; Ruiz, D.; and Meneguzzi, F. 2020. Haprec: Hybrid activity and plan recognizer. *arXiv preprint arXiv:2004.13482*.

Ha, E. Y.; Rowe, J. P.; Mott, B. W.; and Lester, J. C. 2011. Goal recognition with markov logic networks for player-adaptive games. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

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

Höller, D.; Behnke, G.; Bercher, P.; and Biundo, S. 2018. Plan and goal recognition as htn planning. In *IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, 466–473. IEEE.

Hooshyar, D.; Yousefi, M.; and Lim, H. 2018. Data-driven approaches to game player modeling: a systematic literature review. *ACM Computing Surveys (CSUR)* 50(6):1–19.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the fifth National Conference on Artificial Intelligence (AAAI-86)*, 32–37. Philadelphia, PA (USA): AAAI Press.

Kautz, H. 1987. *A Formal Theory of Plan Recognition*. Ph.D. Dissertation, University of Rochester.

Kulkarni, A.; Srivastava, S.; and Kambhampati, S. 2020. Signaling friends and head-faking enemies simultaneously: Balancing goal obfuscation and goal legibility. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*.

Min, W.; Mott, B. W.; Rowe, J. P.; Liu, B.; and Lester, J. C. 2016. Player goal recognition in open-world digital games with long short-term memory networks. In *IJCAI*, 2590–2596.

Mooney, R. J. 1990. Learning plan schemata from observation: Explanation-based learning for plan recognition. *Cognitive Science* 14(4):483–509.

Ortiz-Laguna, J.; Ángel García-Olaya; and Borrajo, D. 2013. Using activity recognition for building planning action models. *International Journal of Distributed Sensor Networks* 2013.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2020. Landmark-based approaches for goal recognition as planning. *Artificial Intelligence* 279.

Pynadath, D., and Wellman, M. 1995. Accounting for context in plan recognition with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 472–481.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 1121–1126. AAAI.

Richter, S., and Westphal, M. 2010a. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

Richter, S., and Westphal, M. 2010b. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *Proceedings of IJCAI*.

Sukthankar, G.; Goldman, R. P.; Geib, C.; Pynadath, D. V.; and Bui, H. 2004. *Plan, Activity, and Intent Recognition*. Morgan Kaufmann.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Vered, M., and Kaminka, G. A. 2017. Heuristic online goal recognition in continuous domains. In *IJCAI*, 4447–4454.

Domain-independent Generation and Classification of Behavior Traces

Daniel Borrajo[*], **Manuela Veloso**[†]
J.P.Morgan AI Research, New York, NY (USA)
{daniel.borrajo,manuela.veloso}@jpmchase.com

Abstract

Financial institutions mostly deal with people. Therefore, characterizing different kinds of human behavior can greatly help institutions for improving their relation with customers and with regulatory offices. In many of such interactions, humans have some internal goals, and execute some actions within the financial system that lead them to achieve their goals. In this paper, we tackle these tasks as a behavior-traces classification task. An observer agent tries to learn characterizing other agents by observing their behavior when taking actions in a given environment. The other agents can be of several types and the goal of the observer is to identify the type of the other agent given a trace of observations. We present CABBOT, a learning technique that allows the agent to perform on-line classification of the type of planning agent whose behavior is observing. In this work, the observer agent has partial and noisy observability of the environment (state and actions of the other agents). In order to evaluate the performance of the learning technique, we have generated a domain-independent goal-based simulator of agents. We present experiments in several (both financial and non-financial) domains with promising results.

Introduction

Given some training traces obtained by observing at least two kinds of agents, the goal of this research consists of learning a classifier that can differentiate among those types of agents by observing traces of their behavior. We assume there is a, usually hidden, rationale for the behavior of agents when taking actions in the environment that depends on some (again hidden) goals and the states they encounter while taking actions to achieve those goals. And we also assume goals, states and actions can be represented using standard planning representation languages.

We leverage on previous work on sequence classification in contexts where there was no domain model and the representation of traces was a vector of features (Xing, Pei, and Keogh 2010). Some of those approaches did a manual definition of the relevant features to be used in the classification, which usually resulted in domain-dependent approaches. And none of these approaches used a relational representation of data in the form of goals, states and actions. Instead, we assume the other agents use a hidden planning model and the relevant aspects to make the classification depend on the actions executed and the related states.

Given the setup of an observer agent and a planning-execution agent, several decision-making tasks can be defined. Within this setting, most works in automated planning have focused on goal/plan recognition, where the observer has to infer the goals the planning agent is pursuing (Ramírez and Geffner 2010) or the plan it is using to achieve some goals (Avrahami-Zilberbrand and Kaminka 2005). Once the goals/plans are recognized, other planning-related tasks can be solved such as generating plans to stop an opponent to reach its goals (Pozanco et al. 2018) or change the environment to improve the goal recognition task (Keren, Karpas, and Gal 2014). Other uses of traces include learning action models (Aineto, Celorrio, and Onaindia 2019) or predicting the next action or sequence of actions another agent is going to perform (Bernard and Andritsos 2019; Tax et al. 2017). However, as far as we know, the sequence classification task has not been addressed yet within the planning community.

Even if it has been less studied than related tasks in the context of automated planning, many real-world tasks benefit directly from this research. Some of these domains have been studied in the context of domain-dependent approaches (Xing, Pei, and Keogh 2010). Examples are: predicting whether someone will buy a product from the web clicks sequence; detecting intrusions in network or standalone computer systems; classification of anomalous behavior in public spaces (e.g. terrorism); machines monitoring the behavior of other machines; or labeling an opponent's behavior in a game. In the case of financial applications there are numerous examples of the use of this task such as: fraud or anti-money laundering detection; classifying malicious traders; attrition prediction; offering new services to customers; or detection of users that will complain.

We present as contributions: a learning technique that can classify in agents' types based on their behavior expressed in observation traces; and a domain-independent simulator of agents' behavior based on dynamic goal generation, plan-

[*]On leave from Universidad Carlos III de Madrid. The position at the lab is as a consultant.

[†]On leave from Carnegie Mellon University

ning and execution. We name the first contribution Classification of Agents' Behavior Based on Observation Traces (CABBOT). Some of the simulator features are: explicit reasoning on goals generation, modification and removal; ability to inject new instances when needed; several methods for generating goals (goals schedule, behavior-based random generation); exogenous events; non deterministic execution of actions; and partial and noisy observability.

A version of this work was published in (Borrajo, Veloso, and Shah 2020). We focused before on its application to money laundering, while this paper focuses on its general applicability to planning tasks. Therefore, the description of the techniques and the simulator are centered on the underlying planning tasks, and the experiments report on several domains. Thus, we also present as contribution several domains designed for this task, whose detailed description is included in the experimental section. The domains range from a simplified terrorist domain to a service cars domain and two financial services related ones. The results show that CABBOT can accurately classify agents in those domains.

Background

Given that we assume agents' rational behavior to be based on the concepts of goals, states and actions, we will use the automated planning formalism to describe the tasks (Ghallab, Nau, and Traverso 2004).

Automated Planning

We use the standard classical STRIPS definition of a planning task, augmented with numeric variables (functions). A planning task is defined as $\Pi = \langle F, A, I, G \rangle$, where F is a set of boolean and numeric variables, A is a set of actions, $I \subseteq F$ is the initial state and $G \subseteq F$ is a set of goals. Each action $a \in A$ is defined in terms of its preconditions (pre(a)) and effects (eff(a)). Effects can set to true the value of a boolean variable (add effects, add(a)), set to false the value of a boolean variable (del effects, del(a)), and change the value of a numeric variable (numeric effects, num(a)). We will denote with S the set of all states. A (full) state is a valuation of all the variables in F; a boolean value for all the boolean variables and a numeric value for the numeric ones. Action execution is defined as a function $\gamma : S, A \rightarrow S$; that is, it defines the state that results of applying an action in a given state. It is usually defined as $\gamma(s, a) = (s \backslash \text{del}(a)) \cup \text{add}(a)$ if pre(a)$\subseteq s$ when only boolean variables are considered. When using numeric variables, γ should also change the values of the numeric variables (if any) in num(a), according to what the action specifies; increasing or decreasing the value of a numeric variable or assigning a new value to a numeric variable. If the preconditions do not hold in s, the state does not change.

The solution of a planning task is called a plan, and it is a sequence of instantiated actions that allows the system to transit from the initial state to a state where goals are true. Therefore, a plan $\pi = \langle a_1, a_2, \ldots a_n \rangle$ solves a planning task Π (valid plan) iff $\forall a_i \in \pi, a_i \in A$, and $G \subseteq \gamma(\ldots \gamma(\gamma(I, a_1), a_2) \ldots), a_n)$. In case the cost is relevant, each action can have an associated cost, $c(a_i), \forall a_i \in A$ and the cost of the plan is defined as the sum of the costs of its actions: $c(\pi) = \sum_i c(a_i), \forall a_i \in \pi$.

The planning community has developed a standard language, PDDL (Planning Domain Description Language), that allows for a compact representation of planning tasks (Ghallab et al. 1998). Instead of explicitly generating all states of Π, a lifted representation in a variation of predicate logic is used to define the domain (predicates and actions) and the problem to be solved (initial state and goals).

Multi-Agent Framework

In this work we consider at least two agents: acting agent, C (e.g. bank customer) and observer agent, B (e.g. financial institution or bank). In order to create a realistic environment, we will consider that they have different observability of the environment. Thus, each one of them will have its own definition of a planning task, as it has already been defined in cooperative (Torreño et al. 2017) and adversarial (Pozanco et al. 2018) multi-agent settings. In the case of C, its planning task can be defined as $\Pi_C = \langle F_C, A_C, I_C, G_C \rangle$. In the case of B, we do not consider here its ability to plan.

B has a partial (public) view of C's task. This view can be defined as $\Pi_{B,C} = \langle F_{B,C}, A_{B,C}, I_{B,C}, \emptyset \rangle$, where $F_{B,C} \subseteq F_C$, $A_{B,C} \subseteq A_C$, $I_{B,C} \subseteq I_C$ and the goals are unkown, represented as $\emptyset$. This view corresponds to the public part of those variables in other Multi-Agent Planning works (Torreño et al. 2017). It also has a partial view of the initial state and the actions; since there will be some actions executed by C, or some preconditions or effects of those actions that B will not observe. B has no observability of C's goals. This assumption contrasts with goal and planning recognition work that assumes a set of potential goals are known (Ramírez and Geffner 2010). In our case, this set would amount to all possible goals that can be defined given a domain (infinite in most cases). Finally, we relax previous works' requirement on C rationality; C can generate optimal or sub-optimal plans.

As an example, a customer might have goals that are not observed by the financial institution, such as having committed a crime, or laundered money. Other goals will be observable only after the customer has executed actions within the financial system that might reveal them, such as having opened an account, worked for a company, made a money transfer, or withdrawn money from a bank. In relation to states, there will be information known by the customer that is not observable by the financial institution, such as how many hours the customer works, or products bought using cash. Similarly, some information will be known, such as products or services bought using financial instruments of the corresponding financial institution, or bills payed to utility companies. Finally, there will be actions performed by the customer that will not be observed by the financial institution, such as committing a crime, while others will be observable, such as making a money transfer.

Once C starts generating plans and executing the actions on those plans, B will be able to see: if the actions in $A_{B,C}$ are executed; and the components of the state related to variables in $F_{B,C}$. A planning trace t_C is a sequence of states and actions executed by C in those states:

$$t_C = (I_C, a_1, s_1, a_2, s_2, \ldots, s_{n-1}, a_n, s_n)$$

where $s_i \in S_C, a_i \in A_C$. An observation trace is also a sequence of states and actions of C from the point of view of B $t_{B,C} = (I_{B,C}, a'_1, s'_1, a'_2, s'_2, \ldots, s'_{n-1}, a'_n, s'_n)$, where $s'_i \in S_{B,C}, a'_i \in A_{B,C}$. Each state s'_i corresponds to the partial observability of C's state s_i by B. Also, each action a'_i corresponds to either an action that can be observed from C, a_i, or a ficticious `no-op` action if a_i cannot be observed by B. There is no actual need of requiring the states to be part of the observation; given that B has a model of C's domain, B can always reproduce the corresponding observable states, by simulating the execution of the observable actions. We will call $T_{B,C} = \{t_{B,C}\}$ the set of traces of agent C observed by agent B.

In the classification task we are addressing in this paper, there are two C agents we would like the learning system to differentiate by observing their behavior traces. As an example, consider a criminal and a regular customer. We want to address non-trivial learning tasks. Therefore, we assume there is nothing in the observable state that directly identifies one or the other type of C agent. Nor there is any difference on the observable actions between the ones that can be executed by one or the other type of C. Formally, given two different types of C, C_1 and C_2, B's observable information on both should be the same:

$$\Pi_{B,C_1} = \Pi_{B,C_2} = \langle F_{B,C_1}, A_{B,C_1}, I_{B,C_1}, \emptyset \rangle$$

Learning to Classify Behavior

B's main task consists of learning to classify among the different types of C (behaviors). The learning task can be defined as follows:

- Given: (1) a set of classes of behavior (labels) $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$; (2) a set of labeled observed traces $T_{B,C_i}, \forall C_i \in \mathcal{C}$; and (3) a partially observable domain model of each C_i given by Π_{B,C_i}
- Obtain: a classifier that takes as input a new (partial) trace t (with unknown class) and outputs the predicted class

A main requirement of CABBOT is to be domain-independent. Therefore, we will not use any hand-crafting of features for the learning task. Another characteristic of this learning task is that it works on unbounded size of the learning examples. Traces can be arbitrarily large, as well as states within the trace and action descriptions (both in the number of different action schemas, and grounded actions). There is no a priori limit on these sizes. Using fixed-sized input learning techniques can be difficult in these cases and some assumptions are employed to handle that characteristic. Hence, we will consider here only relational learning techniques (Dzeroski and Lavrac 2010), and, in particular, relational instance-based approaches (Emde and Wettschereck 1996). Relational learning techniques have been extensively used in the past to learn control knowledge (Veloso et al. 1995), or planning policies (Yoon, Fern, and Givan 2008; García-Durán, Fernández, and Borrajo 2012), among other planning tasks (Jiménez et al. 2012). But, as far as we are aware of, they have not been used for this learning task.

The key parameter of these techniques is the relational distance between two traces, $d : T \times T \to \mathbb{R}$. In order to define the distance between two traces, t_1 and t_2, we have several alternatives.

