Fall 2021

Eliciting & Visualizing Bias in Hiring Practices

Tsitsi Mambo
Bard College

Eliciting & Visualizing Bias in Hiring Practices

Senior Project Submitted to

The Division of Science, Math, and Computing

of Bard College

by

Tsitsi Mambo

Annandale-on-Hudson, New York

December 2021

Acknowledgments

I would like to thank my advisor, Keith O'Hara, for inspiring and encouraging me throughout my time here at Bard. I truly appreciate your patience, guidance, and all the lessons you have taught to prepare me for life after Bard.

I want to thank my Posse, without whom I would not have ever found myself at Bard.

I would also like to thank The Bois. I love you guys. You have made my time at Bard one of my happiest memories and have truly been such an amazing and inspiring support system. I will forever cherish my time here with all of you and the adventures we went on.

Finally, thank you to my family for your endless support and love.

Abstract

This project seeks to develop a way to elicit and visualize bias in the hiring process through the use of Markov Decision Processes, a mathematical framework for modeling decision processes. Three forms of the simulation: User-defined, Random, and Q-learning, were created and their policies were analyzed and compared. Heat Map and Donut Pie visualizations are utilized to present the Policies created from the Models. This project is designed to display the decisions as a form of countering bias during the hiring process.

Contents

Acknowledgments

Abstract

1 Introduction **1**

1.1 Purpose **2**

1.2 Ethics in Computing 2

1.3 Related Work 6

1.3.1 Socio-Anthropological Works 6

1.3.2 Technical Works 9

2 Background **12**

2.1 Machine Learning: Supervised vs Unsupervised vs Reinforcement Learning 13

2.2 Markov Decision Process 15

2.3 Components of Markov Decision Processes 17

2.3.1 Stochastic vs Deterministic 18

2.3.2 Value Iteration 18

2.3.3 Q-learning 20

2.4 Levenshtein and Hamming Distance 22

3 Implementation and Method **25**

3.1 Tools 26

3.2 Decision Model 26

3.3 Transition Function 30

4 Results and Analysis **31**

4.1 Analysis 32

5 Conclusion **47**

5.1 Areas of Impact 47

5.2 Limitations 47

5.3 Experimental Proposal 49

1 Introduction

Over the past several decades, there have been rapid advancements in technology. Simultaneously, the social landscape has seen a rise in support for change across many areas of daily life. One area that has seen constant calls for change is the professional realm, where more people are pushing for the elimination of bias and prejudice as a means to achieve equality in the workplace. The goal of this project is to create a module that identifies the areas where bias exhibits itself during the hiring process as a way of countering it in individuals with the power to enact change. Through identifying these prejudices, each individual can address the areas where their bias manifests and dismantle them. By changing one area in the structure of the hiring process, a cascading effect can result where the initial algorithms that are developed to find ideal candidates will then have less biased data being utilized for the inputs. Therefore, the output can represent a product that has been mitigated for bias. The identification of the areas where bias persists will also help determine which of the aspects that are being taken into consideration might need to be reconsidered. Some studies have suggested the possible omission of names, addresses, and alma maters in the résumé of individuals as forms to aid in the effort to reduce areas where bias manifests itself[2, 9]. This is an ongoing area of debate however as the opposing side also believes that only through the use of these social categorizing identifying factors will companies be able to identify areas where bias exists[ ]. Nevertheless, one of the complexities involved in accounting for bias is identifying the areas that it exists on an individual basis.

1.1 Purpose

This project explores the implementation of a Markov Decision Process (MDP) through game design as a means to replicate and model decision-making behavior. The game-based design follows the approach outlined in "Game Design For Eliciting Distinguishable Behavior."[5] The focus is on applying MDPs to parameterize the game design, utilizing Levenshtein distance to calculate the player reward function, and finally, visualizing the results in various ways to communicate the results discovered effectively. The technical hiring process has evolved to be an intricate and often complicated system. There is potential for change to be made in all areas. However, this project specifically focuses on the résumé review, when candidates have been passed through the selection algorithm and are being reviewed by hiring managers. It seeks to further evaluate the individual, nuanced, biases that are exhibited by potential employers.

1.2 Ethics in Computing

As computing has seen widespread adoption in daily life and application of individuals as well as by organizations, many have come forward with concerns surrounding the rapid growth seen in technology. There have been questions related to the extent of the ethical measures taken in the creation of these technologies. These questions relate to the gathering of demographic data and the lack of accountability for many of the unethical results that are obtained. Understanding the societal impact of the recent evolution of computing, and how it relates to the hiring process,

continues to be an area of interest. The significance of which proves to be of great importance in the equitable distribution of knowledge and power.

In the realm of hiring at technology companies, many companies go through various steps that seek to evaluate, categorize, and place qualified candidates in positions that will allow them to perform at high levels and contribute to the company in more positive ways. From a candidate's perspective of the hiring process, they first submit a résumé and sometimes a cover letter. This can be preceded by meeting with a recruiter via a referral who will then ask for a résumé. From there they are called for a behavioral interview followed by a technical interview. These can happen in any order and for some companies, candidates will see anywhere from two to six technical/behavioral interviews total. After the interview process, if the candidate passes all the evaluations, they will receive an offer that they can choose to negotiate or accept. With such an extensive, and often varied, interview process, and with some companies receiving anywhere from 20-60 résumés a day, the process can be resource-intensive[13,15]. With the explosion of technology in the last several decades, many organizations seek ways to employ it as a cost-reducing measure in various functions of their businesses. Biased decisions increase the efficiency of decision-making, so one does not have to spend copious amounts of time making decisions; instead, they default to biases. This bias can begin even before the interview stage of an application.

Technology companies' use of biased algorithms to select résumés during the hiring process is biased due to the utilization of biased data. When the algorithms are first developed, they make use of data that has been collected based on the current employees and the successes that a company has seen among them. This leads to bias especially if a company is not diversely

represented. The social capital that employers are looking for in their candidates contributes to the hiring decisions that they make. This contributes to an inbreeding bias in the hiring process.

To accommodate the extensive amount of résumés they receive, many companies make use of résumé parsing engines. Big tech companies such as Google receive upwards of 2M+ applications each year [18]. To account for the drastic increase in applicants, they have employed the use of such algorithms in their hiring process. Algorithms serve as the first point of filter for candidates that determine whether the résumés that candidates submit meet the requirements that the company is looking for. These are designed to filter out the candidates who do not possess the skills that the company is searching for. If a résumé fails to contain some of the necessary skills and keywords that a position requires, they send the candidate an automated rejection email. These algorithms do not function transparently or clearly which leads to them being referred to as black-box algorithms.

Black-box algorithms, as used in AI and machine learning, refer to the use of a predictive model whose creation was derived from an algorithm that was run and rerun from inputs. The algorithm recreates itself so often that when looking at how it computes, the original creators are no longer able to understand how it functions. These algorithms often take input from human users and use this input in their unknown calculations to reproduce an algorithm that will be desirable and output desired results. These algorithms have been a source of much debate regarding their accuracy and abilities to not uphold and magnify implicit bias that is present in many data sources. The algorithms themselves are complicated. They are not easily understood or made public for consumer inspection.

After the utilization of a company's parsing engines, hiring managers become involved in the application process. Résumés that make it past the initial screening, are passed along to hiring professionals from the various departments who categorize candidates based on their observations of what fit should look like for each position. There lacks a cohesive and systemized structure for organizing and defining what is an ideal fit for each position. As a result, this determination is left to individual inclinations. The data from the choices that are made will be used in later cohorts of data that will be implemented and potentially perpetuate the same systems that have been put in place in these corporations; these are the systems that choose candidates based on historically biased data. In these ways, the potential for biases to erode hiring choices exists at many stages of the process, from the parsing engines that dictate appropriate resumes, to the hiring managers who conduct the interviews[15].

Many Big Tech companies in recent years are taking a stand for more ethical practices in the operations of their businesses; however, some areas of scrutiny remain. There are still reports that most tech companies have less than 5% Black, Indigenous, and People of Color (BIPOC) employees and very low rates of women employed at these corporations[6,15]. Although they have made commitments to increase the diversity of their workforce, more work remains to be done. Through the understanding of the complex ways in which bias manifests itself, individuals will be able to better understand the effects it can have on the workforce and how to combat it.

Ethics in computing encompasses a wide range of topics and technologies. One such field relates to the use of historical data in machine learning and how this data has been used to derive insights across various fields. The gathering of data is in itself a time-consuming and

space-intensive process but is necessary for obtaining the information needed to begin making more educated decisions.

1.3 Related Work

1.3.1 Socio-Anthropological Works

The following papers contributed to the development of this project into one that seeks to examine decision-making and how that can continue to perpetuate a belief. “Algorithms, Platforms, and Ethnic Bias: An Integrative Essay” focuses on bias in algorithms and looks at nine potential sources of bias that could be present in algorithms including training data bias, algorithmic focus bias, algorithmic processing bias, transfer context bias, misinterpretation bias, non-transparency bias, consumer bias, and others[22]. They seek to determine the ways bias exists and can be countered while situating this study within the ethics of big tech. It’s stated that the remedy for these biases is reliant upon the determination of fairness and what that looks like for each group. This article is salient in helping to narrow down the points that this project focuses on, training data bias and feedback loop bias. These two forms specifically describe bias that is present in the data sets used in machine learning algorithms and bias that is self-reinforcing by using biased outputs in the inputs of algorithms. It contemplates the need for “active countermeasures” in the process to help remove biases that have been produced in previously unforeseen ways. Through the continued study of the nuances and records that led to and magnified bias, companies and individuals can develop effective countermeasures.

The lens with which bias exists can also be viewed from another area of the hiring process. “Are You One of Us?: Current Hiring Practices Suggest the Potential for Class Biases in Large Tech Companies” examines the interview process for internships at technical companies and discovers that evaluators often assess candidates based on three themes for establishing fit: industrial fit, organizational fit, and individual fit. They labeled industrial fit as a “candidate's ability to strategically and selectively adopt the language of an “industrial researcher” in their oral communication and written materials.” Organizational fit relates to “showing an awareness of the values and people of a particular organization” which is established based on an applicant’s social connections and referrals within the company. Similarly, “Social Capital at Work: Networks and Employment at a Phone Center'” describes the notion of social capital where an employees’ value is determined by the value of their connections, and these connections have proven to yield significant economic return by reducing costs associated with retaining new employees. Lastly, individual fit relates to “the elusive sense that an applicant is somehow similar to the evaluator” and would be someone they share lifestyle preferences with. All these metrics contribute to a system that categorizes applicants based on their network and similarities to the evaluators. These become areas where bias and a feedback loop resurfaces as a specific image is perpetuated throughout the hiring process that excludes groups of individuals whose background is not similar to the trend for the company, which are usually those in minority groups[15, 18].

“How Algorithms Discriminate Based on Data They Lack: Challenges, Solutions, and Policy Implications” is another in the series that touches on the topic of how algorithms can exacerbate discrimination. It further investigates the ideas of including vs excluding social

category data, such as race or gender, in various algorithms and brings forth the idea of being able to use information that is gathered to supplement already existing practices in detecting bias[2].