- Compute a distance between the sets of actions on each trace. A simple, yet effective, distance function consists of using the inverse of the Jaccard similarity function (Jaccard 1901) as:

  $$d_a(t_1, t_2) = 1 - \frac{|an(t_1) \cap an(t_2)|}{|an(t_1) \cup an(t_2)|}$$

  where $an(t_i)$ is the set of actions' names in t_i. This distance is based on the ratio of common action names in both traces to the total number of different action names in both traces.

- Compute distances between sequences of states differences. Given two consecutive states s_1 and s_2 in a trace, we define their associated difference or delta, that represent the new literals in the state after applying the action. They are defined as: $\delta_{s_i,s_{i+1}} = s_{i+1} \setminus s_i$. We can compute a distance between the sets of deltas on each trace by using the Jaccard similarity function as before.

  $$d_\Delta(t_1, t_2) = 1 - \frac{|\Delta(t_1) \cap \Delta(t_2)|}{|\Delta(t_1) \cup \Delta(t_2)|}$$

  where $\Delta(t_i) = \{\delta_{s_j,s_{j+1}} \mid \forall s_j, s_{j+1} \in t_i, 0 \leq j \leq n-1\}$ is the set of deltas of a trace t_i. Again, we only use the predicate and function names.

- The two previous distances only consider actions and deltas as sets. If we want to improve the distance metric, we can use a frequency-based approach (equivalent to an n-grams analysis with $n = 1$). Each trace is represented by a vector. Each position of the vector contains the number of times an observable action appears in the trace. The distance between two traces, d_g, is defined as the squared Euclidean distance of the vectors representing the traces. As before, a new trace is classified as the class of the training trace with the minimum distance to the new trace.

- Instead of using only counts, the distance function can also consider actions and state changes as relational formulae and use more powerful relational distance metrics. We have defined a version of the RIBL relational distance function (Emde and Wettschereck 1996) adapted for our representation of traces, d_r. We needed to adapt it given the different semantics of the elements of the traces with respect to generic RIBL representation of examples. Given two traces, we first normalize the traces by substitution of the names of the constants by an index of the first time they appeared within a trace. For instance, given the following action and state pair:

  ```
  ⟨ create-account(customer-234,acc-345),
    {acc-owner(customer-234,acc-345),
     balance(acc-345)=2000} ⟩
  ```

  the normalization process would convert the trace to:

⟨ `create-account(i1,i2)`,
`{acc-owner(i1,i2),balance(i2)=2000}` ⟩

This process allows the distance metric to partially remove the bias related to using different constant names in the traces. The distance d_r is then computed as:

$$d_r(t_1, t_2) = \frac{1}{2}(d_{ra}(t_1, t_2) + d_{r\Delta}(t_1, t_2))$$

i.e. as the average of the sum of d_{ra} (distance between the actions of the two traces) and $d_{r\Delta}$ (distance between the deltas of both traces). d_{ra} is computed as:

$$d_{ra}(t_1, t_2) = \frac{1}{Z} \sum_{a_i \in a(t_1)} \min_{a_j \in a(t_2)} d_f(a_i, a_j)$$

where $a(t_i)$ is the set of ground actions in t_i, d_f is the distance between two relational formulas and Z is a normalization factor ($Z = \max\{|a(t_1)|, |a(t_2)|\}$). We normalize by using the length of the longest set of actions to obtain a value that does not depend on the number of actions on each set, so distances are always between 0 and 1. d_f is 1 if the names of a_i and a_j differ. Otherwise, it is computed as:

$$d_f(a_i, a_j) = 0.5 - 0.5 \frac{1}{|\text{arg}(a_i)|} d_{\text{arg}}(a_i, a_j)$$

where $d_{\text{arg}}(a_i, a_j)$ is the sum of the distances between the arguments in the same positions in both actions. Each distance will be 0 if they are the same constant and 1 otherwise. Again, we normalize the values for distances. Also, when two ground actions have the same action name, we set a distance of at most 0.5. For instance, if l_1 =`create-account(i1,i2)` and l_2 =`create-account(i3,i2)`,

$$d_f(l_1, l_2) = 0.5 - 0.5\frac{1}{2}(1 + 0) = 0.25.$$

As a reminder, each trace contains a sequence of sets of literals that correspond to the delta of two states. Therefore, $d_{r\Delta}$ is computed as the distance of two sets of deltas of literals ($\Delta(t_1)$ and $\Delta(t_2)$). We use a similar formula to the previous ones:

$d_{r\Delta}(t_1, t_2) = \frac{1}{Z_\Delta} \sum_{\delta_1 \in \Delta(t_1)} \min_{\delta_2 \in \Delta(t_2)} d_{r\delta}(\delta_1, \delta_2)$

where $Z_\Delta = \max\{|\Delta(t_1)|, |\Delta(t_2)|\}$, and $d_{r\delta}$:

$d_{r\delta}(\delta_1, \delta_2) = \frac{1}{\max\{|\delta_1|, |\delta_2)|\}} \sum_{l_i \in \delta_1} \min_{l_j \in \delta_2} d_{f'}(l_i, l_j)$

$d_{f'}(l_i, l_j) = d_f(l_i, l_j)$ when the literals correspond to predicates. We use d_f since actions and literals in the state (l_j, l_j) share the same format (a name and some arguments). However, when they correspond to functions, since functions have numerical values, we have to use a different function d_n. In this case, each l_i will have the form $f_i(\text{arg}_i) = v_i$. $f(\text{arg}_i)$ has the same format as a predicate (or action) with a name f_i and a set of arguments arg_i, so we can use d_f on that part. The second part is the functions' value. In that case, we compute the absolute value of the difference between the numerical values of both functions and divide by the maximum possible difference (M) to normalize:[1]

$$d_n(l_i, l_j) = d_f(f_i(\text{arg}_i), f_j(\text{arg}_j)) \times \frac{\text{abs}(v_i - v_j)}{M}$$

We multiply both, since we see the distance on the arguments as a weight that modifies the difference in numerical values. For example, if δ_1=`{acc-owner(i1,i2),balance(i2)=20}`, δ_2=`{acc-owner(i1,i3),balance(i3)=10}`,

$$d_{r\Delta}(\delta_1, \delta_2) = \frac{1}{2}(\min\{0.25, 1\} + \min\{1, 0.5 \times \frac{|20-10|}{M}\})$$

Once we have a distance metric between traces, we use an instance-based technique, as kNN, to classify a new trace according to the k traces with minimum distance, and computing the mode of those traces' classes. Since the classifier takes a trace as input, CABBOT also allows for on-line classification with the current trace up to a given simulation step. A nice property of kNN is that we can explain how a behavior was classified by pointing out the closest previous cases.

Generation of Synthetic Behavior

In real world applications, traces will come from observations of other agents' actions. In this paper, we have also developed a simulator that can produce those traces for the C agent. Figure 1 shows a high level outline of the simulator. C takes actions in the environment by using a rich reasoning model that includes planning, execution, monitoring and goal generation. It is inspired in some planning and execution architectures (Guzmán et al. 2012), where the main difference lies on the dynamic generation of goals. In particular, the goal generation component allows the agent to change or generate new goals on-line as in past work on goal reasoning (Roberts et al. 2018).

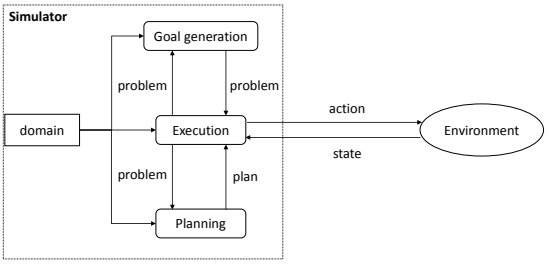

Figure 1: High level view of the simulator.

The components of the simulator for the planning agents are: the *Execution*, that takes a domain and problem description and follows a reasoning cycle that involves generating a new plan by calling *Planning*, executing the next action(s) from the current plan in the environment and observe the next state, and obtaining new goals or state components from *Goal generation*. The simulator is domain independent, except for the Goal reasoning that needs to generate behavior corresponding to at least two types of agents in the same domain. Now, we present a description of each module.

Execution

Execution performs several tasks for some iterations:

- if there is no plan, or there is a reason for replanning, it calls Planning to generate a new plan. Reasons for replanning include: the state received from the environment is

[1] We use a large constant in practice.

not the expected one (it does not fully match the state predicted by the effects of the most recently action); and Goal generation has returned new goals and/or changes in the state. We are using a standard planner for replanning, but it can be substituted by replanning algorithms (Fox et al. 2006; Borrajo and Veloso 2012).

- if there is a plan in execution, it selects the next action to execute and sends it to the environment. The environment simulates the execution of the action and returns a new state. As mentioned above, the new state can be the one defined by the effects (deterministic execution). Our simulator also includes the possibility of defining non-deterministic execution of actions, as well as the appearance of exogenous events.
- at each step, it also calls Goal generation for changes in the goals or partial descriptions of states, as explained below.
- the interaction with the environment also generates a trace of observations that will be used for both training and testing of the learning component of B. As explained before, the trace contains a sequence of actions and states from the point of view of B. Therefore, Execution applies a filter on both so that it only includes in the trace its observable elements. Observability is defined for each domain. We opted for a simplified way to define it as the sets of lifted actions and predicates that can be observed by B. Any ground action or state literal of a lifted action or predicate on those sets will be observable. Besides, B might not see the actual executed action but another one (noisy observations). Also, it might not be able to see some of the actions even if they are in the observable set (a further aspect of partial observability).
- each simulation finishes after a predefined number of simulation steps (horizon) that is a parameter, or after a plan has not been found in a given time bound. We set the time bound with a low value (10 seconds), since this is enough in the experimental domains we have used in most cases.

Goal generation

This component allows agents to generate believable behavior whose goals evolve over time depending on the current state of the environment. It takes as input the current problem description (state, goals and instances) and returns a new problem description. The first obvious effect of this module is to change goals. In order to do so, we have defined two kinds of behavior for each domain by changing the goals of each type of behavior. For instance, in the case of a terrorist domain, we define two types of agents: regular person and terrorist. The regular person would generate goals of going from one place to another. When the simulator has achieved the previous goal (moving to a place), this module will generate a new goal of being somewhere else randomly chosen. However, randomly, the knapsack that it carries might fall down and be forgotten by the person. So, when the person notices that it does not carry the knapsack, it will generate a new goal to hold it again. In the case of the terrorist, this module will randomly generate the goal of not carrying the knapsack. And even if it knows that it is not carrying the knapsack, it will not generate as goal to carry it again, as in the case of the regular person. As a reminder, the observer does not know the goals of the other agent.

This module can also change the problem state and instances. This is useful for generating new components of the state on-line, as with partial observability of a rich environment. Suppose, we want to simulate an open environment where agents wander around and go to places that were not defined originally in the initial problem description. One alternative consists of defining a huge state (and associated instances) in the initial problem description to account for the whole map. This forces the planner to generate many more instantiations than the ones actually needed to plan in the first simulation steps. The ability of Goal generation to change the state and instances descriptions, allows the simulator to generate new parts of the world (or even remove visited ones if not further needed) on the fly, making the process more efficient and dynamic.

Planning

We are working in a domain-independent setting. Therefore, domain and problem models are specified in PDDL. Thus, any PDDL complaint planner could be used for this purpose. In particular, we are using some domains with extensive use of numeric variables (using PDDL functions). So, we are constrained to planners that can reason with numeric preconditions and effects. Examples of planners we are using are: LPG (Gerevini, Saetti, and Serina 2003); CBP (Fuentetaja, Borrajo, and Linares-López 2010), or SAYPHI (de la Rosa, García-Olaya, and Borrajo 2013). As expected, planners take as input a domain and problem description in PDDL, and return a plan that solves the corresponding planning task. All these planners generate sub-optimal solutions.

Experiments

Experimental setting

Due to the lack of existing domains in the planning community that address the task of behavior classification from planning-execution traces, we have defined several new domains:

- **terrorist**: a domain where people move around a grid that represents an open place (station, airport, square, ...) holding a knapsack. Regular people might accidentally drop the knapsack (with probability 0.2), but they try to recover it when they find out. Terrorists drop the knapsack (with probability 0.4) and leave it there. The model is composed of three actions (move, drop and take) and four predicates. The goal is to classify in terrorist or regular behavior from the observed traces. There is full observability in this domain, given that all actions and states are observable, and cannot differentiate between the two types of agents.
- **service cars**: some vehicles move around the streets of a city network. The model comprises seven actions and seven predicates. Actions include: moving from one street section to another connected one, boarding and unboarding a vehicle, stopping a vehicle and moving it again. The

goal is to classify the vehicles that are particular cars from the service cars (taxis or equivalent). All actions and predicates are observed, except for two predicates (whether a driver of a car owns the car, and whether there is a passenger inside a service car or not). There are two board actions depending on the type of vehicle, but the observer cannot differentiate between the two. The same applies to debark actions. The probability a new goal related to moving someone appears is 0.6 in the case of service cars, while the probability a new goal related to moving the owner appears is 0.2 in case of private cars.

- **customer journeys (journey)**: customers access the mobile application of a bank and perform several operations. The model comprises 22 actions, 24 predicates and 2 functions. Actions include: logging in, checking or changing diverse information on their accounts, or performing financial operations. The goal is to classify between customers that are active with the application from the ones that do not use it. The observable actions and predicates are equal for both types of customers. The main difference is the probability of a goal appearing at some point (need of a customer of performing some operation). Active customers will have a higher probability than non-active ones.
- **customer journeys (digital-journey)**: another version of the previous domain, where the task consists of classification between digital users and traditional users. In terms of behavior, digital users have a higher probability of performing digital-based operations (such as quick payments) and traditional users tend to have a lower probability on those operations, but a higher one on traditional operations (such as paying bills).
- **anti-money laundering (AML)**: customers of a financial institution perform operations such as money transfers, payments, or deposits. In the meantime, not observable by B, these customers are either involved in criminal activities, or are regular customers. The challenge in this domain consists of characterizing the type of behavior from observations related to standard activities with the bank. The model comprises 33 actions, 37 predicates and 12 functions. Actions include: criminal activities, getting a job, getting a payroll, or making financial operations. The goal consists of classifying between money laundering individuals and regular individuals. Observability is restricted to the information that a bank can have on a given customer. Therefore, predicates as someone being a criminal or getting dirty money are not observable, while predicates related to making transactions and opening accounts are. We have tried to make this domain rich in terms of the different traces generated by the simulator. Therefore, we have defined several probability distributions that affect issues such as probabilities of selecting different money laundering strategies by criminals, or buying different kinds of items by criminals and regular customers.

For each domain, we have randomly generated 10 traces of each type of behavior for training and 20 for test (where classes are uniformly randomly selected). We measure the accuracy of the prediction. We have used $k = 1$ for the experiments, given that we already obtained good results with that value. We have varied the following parameters to see the impact they have on the results:

- length of the traces (simulation horizon). We used the values: 5, 10, 20, 50 and 100. Default is 50.
- similarity function. We have used the defined ones: d_a, d_Δ, d_g and d_r. Default is d_r.
- probability-goal-appears. We have defined a probability that a set of goals appear at a given time step. Once a set of goals appears, it might take several time steps to execute the plan to achieve all goals. In the meantime, we do not generate new goals, though the simulator is ready to work with that case too. We used the values 1.0, 0.8, 0.5, 0.1, 0.05 and 0.01. Default is 1.0.

We will present results by varying these parameters one at a time to observe the impact they have on the performance of CABBOT.

Results

Table 1 shows the results for the `journey` domain. In this domain, the behavior depends on the probability of a goal arriving for both types of customers: active and non-active. We varied those probabilities to analyze how their values affect the accuracy of CABBOT. We can observe that when the difference between the two probabilities gets smaller, the behavior becomes more similar (in terms of activity level of customers) and accuracy of classification degrades. In the extreme, when the two probabilities are equal – (0.5, 0.5) case –, the classification accuracy is equivalent to a random classification (0.55). We will use the combinations $\langle 0.8, 0.01 \rangle$ (named journey-B for bigger difference) and $\langle 0.5, 0.1 \rangle$ (named journey-S for smaller difference) for the remaining comparisons.

	Prob. non-active			
Prob. active	**0.01**	**0.05**	**0.1**	**0.5**
0.5	1.00	1.00	0.85	0.55
0.8	1.00	0.95	0.85	0.70
1.0	1.00	1.00	0.95	0.80

Table 1: Classification accuracy in the customer journey domain varying the probability of appearing goals for the two kinds of customers, active and non-active.

The next results of the experiments are presented in Table 2. Rows represent the domains, and the columns are different lengths of the traces (horizons). The values correspond to the accuracy of CABBOT fixing all other parameters to their default values. The results show that CABBOT is able to correctly classify behavior traces in a high percentage of cases. We observe that we do not need a high number of traces nor lengthy traces to obtain good results. As expected, CABBOT had less accuracy in shorter traces, since it has observed less number of actions/states, so it is harder to correctly classify the behavior. In the case of the `journey`

domain, the longer traces allow for more goals to appear in the case of non-active customers, making the classification harder. Also, as it was observed before, the results with a smaller difference of probability values are worse than with a bigger difference, specially in the case of shorter traces' lengths.

	Length of traces				
Domain	**5**	**10**	**20**	**50**	**100**
terrorist	0.60	0.90	0.95	1.00	1.00
service car	0.60	0.95	1.00	1.00	1.00
journey-B	1.00	0.95	0.85	0.80	0.95
journey-S	0.45	0.80	0.60	0.95	0.85
digital-journey	0.75	0.85	0.90	1.00	1.00
AML	1.00	1.00	1.00	0.90	0.95

Table 2: Classification accuracy in different domains varying the length of the trace.

Table 3 shows the results when we vary the similarity function. As we can see, the accuracy is perfect in most cases for all domains except for the customer journeys one. Even if the intention when generating the two kinds of behavior was to include slight differences, the learning system is able to detect those by using the different similarity functions. In the case of the `journey` domain, we can see that the actions-based distance obtains better results than the one based on comparing goals. Since this domain has many different goals, when goals appear the traces differ more on the goals than on the actions achieving the goals. Also, the similarity function used does not affect much in this domain to differentiate between bigger (B) or smaller (S) probability differences.

	Similarity function			
Domain	d_a	d_Δ	d_g	d_r
terrorist	1.00	1.00	0.95	0.90
service car	0.50	1.00	1.00	1.00
journey-B	1.00	0.50	1.00	0.80
journey-S	0.75	0.50	1.00	0.95
digital-journey	1.00	0.95	1.00	1.00
AML	1.00	1.00	1.00	1.00

Table 3: Classification accuracy in different domains varying the similarity function.

CABBOT can make on-line classification of traces as soon as observations are made. Table 4 shows the average number of observations before making the final classification decision when varying the similarity function. While in the `AML` and `service car` domains, it takes a small number of steps to make the final decision, the number of steps required in the other two domains is higher. This is specially true in the case of the `journey` domain for the same reasons discussed above; i.e. goals could take some time to appear.

Table 5 shows the results when we vary the probability

	Similarity function			
Domain	d_a	d_Δ	d_g	d_r
terrorist	4.20	6.40	26.40	15.90
service car	0.00	2.90	26.20	0.70
journey-B	15.90	7.30	17.90	2.30
journey-S	25.00	25.00	16.40	11.10
digital-journey	2.80	10.60	10.55	5.90
AML	2.30	1.40	5.25	1.60

Table 4: Average number of observations before making the final classification decision when varying the similarity function in several domains.

of partial observability. We can see that when the probability of making an observation at a given time step decreases, so does the accuracy of the learning system and correspondingly the number of steps it takes the learning system to converge to the final classification increases. In the extreme, when the probability is 0.01 for a length of history of 50, the traces will at most consist of one or two elements, so classifying the traces becomes a hard task as shown by the low probabilities. The rate at which the accuracy decreases varies across domains. In the case of `AML`, `digital-journey` and `journey-B` domains, there is a slow decrease in accuracy. In the other three domains, the drop in accuracy is more acute starting at even a probability of observation of 0.5 in the `terrorist` domain.

	Probability of partial observations			
Domain	**1.0**	**0.5**	**0.1**	**0.01**
terrorist	0.95	0.45	0.50	0.50
service car	1.00	1.00	0.85	0.45
journey-B	0.90	0.90	1.00	0.60
journey-S	0.85	0.85	0.80	0.45
digital-journey	0.90	0.75	0.55	0.45
AML	1.00	0.90	0.80	0.35

Table 5: Classification accuracy in different domains varying the probability of partial observability.

Table 6 shows the results when we vary the probability of an execution failure of individual action (degree of non-determinism). When an action fails, it stays in the same state. Since the length of the history is 50 steps, even if some actions fail, CABBOT is still getting enough observations to make accurate classifications.

Related work

Given some sequence of events, there have been several learning tasks defined: sequence prediction (what the next step is going to be) (Bernard and Andritsos 2019); sequence generation (learning to generate new sequences, e.g. simulation); sequence recognition (determine whether the sequence is legitime or belongs to a given type); sequential decision making (how to make decisions over time, e.g. plan-

Domain	Probability of execution failure 0.0	0.2	0.4
terrorist	0.95	0.90	0.80
service car	1.00	1.00	1.00
journey-B	0.90	0.95	0.95
journey-S	0.90	0.95	0.80
digital-journey	0.70	0.85	0.75
AML	1.00	1.00	1.00

Table 6: Classification accuracy in different domains varying the probability of individual action execution failure. In parenthesis, the number of steps until it converges to the final decision.

ning). This paper deals with sequence recognition or classification.

This task has been addressed by using different types of techniques (Xing, Pei, and Keogh 2010) based on: features, distances or models. Features can be the presence or frequency of k-grams for all grams of size k. Model-based assumes an underlying probabilistic model and learns the parameters (Naive Bayes, HMM, ...). In our case, the number of symbols in the alphabet is huge (if groundings), so computing conditional probabilities is intractable, or is very small (action schemas) and probably not useful. Otherwise, we would have to rely on domain knowledge to know, for instance, that the transaction amounts (not part of the actions) are relevant, or the sum of amounts of several consecutive transactions. So, we have opted to use a distances-based approach. Our learning task is also related to detecting anomalous behavior or outliers detection (Chandola, Banerjee, and Kumar 2010; Gupta et al. 2013) where the techniques are the same ones. The main difference with respect to previous work is that their definition of traces is very simplistic in most cases: small number of action labels; no representation of state nor goals; and they do not handle relational data. From the point of view of classical automated planning, there has been related work on goal/plan recognition (Ramírez and Geffner 2010). However, as we discussed in the introduction, the task we deal with here is not about predicting the goal/plan, but about classifying a given behavior in a set of behavior classes.

We have used several similarity functions such as the ones based on Jaccard distance or RIBL. Other similarity functions have been defined in related tasks, such as process mining (Becker and Laue 2012), plan diversity (Roberts, Howe, and Ray 2014) or plan stability (Fox et al. 2006) (see (Ontañón 2020) for an extensive review). These previous similarity functions used mainly the actions in the plan, but did not include the corresponding states.

Some of our domains have been analyzed previously by similar approaches: understanding customer journeys in the field of marketing (Lemon and Verhoef 2016); predicting an on-line buy action from the sequence of clicks (Bertsimas, Mersereau, and Patel 2003); process mining (van der Aalst 2016); intrusion detection in a computer network or system (Scholau et al. 2001); or anti-money laundering (Lopez-Rojas and Axelson 2012). None of them used a representation based on planning tasks, nor any relational learning approach. So, their approaches relied on carefully selecting the features to be used for defining the learning examples.

Conclusions

We have presented four main contributions. The first contribution consists of posing the sequence classification task in terms of a richer representation framework than previous work. We use goals, states and actions to include the traces rationale in the traces description. The second contribution consists of a learning technique that takes a set of training traces of other agents's behavior and can classify later traces in different classes. The third contribution is a simulator that generates synthetic behaviors where agents can dynamically change their goals, and therefore their plans. Execution of those plans is stochastic, so those agents are able to monitor the execution and replan when needed. Finally, the fourth contribution is a set of automated planning domains that can be used for comparison in future work. Experimental results show that this approach performs well in some domains, including variations of real finance-related domains.

Acknowledgements

This paper was prepared for information 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. The authors thank Alice Mccourt for her useful revision of the paper. The authors would like to thank Sameena Shah for her discussions on the applications of this work to the AML task.

References

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence* 275:104–137.

Avrahami-Zilberbrand, D., and Kaminka, G. 2005. Fast and complete symbolic plan recognition. In *Proceedings of IJCAI-05*.

Becker, M., and Laue, R. 2012. A comparative survey of business process similarity measures. *Computers in Industry* 63(2):148–167.

Bernard, G., and Andritsos, P. 2019. Accurate and transparent path prediction using process mining. In *European Conference on Advances in Databases and Information Systems*, 235–250. Springer.

Bertsimas, D.; Mersereau, A.; and Patel, N. 2003. Dynamic classification of online customers. In *Proceedings of the 2003 SIAM International Conference on Data Mining*.

Borrajo, D., and Veloso, M. 2012. Probabilistically reusing plans in deterministic planning. In *Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain-Independent Planning*, 17–25.

Borrajo, D.; Veloso, M.; and Shah, S. 2020. Simulating and classifying behavior in adversarial environments based on action-state traces: An application to money laundering. In *Proceedings of the 2020 ACM International Conference on AI in Finance*.

Chandola, V.; Banerjee, A.; and Kumar, V. 2010. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering* 24(5):823–839.

de la Rosa, T.; García-Olaya, A.; and Borrajo, D. 2013. A case-based approach to heuristic planning. *Applied Intelligence* 39(1):184–201.

Dzeroski, S., and Lavrac, N. 2010. *Relational Data Mining*. Springer.

Emde, W., and Wettschereck, D. 1996. Relational instance based learning. In Saitta, L., ed., *Machine Learning - Proceedings 13th International Conference on Machine Learning*, 122 – 130. Morgan Kaufmann Publishers.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, 212–221.

Fuentetaja, R.; Borrajo, D.; and Linares-López, C. 2010. A look-ahead B&B search for cost-based planning. In Meseguer, P.; Mandow, L.; and Martínez-Gasca, R., eds., *Current Topics in Artficial Intelligence, CAEPIA 2009 Selected Papers*, volume LNAI 5988 of *Lecture Notes on Artificial Intelligence*, 201–211. Sevilla (Spain): Springer Verlag.

García-Durán, R.; Fernández, F.; and Borrajo, D. 2012. A prototype-based method for classification with time constraints: A case study on automated planning. *Pattern Analysis and Applications Journal* 15(3):261–277.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239–290.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning. Theory & Practice*. Morgan Kaufmann.

Gupta, M.; Gao, J.; Aggarwal, C. C.; and Han, J. 2013. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering* 26(9):2250–2267.

Guzmán, C.; Alcázar, V.; Prior, D.; Onaindía, E.; Borrajo, D.; Fdez-Olivares, J.; and Quintero, E. 2012. PELEA: a domain-independent architecture for planning, execution and learning. In *Proceedings of ICAPS'12 Scheduling and Planning Applications woRKshop (SPARK)*, 38–45. Atibaia (Brazil): AAAI Press.

Jaccard, P. 1901. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bulletin de la Société Vaudoise des Sciences Naturelles* 37:547–579.

Jiménez, S.; de la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4):433–467.

Keren, S.; Karpas, E.; and Gal, A. 2014. Goal recognition design. In *Proceedings of ICAPS'14*.

Lemon, K. N., and Verhoef, P. C. 2016. Understanding customer experience throughout the customer journey. *Journal of Marketing: AMA/MSI Special Issue* 80:69–96.

Lopez-Rojas, E. A., and Axelson, S. 2012. Money laundering detection using synthetic data. In *The 27th workshop of Swedish Artificial Intelligence Society (SAIS)*, 33–40.

Ontañón, S. 2020. An overview of distance and similarity functions for structured data. *Artificial Intelligence Review* 1–43.

Pozanco, A.; Escudero, Y.; Fernández, S.; and Borrajo, D. 2018. Counterplanning using goal recognition and landmarks. In *Proceedings of IJCAI'18*, 4808–4814.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, 1121–1126. AAAI.

Roberts, M.; Borrajo, D.; Cox, M.; and Yorke-Smith, N. 2018. Special issue on goal reasoning. *AI Communications* 31(2):115–116. DOI: 10.3233/AIC-180754.