"Achieving Ethics and Fairness in Hiring: Going beyond the Law" focuses on discussing what ethical hiring practices look like and how they are organized to fit lawful requirements but how they can also transcend the requirements from the law. It's a discussion on the benefits of how ethical hiring practices contribute to better hiring decisions. Utilitarianism is the ideology that many evaluators take when making a decision and explains how the best value is one that supports the greatest good for the greatest number of people. The paper speaks on the conditions of which utilitarianism may need reconsideration, including when hiring practices lead to serious disadvantages of a demographic group. The conclusion becomes that hiring managers should take into consideration law and regulations, as well as ethics and moral standards that adhere to a systematic hiring process based on facts more so than intuition, which can lead to personal bias and self-interest[7].

"Why Are We Using Black Box Models in AI When We Don't Need To? A Lesson From An Explainable AI Competition" spoke to the use of black-box algorithms in AI has become a recent widespread occurrence. These algorithms are complicated, uninterpretable algorithms created by AI and used in machine learning models. The use of interpretable and transparent algorithms often comes with constraints surrounding the definition of many variables but also provides a more ethical alternative. The Black box algorithms that are widely used take advantage of a framework that isn't able to be fully understood and also has the potential to inherit biases present in the datasets that they are trained on. The Explainable Machine Learning

Challenge saw a wide range of participants submit solutions that utilized black box algorithms but also had one team who went against the grain and used an interpretable AI, which brought to question the need for these algorithms on such a wide scale. Especially if they continue to perpetuate aspects of society that should be examined closer. This paper urges the construction of more interpretable algorithms to solve problems and make high-stakes decisions in place of these black-box algorithms[3].

1.3.2 Technical Works

The following works contributed to the moments when contemplating the saliency in examining social categorical information and its role in possible bias detention. “How Congress Could Reduce Job Discrimination by Promoting Anonymous Hiring” examines ways to reduce bias and argues for the removal of identifying information in the hiring process as a means to reduce discrimination and prejudice. This includes “stripping résumés of all information related to race or sex, and eliminating selection interviews.” It presents a tangible implementation of the idea with how a voluntary system could be utilized that would give the employers immunity from disparate treatment lawsuits[4]. Although this was one support for the removal of social category information, others argue for the use of this information to help identify areas of bias and thereby effectively provide a remedy. It influences the decision to withhold some social categorical information when presenting information to an agent to control which aspects could be observed during this simulation.

“Investigating the Impact of Gender on Rank in résumé Search Engines” and “Evaluating Fairness Metrics in the Presence of Dataset Bias” both try to understand how fairness metrics are

measured in different contexts. The quantification of “fairness” will contribute to the modeling of bias since this measure contributes to the detection of bias in data. Different metrics were proposed with regards to the fairness metric including a difference in means, difference in residuals, disparate impact, and various others. These metrics are difficult to interpret without external content, however, and require some awareness of bias already present in order to detect bias. Ultimately, metrics for determining fairness are not included in this project due to the focus on examining how Markov Decision Processes can be used to model hiring processes and express the findings.

“A survey on measuring indirect discrimination in machine learning” provides one such perspective from the viewpoint of individuals who believe in the use of these protected social characteristics such as race and ethnic origin, age, sexual orientation, gender, and others. The paper describes how the typical machine learning algorithm is a procedure used for producing a predictive model, or resulting collection of rules, from historical data[9]. The resulting model is used for decision-making for new incoming data. The model would take personal characteristics as inputs and output a prediction as shown in **Fig.1**.

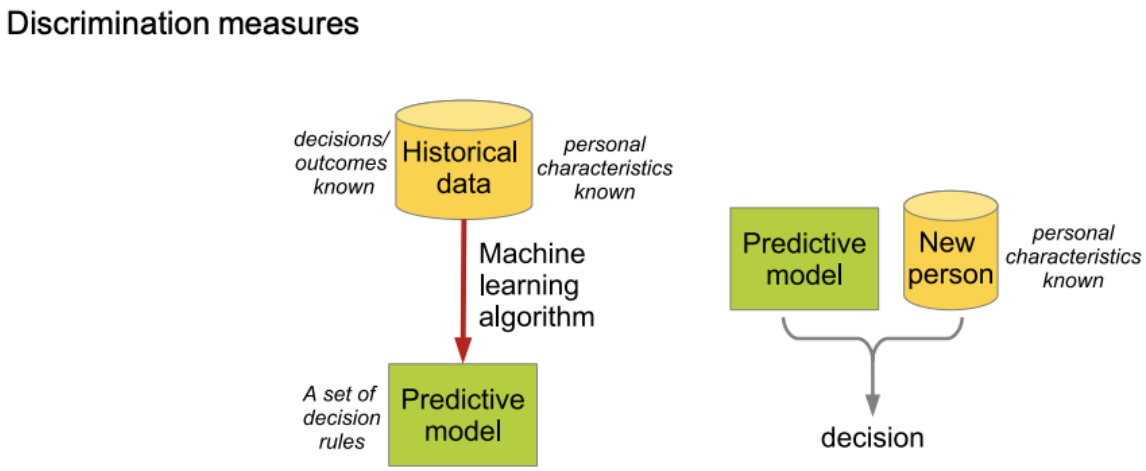


Fig. 1. A typical machine learning setting.

Fig. 1 [9]

They explain how models that are used may make biased decisions, but algorithms can be designed to be aware of discrimination for the model construction phase to enforce non-discriminatory constraints into the models. This is why "one of the main goals of discrimination-aware machine learning and data mining is to develop discrimination-aware algorithms, that would guarantee that non-discriminatory models are produced." They conclude to not use protected characteristics as model input due to the potential for direct discrimination, but instead use it in the model learning process, which often may help to enforce non-discrimination constraints as long as the resulting model does not take these characteristics as input[9]. This paper provides a deeper understanding of how the machine learning models work and how they may impact the models used during the hiring process. It presents a framework for understanding how these predictive models are created and used.

The focus of "Game Design for Eliciting Distinguishable Behavior" is to be able to infer latent psychological traits from human behavior through game development. This study utilizes

Markov Decision Processes to formulate the game design and categorizes and models the results based on Prospect Theory, which is the concept that different people can perceive the same numerical values differently. The players in the game study are given a set of actions in an environment, and at each state, they can either lose a certain amount of the reward or gain it. The goal was to model the different types of players (Loss-Neutral, Gain-Seeking, Loss-Averse) and the reward that contributed to each player's policy[5]. This study heavily influenced the visualization goals and function of the MDPs in this project.

2 Background

This section describes the technical background for information that is used throughout this project.

The visualizer game module seeks to help hiring managers to understand and visualize intrinsic biases that they own. It accomplishes this goal through the use of reinforcement learning. Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment to maximize the notion of cumulative reward. Our conceptual agents are the hiring professionals, and the reward will be a qualified candidate and a visual depiction of the choices that the agents make and their significance. The reinforcement

aspect will occur throughout the game as a measure of how the professional rates/quantifies various forms of fit. Reinforcement Learning makes use of A.R.E.A. which describes an Agent, Environment, Reward, and Action. Through the examination of the Markov Decision Process, this game quantifies and reproduces the model for how decisions are made.

2.1 Machine Learning: Supervised vs Unsupervised vs Reinforcement Learning

The ability to understand the latent psychological bias present in the hiring process will be a contributing factor in designing these data-driven systems that are widely used to reinforce and support many of the hiring systems that big tech companies have created and implemented. Machine Learning is a rising field where the application of AI allows systems to be able to automatically learn and adapt to situations without being explicitly programmed to perform an action[]. This can be accomplished through the use of collected data to observe patterns that help the machine to make better more efficient decisions based on a provided metric. In Machine Learning, there are several learning paradigms: Supervised, Unsupervised, and Reinforcement Learning. A vast majority of machine learning is encompassed in the use of Supervised and Unsupervised Learning. Supervised learning is learning that occurs from a set of carefully curated labeled data. In this method, the system is learning to extrapolate its responses so that when presented with a new situation not previously seen in the data, it will be able to generalize the correct action to take based on previously observed trends. Unsupervised learning distinguishes itself because it is learning based on a set of unlabeled data. The system uses the data points to make clusters and discover hidden patterns among the data points, which are then

used in many facets. The last paradigm, reinforcement learning, explores the idea of learning the most optimal path to completing an action with the overall goal being to maximize the reward for the action.

The main difference occurs in that reinforcement learning is attempting to maximize the reward while unsupervised learning is trying to find a hidden structure/pattern(SuttonBartol). This demonstrates the decision to utilize Markov Decision Processes, a form of reinforcement learning. Intending to try to visualize the optimal candidate for a specific job, the project focuses on the attainment of a reward when a correct decision is made and thereby aims to maximize the reward of a correct decision being made. We choose to attempt a method that diverges from the standard where large data collection would be needed from a company. This data on employees is difficult to obtain because of confidentiality restrictions and would be difficult to directly imitate. Using reinforcement learning allows the data to be built as the model is created and the optimal policy to be observed.

Supervised learning requires the utilization of data that already holds the correct answer, or in the case of the present problem, the superior candidate for a job position. It requires data that tells not only how good or bad an action is, but also what action should be taken given a problem. Unsupervised learning would also require massive data sets of current employees and from that data, the algorithm would observe patterns that would form the equation to express what an ideal candidate would look like in this position. This would not present the most optimal solution seeing as most data would be based on the company standards that have already been set and would therefore continue to perpetuate any bias that is present in the data. There are various

ways that restrictions could be put in place to mitigate this problem, however, this project explores a different approach.

2.2 Markov Decision Process

The objective of the simulation is to illuminate the cognitive model in the decision process that is subconscious among hiring professionals[1]. The basic concept will allow the agent, the hiring professional, to go through different tasks/actions. These actions will have various rewards associated with them and at each point, the agent is measured in some capacity. The agent has the option to further explore and determine if there is a greater reward deeper into the game, or exploit their current state/candidate. In the context of hiring, this might have an outlook of the hiring professional deciding that they have identified enough distinguishable and sentient information to make their hiring decision and so no longer have to explore different states.

The process of choosing which actions/choices are best can be expressed as a Markov Decision Process. The Markov Decision Process follows the Markov property which states that "The future is independent of the past given the present"[]. This is a process that has specified transition probabilities from state to state. The transition probabilities are the probability that the agent will move from one state to a successor state and these are formalized in this project by defining a random variable and programming various conditionals that depend on this output [ ]. These are plausible models to describe the choices because they hold specific characteristics that are reflective of this project. Markov Decision Processes have a finite set of states, a set of

actions available in each state, transitions between states, rewards associated with each transition, A discount factor γ between 0 and 1, and Memorylessness. The Markov Decision Process would help to quantify and identify the factors at each point that a decision is made and contribute to the end model that depicts any biases that are held by the hiring professionals[10,11].

In a MDP, the states where an agent begins can be random or constant and they represent the point at which an agent is currently. States can be discrete, where they take a finite set of values, or continuous, where the values are not finite. The state in which an agent begins is denominated as the starting state, S_0[ ]. The actions express what an agent can do in each state.