Roberts, M.; Howe, A. E.; and Ray, I. 2014. Evaluating diversity in classical planning. In *Proceedings of ICAPS*.

Scholau, M.; DuMouchel, W.; Ju, W.-H.; Karr, A. F.; Theus, M.; and Vardi, Y. 2001. Computer intrusion: Detecting masquerades. *Statistical Science* 16(1):58–74.

Tax, N.; Verenich, I.; La Rosa, M.; and Dumas, M. 2017. Predictive business process monitoring with lstm neural networks. *Lecture Notes in Computer Science* 477–492.

Torreño, A.; Onaindia, E.; Komenda, A.; and Stolba, M. 2017. Cooperative multi-agent planning: A survey. *ACM Computing Surveys* 50(6):84:1–84:32.

van der Aalst, W. 2016. *Process Mining: Data Science in Action*. Springer.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Xing, Z.; Pei, J.; and Keogh, E. 2010. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter* 12(1):40–48.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9:683–718.

Probabilistic Policy Reuse for Similarity Computation Among Market Scenarios

Enrique Martínez, Javier García and Fernando Fernández
Universidad Carlos III de Madrid
Avenida de la Universidad, Madrid, Spain
fjgpolo@inf.uc3m.es

Abstract

Probabilistic Policy Reuse (PPR) is a transfer learning approach to improve a reinforcement learner with guidance from previously learned similar policies. It uses the past policies as a probabilistic bias where the learner chooses among the exploitation of the ongoing learned policy, the exploration of random unexplored actions, and the exploitation of past policies. An interesting side-effect of PPR is its capability of revealing the similarity between the past and new task, which has interesting applications from a financial point of view. In fact, in this paper we propose to use PPR to identify the similarity among different market scenarios in the context of pricing. Identifying these similarities allows us 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. We show how the degree of similarity affects the transfer efficacy. To carry out this study, in this paper we focus on the pricing problem in the insurance industry using a synthetic dataset provided by BBVA, one of the largest Spanish companies in the banking sector.

Introduction

Pricing is the determination of the price at which a product or service will sell or the yield at which bonds will sell as new issues (Phillips 2005; Hinterhuber 2017). If the price is set too high or the yield is set too low, the issue will not sell out. If the price is set too low or the yield is set too high, the issuer will pay more than necessary in dilution or interest to sell it. Therefore, pricing is a challenging problem because of the huge number of factors influencing the pricing policies: customer characteristics, product features, risk, price sensitivity, etc. Recently, Machine Learning and, in particular, Reinforcement Learning has proposed to learn pricing policies (Krasheninnikova et al. 2019). However, such policies are adapted to a particular market scenario, and it is not possible to easily adapt them to the changing circumstances of the environment. For instance, let us assume we have a pricing policy adapted to a particular scenario in which the general trend by customers is to accept high prices. This policy will have a nefarious performance if an economic crisis occurs and customers no longer accept such high prices. In this new scenario, it is required to relearn a new pricing policy that better suits to the new circumstances. Typically, this new policy can be learned from scratch, but what we propose in this paper is to use the pricing policies of past market scenarios to learn a new policy that adapts to the new circumstances of the environment.

This problem has a direct mapping to transfer learning (Taylor and Stone 2009; Da Silva and Costa 2019). The key idea of transfer is that experience gained in learning to perform one *source* task can help to improve learning performance in a related, but different, *target* task (Taylor and Stone 2009). In this paper we pursue to apply this idea for transferring the knowledge among different market scenarios. In this manuscript, we report on Probabilistic Policy Reuse (PPR) (Fernández and Veloso 2006) as transfer approach. When solving a new problem, PPR utilizes the past policies as a probabilistic bias where the learner faces three choices: the exploitation of the ongoing learned policy, the exploration of random unexplored actions, and the exploitation of past policies. As a past policy becomes relevant to solving a new task, such effective reuse reveals the similarity between the past and new task. From a financial point of view, such similarities allows us to identify how a market is similar to another, and thus how a pricing policy learned in a market scenario is useful to learn in a related, but different, market scenario. The accuracy of the transfer depends on how similar the *source* markets and the *target* market are. Therefore, the main contributions of this manuscript are: (i) show empirical results about how the similarity metric among market scenarios work, and (ii) to demonstrate how PPR uses these similarities as a probabilistic bias in the exploration/exploitation process. In this paper, we focus on the pricing problem of an insurance company in order to demonstrate the benefits of reusing pricing policies among different market scenarios. For this we have the data provided by the insurance division of BBVA, one of the largest Spanish companies in the banking sector.

The organization of the paper is as follows. Section *Background* offers the theoretical background on pricing and the basis of RL required to understand the proposed research. Section *Mapping Pricing onto Reinforcement Learning* for-

mulates the problem of pricing as an MDP. Such modeling requires defining of all the elements of an MDP: the state and action spaces, and the reward and the transition functions. Section *Policy Reuse* presents the π-reuse exploration strategy used to learn in the proposed MDP, and the PRQ-learning algorithm. Finally, Section *Experimental Results* shows some results and Section *Conclusions* concludes and introduces future research.

Background

This section provides a detailed description of the problem to be solved by RL, and the basis of RL.

Pricing

Generalized Linear Models (GLM) is the standard in the pricing industry (Murphy, Brockman, and Lee 2000) as it is easy to fit and interpret. GLMs are a rich class of statistical methods that include linear regression, logistic regression, log-linear models, poisson regression and multinomial response model (Agresti 2015). From a mathematical point of view, pricing can be solved using GLM as a general approach. For instance, we might use standard linear regression to learn a model to predict the price for a given customer and product. We can define a GLM (Nelder and Wedderburn 1972; Lee and Nelder 2003) as a relation between dependent and independent variables which takes the following form:

$$\varphi = g(\mu) = E(Y) = \mathbf{X}\beta \tag{1}$$

where g is the link function, $\mathbf{X}$ is the model matrix and β is the vector of unknown parameters, with Y being independent and distributed as some exponential family distribution. Note that the standard linear regression model is a special case of GLM where the link function, g, is the identity function.

While setting a price for a given customer and product can be solved with standard linear regression models, the pricing strategy optimization deals with two main components: on the one hand, the pricing model represented by Equation (1) and, on the other, the customer's degree of acceptance of a given price (sensitivity to the price).

Customer sensitivity represents a valuable piece of information to correctly adjust the price and simulate several probabilistic scenarios. As the grade of acceptance of a given price can be represented as a probability function, it can use a logistic regression model, i.e., a GLM with a logistic link function (Germán 2007) to represent the probability of renewal for a given price. The distribution of the target variable (i.e., if the customer will renew the policy) is set to be the binomial distribution. The mean of the binomial distribution, that is, the prediction generated by the model, is the probability that the event will occur. Equation (2) represents the $logit$ function (Kleinbaum and Klein 2010):

$$\eta = \ln\left(\frac{\rho}{1-\rho}\right) = \mathbf{X}\beta \tag{2}$$

where η is the logit or log-odd function and ρ is the probability of renewal, $\mathbf{X}$ is the model matrix and β is the vector of unknown parameters. Knowing the GLM model parameters, β, we can obtain the probability of renewal directly from the above equation:

$$\rho = \frac{\mathrm{e}^{\mathbf{X}\beta}}{1+\mathrm{e}^{\mathbf{X}\beta}} = \frac{1}{1+\mathrm{e}^{-\mathbf{X}\beta}} \tag{3}$$

The logistic regression model in Equation (3) predicts the probability acceptance for a given renewal ratio and customer. The renewal ratio, v, is the ratio at which customers renew the price of their insurance. This renewal ratio decrements the price when $v \in [0.0, 1.0)$ or increments the price when $v \in (1.0, 2.0]$. Therefore, when v is equal to 1.0 it represents neither incremental nor decremental, i.e., the same price of renewal.

As we saw, the optimization of a pricing strategy requires handling two main goal functions:

1. Revenue

$$f_1(v) = \sum_{i \in v} \mathcal{B}(\rho_i(v_i)) * price_i * v_i \tag{4}$$

where $\mathcal{B}$ represents the Bernoulli distribution for probability ρ_i of acceptance of a given renewal ratio v_i for the i-th customer in the portfolio, and $price_i$ is the current price paid by the i-th customer in the portfolio to the insurance company with $f_1 \in R$.

2. Retention

$$f_2(v) = \frac{1}{N}\sum_{i \in v} \rho_i(v_i) \tag{5}$$

where ρ_i is the probability of acceptance of a given renewal ratio v_i for the i-th customer in the portfolio, and N is the total number of customers in the portfolio with $f_2 \in [0, 1]$.

Therefore, although GMLs are the standard in the industry to model the pricing problem as described in Equation 1, such modelization adopts a static view of the problem and treats it as a linear prediction problem. Furthermore, the addition of the probability of customer acceptance transforms the problem into a stochastic problem which makes it difficult to solve it as a linear problem. For these reasons, what we suggest in this paper is precisely that it is more appropriate to model the pricing problem as a sequential stochastic decision problem as described in Section *Mapping Pricing onto Reinforcement Learning*.

Additionally, we assume the underlying model dynamics are completely unknown for the learning agent. This makes necessary the use of other types of techniques such as those based on *model-free* RL. In this paper, the proposed approach is able to go *model-free* and learn to price from data generated from a dynamic environment. In fact, the proposed approach fits in better with what happen in the real life, in which the insurer offers renewal prices to a sequence of customers whom he attends one by one.

Finally, one of the important points is the consideration of the business constraints that prices may have. In our case, we consider such constraints C_i are defined for each customer i in the portfolio. It represents the ranges in which each renewal ratio can be chosen, i.e., $v_i \in [\mathrm{C}_{i0}, \mathrm{C}_{i1}]$. These ranges can be adjusted by taking into account the company's objectives, marketing campaigns, changes in legislation or even the competence of other insurance companies. As an example, the lower limit of these ranges can be reduced to offer lower renewal prices in response to a market of aggressive competition or marketing campaigns that aim to retain the highest number of customers.

Reinforcement Learning

A RL environment is typically formalized by means of a Markov Decision Process (MDP) (Sutton and Barto 2018). A MDP consists of a set of states S, a set of actions A available from each state, the reward function $R : S \times A \rightarrow \Re$ which assigns numerical rewards to transitions, and transition probabilities $T : S \times A \times S \rightarrow [0, 1]$ that capture the dynamics of a system. The goal is to learn a policy π, which maps each state to an action, such that the return $J(\pi)$ is maximized:

$$J(\pi) = \sum_{k=0}^{K} \gamma^k r_k \tag{6}$$

where r_k is the immediate reward received in step k, and γ is the discount factor and affects how much the future is taken into account (with $0 \leq \gamma \leq 1$). We assume that the interaction between the learning agent and the environment is broken into episodes, where K is a time instant at which a terminal state is reached, or a fixed length for a finite horizon problem. Traditional methods in RL, such as TD-learning (Sutton and Barto 1998), typically try to estimate the return (sum of rewards) for each state s when a particular policy π is being performed. This is also called the value-function $V^\pi(s) = E[J(\pi)|s_0 = s]$.

The value of performing an action a in a state s under policy π is represented as $Q^\pi(s, a) = E[J(\pi)|s_0 = s, a_0 = a]$- this value represents the estimated return, i.e. sum of rewards, the system will receive when it performs action a in the state s, and follows the policy π thereafter. The Q-function is also called the action-value function. The Q-learning algorithm (Watkins 1989) is one of the most widely used for computing the action-value function. Given any experience tuple of the type $< s, a, s', r >$ - where s is a state, a is an action, s' is the state achieved when executing a from s, and r is the immediate reward - it updates the Q-function following Equation 7

$$Q^\pi(s, a) \leftarrow Q^\pi(s, a) + \alpha(r + \gamma \max_{a'} Q^\pi(s', a') - Q^\pi(s, a)) \tag{7}$$

where γ is the discount factor, and α is a learning rate. To correctly approximate the Q-function, the Q-learning algorithm uses an exploration strategy (e.g., ϵ-greedy, softmax) as a balance between the exploration of random unexplored actions and the exploitation of the ongoing learned policy (Tijsma, Drugan, and Wiering 2016). In domains with a discrete state-action space it is usual to use a tabular representation of the Q-function. Such Q-tables have as many rows as states existing in the domain, and as many columns as actions that can be executed in each of these states. Each of the positions in the Q-table is the value of the Q-function for the corresponding state and action.

Mapping Pricing onto Reinforcement Learning

Once the background has been introduced, this section formally presents from a RL perspective the problem this paper deals to. The problem considered in this paper is as follows: The insurer company has a portfolio of customers. Then, when it is time to renew the customer prices, the insurer takes the first client from the portfolio and makes a decision: which renewal price to offer him/her taking into account the current situation of the company (i.e., current revenue, retention). Whether the customer accepts the renewal price or not, the insurer's decision leads to the company to a new situation (e.g., if the customer does not accept the renewal price, the insurer company will have one customer less and, hence, lower revenue). Additionally, we can know if the insurer's decision has been good or bad (e.g., if the insurer's decision increases the revenue we can consider that the decision has been good, and bad otherwise). Thus, one can take into account the utility of each particular decision. Then, given the new situation of the company, the insurer takes the next customer from the portfolio, makes a decision, and so on until the insurer makes a decision for each of the customers in the portfolio. In this way, the problem can be seen as a succession of situations (states), decisions (actions), and utilities (rewards).

Therefore, this task can be modeled as an MDP. In fact, this task is an episodic task, where each episode has as many steps as customers N are in the portfolio. Therefore, each one of these episodes can be considered as a real-life renewal period, in which the company renews one by one all the customers in its portfolio. For each step n, the learning agent receives a state s_n. In this paper, a state s_n is a tuple in the form $s_n = \langle f_1^n, f_2^n, tier_n, price_n \rangle$, where f_1^n and f_2^n are respectively the revenue and the retention at step n, and $tier_n$ and $price_n$ are respectively the customer segment and the insurance price for customer n in the portfolio. Therefore, each state s_n is composed of global features relating to the status of the insurance company (i.e., f_1 and f_2) and customer features (i.e., $tier$ and $price$). From a RL point of view, it is also important to note that the customer segmentation can be considered as a way of compacting several customer features into only one feature already provided by the insurance company, $tier$, which allows reducing the state space. Therefore, the features $tier$ and $price$ perfectly describes the particular situation of a given customer. In respect to the global features, we have included in the state description those directly related to the considered objectives.

For the first state of each episode, s_1, the initial revenue and retention are computed as in Equations (8) and (9), i.e., we start from a situation where we have the profit gener-

ated by all the customers belonging to our portfolio (Equation (8)), but we also assume that we are going to lose some of these customers even offering them the same price, which is simulated by a renewal ratio of 1 (Equation (9)). This starting point is, in fact, the one that best fits the starting point in the real world.

$$f_1^1 = \sum_{i=1}^{N} price_i \tag{8}$$

$$f_2^1 = \frac{1}{N} \sum_{i=1}^{N} \rho_i(1) \tag{9}$$

At state s_n the agent performs an action a_n. In this paper, we consider action a_n is the renewal ratio v_i for the current customer n. The renewal ratios for each client are limited with the constraints applied to each customer, $[C_{n0}, C_{n1}]$, as explained in Section *Pricing*. This is to say that each customer has their own constraints for renewal ratios. Therefore, a_n is sampled from $[C_{n0}, C_{n1}]$.

After performing an action a_n in state s_n, the agent transits to a new state $s_{n+1} = \langle f_1^{n+1}, f_2^{n+1}, tier_{n+1}, price_{n+1} \rangle$. Therefore, a transition function is required in order to compute the values for f_1^{n+1} and f_2^{n+1}. These values are respectively computed using Equations (10) and (11) where $n = 1, \ldots, N$.

$$f_1^{n+1} = \sum_{i=1}^{n} \mathcal{B}(\rho_i(a_i)) * price_i * a_i + \sum_{i=n+1}^{N} price_i \tag{10}$$

$$f_2^{n+1} = \frac{1}{N} \left(\sum_{i=1}^{n} \rho_i(a_i) + \sum_{i=n+1}^{N} \rho_i(1) \right) \tag{11}$$

Both Equations (10) and (11) are divided into two main parts: the first takes into account the customers for whom a renewal ratio a_n has already been selected in the episode, and the second the customers in the portfolio for whom this renewal ratio has not yet been selected. For the latter, we assume a renewal ratio of 1. In this way, Equation (10) and (11) remain consistent with Equations (8) and (9). It is important to be aware of the fact that the transition function only affects the f_1 and f_2 part of the state space, i.e., to the global situation of the company, since no matter what action the learning agent takes in step n, $tier_{n+1}$ and $price_{n+1}$ will be the customer segment and the insurance price of the next customer in the portfolio.

Finally, when the learning agent performs an action a_n in a state s_n and moves to a state s_{n+1}, it also receives a reward signal r_n. Therefore, the definition of a reward function is also required. In this paper, we formulate a reward function for each of the two proposed strategies to optimize the renewal price: the maximization of the revenue, and the maximization of the revenue subject to the retention does not fall below a given threshold. The former strategy is related to an MDP with a single criterion to be optimized. In this case, the reward function is formulated as in Equation (12), i.e. at step n the reward is the difference between the value of revenue function in state s_{n+1} and state s_n.

$$r_n = (f_1^{n+1} - f_1^n) \tag{12}$$

It is important to be aware of the fact that an interesting property of this model is that the competence is indirectly modeled by means of the probability of customer acceptance ρ used in Equations 9 and 11. The probability ρ is related to the prices offered by other companies.

Policy Reuse

In this section, we describe the basic algorithms of Policy Reuse. We first describe how to reuse just one past policy by the π-reuse exploration strategy. Then, we show how to reuse a set of past policies by PRQ-Learning. It is important to bear in mind that these algorithms have been previously proposed by Fernández and Veloso (Fernández and Veloso 2006). This section reprises the descriptions given in that work for ease of reference.

π-reuse

As mentioned previously, this paper transfers the pricing policies among different market scenarios by using π-reuse. The π-reuse strategy is an exploration strategy able to bias a new learning process with a past policy. 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 direct RL method 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 13.

$$a = \begin{cases} \Pi_{past}(s) & \text{w.p. } \psi \\ \epsilon - greedy(\Pi_{new}(s)) & \text{w.p. } (1 - \psi) \end{cases} \tag{13}$$

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. Algorithm 1 shows a procedure describing the π-reuse strategy integrated with the Q-Learning algorithm.

The procedure gets as an input the past policy Π_{past}, the number of episodes K, the maximum number of steps per episode H,and the ψ parameter. An additional υ parameter is added to decay the value of ψ in each learning step. The procedure outputs the Q function, the policy, and the average Reuse Gain obtained in the execution, W, which will play an important role in similarity assessment.

PRQ-learning

PRQ-learning is depicted in Algorithm 2. The algorithm gets as input: a new task to solve Ω; the policy library $L = \{\Pi_1, \ldots, \Pi_n\}$ composed of n past policies that solve n different tasks, respectively; the temperature parameter of

ALGORITHM 1: π-reuse

Input: Π_{past}, P, H, ψ, υ
Output: $Q^{\Pi_{new}}(s,a)$, Π_{new}

Initialize $Q^{\Pi_{new}}(s,a) \leftarrow 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$, $\psi_0 \leftarrow \psi$;
repeat
  Set the initial state, s, randomly;
  $h \leftarrow 1$;
  repeat
    With a probability of ψ_p, $a \leftarrow \Pi_{past}(s)$;
    With a probability of $1-\psi_p$, $a \leftarrow \epsilon$-greedy$(\Pi_{new}(s))$;
    Receive the next state s', and reward, $r_{k,h}$;
    Update $Q^{\Pi_{new}}(s,a)$, and therefore, Π_{new};
    $Q^{\Pi_{new}}(s,a) \leftarrow (1-\alpha)Q(s,a)^{\Pi_{new}} + \alpha[r + \gamma \max_{a'} Q^{\Pi_{new}}(s',a')]$;
    $s \leftarrow s'$;
    $h \leftarrow h+1$;
    $\psi_{h+1} \leftarrow \upsilon \times \psi_h$;
  until $h < H$;
  $k \leftarrow k+1$;
until $k < K$;
$W = \frac{1}{K}\sum_{k=0}^{K}\sum_{h=0}^{H}\gamma^h r_{k,h}$

ALGORITHM 2: PRQ-learning

Input: Ω, L, τ, $\triangle\tau$, K, H, ψ, υ, γ, α
Output: $Q^{\Pi_\Omega}(s,a)$, Π_Ω

Initialize $Q^{\Pi_\Omega}(s,a) \leftarrow 0, \forall s \in \mathcal{S}, a \in \mathcal{A}$;
Initialize W_i and W_Ω to 0;
Initialize U_i and U_Ω to 0;
repeat
  Choose Π_k following the probabilities;

$$P(\Pi_j) = \frac{e^{\tau W_j}}{\sum_{p=0}^{n} e^{\tau W_p}} \qquad (14)$$

  if $\Pi_k == \Pi_\Omega$ **then**
    Q-Learning with fully greedy strategy;
  else
    π-reuse(Π_k,1, H, ψ, υ);
  end
  Compute reward R in episode k;
  $W_k = \frac{W_k U_k + R}{U_k + 1}$;
  $U_k = U_k + 1$;
  $\tau = \tau + \triangle\tau$;
  $k \leftarrow k+1$;
until $k < K$;

the softmax policy selection equation τ, and a decay parameter $\triangle\tau$; and a set of previously defined parameters: K, H, ψ, υ, γ and α.

The algorithm initializes the new Q function to 0, as well as the estimated reuse gains W_i and W_Ω of the policies. Then the algorithm executes the K episodes iteratively. In each episode, the algorithm decides which policy to follow with a softmax strategy which uses the values W_Ω and W_i with the temperature parameter τ (Equation 14). In the first iteration, all the policies have the same probability to be chosen, given that all W_i values are initialized to 0. Once a policy is chosen, the algorithm uses it to solve the task, updating the Reuse Gain for such a policy with the reward obtained in the episode, and therefore, updating the probability to follow each policy. If the policy chosen is Π_Ω, the algorithm follows a completely greedy strategy. However, if the policy chosen is Π_i (for $i = 1, \ldots, n$), the π-reuse action selection strategy, defined in previous section, is followed instead. In this way, the Reuse Gain W_i of each of the past policies can be estimated on-line with the learning of the new policy. After executing several episodes, it is expected that the new policy obtains higher gains than reusing the past policies, so it will be chosen most of the time.

It is important to note that the Reuse Gain W_i measures the usefulness of reusing the policy Π_i to learn the new policy Π_Ω, where the most useful policy to reuse, Π_k, from the Library, $L = \{\Pi_1, \ldots, \Pi_n\}$, is the one that maximizes the Reuse Gain when learning such a task. In fact, we can compute the distance between a past policy Π_i and the new policy Π_Ω as defined in Equation 15.

$$d(\Pi_i, \Pi_\Omega) = W_\Omega - W_i \qquad (15)$$

We define this distance not by direct comparisons between the policies, but comparing the result of applying them.

Experimental Results

This section presents the experimental results collected from the use of the PRQ-learning algorithm in the renewal price adjustment problem. First, we describe the dataset provided by BBVA with the synthetic information about the customers and their probabilities of acceptation. Then, we describe the market scenarios used for transferring pricing policies among them. Later, we describe the parameter setting, and finally the results.

Synthetic Dataset

The real data that we have from BBVA are a dataset of 83200 decisions with 11 features from a home insurance portfolio. We will use it to validate the proposed framework of usage of RL. However, during the research stage we will use a synthetic dataset in which only the main features of the real dataset are taken into account. The selection of these features has been proposed by insurer experts from BBVA who have discarded those features that were not relevant for the considered task. Additionally, the values for the customer' features in the synthetic dataset are generated by drawing from a model fitted to the real data, ensuring that the synthetic and the original dataset have the same statistical properties, but without compromising the privacy of the original dataset. The final synthetic dataset has 7 features as shown in Table 1. Table 1 shows the feature names, the feature types (numeric or nominal), the mean value and standard deviations of the numeric features (with the notation $A \pm B$, being A the mean and B the standard deviation), and a brief description of the corresponding feature.

Table 1: Synthetic dataset

Feature	Type	Mean	Description
price	Numeric	207.1 ± 89.1	Insurance price
tier	Nominal	–	Financial segment of customer
v	Numeric	1.1 ± 0.1	Renewal ratio
$\rho(v)$	Numeric	0.7 ± 0.2	Probability of customer acceptance
C	Numeric	$[0.9 \pm 0.1; 1.1 \pm 0.1]$	Range renewal ratio
gain	Numeric	224.7 ± 97.9	$price * v$
class	Nominal	–	Price acceptance $\sim \mathcal{B}(\rho(v))$

Therefore, in this synthetic dataset we consider five main features for each customer: $price$, $tier$, v, ρ, and C. Feature $price$ is the price given from Equation 1, i.e., the insurance price. $tier$ is a well known feature in the insurance sector that indicates a financial segment of a customer. Customer segmentation is usually performed through a clustering process considering several customer features (e.g., age, number of claims during the last year) (Abolmakarem, Abdi, and Damghani 2016). In this way, in order to make a decision for a given customer, the company does not analyze each one of its particular characteristics, but simply the segment to which it belongs to. It is expected that all customers of the same group react in the same way to company decisions. Therefore, companies can even customize the decisions for each segment (e.g., marketing strategies, commercial plans, renewal prices). In this paper, the segment to which a customer belongs to is provided by the insurance company and it indicates a different customer risk. In particular, we consider four segments and, accordingly, we consider four values for this feature: $tier1$, $tier2$, $tier3$ and $tier4$, where the greater the value of $tier$, the greater the risk of the customer. Feature v is the renewal ratio offered to the customer and $\rho(v)$ is an artificially created probability of customer acceptance for that renewal ratio used for experimentation purposes. Finally, feature C are the business constraints, i.e., the range in which the renewal ratio v can be chosen for that particular customer. The features $gain$ and $class$ are derived from the previous features. In particular, $gain$ is the new insurance price offered to the customer (i.e., $gain = price * v$) and $class$ is the renewal acceptance class sampled from $\sim \mathcal{B}(\rho(v))$. Figure 1 graphically represents the synthetic dataset including these features. Note that, in Figure 1, higher prices are applied to customers with higher risk. Both variables $price$ and ρ exhibit a non-lineal increment.

This synthetic dataset is used for two purposes. The first is to train the logistic regression function ρ described previously. For training, the selected input attribute is the feature v in Table 1, and the outcome attribute is the feature $class$. If $class = 1$ it means that the customer accepts the new price, if $class = 0$ s/he doesn't. After training, the function ρ is able to predict for a given increment the grade of acceptance. The function ρ will be used to model the

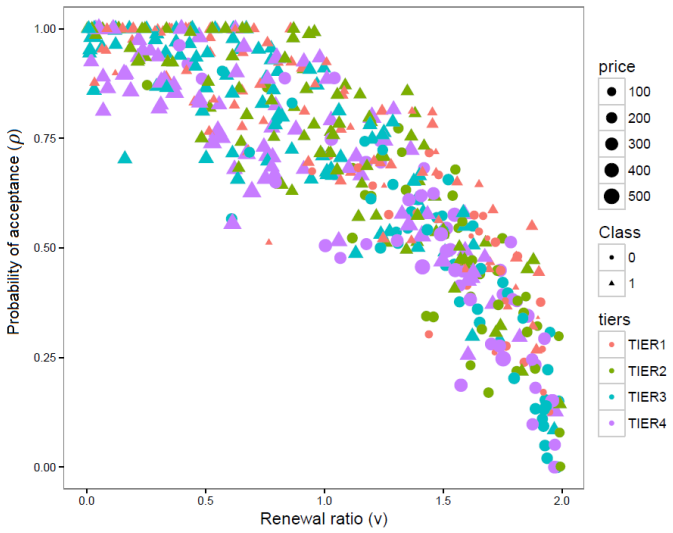

Figure 1: Synthetic dataset with $price$, $tier$, v, ρ and $class$.

Pricing Strategy Optimization problem as an MDP as described previously. Secondly, we assume that the customers in this synthetic dataset are the customers in our portfolio, and their features ($price$ and $tier$) are part of the description of the state space. Then, each of these customers is considered in a different state as described in Section *Mapping Pricing onto Reinforcement Learning*. Therefore, such customers are used to create the synthetic MDP described in Section *Mapping Pricing onto Reinforcement Learning*, in which the agent travels for learning. It is important to note that with these two ingredients we have what it takes to build the price renewal simulator used in the proposed experimentation.