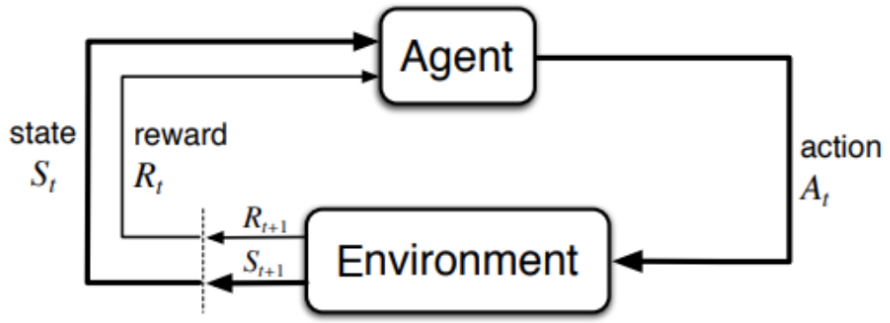


Fig. 2 An image of the Reinforcement Learning process?[]

This simulation is formalized by following **Fig.2** and "The behavior game diagnostic game design", where the player will have different states that they can navigate through. At each state, there is a "reward" that the agent accumulates. Also at each state, they will be given a score

in one of several categories designed to investigate different forms of bias. Similar to the study, the observations obtained from the simulation create a model that depicts certain behaviors among each agent.

"Reinforcement Learning: A Survey" describes the history of RL from a computer science perspective. This is an important vantage point considering topics surrounding reinforcement learning can be heavily influenced by the field of psychology. The description for the three models of optimality in this paper was instrumental in being able to decide which method would be used for the agent. This was about how an agent's actions are decided at any given point. The Finite-horizon model is the easiest to think about; at a given moment in time, the agent should optimize its expected reward for the next steps. Another model is the infinite-horizon model where the long-run average reward is accounted for but the rewards for the future are discounted by a factor γ(gamma). The last model is the average-reward model where the agent takes actions that optimize its long-run average reward[11].

2.3 Components of Markov Decision Processes

Through the use of the Markov Decision Process (MDP) to parametrize the choices the agent makes, the choices made will be able to be observed. MDP's are frameworks that can model decision-making and allow agents to arrive at an optimal policy. At each point, the agent will make a decision, and that decision will have a contributing factor to the end policy.

2.3.1 Stochastic vs Deterministic

In a Markov Decision Process, when the process is said to be deterministic, this implies that the same action will always result in the same outcome. There is no transition probability in deterministic MDP because the probability of going from s to s' is always 1. Whereas in a stochastic MDP, performing the same action in a state won't always yield the same outcome and there is a chance of a different outcome resulting[16]. Stochastic models are used to model conditions where the outcome, the state an agent ends up in from the current state, is partly random. The agent draws from a probability distribution encoded by the policy, how the agent acts in a given state, to determine the new state, s', that the agent advances. The reward that is obtained therefore is in correspondence with the new state, s'.

2.3.2 Value Iteration

Markov Decision processes provide a fundamental way to both examine real-world problems and solve them using reinforcement learning. Markov Decision Processes allow the sequential modeling of decision-making problems where a decision-maker sequentially interacts with the environment, deciding at each step of a problem[]. The solution of a Markov Model can be found in the policy that is discovered. This policy defined as π(als) is the probability distribution over Actions (a $\in$ A) for each state (s $\in$ S) at a particular time step(t). Actions that have a higher reward will also have a higher probability(). Two alternative methods to focus on

that allow one to solve this probability distribution and also guarantee an optimal policy are Value Iteration and Policy Iteration[16, 23].

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
  $\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Fig. 5 Pseudo-code for the Value Iteration algorithm. The Value array is initialized with zeros and goes through the process of updating V(s) until it converges upon its optimal values[10].

Value Iteration allows us to calculate the optimal value function. It requires a transition function for the MDP model and a discount reward. The Value iteration algorithm takes advantage of the Bellman equation, an equation that allows us to calculate state values by using an iterative approach. We begin with a value function, which is updated by looking ahead one step and finding the maximum V(s) of all possible actions. The value is updated in one step since all possible rewards are calculated by looking ahead[24].

2.3.3 Q-learning

Q-learning was an important breakthrough in reinforcement learning. It is a model-free off-policy reinforcement learning algorithm whose goal is to find the best action to take in a given state. Q-learning is considered off-policy because it takes advantage of performing random actions to learn a policy that maximizes the total reward. Q-learning is implemented by first creating a Q-matrix which will hold all of the updated values of the Q-function. At each episode or iteration of the algorithm, the Q-matrix is updated until after all the iterations have been completed and the values in the table converge onset values. The table will then be a source for the agent to reference what the best action is.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

Fig. 3 Pseudo-code for the Q-learning algorithm. It describes the process of initializing the Q matrix and iterating through multiple episodes where the Q function is continuously updated[16].

The agent has to take action to update the Q-learning matrix. To do this, it has to decide between exploring the environment at random or exploiting the values in the Q-matrix. By exploring the environment, the agent discovers new states and the rewards associated with those

states and is then able to further update the Q-table. By exploiting the Q-table, the agent uses the values already present and known in the matrix to determine which action in a given state will give you the highest possible reward.

Updates to the Q-Matrix occur after each action and when the iteration reaches its goal. The goal for this case is to reach the end of the simulation. The Q-learning algorithm is aided by multiple iterations that result in the Q-matrix being continuously updated until the values converge upon the optimal Q-value[24]s.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

Fig. 4 This Q-learning Equation describes the inputs that result in a new Q-value, Q(S, A)[16].

In the above equation, the alpha corresponds to the learning rate which relates to the rate at which we accept the new Q-value versus the Old Q-value. If this value is 1, then the new estimate will be the new Q-value and thereby express the agent exploring, while a factor of 0 expresses the agent exclusively exploiting prior knowledge. Gamma corresponds to the discount factor and describes how much the future reward will be discounted. This means that as time progresses, the reward becomes less and less valuable[19, 21].

Q-learning takes advantage of the Bellman Equation as part of its core algorithm. This equation is recurrent in Reinforcement Learning and helps to solve MDPs by finding the optimal policy and value functions.

The Q-learning algorithm falters as the number of states and actions grows due to the chance that the agent visits a particular state or action decreasing. For the current example, Q-learning allows the optimal policy to be discovered because of the finite number of states and actions present in our scenario/model.

2.4 Levenshtein and Hamming Distance

During the implementation of the MDP Game, the model required a reward to be calculated. This reward presented an opportunity to become more creative in how it would be implemented. When analyzing the differences in a résumé versus a Job Description, you realize that there are often a lot of skills that employers look for but also a lot of skills that are not mentioned that a potential candidate would have. To analyze the differences between the two, we take advantage of the Levenshtein Distance. This is a string metric for calculating the differences between two string sequences [17]. In the case of the MDP, the job ad and résumé are put through a filter that gets rid of common filler/stopwords such as “the, it, an” etc. This is all done in python. After the two files have been filtered, the python library RapidFuzz is utilized to calculate the Levenshtein Distance using `fuzz.token_set_ratio` function.

```
from nltk.corpus import stopwords
nltk.download('stopwords')
from nltk.tokenize import word_tokenize

f = open('Job_Desc/JobDescr2.txt', 'r')
d = open('Resumes/ResumeSample3.txt', 'r')

#Splits values in txt file into list
data = f.read()
split_job = data.split()
```

```
split_job[:] = (value for value in split_job if value != '\t')

data = d.read()
split_resume = data.split()
split_resume[:] = (value for value in split_resume if value != '\t')

#Remove stopwords
resume_tokens_without_sw = [word for word in split_resume if not word in
stopwords.words()]
ad_tokens_without_sw = [word for word in split_job if not word in
stopwords.words()]

#Calculate the Levenshtein distance
"""
  The Levenshtein distance is a string metric for measuring the difference
  between two sequences.
  It is calculated as the minimum number of single-character edits necessary to
  transform one string into another.
  """
Levenshtein = fuzz.token_set_ratio(resume_tokens_without_sw,
ad_tokens_without_sw)
```

Fig. 7

The Levenshtein Distance has several bounds which are represented by the following:

1. It is at least the difference of the sizes of the two string

2. It is at most the length of the longer string

3. It is zero if and only if the strings are equal

4. If the strings have the same size

5. the Hamming distance is an upper bound on the Levenshtein distance

6. The Levenshtein distance between two strings is no greater than the sum of their Levenshtein distances from a third string[17].

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	2

The distance is in the lower right hand corner of the matrix, i.e., 2.

Fig. 8 [17]

The Levenshtein distance calculates the substitution, deletion, and insertions necessary to convert one string to another. In figure(above), it describes the process of calculating the Levenshtein distance where a is the first string, b is the second string, i and j are the ending character position of strings 1 and 2 respectively. Also, we recognize that 1 is only added if the characters are not the same at positions i and j. At each position in the string, we are checking that the strings are not equal, and incrementing the edit distance if that is the case.

The Hamming distance is another string metric that calculates the number of bit positions in which the two bits are different. It has the potential to be utilized when comparing across candidates, after removing words that do not add value to the requirements needed for a position. It is a faster and lighter-weight string metric as it only accounts for the substitutions necessary to find the differences in two strings. The pitfalls with using the Hamming distance relate to its requirements to have strings of equal length. The strings from the documents would need to be

padded to fulfill this requirement and when that occurs, that has the potential to change the value of the Hamming distance in ways that make it more complicated to account for. If there existed a padded string “ABC___” and string “ABCDEF”, the hamming distance is 3, however, the string “___ABC” compared to the same string “ABCDEF” produced a hamming distance of 0. Both metrics have a space in the project, but the abilities of the Levenshtein distance mark it as more apt for the comparison of the résumé and job description [25].

3 Implementation and Method

This section begins by describing tools, languages, and libraries used to implement the MDP model and create the beginnings of the simulation design including the simulation space, the models, and the metric for evaluating the reward function. . It explores the implementation of the creation of optimal policy that is used to compare the results from a user. The section discusses the implementation of the human user space where the decisions are made based on one individual and compared to the optimal policy discovered by the Q-learning model to aid the objective.

3.1 Tools

Python is heavily used to develop the simulation. Its vast array of libraries are efficient in helping to create the models and visualizations necessary for the project. These python libraries include two visualization libraries: Seaborn and matplotlib. Illustrations of the state spaces and transition functions for the simulation were also created using Excalidraw.

3.2 Decision Model

The simulation states are a direct representation of a candidate's résumé. The format focuses on the categories in a technical résumé, which differs from non-technical and artistic résumés. These formats often include a technical project section which sometimes replaces the traditional hobbies section of a standard résumé format. And although interpersonal skills are included, technical skills are also highly recommended for a technical résumé. Other sections such as education, previous experience, personal statement, and certifications still follow the basic design seen in regular résumés. The sections from the résumé go on to define the states for the RL model.

States: Education, Skills, Certifications, Experience, and Referrals.

In each of these states: Education, Skills, Experience, Certifications, and Referrals, the agent performs an action. These actions directly reflect the choices that can be made when placed in one of these conditions. An agent can accept the candidate, decline the candidate, continue to

examine the candidate's résumé, and examine another candidate. The agent is presented with an unrecognizable arrangement of candidates and they begin by choosing a candidate that they wish to examine. At no point is a candidate's identity distinguishable from the others based on the initial image and the candidates are named with non-descriptive names such as "candidate 1". Upon selection of a candidate, the agent can choose which of the above states they will proceed to examine. After that point, the candidate selects which action to perform in a given state.

Actions: Accept, Decline, Continue to Examine Candidate, Examine Another Candidate

The agent is initially presented with a job description. After which, they decide which candidate to begin examining. At each point, they are presented with a choice of which action to choose that will take them from the current state, S(t), to the next state, S(t+1). The goal is to achieve the highest possible reward with the most optimal set of choices by requiring the agent to match the ideal candidate to the job description. The reward function is based on each action and the corresponding Levenshtein(Lev) distance. This numerical value is used to calculate how similar the job description and résumé are and for a candidate with the highest possible total being 100. For each state, when A(t) = accept, if Lev > 75, 1 is added to the score, else a default value of .33 is subtracted. When A(t) = decline, if Lev > 75, 1 is subtracted from the score. This indicates a candidate who has a high match to a résumé but is ultimately declined from the position. Similarly, when an agent chooses to continue evaluating a candidate with a Lev > 50, they are rewarded .33, and when A(t) = examine other candidates, with a Lev > 50, .33 is subtracted from the total reward. This method follows to punish negative decisions for high Lev

values and to reward positive decisions to these same Lev values. Ultimately, the highest value for R(s) is obtained when a candidate A(t) = accept and a high Lev is associated with the résumé and the lowest R(s) is obtained when candidate A(t) = decline and a high Lev is associated with the résumé.

The simulation has several different forms that it seeks to compare through visualizations of the policies that are created.

- A User-defined MDP Model was created which takes in user input to create the policy for the MDP and is meant to be used by an agent in a training simulation.
- An Automated MDP Model was also created where the decisions are randomly chosen to create the policy of the agent. This model is meant to be used when no experiment is conducted.
- The last model is the Optimal MDP Model which utilizes Q-learning to create the optimal policy for the agent.

```
prob = rm.uniform(0, 1)
    print(prob)


    if (prob > .2):
        state = input_state #Highest chance of following the state chosen
        print("In STATE", state)
    else:
        if(prob > .7): #
            state = rm.randint(3,5) #more likely to look at referrals, certifications, experience
                print("In STATE", state)
              else:
            state = rm.randint(1,5)
                print("In STATE", state)

```

Fig. 6 Algorithmic implementation that controls the stochastic nature of the MDP models

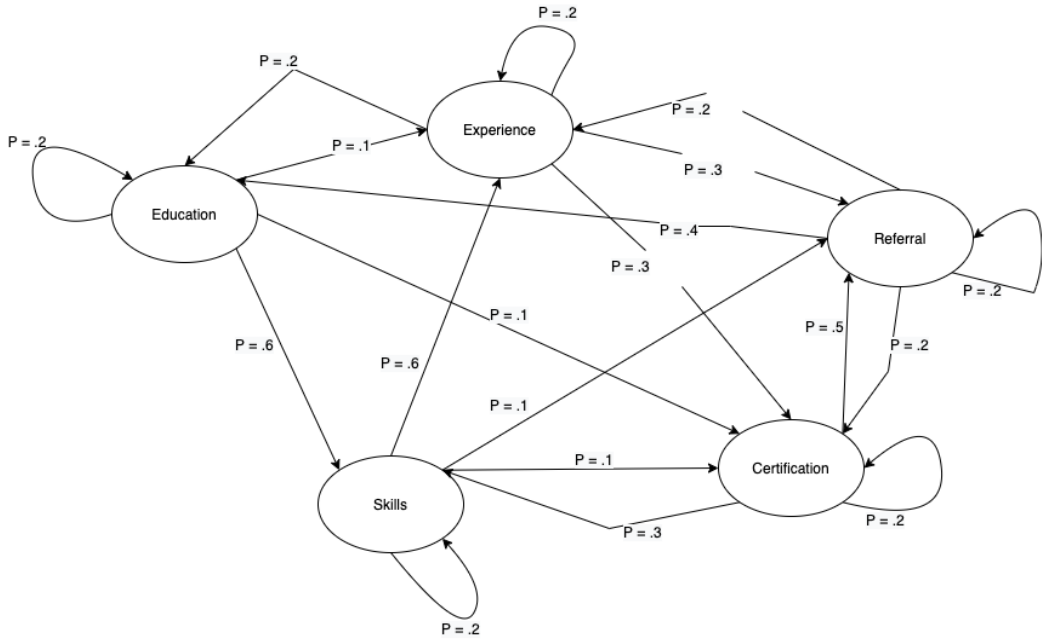


Fig. 9 Transition Diagram of States

3.3 Transition Function

The deterministic and stochastic versions of the simulation are created for each model. The deterministic designs show the policy that results when an agent chooses an action and that action is executed. And the stochastic designs show a more likely element where an agent can make a decision, and the action chosen may not be the action that is executed. The stochastic model follows the logic displayed in Fig. 6.

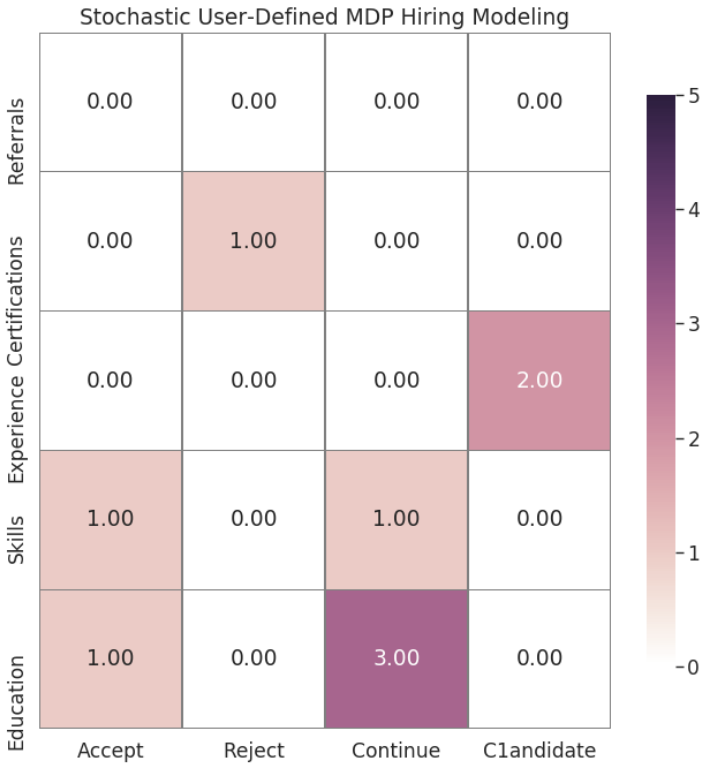
The end of the simulation will give a visual representation of the qualities that each agent determined to be most salient in the process of completing their objective. These visual presentations will showcase the Policy and Reward for the models, The visuals are presented in the form of heat maps that show the optimal policy and reward functions. They only seek to show

overall values for each $S_{(t)}$, $A_{(t)}$ pair and do not break down choices made among the individual candidates. The simulation provides the opportunity to compare the results observed from a user perspective to the most optimal policy that the agent could have taken. It serves to show the agent a comparison of the decisions that are made versus potentially better decisions that would have obtained a more qualified candidate[20]. The visualizations are mainly for the purpose of comparison to standards set by the agent.

4 Results and Analysis

This section presents the key results and analysis that are seen from many iterations of the simulation in the three forms. Various graphical representations are presented along with comparisons to the optimal policy achieved using Q-learning. It is broken down into the following sections: visualization results for MDP policies and rewards, Optimal MDP policy discovered, and the analysis of what the discoveries entail and how they can be used for future references.

4.1 Analysis


Stochastic User-Defined MDP Hiring Modeling
Referrals
Certifications
Experience
Skills
Education
0.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00
0.00 0.00 0.00 2.00
1.00 0.00 1.00 0.00
1.00 0.00 3.00 0.00
Accept
Reject
Continue
C1andidate
5
4
3
2
1
0

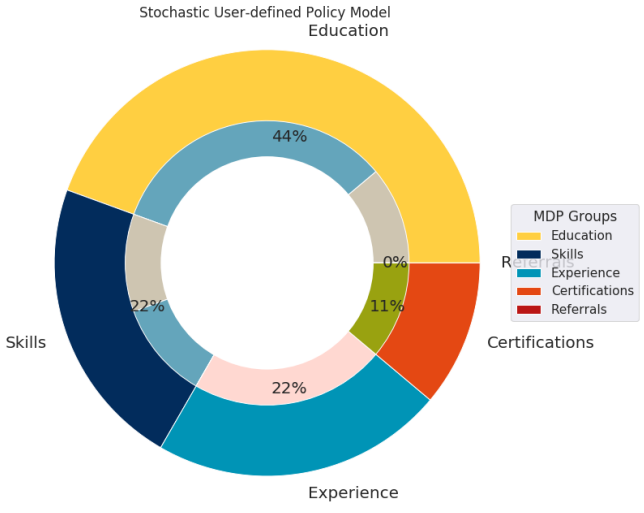
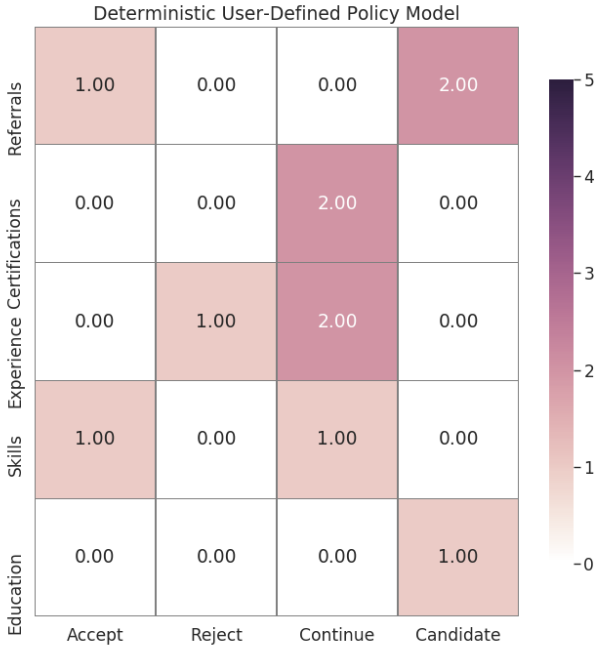

Stochastic User-defined Policy Model
Education
44%
0%
11%
22%
22%
Skills
Certifications
Experience
MDP Groups
Education
Skills
Experience
Certifications
Referrals