Market Scenarios

We have configured different market scenarios in which the difference among all of them is the probability of acceptance ρ of the renewal ratios v by the customers. Figure 2 shows the probability ρ of acceptance of renewal ratios for each of the proposed market scenarios.

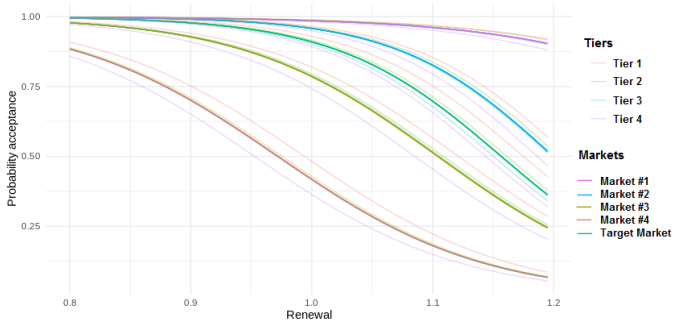

Figure 2: Probability ρ of acceptance of a given renewal ratio v in different market scenarios.

In particular, Figure 2 shows the probability of acceptance

ρ for four market scenarios, and a *target* market scenario. *Market* $\#1$ is the *easiest* market where customers tend to accept the renewal price offered to them, regardless of the quantity. Instead, *Market* $\#4$ is the *hardest* market where the opposite situation exists: customers tend to reject the renewal price offered to them. The rest of the market scenarios consider intermediate acceptance situations. Additionally, it is important to note that for each market scenario the probability of acceptance depends both the renewal ratio (ρ will be lower, the higher the renewal price υ), and to the segment of risk the customers belongs to, i.e., to the *tier* (ρ will be lower, the higher the *tier*). Figure 2 the probability of acceptance of the *target* market scenario, i.e., the market used as the target in the experimentation.

Parameter setting

In the experiments, we have used different parameters. For instance, in all cases, we set the initial values of the Q function to zero. This is a optimistic initialization of Q, since expected average rewards tend to be under zero in the long-term. In the case of PRQ-learning, the parameter ψ also start with a value of 1.0 and exponentially decay every step with $\psi = \psi \times \upsilon$, where the value of υ is 0.997. As a result, the probabilities of selecting a random action, the transferred policy or the policy being learning in current episode is shown in Figure 3. At the beginning of each episode, there is a higher probability of performing actions of the past policy, along with random actions, but as the learning process proceeds, there is a higher probability of selecting the actions proposed by the new policy.

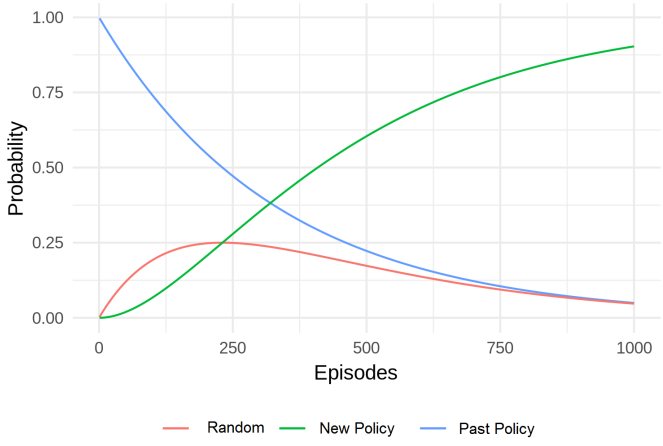

Figure 3: π-reuse probabilities

Finally, we have run the algorithms 5 times due to the stochastic nature of the learning processes. Therefore, we describe the average results.

Results

This section shows the results collected from the use of π-reuse for learning a pricing policy in the *target* market scenario using the pricing policies learned in each of the *source* market scenarios described in Section *Market Scenarios*. The first step, is however, learning from scratch the previous policies in the *source* market scenarios.

Learning from Scratch in the *Source* Markets Scenarios Figure 4 shows the four learning curves of applying Q-Learning from scratch in the four source markets scenarios defined above.

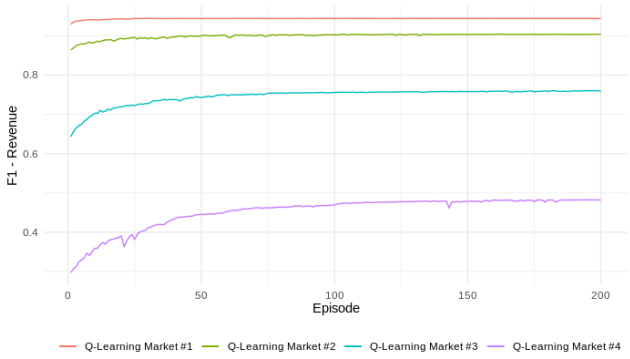

Figure 4: Four Q-Learning processes corresponding to learning from scratch in *Market* $\#1$, *Market* $\#2$, *Market* $\#3$ and *Market* $\#4$, respectively.

Figure 4 shows that for *Market* $\#1$ where acceptance probabilities are higher, the performance achieved is quite high, while for markets with low acceptance rate, final performance is smaller. In this case, exploration strategy used was very soft, and a linear reduction of the ϵ parameter was set to ensure get closer to optimality. All the previous learned policies can be transferred to new learning processes, as will be described below.

Computing the Reuse Gain with PRQ-learning The goal of PRQ-learning is to learn an action policy Π_Ω in the *target* market scenario Ω. The policies learned in the *source* market scenarios are the past policies used as a probabilistic bias in the new exploratory process in the *target* market scenario, i.e., $L = \{\Pi_1, \Pi_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6\}$, where Π_i is the policy learned in the task Ω_i corresponding to the *Market* $\#i$, with $1 \leq i \leq 4$. In the library L, there are two additional policies, Π_5 and Π_6, corresponding to always offering the minimal renewal rate and the maximum renewal rate, respectively.

The process of learning the most similar policy is illustrated in Figure 5. Figure 5 (a) shows the evolution of the Reuse Gain computed for each policy involved, W_1, ..., W_6, and the gain W_Ω. Such as Reuse Gains are computed according to Equation in line 12 of Algorithm 2. On the x axis, the number of episodes is shown, while the y axis shows the gains. Initially, the Reuse Gain of all the policies is set to 0. After a few episodes, $W1$, $W2$ and $W3$ stabilize at around -0.023. However, $W4$ and $W5$ increases up to around -0.018. Instead, $W6$ stabilizes at around -0.028. These values demonstrate that the most similar policies are Π_4 and Π_5 corresponding to the policies of the *hardest* market scenario $Market\#1$ (Π_4) and to always offering the

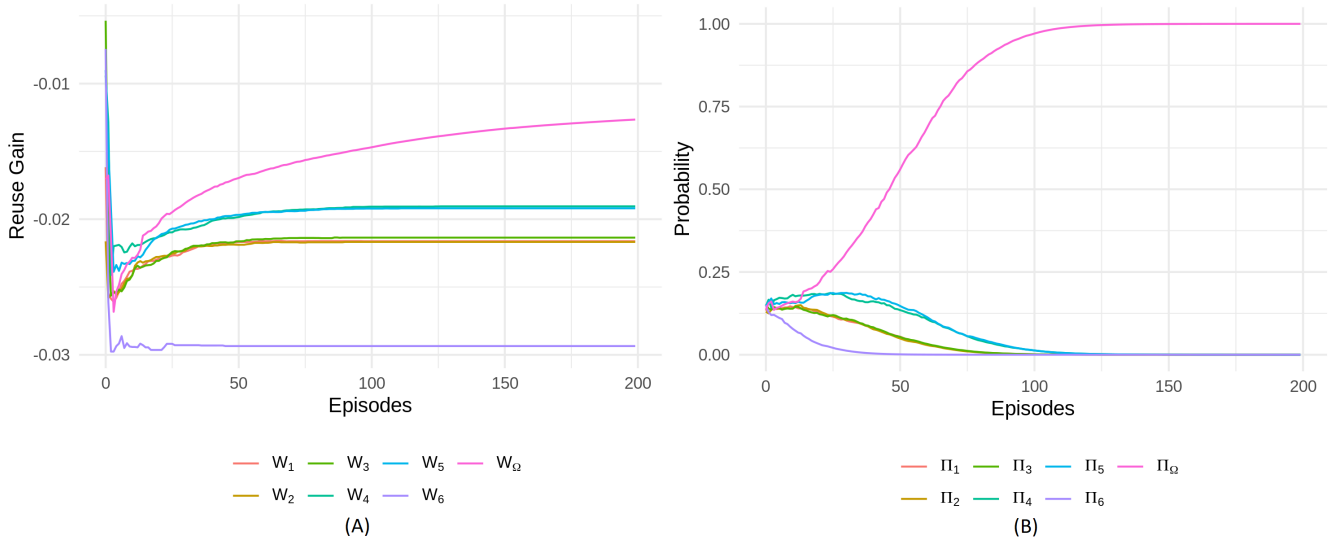

Figure 5: (a) Evolution of the reuse gains, $W_1, \ldots, W_\Omega$, and (b) evolution of the selection probabilities, $P(\Pi_1), \ldots, P(\Pi_\Omega)$, for each one of the policies involved.

minimal renewal rate (Π_5). It is important to bear in mind that the *hardest* market is the market where the acceptance rate of the client is lower and, where the most frequent actions are those that use lower renewal rates. Somehow, these results say that reusing where the renewal rate is very low (safe policies from the retention point of view), is better for the reuse, and it is worse when the renewal rates are higher. This is an interesting collateral result: a source policy is the best for reuse, not only because it is learned in the most similar task, but also because it is safer. In fact, the experiments demonstrate that the safest policies are the best for reuse.

The gain of the new policy, W_Ω, starts to increase around episode 20, achieving a value higher than -0.013 by episode 200, demonstrating that the new policy is very accurate. The values of the Reuse Gain computed for each policy are used to compute the probability of selecting them in each iteration of the learning process, using the formula introduced in Equation 14 (initial $\tau = 2$, and $\triangle\tau = 2$). Figure 5 (b) shows the evolution of these probabilities. In the initial steps, all the past policies have the same probability of being chosen given that the gain of all them is initialized to 0. While the gain values are updated, the probability of Π_4 and Π_5 grow. For the other past policies, the probability decreases down to 0. The probability of the new policy also increases, and after 20 episodes, it is bigger than the rest. After a few iterations more, it achieves the value of 1, given that its gain is the highest, as shown in Figure 5 (a).

Figure 5 (b) demonstrates how the balance between exploiting the past policies or the new one is achieved. It shows how in the initial episodes, the algorithm chooses to reuse the past policies to find the most similar. Then, it reuses the most similar policy until the new policy is leaned and improves the result of reusing any past policy. In summary, we can say that the PRQ-learning algorithm has demonstrated to successfully reuse a predefined set of policies, and how it can compute the reuse gain for each of the past policies.

Conclusions

In this paper we have provided a novel study about reusing pricing policies, linked to particular market scenarios, in a related, but different, market scenario. PRQ-learning is a powerful algorithm to improve learning performance in new tasks by reusing policies learned in previous tasks, as it has been shown in this manuscript. In the context of pricing, π-reuse has also been postulated as a successful technique that allows transferring knowledge from one market scenario to another. The paper also demonstrates how PRQ-learning can be used to compute similarity metrics between the past policies and the new one. The experiments demonstrate that policies tending to offer low renewal prices are better for the reuse. We consider that from a financial point of view creating policy libraries, reusing them as source for new learning processes, and computing similarity metrics is a promising research line.

In future work, we plan to use PPR in different financial problems. Also, in this scope, the concept of safety arises, since in these areas, it is interesting the use of learning processes that does not produce catastrophic situations (like a large number of clients that decline its renewal, reducing client retention), overall when we transfer policies from simulation to real environments.

Acknowledgments

This work was partially funded by a J.P. Morgan AI Faculty Research Award to Fernando Fernández, and grants TIN2017-88476-C2-2-R and RTI2018-099522-B-C43 of FEDER/Ministerio de Ciencia e Innovación - Ministerio

de Universidades - Agencia Estatal de Investigación. Javier García is partially supported by the Comunidad de Madrid funds under the project 2016-T2/TIC-1712.

References

Abolmakarem, S.; Abdi, F.; and Damghani, K. K. 2016. Insurance customer segmentation using clustering approach. *International Journal of Knowledge Engineering and Data Mining* 4(1):18–39.

Agresti, A. 2015. *Foundations of Linear and Generalized Linear Models*. Wiley Series in Probability and Statistics. Wiley.

Da Silva, F. L., and Costa, A. H. R. 2019. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research* 64:645–703.

Fernández, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 720–727.

Germán, R. 2007. Lecture notes on generalized linear models.

Hinterhuber, A. 2017. Implementing pricing strategies. *Journal of Revenue and Pricing Management* 17:51–80.

Kleinbaum, D. G., and Klein, M. 2010. *Logistic regression: a self-learning text*. New York: Springer, 3rd edition.

Krasheninnikova, E.; Garcia, J.; Maestre, R.; and Fernandez, F. 2019. Reinforcement learning for pricing strategy optimization in the insurance industry. *Engineering Applications of Artificial Intelligence* 80:8 – 19.

Lee, Y., and Nelder, J. 2003. Robust design via generalized linear models. *Journal Of Quality Technology* 35(1):2–12.

Murphy, K.; Brockman, M.; and Lee, P. 2000. Using generalized linear models to build dynamic pricing systems for personal lines insurance. In *Casualty Actuarial Society Forum*, 107–139.

Nelder, J., and Wedderburn, R. 1972. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)* 135(3):370–384.

Phillips, R. 2005. *Pricing and Revenue Optimization*. Stanford Business Books. Stanford University Press.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning : An Introduction*. MIT Press.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Second edition.

Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10(Jul):1633–1685.

Tijsma, A. D.; Drugan, M. M.; and Wiering, M. A. 2016. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*, 1–8.

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

An Exploration of the Use of AI Planning for Predicting Stock Market Movement

Sumit Mund and **Mauro Vallati** and **Thomas McCluskey**
School of Computing and Engineering,
University of Huddersfield, UK

Abstract

Reasoning about stock market and projecting its movement is of great interest to the investors and traders. Recent advances in the stock market field focused on designing automated agents for performing trading tasks. However, there is no diminishing interest in approaches for predicting fluctuations in the market and, more importantly, to reason on different scenarios and possibilities because these not only impact the automation but also important from risk management perspective even without automation.