Fig. 10 Stochastic User-Defined Policy



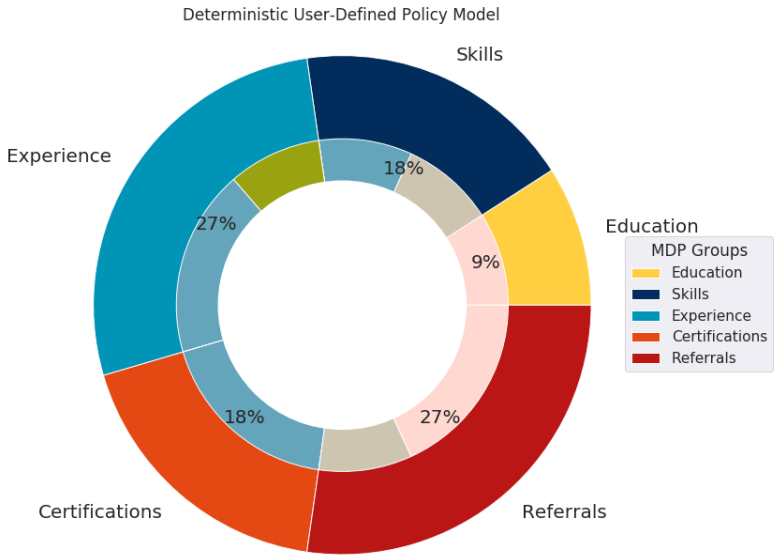
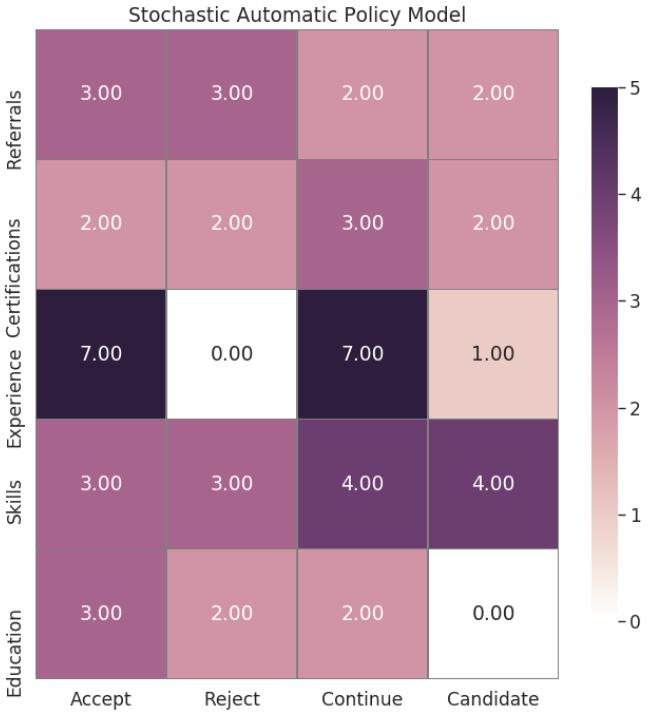


Fig. 11 Deterministic User-defined Policy

The Stochastic User-defined MDP policy presented in the visualization results are seen in **Fig. 10 and Fig. 11**. This policy results from a user perspective of completing the MDP tasks. The stochastic policy is more applicable to real-world applications due to the element of randomness that comes with completing a task. Utilizing MDPs allowed a probability distribution to be created for traversing from one state to the next. The model format allows a state that is chosen to be actively pursued 80% of the time. The states were then chosen at random from the state that the agent is in. It was uniformly likely that an agent could choose to be in the state that they currently were in. Upon closer observation of the above results, it is seen how this particular user

chose the action to continue to observe a candidate 71% of the time and only accepted after observing the states for Experience and Certifications.


Stochastic Automatic Policy Model
Referrals
Certifications
Experience
Skills
Education
3.00
3.00
2.00
2.00
2.00
2.00
3.00
2.00
7.00
0.00
7.00
1.00
3.00
3.00
4.00
4.00
3.00
2.00
2.00
0.00
Accept
Reject
Continue
Candidate
5
4
3
2
1
0

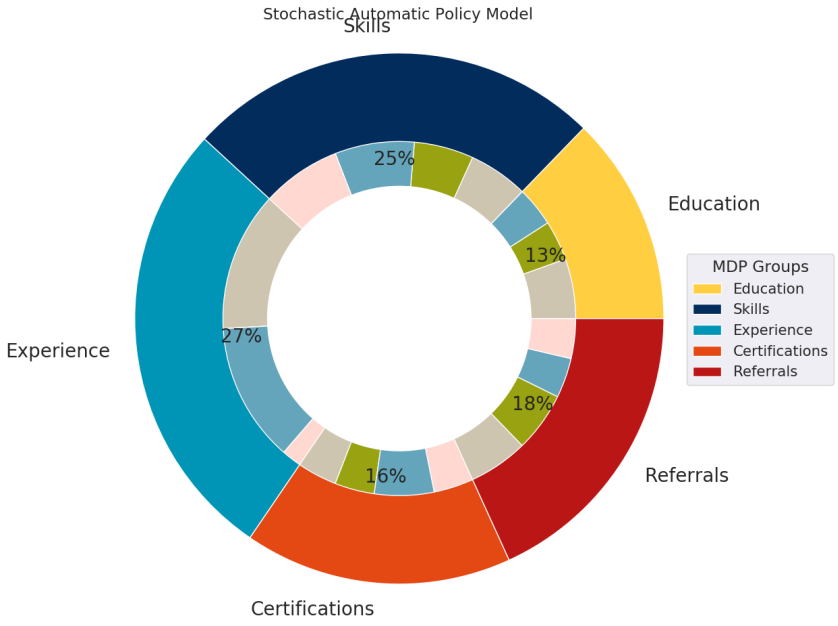
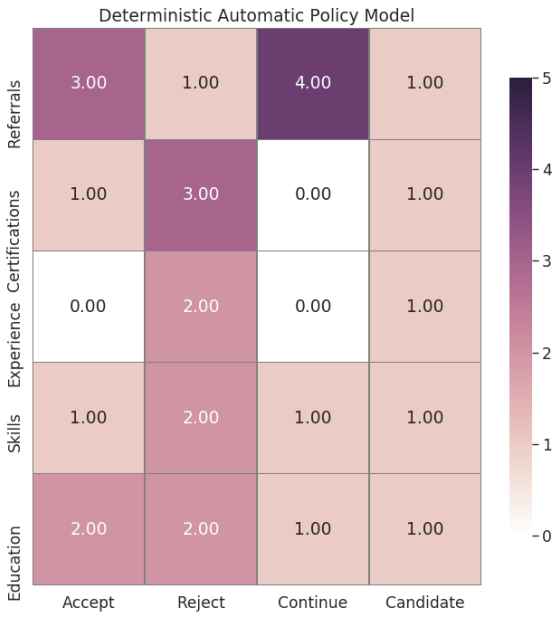

Stochastic Automatic Policy Model
Skills
Education
Experience
Referrals
Certifications
25%
13%
27%
18%
16%
MDP Groups
Education
Skills
Experience
Certifications
Referrals

Fig. 12 Stochastic Random Policy



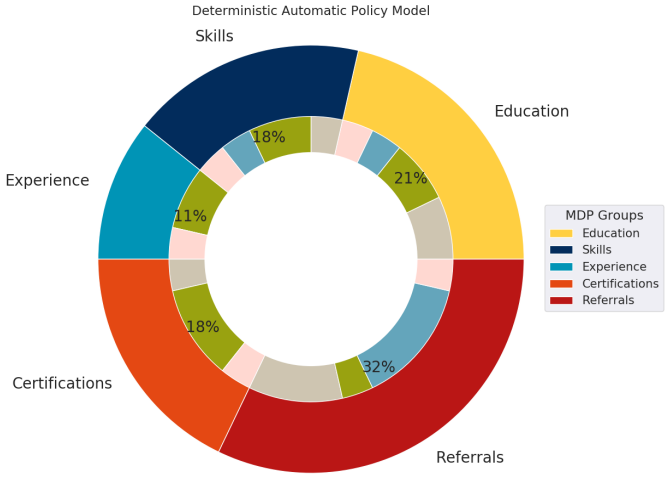
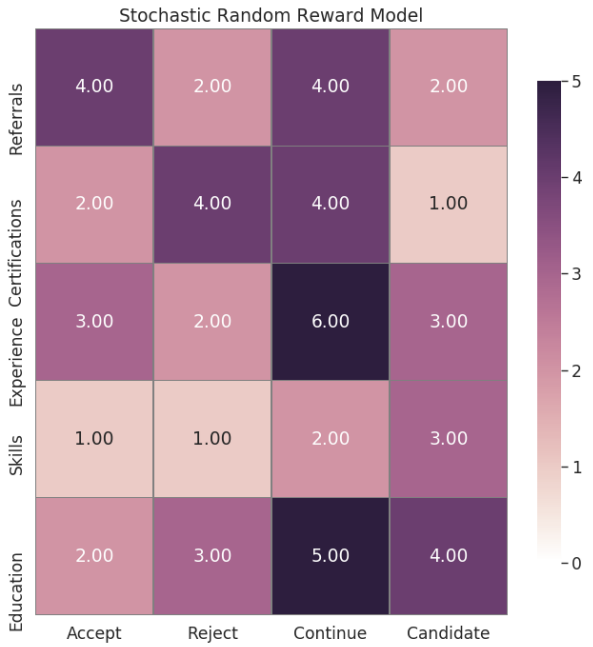


Fig. 13 Deterministic Random Policy



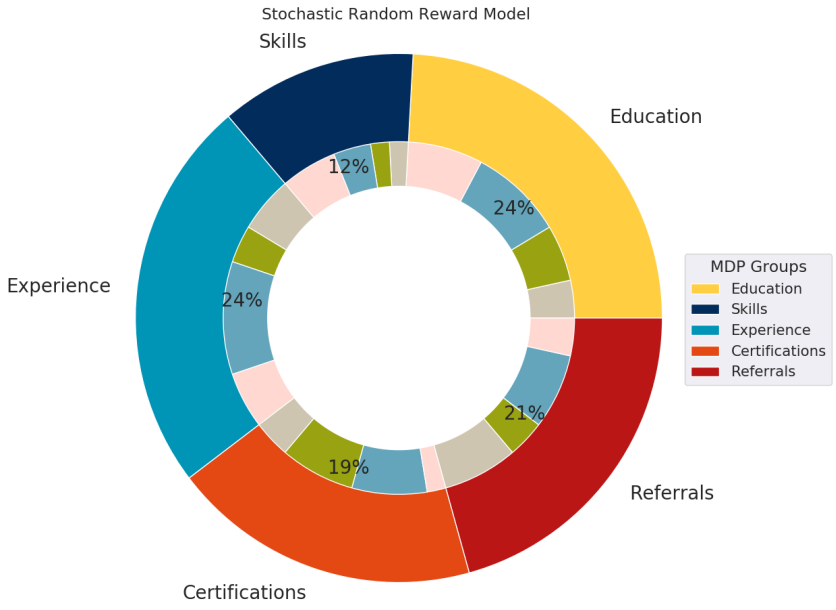
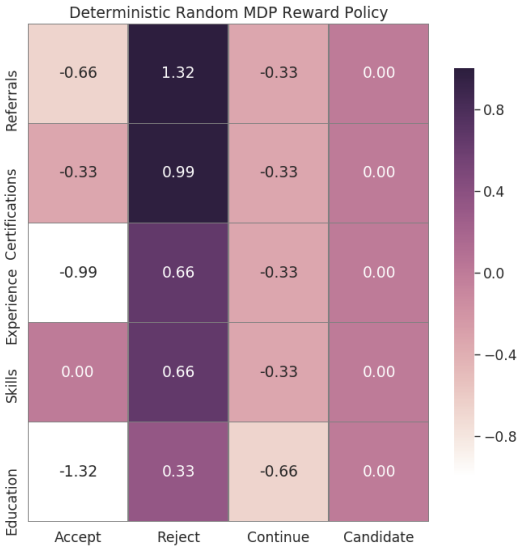