In this work, we are exploring the use of plan recognition as planning approach via AI Planning technique to reason about the observations made from the market, and predict the direction of the market movement. The benchmark broad based stock market index being the representative of the overall market, we are using benchmark index, Nifty to project movement of Indian stock market. Our proposed approach considers the available domain knowledge and a sequence of observations derived from the overall market to produce a number of plans that could provide explanations for the observed behaviour as well as the prediction by computing posterior probability.

Introduction

Stock market movement is of particular interest to traders and investors. While value investors might be interested in broader movement of the market, traders, especially day-traders are interested in market movement on a daily basis and also intra-day basis. Being aware of the movement in advance can be useful for all from a risk management perspective, as well as for maximising profit and return of investments. At the same time, reasoning about the movement can be useful for both building trading strategy and risk management.

While stock market constitutes of many individual stocks, a broad-based index[1] involving a basket of prime stocks can represent the entire market and the change in the index value represents the market movement. Usually, market movements are predicted and reasoned by human analysts with the use of all or a combination of domain expertise/experience, technical analysis, macro-economic factor, daily news analysis, quantitative models etc. The movement prediction can consider different time horizons. It can focus on the next few minutes or hours (near-term), or it can focus on next day(s) or next a few weeks ahead (short-term). Long-term predictions are generally made in terms of months, a quarter or even year(s) ahead.

There are many studies and approaches to predict the market using market macros, fundamental and technical analysis and statistical or machine learning approaches (Nti, Adekoya, and Weyori 2019; Bustos and Pomares-Quimbaya 2020). While approaches with market macros and fundamental analysis to prediction involves cause and effect analysis and some form of reasoning, other approaches including the ones making use of news, mostly infer or find patterns from historical data and make prediction based on that. Prediction related to market macros and market events (news flow) can make use of AI planning techniques to do prediction based on a domain theory and we will explore that in this work.

There are cases where historical data is inadequate or every now and then we get into a new scenario which brings difficulty to relate to the near past. A nice example is the ongoing COVID19 pandemic, the mayhem it brought across the global financial markets and dramatic recovery in most of the broader markets recently. Usually when US federal reserve cut interest rates, the market goes up. But in a prevailing negative economic sentiments, when fed suddenly announced a rate cut, market went down[2] as it kinds of confirmed that the economy is indeed going downhill. While it might be difficult in inferring from historical data in such scenarios; with a proper domain theory by expert(s) these can be modeled to do prediction as well as to build trading strategies.

In this context, we explore the use of AI Planning techniques for predicting stock market movement in the near-term.

As a first exploratory step, we are aiming to predict and reason about the market movement (near-term) during the

[1] https://www.investopedia.com/terms/b/broad-basedindex.asp

[2] https://finance.yahoo.com/news/stock-market-news-live-updates-march-16-2020-220735000.html

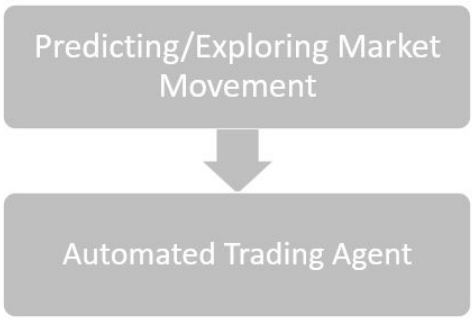

Figure 1: Overview of the roadmap towards a fully automated trading agent.

market session of a day. The ultimate goal is to develop a fully automated trading agent based on AI planning, that is making best use of domain knowledge, necessary data transformation, and available information. Notably, the proposed automated trading agent will not only be able to predict near-term market movement, but will also provide an ideal framework for analysing and comparing different scenarios and hypotheses. Market movement prediction or exploration with different hypothesis can be extended further to develop an automated trading agent incorporating risk management. An overview of the 2-step roadmap is provided in Figure 1.

In this exploratory study we are limiting the scope of considered movement by increase or decrease of current value of the index for the day, as compared to the close value of the trading session. If current value is more than close value then we will define it as DOWN otherwise UP. In our experiment we are using benchmark index, Nifty[3] to project movement of Indian stock market, however the same approach can be applied for any stock market represented by an index.

Our proposed approach not only makes prediction with a probability associated with it, but it will also be in the position to explain the observed events by forming a trajectory where a trajectory can be seen as series of event leading to the upcoming state of the market. Hence, in a sense, it provides a plausible reasoning for the prediction.

We take inspiration from the work (Sohrabi, Riabov, and Udrea 2016) and are using plan-recognition-as-planning technique to explain the observations, derived from stock market and predict the movement as UP or DOWN. Our prototype takes the domain knowledge, a sequence of observations derived from the overall market (and from the relevant news of the corresponding date), k as number of trajectories or explanations to produce. It then computes the posterior probability of the goal given the optimum trajectory using the costs of k trajectories that a top-K planner generates as a solution to the transformed planning problem.

In the rest of the paper, we first discuss plan recognition as planning problem, and we then introduce the Prediction problem ans solution through a top-K planner as a goal recognition problem. Finally, conclusions are given followed by a brief details on experiments and analysis.

[3]https://www.nseindia.com/products-services/indices-nifty50-index

Background on Plan Recognition as Planning

In their seminal work, Ramírez and Geffner (2009) used classical planning algorithm for plan recognition: Given a domain theory, through necessary problem transformation, the set G^* of goals G can be recognized which can explain a sequence of observations, provided there exists an optimal plan consisting of an action sequence π for both the goal G and the goal G extended with extra goals representing the observations.

This approach has been extended in (Ramírez and Geffner 2010) to a general problem of probabilistic plan recognition. Assuming that actions have deterministic effects and there is complete information about the initial state, the plan recognition can be solved efficiently using classical planners with input of a probability distribution over the set of goals. The posterior goal probabilities representing the costs of achieving the goal are estimated by calling a classical planner twice without further modification.

Sohrabi, Riabov, and Udrea (2016) enhanced the above and provided a foundation for our prediction problem. They introduced a plan recognition approach to address observations (including missing or noisy ones) over fluents. While in the above (Ramírez and Geffner 2010), it was required to have a probability distribution over the set of goals as input, in this case the authors approximated posterior probabilities by taking the total costs and normalizing over a combined costs of a set of plans generated by high-quality plans or diverse plans.

Before defining our problem we present the background definitions. We mostly rely on terminology and definitions introduced in (Sohrabi, Riabov, and Udrea 2016).

Definition 1 (Planning Problem) *A planning problem is a tuple $P = (F, A, I, G)$, where F is a finite set of fluent symbols, A is a set of actions with preconditions,* PRE*(a), add effects,* ADD*(a), delete effects,* DEL*(a), and action costs,* COST*(a), $I \subseteq F$ defines the initial state, and $G \subseteq F$ defines the goal state.*

A fluent can be true or false. A set of fluents with true values can be defined as a state, s. A sequence of actions, $\pi = [a_0, ..., a_n]$ is a solution or plan to the problem, P if it maps initial state, I to the goal state, G. Each action, a needs to have a non-negative cost, $COST(a)$ and cost of the plan or solution will be $COST(\pi) = \sum COST(a)$. A plan π is optimal if it has minimum cost.

Definition 2 (Plan Recognition Problem) *A plan recognition problem is a tuple $R = (F, A, I, O, \mathcal{G},$ PROB*), *where (F, A, I) is the planning domain as defined above, $O = [o_1, ..., o_m]$, where $o_i \in F, i \in [1, m]$ is the sequence of observations, $\mathcal{G}$ is the set of possible goals $G, G \subseteq F$, and* PROB *is the goal priors or a probability distribution over $\mathcal{G}$.*

The observations are sequenced, or ordered and each observation is an observable fluent. In real world, actions may not be directly observable; for example the exact action(s) causing the price of a stock to rise may not be observable. However, the observations over fluents as some of the effects of observation(s) can be observed; for example stock price movement as an observation.

An observation can fall into three categories. A sequence of observations is *satisfied* as explained or discarded by an action sequence and its execution trace if there exists a mapping through a non-decreasing function from observation indices to the state indices . An observation can also be *noisy* or *missing* (Sohrabi, Riabov, and Udrea 2016). For a given goal and plan, an observation can be *noisy* or unexplained if it has not been added to the state. When an observation is added but is not observed then it would be categorised as *missing* observation. For a given plan and goal, an observation can be categorised as one and for other plans or goals, the same may fall into other categories.

The solution to the plan recognition is the posterior probability $P(\pi|O)$ and the posterior probability of goal, $P(G|O)$ given a set of observations, O.

Using the weighted factor, $V_{O,G}(\pi)$ and other calculations in (Sohrabi, Riabov, and Udrea 2016), the posterior probability of goal, $P(G|O)$ given a set of observations can be as follows:

$$V_{O,G}(\pi) = COST(\pi) + b_1.M_{O,G}(\pi) + b_2.N_{O,G}(\pi)$$

where π *is a plan that meets the goal* G *and satisfies* O, $M_{O,G}(\pi)$ *is the number of missing observations in* O, $N_{O,G}(\pi)$ *is the number of noisy observations in* O, *and* $b1$ *and* $b2$ *are the corresponding coefficients assigning weights to the different objectives.*

$$P(G|O) = \beta \left[1 - \frac{\beta^{'} V_{O,G}(\pi)}{\sum_{\pi^{'} \in \Pi} V_{O,G}(\pi^{'})} \right]$$

where β and $\beta^{'}$ are normalizing constants.

Prediction Problem

Next, we explain the prediction problem. We extended the plan recognition theory(Sohrabi, Riabov, and Udrea 2016) to model the prediction problem. The transformed planning problem allows for explanation of the observations as well as the prediction which would be the recognised goal or the most probable goal.

The Prediction Problem can be presented as $PP = (F, A, I, O, \mathcal{G}, K)$, where (F, A, I) is the planning domain as in the previous section, $O = [o_1, ..., o_m]$, where $o_i \in F, i \in [1, m]$ is the sequence of observations, $\mathcal{G}$ is the set of possible but finite goals $G, G \subseteq F$, K is the number of paths to produce. In this case, $\mathcal{G} = [market\_up, market\_down]$.

Here, the set of goals, $\mathcal{G}$ are part of the problem definition, but probability distribution over $\mathcal{G}$ is not included in the definition. However, the solution need to find posterior probabilities $P(G|O)$ given a sample of K path or trajectories. It can be treated as a goal recognition problem and its solution will be a sample of K path or trajectories (Sohrabi, Riabov, and Udrea 2016) from which the posterior probabilities $P(G|O)$ can be calculated as in the previous section.

Finding k Plans

Similar to (Sohrabi, Riabov, and Udrea 2016), we compute the k plans using a top-k planner, capable of finding a given set of plans with high quality. Alternatively, top quality planning (Katz, Sohrabi, and Udrea 2020), diverse planning (Katz, Sohrabi, and Udrea 2020) or recent top-k planner with symbolic search (Speck, Mattmüller, and Nebel 2020) can also be used.

A top-k planner (Riabov, Sohrabi, and Udrea 2014) solves the problem of finding k plans of highest quality. Depending on k, the output of top-k plans can be a set of optimal plans, or a mix of optimal plans and sub-optimal plans. The top-k planning planner, *TK** uses a k shortest paths algorithm called K^* (Aljazzar and Leue 2011).

```
(define (problem mkt-prob-1)
    (:domain market)
    (:objects Positive_Economy_Sentiment Negative_Economy_Sentiment
       Fed_Interest_Rate_Cut Market_Up Market_Down)
    (:init
        (SOURCE Positive_Economy_Sentiment)
        (SOURCE Negative_Economy_Sentiment)
        (TARGET Market_Up)
        (TARGET Market_Down)
        (CONNECTED Positive_Economy_Sentiment Fed_Interest_Rate_Cut)
        (CONNECTED Fed_Interest_Rate_Cut Market_Up)
        (CONNECTED Negative_Economy_Sentiment Fed_Interest_Rate_Cut)
        (CONNECTED Fed_Interest_Rate_Cut Market_Down)
        (= (total-cost) 0)
        (= (discard-cost) 100)
        (= (starting-cost Positive_Economy_Sentiment) 10)
        (= (starting-cost Negative_Economy_Sentiment) 1)
        (= (connected-cost Positive_Economy_Sentiment Fed_Interest_Rate_Cut) 5)
        (= (connected-cost Negative_Economy_Sentiment Fed_Interest_Rate_Cut) 5)
        (= (connected-cost Fed_Interest_Rate_Cut Market_Up) 5)
        (= (connected-cost Fed_Interest_Rate_Cut Market_Down) 1)
    )
    (:goal (and
        (__goal-achieved)
        )
    )
    (:metric minimize (total-cost))
)
```

Figure 2: Sample PDDL Problem (Partial Encoding)

Path / Trajectory	Cost
negative_economy_sentiment -> fed_interest_rate_cut -> **market_down**	7
negative_economy_sentiment -> fed_interest_rate_cut -> **market_up**	11

Figure 3: Trajectories

Experiment and Analysis

We have modelled the market domain in a generic way using the PDDL (McDermott et al. 1998) classical planning language. In this way, a problem can be formulated with different kinds of observations derived from overall market, news and others including findings from financial analysts. The domain modelling can be done similar to (Sohrabi et al. 2018). We have extended the published benchmark PDDL domain[4] for our requirement. The domain PDDL describes in a generic way to traverse a graph from identified start locations (SOURCE) to goal locations (TARGET).

Partial encoding of the sample problem is shown in Fig.2. The mentioned costs here are arbitrary and for illustration and can be adjusted or learned from data. We have used the Top-k planner (Katz et al. 2018) for the solution. Starting with negative economic sentiment, it generates two trajectories; one to each goal as shown in figure 3. As the trajectory leading to market down having the shortest cost, the probability of market being down given the trajectory will be high and hence the prediction.

[4]https://github.com/IBM/risk-pddl/blob/master/domain.pddl

This is an ongoing work and we are continuing with experimenting and empirical analysis.

It is noteworthy here is that our prediction approach is different from the typical statistical inference or machine learning modeling which purely makes prediction from the historical data. In our case, we model the domain knowledge and reason about the world through the recent series of observations and make predictions using classical planning techniques. However, there are instances where we have captured statistical parameters and relationships based on historical data as part of capturing our domain knowledge.

Conclusion

We proposed a prototype which can predict the immediate future or goal state given a series of past observations. For this we have achieved capturing the domain by PDDL through knowledge engineering. We have formulated the prediction problem for Stock Market Movement and produced the solution with necessary modification from existing solutions. This is a fist step with a prototype for our ongoing research and development towards an automated trading agent. Similar technique can be employed to predict and reason about market movement for short term and even long term. This can form the base to proceed developing an agent capable to automated trading with inbuilt risk management.

In this work, we have proposed an idea of developing an automated trading agent and tried to produced a roadmap. We have built a prototype as a first step to lay the foundation for an automated agent. As part of the prototype developed, we applied existing works in plan recognition as planning (Sohrabi, Riabov, and Udrea 2016), state projection (Sohrabi, Riabov, and Udrea 2017) and scenario planning (Sohrabi et al. 2018) to Stock market domain and achieved predicting market movement by solving the prediction problem. As part of knowledge engineering, we modeled the stock market domain using PDDL and transformed raw data into observations.

Acknowledgements

We sincerely thank Shirin Sohrabi from IBM Research for all her help and support.

References

Aljazzar, H., and Leue, S. 2011. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence* 175(18):2129–2154.

Bustos, O., and Pomares-Quimbaya, A. 2020. Stock market movement forecast: A systematic review. *Expert Systems with Applications* 1134–64.

Katz, M.; Sohrabi, S.; Udrea, O.; and Winterer, D. 2018. A novel iterative approach to top-k planning. In *Proceedings of the Twenty-Eigth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press.

Katz, M.; Sohrabi, S.; and Udrea, O. 2020. Top-quality planning: Finding practically useful sets of best plans. In *AAAI*, 9900–9907.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL-the planning domain definition language.

Nti, I. K.; Adekoya, A. F.; and Weyori, B. A. 2019. A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review* 1–51.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI International Joint Conference on Artificial Intelligence*.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Riabov, A.; Sohrabi, S.; and Udrea, O. 2014. New algorithms for the top-k planning problem. In *Proceedings of the Scheduling and Planning Applications woRKshop (SPARK) at the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 10–16.

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 *IJCAI*, 3258–3264.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2017. State projection via AI planning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Speck, D.; Mattmüller, R.; and Nebel, B. 2020. Symbolic top-k planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9967–9974.

Managing Risks to Assets in Corporate Finance with NLP and Planning

Biplav Srivastava[1], **Javid Huseynov**[2]
[1]AI Institute, University of South Carolina, Columbia, SC 29208
[2]IBM Chief Analytics Office, Armonk, NY, USA 10504

Abstract

Corporate finance refers to how a company manages its own money, i.e., its source of funds, expenses and long-term investments. In contrast to other areas of finance like investment banking and customer relationship, corporate finance has drawn little attention from AI researchers. In this position paper, we highlight how Natural Language Processing techniques and planning can help a company identify risks and take action to preserve its financial assets.

Introduction

Artificial Intelligence (AI) is impacting all aspects of global economy including financial services. For example, in (McWaters and Galaski 2018), the study surveys the impact of AI on capital markets, market infrastructure, deposits and lending, insurance, payments and investment management. It finds that AI is leading to change in operations at financial companies providing new opportunities to innovate on services, collaborate on data, but they need to engage workforce and look at new ways to stay competitive.

However, there is little guidance offered by previous AI studies on corporate finance. Managing finances is an important corporate function at any company. As a simplified view, it deals with how a company raises its funds, utilizes its funds to invest in its business, earns revenue and pays its vendors and employees, and manages cash flow to keep the business running.

In this position paper, we explore how the AI techniques of natural language processing (NLP) and planning can be useful to improve corporate finance. We review methods applicable for multi-national companies, introduce a novel use-case of how a company can move its money efficiently anticipating risks and discuss practical considerations in getting them adopted by financial specialists.

The paper is organized as follows: we start by describing a hypothetical multi-national corporation that wants to manage its finances. Inspired by actual deployments at a large company, we discuss how available AI technologies may be relevant for the hypothetical company. Then, we introduce a new scenario of detecting sovereign risk (e.g., political risk in a country) and in response, moving money across the hypothetical company's different entities efficiently using planning. We conclude by discussing features that financial specialists value from AI technologies in this domain.

Analyzing Risks Scenarios in a Hypothetical Multi-National Company

Consider a hypothetical company, *Hypo Inc.*, which is based in one country but does business around the world using subsidiaries in each major market or country. *Hypo Inc.* will be booking sales, paying vendors, maintaining bank accounts and paying taxes in all regions it does business. As a corporate function, it would have teams of risk and financial specialists looking at events around the world for signs of risks and how to respond to them.

Identifying Risks from News

In such a setting, news sources act as a powerful data source to detect risks by using techniques like Natural Language Understanding (NLU) from NLP. Typical data sources used for building the capability, namely, curated news or web crawls and firmographic knowledge bases, are publicly or commercially available. By detecting entities and analyzing for topics, in theory, one can observe for signs of risks.

But there are two primary challenges in developing a solution. Firstly, evolving news topics can vary widely by context over time. Thus, topic classification based on supervised learning, which is the predominant approach in literature, is not scalable or extensible when dealing with a volume of over 25 million news feeds per month. Secondly, there are over 160 million active business entities (companies) across the world. Accurate identification of these entity mentions amidst the contextually diverse and fast moving news streams requires complex yet efficient methodology.

We describe an actual system that has addressed these challenges at a large corporation[1]. The system is implemented and demonstrable as a web application as seen in Figures 1 and 2. The user selects an entity (*Chase* Bank) one may be interested in and a time range of choice. As seen in Figure 1, the system responds by showing news with mentions of the entity, details of any news the user has selected with all entities detected, and their sentiments along

[1]For business reasons, the identity of the system is anonymized.

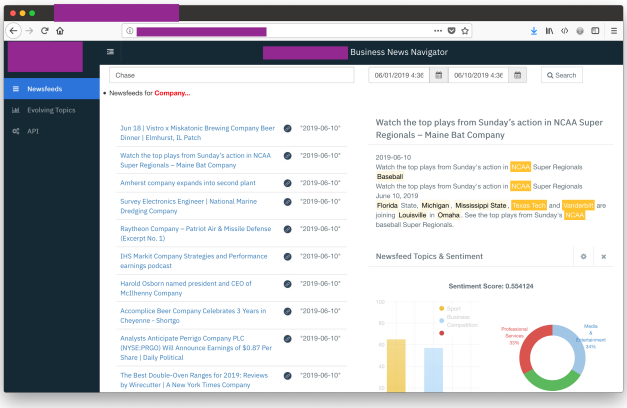

Figure 1: Looking for an entity (*Chase* Bank) in a time range.

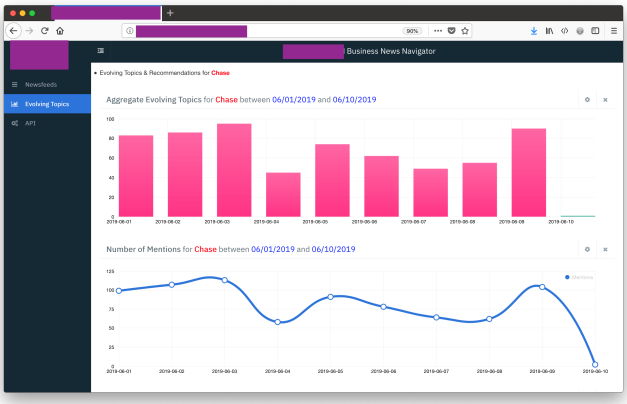

Figure 2: Evolution of entity (*Chase* Bank) in a time range.

user-defined categories. Figure 2 shows the evolution of the entity's mentions over time.

The system allows results to be *filtered* based on a domain-specific taxonomy that specialists prefer - a usability consideration we will return to in discussion section. The risk specialist interprets entity information to identify business risks (Jolly 2003) like sovereign risk (risk about country), reputation risk, competition risk, compliance risk (risk from new rules) and financial risk.

Identifying Scenarios from Detected Observations and Risks

Once risks have been identified, they can be used to investigate emerging scenarios. In most companies, this is a manual task.

However, an exciting application of planning has emerged recently to identify enterprise risk. Specifically, in (Sohrabi et al. 2018), the authors describe a system which, given observed events, can project them into future to generate what-if scenarios. Internally, the system first builds action models from textual description of the domain. Then, given observations of risks, the system formulates the scenario generation task as a plan recognition problem to be solved with planning techniques which optimally explain the observations (Sohrabi et al. 2019).

Planning Asset Movements to Mitigate Risks

We now introduce a usecase about how a multi-national company can react to risks to its financial assets held in a foreign country. Consider the case where one of the countries where *Hypo Inc.* operates in, *Country-1*, is promoting new investments into it and giving tax break while another country, *Country-2*, is planning to introduce new taxes for money held by corporations in banks. *Hypo* would like to move its assets away from *Country-2* in the short-term and consider *Country-1* as a future investment opportunity. Another case may be when *Hypo* wants to sell some of its business in a foreign country and wants to brings the proceed back to its home country with minimum overheads.

In situations like these, the current approach is for multi-disciplinary teams of financial analysts, tax experts and lawyers to collaborate on the laws of different countries related to companies, taxes and markets, and recommend course of actions. *Hypo*'s business leaders will consider the recommendations while making actual decisions. It is not uncommon for a large company to have entities in 150+ countries, in 30+ currencies and move asset in tens of billions of dollars. So, movement of assets involves tens of people and involves months of manual planning.

We consider the problem of determining efficient movement of assets as a planning problem. The domain consists of predicates specifying companies and their entities, the type of assets they hold and in which currencies, and how the assets can be moved. Examples of actions are:

- Convert currency between two currencies in a country
- Transfer money across countries within a company in a given currency
- Move money from a subsidiary to home company at a given location

The problem file uses the domain predicates and one can instantiate in which country a company's entities are, in which currencies its entities hold assets and where the company wants its assets to go. In Figure 3, we show a very small situation of a company having assets in US and non-US (foreign) location and with assets in two currencies. There are two plans found by an off-the-shelf planner - the left plan calls for money to be first moved from subsidiary entity's foreign country to home country (US), then converted to US dollar in US, and finally transferred from subsidiary to home company in US. The right plan calls for the money to be converted to US dollars in the foreign location, then the money be moved from subsidiary to home company but at foreign location, and finally the money be moved within home company from foreign location to US.

In this domain, AI planning can help with:

- generating one or more plans as desired

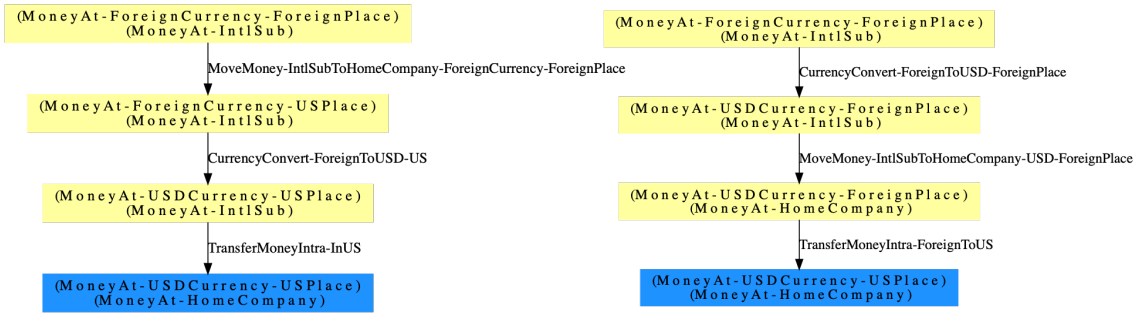

Figure 3: Two plans to move assets.

- showing the most optimal plan (least cost, risk, duration, . . .) based on user's choice
- help expert create a plan interactively
- learn patterns and preferences that an organization prefers
- given a transfer patterns that a company wants to avoid (for whatever reason), produce plans consistent with those constraints

Discussion

In previous sections, we have outlined how AI methods can help in corporate finance activities. However, users in the corporate finance domain have raised hurdles addressing which can make them use the techniques more conveniently. We discuss some of them here.

1. Evolution of entities over time and their inter-relationships: Users want to understand how entities of interest change over time and want to verify the system's results with their own world knowledge of events. Furthermore, they are more convinced of system's capabilities if they see that the system has extracted inter-related concepts (like a company and its subsidiaries) since user's specification is often partial. These inputs guided the design of system shown in Figures 1,2.

2. Explanation of risks identified and underlying events and entities: Extracted entities, time and location (space) from news can be processed by rules to identify simple risks, while the information can be made available to specialists to identify risks comprehensively. The specialists rely on explanation of risks among themselves and also any by the system.

3. Visualization of plans and interaction interfaces: Regardless of how the plans are produced (e.g., manually, with a planner or mixed), users want to explore how a plan will unravel over time. This is because users want to understand the implications when a plan will be *executed* in practice and impact real-world entities. Plans generated by planners may help specialists but because of the financial implications of actions, they do not want to accept results despite theoretical results of a planner's search being sound and complete.

4. Manual oversight over end-to-end integration: In theory, it may be possible to integrate and automate the process of identifying entities from news, identify risks from extracted entities and events, consider scenarios with highest certainty and priority(Sohrabi et al. 2018), and then taking subsequent actions to promote longer term goals. In practice, these activities are performed by different teams who specialize in the respective domain. Manual oversight is highly valued throughout the process in companies like *Hypo*.

Conclusion

In this paper, we explored the role on AI techniques of NLP and planning to address select problems in corporate finance - how a company manages its own funds. This problem can be particularly complex for multi-national companies with hundreds of their own subsidiaries, vendors and customers, and doing business and investment in many currencies around the world. Although AI can lead to significant impact, there are also practical issues for human-AI collaboration that need to be overcome.

References

Jolly, A. 2003. Managing business risk: A practical guide to protecting your business. In *Kogan Page Limited. pp. 6–7. ISBN 0-7494-4081-3*.

McWaters, R. J., and Galaski, R. 2018. The new physics of financial services - understanding how artificial intelligence is transforming the financial ecosystem. In *World Economic Forum (WEF) report, at http://www3.weforum.org/docs/WEF_New_Physics_of_Financial_Services.pdf*.

Sohrabi, S.; Riabov, A.; Katz, M.; and Udrea, O. 2018. An ai planning solution to scenario generation for enterprise risk management. In *32nd Conference on Artificial Intelligence (AAAI-18)*.

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. In *AI Communications 32(1), 1-13*.