Fig. 14 Stochastic Random Reward Policy



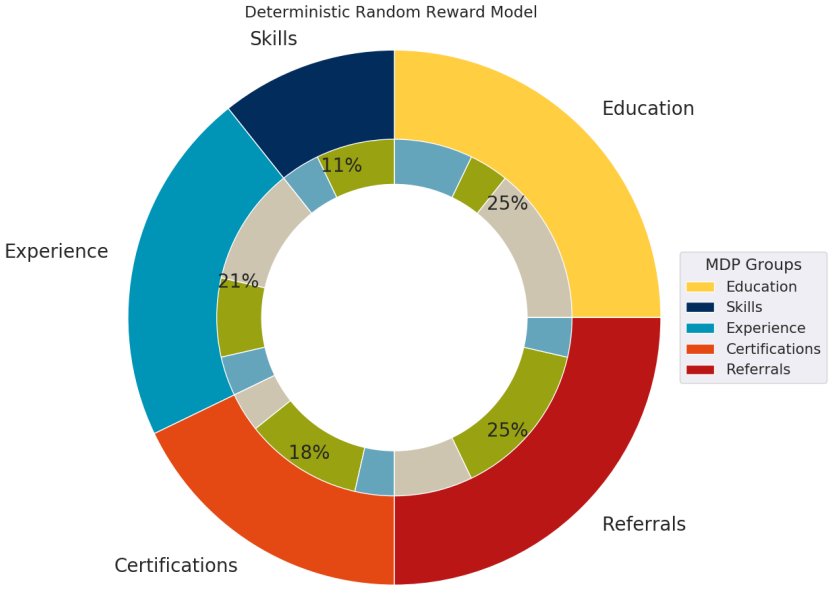
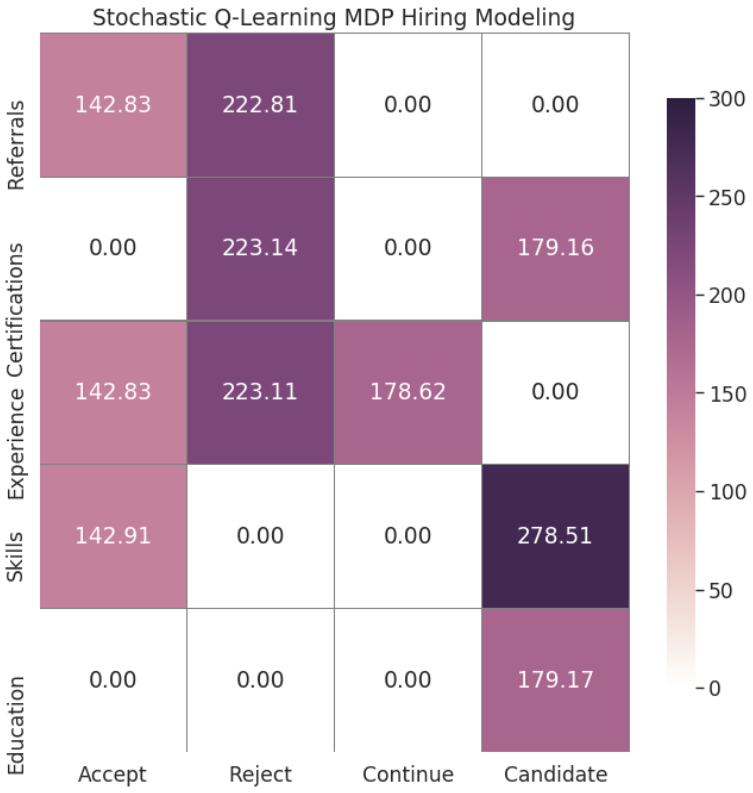


Fig. 15 Deterministic Random Reward

The Random Policy presented in the visualization results is seen in **Fig. 12-15**. These depict the deterministic and stochastic variations and also include the Deterministic Reward Policy. They exhibit greater variation in the resulting Policies with regard to actions taken and states entered. The Deterministic Policy placed a great emphasis on Referrals, while the stochastic policy placed emphasis on Education and Experience. These results are what an agent can use to determine whether actions are in line with the optimal policy and the procedures of the company. They can be used to evaluate the procedures that companies use in order to provide greater structure and prevent deviation.


Stochastic Q-Learning MDP Hiring Modeling
Referrals
Certifications
Experience
Skills
Education
142.83
222.81
0.00
0.00
0.00
223.14
0.00
179.16
142.83
223.11
178.62
0.00
142.91
0.00
0.00
278.51
0.00
0.00
0.00
179.17
Accept
Reject
Continue
Candidate
300
250
200
150
100
50
0

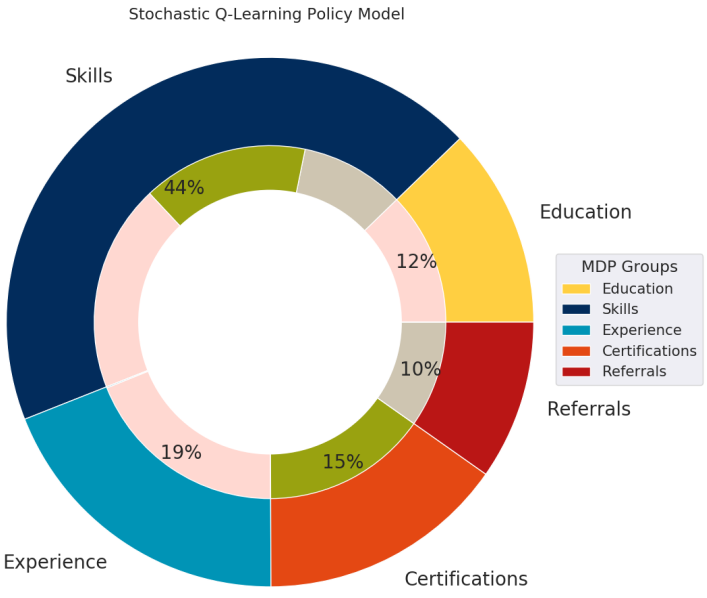
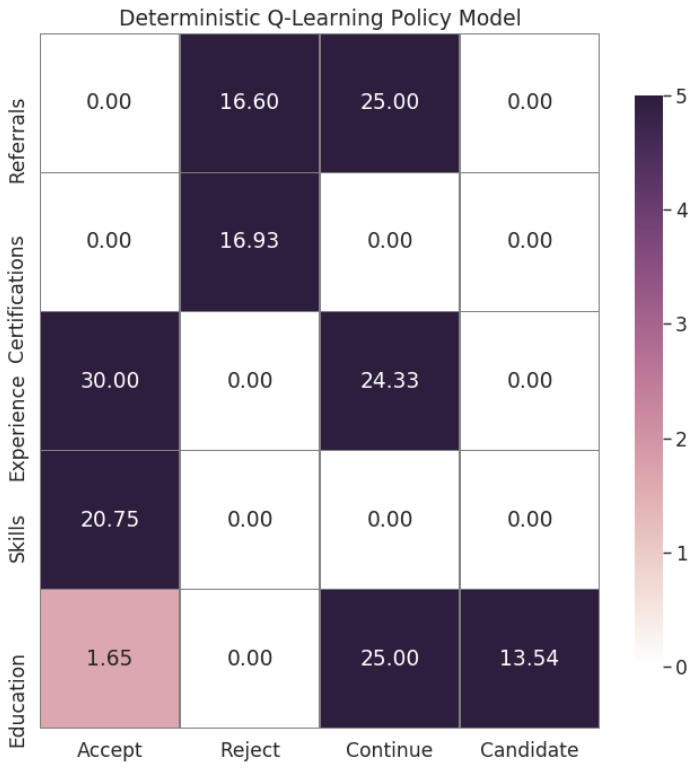

Stochastic Q-Learning Policy Model
Skills
Education
Referrals
Experience
Certifications
44%
12%
10%
19%
15%
MDP Groups
Education
Skills
Experience
Certifications
Referrals

Fig. 16 Stochastic Q-Learning Policy



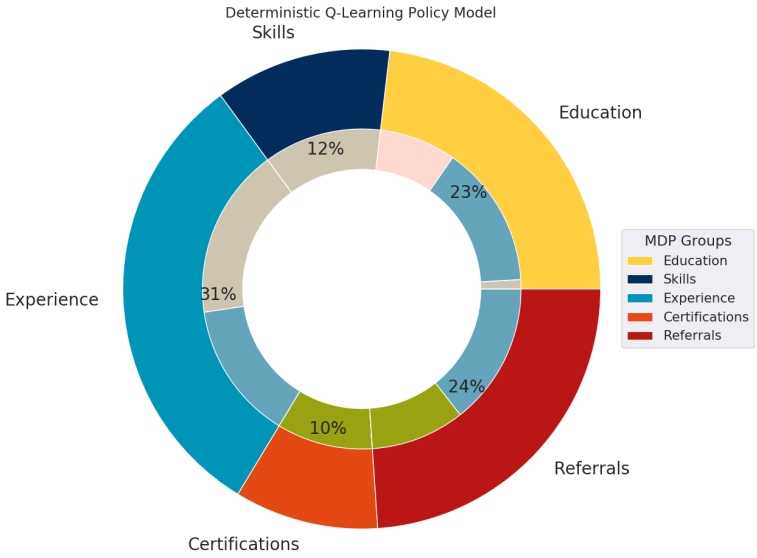


Fig. 17 Deterministic Q-Learning Policy

The Q-Learning Policies highlighted in **Fig. 16** and **Fig. 17** depict the optimal Policy for an agent in this environment. The values from both the Deterministic and Stochastic Policies show closer similarities in the optimal path than in the User-defined and Random Model Policies. Ultimately, the Q-learning Policy shows the states that hold the highest importance as defined by the reward function and can be used as a reference in comparison to a User-Defined Policy.

5 Conclusion

This section concludes the findings and states the main objective of this project, as well as further points of consideration and limitations.

5.1 Areas of Impact

The goal of this project is to create a game that seeks to understand and calculate areas of bias. The game has the potential to be used in the training for hiring professionals across different industries. The hope is that it will act as one of many tools that corporations will be able to use as part of their bias training. Through the continued use of this tool at regular intervals throughout a hiring professional's employment, they will be able to gauge which criteria they value over others and how these areas exhibit their definition of "fit" for a position. Once these criteria are discovered, it will help in the further valuation of their system and how it can lead to a lack of diversity in the workplace. Currently, there is no method capable of completely removing bias, however, generating awareness has been shown to mitigate cognitive biases that individuals experience.

5.2 Limitations

There are several points at which this game can be improved. These include allowing for different levels of difficulty, where candidates have vastly closer similarities in résumé talking points, and at which the optimizer will also have to become more critical for the criteria to choose the best candidate.

The goal in the creation of this module was to counteract the bias in individuals by making them more aware of the components that they looked at and how those might be perpetuating pre-existing stereotypes in the company. As with many forms of technologies that have been developed in recent years, there is a concern for how the intended use of the technology will not always be adhered to by the consumers of the product. Oftentimes, unwittingly, technology ends up either with a multifaceted purpose for the users and/or with unintended consequences[23]. When developing the current prototype, it's important to acknowledge the concern that it can have the opposite effect where it continues to perpetuate bias that it is trying to counter. Possible reasons that have been contemplated relate to the fact that job ads are written by candidates who possess vernacular and expressions that directly correlate to the job poster's background and upbringing. This can have the effect of excluding people who do not come from this background from even applying and when they do, from being able to perform as well on the ad and résumé comparison depending on the program specifications. The code that is implemented in this project makes use of the Levenshtein distance, which matches the words on a résumé to those of a job advertisement and calculates the difference between the two strings. A high Levenshtein distance indicates greater similarities between the job

advertisement and the résumé and vice versa. If a résumé makes use of synonyms and abbreviations, then they increase the likelihood of having a low distance score which would normally indicate an unqualified candidate, however, in this case, it could indicate that a candidate holds different cultural linguistics. This is one area that would need to be accounted for in the future improvements of this project and delves deeper into areas of Natural Language Processing and how to account for cultural differences in language and communication.

Another limitation of this study is it fails to take into account candidates that might be overqualified for a job posting. This would be seen in areas where a candidate with a master's degree might apply for a position only requiring a bachelor's, as well as candidates who have a host of certifications and skills that might not directly translate to those that are asked for on a résumé but would be virtually similar in understanding, such as having AWS vs Azure cloud certification. This is a component that is similar to the previous limitation and would require a program that directly examines and learns which skills are transferable. The two previous points mentioned indicate the primitive implementation of the program that compares the job advertisement and the candidate résumé and would require further research.

5.3 Experimental Proposal

We propose an experiment where participants are recruited and given a monetary incentive for their participation. Ideally, the experiment would have a wide range of participants in the technology recruiting/hiring field. The participants would be asked to complete the MDP simulation where they choose the best candidates for each job description and then show the

results of their preferences. They will then be asked to complete the same exercise again and the results between the two trials will be compared. They will be asked to do this simulation once again and their results will be compared with those from the first two trials. The results will consist of which action they were most likely to take in each state, and the visual representations of that, as well as the reward, gained in each state for an action, and lastly what the ideal candidate is. The experiment would be designed to determine how effective the MDP Game is at increasing awareness of bias/preferences for certain attributes among participants. The evaluation would focus on whether individuals saw a change in accuracy in finding the most optimal candidate when shown the results of their choices from across the three trials. The experiment would aid in determining the effectiveness of the MDP game in a real-world context in determining if it helps in the reduction of bias and aids participants in identifying areas that they are predisposed to be more partial to that don't necessarily correlate to the areas of qualification for a candidate.

Bibliography

[1] Anna N. Rafferty and Matei Zaharia and Thomas L. Griffiths. 2014. Optimally designing games for behavioral research. Proceedings: Mathematical, Physical and Engineering Sciences, 2167. Royal Society, 1--20. http://www.jstor.org/stable/24508470

[2] Betsy Anne Williams and Catherine F. Brooks and Yotam Shmargad. 2018. How Algorithms Discriminate Based on Data They Lack: Challenges, Solutions, and Policy Implications. Journal of Information Policy. Penn State University Press, 78--115. https://www.jstor.org/stable/10.5325/jinfopoli.8.2018.0078

[3] Cynthia Rudin and Joanna Radin. 2019. Why Are We Using Black Box Models in AI When We Don't Need To? A Lesson From An Explainable AI Competition. Harvard Data Science Review, 2, (2019-11-22). https://hdsr.mitpress.mit.edu/pub/f9kuryi8

[4] David Hausman. 2012. How Congress Could Reduce Job Discrimination by Promoting Anonymous Hiring. Stanford Law Review, 5. Stanford Law Review, 1343--1369. http://www.jstor.org/stable/41511137

[5] Fan Yang and Liu Leqi and Yifan Wu and Zachary C. Lipton and Pradeep Ravikumar and William W. Cohen and Tom M. Mitchell. 2019. Game Design for Eliciting Distinguishable Behavior. CoRR. http://arxiv.org/abs/1912.06074

[6] France Winddance Twine. 2018. Technology's Invisible Women: Black Geek Girls in Silicon Valley and the Failure of Diversity Initiatives. International Journal of Critical Diversity Studies, 1. Pluto Journals, 58--79. https://www.jstor.org/stable/10.13169/intecritdivestud.1.1.0058.

[7] G. Stoney Alder and Joseph Gilbert. 2006. Achieving Ethics and Fairness in Hiring: Going beyond the Law. Journal of Business Ethics, 4. Springer, 449--464. http://www.jstor.org/stable/25123928

[8] Gustavo Tondello and Deltcho Valtchanov and Adrian Reetz and Rina Wehbe and Rita Orji and Lennart Nacke. 2018. Towards a Trait Model of Video Game Preferences. International Journal of Human-Computer Interaction. 1-17. DOI: 10.1080/10447318.2018.1461765

[9] Indre Zliobaite. 2015. A survey on measuring indirect discrimination in machine learning.CoRR, abs/1511.00148. http://arxiv.org/abs/1511.00148

[10] Jacob Russell and Eugen Santos. 2019. Explaining reward functions in Markov Decision Processes. The Thirty-Second International Flairs Conference.

[11] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: a survey. J. Artif. Int. Res.4, 1 (January 1996), 237–285.

[12] Michael D. Cohen and James G. March and Johan P. Olsen. 1972. A Garbage Can Model of Organizational Choice. Administrative Science Quarterly, 1.[Sage Publications, Inc., Johnson Graduate School of Management, Cornell University], 1--25. http://www.jstor.org/stable/2392088.

[13] Mike Monteiro. 2019. Ruined by Design: How Designers Destroyed the World, and What We Can Do to Fix it. Mule Design.

[14] Nicol Turner Lee and Paul Resnick and Genie Barton. 2019. Algorithmic bias detection and mitigation: Best practices and policies to reduce consumer harms. Brookings, (May 2019).

[15] Phoebe K. Chua and Melissa Mazmanian. 2020. Are You One of Us? Current Hiring Practices Suggest the Potential for Class Biases in Large Tech Companies. Proc. ACM

Hum.-Comput. Interact. 4, CSCW2, Article 143 (October 2020), 20 pages. DOI:https://doi.org/10.1145/3415214

[16] Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.

[17] Rishin Haldar and Debajyoti Mukhopadhyay. 2011. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. Computing Research Repository - CORR.

[18] Roberto M. Fernandez and Emilio J. Castilla and Paul Moore. 2000. Social Capital at Work: Networks and Employment at a Phone Center. American Journal of Sociology, 5, University of Chicago Press,1288--1356. http://www.jstor.org/stable/3003768

[19] Ruwaid Louis and David Yu. 2019. A study of the exploration/exploitation trade-off in reinforcement learning : Applied to autonomous driving. Dissertation.

[20] Sean McGregor and Hailey Buckingham and Thomas G. Dietterich and Rachel Houtman and Claire Montgomery and Ronald Metoyer. 2017. Interactive visualization for testing Markov Decision Processes: MDPVIS. Journal of Visual Languages & Computing, 93-106. DOI: https://doi.org/10.1016/j.jvlc.2016.10.007

[21] Sebastian Varges and Giuseppe Riccardi and Silvia Quarteroni and Alexei Ivanov. 2010. The exploration/exploitation trade-off in Reinforcement Learning for dialogue management. 479 - 484. DOI:10.1109/ASRU.2009.5373260.

[22] Selena Silva and Martin Kenney. 2018. Algorithms, Platforms, and Ethnic Bias: An Integrative Essay. Phylon (1960-), 1 & 2, 9--37.https://www.jstor.org/stable/26545017

[23] Steven D. Whitehead and Long-Ji Lin. 1995. Reinforcement learning of non-Markov decision processes. Artificial Intelligence,1. 271-306. DOI: https://doi.org/10.1016/0004-3702(94)00012-P

[24] Stuart J. Russell and Peter Norvig. 2020. Artificial intelligence: a modern approach. 4th edn, Boston: Pearson.

[25] Vladimir Kulyukin and Abraham Bookstein and Timo Raita. 2002. Generalized Hamming Distance. Information Retrieval. DOI: 10.1023/A:1020499411651

Appendix A

The below code base shows only the Iterative MDP Model. It was modified to take in user input to create the User-defined Model and the Q-learning Model.

```
1.
2.  rows = 5
3.  cols = 4
4.  grid2 = [[0 for i in range(cols)] for i in range(rows)] #5States x 4 Actions
5.  reward_grid2 = [[0 for i in range(cols)] for i in range(rows)] #5States x 4 Actions
6.
7.  #Keeping track of how reward is calculated
8.  Lev = Levenshtein
9.
10. #For the reward, I have chosen a gradually increasing reward as the goal is approached
    with obstacles based on the Lev score
11. reward = []
12.
13. n_episodes = 30
14.
15.    print("Decision Modeling MDP Choice")
16.    print("=========================================")
17.    print("Follow the prompted Instructions")
18.
19.    def mdp_game():
20.   r = 0
21.
22.   pi = []
23.   input_state = rm.randint(0,5) #before for loop
24.
25.   prob = rm.uniform(0, 1)
26.       print(prob)
27.
28.
29.       if (prob > .2):
30.      state = input_state
31.          print("In STATE", state)
32.       else:
33.          if(prob > .7):
34.         state = rm.randint(3,5)
35.              print("In STATE", state)
36.          else:
37.         state = rm.randint(1,5)
38.              print("In STATE", state)
39.
40.       if( state == 1 ):
```

```
41.          print("You have selected Education")
42.
43.          print("\nWhich action would you like to take?")
44.          print(" A) Examine Another Section\n B) Accept the Candidate's Application\n
    C) Decline the Candidate's Application \n\nEnter A or B or C:\n ")
45.
46.    #Automate the choice for action
47.       list1=['a', 'b', 'c','d']
48.       b=rm.randint(0,3)
49.       action = list1[b]
50.
51.          if( action == ('A') or action == ('a')):
52.             print("You have chosen to accept the candidate's application")
53.             print("Accept")
54.
55.          grid2[0][0] += 1
56.
57.          #Keeping track of reward
58.               if( Lev >= 75):
59.             r += 1
60.             reward.append(r)
61.             reward[0] += 1
62.               else:
63.             r -= .33
64.             reward.append(r)
65.             reward[0] += 1
66.
67.          reward_grid2[0][0] += r
68.
69.           elif( action == ("B") or action == ("b") ):
70.             print( "You have chosen to decline the candidate's application" )
71.             print("Decline")
72.
73.          grid2[0][1] += 1
74.
75.          #Keeping track of reward
76.               if( Lev >= 75):
77.             r -= 1
78.             reward.append(r)
79.             reward[0] += 1
80.               else:
81.             r += .33
82.             reward.append(r)
83.             reward[0] += 1
84.
```

```
85.         reward_grid2[0][1] += r
86.
87.       elif( action == ("C") or action == ("c") ):
88.          print( "You have chosen to continue viewing the candidate's application" )
89.          print("Continue")
90.       grid2[0][2] += 1
91.       mdp_game()
92.
93.       #Keeping track of reward
94.          if( Lev >= 50):
95.          r += .33
96.          reward.append(r)
97.          reward[0] += 1
98.           else:
99.          r -= .33
100.              reward.append(r)
101.              reward[0] += 1
102.
103.          reward_grid2[0][2] += r
104.
105.       elif( action == ("D") or action == ("d") ):
106.          print( "You have decided to view another Candidate" )
107.          grid2[0][3] += 1
108.          #          choice = int( input("Please Select which candidate you would like to
    examine .\n Press 1 and hit enter.\n Press 2 and hit enter.\n Press 3 and hit enter.\n"))
109.          choice = rm.randint(0,3)
110.          candidates(choice)
111.
112.          #Keeping track of reward
113.          if( Lev >= 50):
114.             r -= .33
115.             reward.append(r)
116.             reward[0] += 1
117.          else:
118.             r += .33
119.             reward.append(r)
120.             reward[0] += 1
121.
122.          reward_grid2[0][3] += r
123.
124.       else:
125.          print("Invalid Action Chosen!")
126.
127.    elif( state == 2 ):
128.       print( "You have selected Skills" )
```

```
129.
130.         print("\nWhich action would you like to take?")
131.         print(" A) Examine Another Section\n B) Accept the Candidate's Application\n
    C) Decline the Candidate's Application \n\nEnter A or B or C:\n ")
132.
133.       #Automate the choice for action
134.         list1=['a', 'b', 'c', 'd']
135.         b=rm.randint(0,3)
136.         action = list1[b]
137.
138.         if( action == ('A') or action == ('a')):
139.           print("You have chosen to accept the candidate's application")
140.           print("Accept")
141.
142.           grid2[1][0] += 1
143.
144.         #Keeping track of reward
145.           if( Lev >= 75):
146.             r += 1
147.             reward.append(r)
148.             reward[0] += 1
149.           else:
150.             r -= .33
151.             reward.append(r)
152.             reward[0] += 1
153.
154.           reward_grid2[1][0] += r
155.
156.         elif( action == ("B") or action == ("b") ):
157.           print( "You have chosen to decline the candidate's application" )
158.           print("Decline")
159.
160.           grid2[1][1] += 1
161.
162.           #Keeping track of reward
163.           if( Lev >= 75):
164.             r -= 1
165.             reward.append(r)
166.             reward[0] += 1
167.           else:
168.             r += .33
169.             reward.append(r)
170.             reward[0] += 1
171.
172.           reward_grid2[1][1] += r
```

```
173.
174.         elif( action == ("C") or action == ("c") ):
175.             print( "You have chosen to continue viewing the candidate's application" )
176.             print("Continue")
177.
178.             grid2[1][2] += 1
179.             mdp_game()
180.
181.             #Keeping track of reward
182.             if( Lev >= 50):
183.                 r += .33
184.                 reward.append(r)
185.                 reward[0] += 1
186.             else:
187.                 r -= .33
188.                 reward.append(r)
189.                 reward[0] += 1
190.
191.             reward_grid2[1][2] += r
192.
193.         elif( action == ("D") or action == ("d") ):
194.             print( "You have decided to view another Candidate" )
195.             grid2[1][3] += 1
196.
197.     #           choice = int( input("Please Select which candidate you would like to
    examine .\n Press 1 and hit enter.\n Press 2 and hit enter.\n Press 3 and hit enter.\n"))
198.             choice = rm.randint(0,3)
199.             candidates(choice)
200.
201.             #Keeping track of reward
202.             if( Lev >= 50):
203.                 r -= .33
204.                 reward.append(r)
205.                 reward[0] += 1
206.             else:
207.                 r += .33
208.                 reward.append(r)
209.                 reward[0] += 1
210.
211.             reward_grid2[1][3] += r
212.
213.         else:
214.             print("Invalid Action Chosen!")
215.
216.     elif( state == 3 ):
```

```
        print( "You have selected Experience" )

        print("\nWhich action would you like to take?")
        print(" A) Examine Another Section\n B) Accept the Candidate's Application\n
    C) Decline the Candidate's Application \n\nEnter A or B or C:\n ")

    #Automate the choice for action
        list1=['a', 'b', 'c','d']
        b=rm.randint(0,3)
        action = list1[b]

        if( action == ('A') or action == ('a')):
            print("You have chosen to accept the candidate's application")
            print("Accept")

            grid2[2][0] += 1
            #Keeping track of reward
            if( Lev >= 75):
                r += 1
                reward.append(r)
                reward[0] += 1
            else:
                r -= .33
                reward.append(r)
                reward[0] += 1

            reward_grid2[2][0] += r

        elif( action == ("B") or action == ("b") ):
            print( "You have chosen to decline the candidate's application" )
            print("Decline")

            grid2[2][1] += 1

            #Keeping track of reward
            if( Lev >= 75):
                r -= 1
                reward.append(r)
                reward[0] += 1
            else:
                r += .33
                reward.append(r)
                reward[0] += 1

            reward_grid2[2][1] += r
```

```
261.
262.         elif( action == ("C") or action == ("c") ):
263.             print( "You have chosen to continue viewing the candidate's application" )
264.             print("Continue")
265.             grid2[2][2] += 1
266.             mdp_game()
267.
268.             #Keeping track of reward
269.             if( Lev >= 50):
270.                 r += .33
271.                 reward.append(r)
272.                 reward[0] += 1
273.             else:
274.                 r -= .33
275.                 reward.append(r)
276.                 reward[0] += 1
277.
278.             reward_grid2[2][2] += r
279.
280.         elif( action == ("D") or action == ("d") ):
281.             print( "You have decided to view another Candidate" )
282.             grid2[2][3] += 1
283.     #           choice = int( input("Please Select which candidate you would like to
    examine .\n Press 1 and hit enter.\n Press 2 and hit enter.\n Press 3 and hit enter.\n"))
284.             choice = rm.randint(0,3)
285.             candidates(choice)
286.
287.             #Keeping track of reward
288.             if( Lev >= 50):
289.                 r -= .33
290.                 reward.append(r)
291.                 reward[0] += 1
292.             else:
293.                 r += .33
294.                 reward.append(r)
295.                 reward[0] += 1
296.
297.             reward_grid2[2][3] += r
298.
299.         else:
300.             print("Invalid Action Chosen!")
301.
302.     elif( state == 4 ):
303.         print( "You have selected Certifications" )
304.
```

```
305.        print("\nWhich action would you like to take?")
306.        print(" A) Examine Another Section\n B) Accept the Candidate's Application\n
    C) Decline the Candidate's Application \n\nEnter A or B or C:\n ")
307.
308.      #Automate the choice for action
309.        list1=['a', 'b', 'c','d']
310.        b=rm.randint(0,3)
311.        action = list1[b]
312.
313.        if( action == ('A') or action == ('a')):
314.           print("You have chosen to accept the candidate's application")
315.           print("Accept")
316.
317.           grid2[3][0] += 1
318.
319.        #Keeping track of reward
320.           if( Lev >= 75):
321.              r += 1
322.              reward.append(r)
323.              reward[0] += 1
324.           else:
325.              r -= .33
326.              reward.append(r)
327.              reward[0] += 1
328.
329.           reward_grid2[3][0] += r
330.
331.        elif( action == ("B") or action == ("b") ):
332.           print( "You have chosen to decline the candidate's application" )
333.           print("Decline")
334.
335.           grid2[3][1] += 1
336.
337.           #Keeping track of reward
338.           if( Lev >= 75):
339.              r -= 1
340.              reward.append(r)
341.              reward[0] += 1
342.           else:
343.              r += .33
344.              reward.append(r)
345.              reward[0] += 1
346.
347.           reward_grid2[3][1] += r
348.
```

```
349.        elif( action == ("C") or action == ("c") ):
350.            print( "You have chosen to continue viewing the candidate's application" )
351.            print("Continue")
352.
353.            grid2[3][2] += 1
354.            mdp_game()
355.
356.            #Keeping track of reward
357.            if( Lev >= 50):
358.                r += .33
359.                reward.append(r)
360.                reward[0] += 1
361.            else:
362.                r -= .33
363.                reward.append(r)
364.                reward[0] += 1
365.
366.            reward_grid2[3][2] += r
367.
368.        elif( action == ("D") or action == ("d") ):
369.            print( "You have decided to view another Candidate" )
370.            grid2[3][3] += 1
371.
372.    #           choice = int( input("Please Select which candidate you would like to
    examine .\n Press 1 and hit enter.\n Press 2 and hit enter.\n Press 3 and hit enter.\n"))
373.            choice = rm.randint(0,3)
374.            candidates(choice)
375.
376.            #Keeping track of reward
377.            if( Lev >= 50):
378.                r -= .33
379.                reward.append(r)
380.                reward[0] += 1
381.            else:
382.                r += .33
383.                reward.append(r)
384.                reward[0] += 1
385.
386.            reward_grid2[3][3] += r
387.
388.        else:
389.            print("Invalid Action Chosen!")
390.
391.    elif( state == 5 ):
392.        print( "You have selected Referral" )
```

```
393.
394.        print("\nWhich action would you like to take?")
395.        print(" A) Examine Another Section\n B) Accept the Candidate's Application\n
    C) Decline the Candidate's Application \n\nEnter A or B or C:\n ")
396.
397.      #Automate the choice for action
398.        list1=['a', 'b', 'c','d']
399.        b=rm.randint(0,3)
400.        action = list1[b]
401.
402.        if( action == ('A') or action == ('a')):
403.          print("You have chosen to accept the candidate's application")
404.          print("Accept")
405.
406.          grid2[4][0] += 1
407.
408.        #Keeping track of reward
409.          if( Lev >= 75):
410.            r += 1
411.            reward.append(r)
412.            reward[0] += 1
413.          else:
414.            r -= .33
415.            reward.append(r)
416.            reward[0] += 1
417.
418.          reward_grid2[4][0] += r
419.
420.        elif( action == ("B") or action == ("b") ):
421.          print( "You have chosen to decline the candidate's application" )
422.          print("Decline")
423.
424.          grid2[4][1] += 1
425.
426.          #Keeping track of reward
427.          if( Lev >= 75):
428.            r -= 1
429.            reward.append(r)
430.            reward[0] += 1
431.          else:
432.            r += .33
433.            reward.append(r)
434.            reward[0] += 1
435.
436.          reward_grid2[4][1] += r
```

```
437.
438.        elif( action == ("C") or action == ("c") ):
439.            print( "You have chosen to continue viewing the candidate's application" )
440.            print("Continue")
441.
442.            grid2[4][2] += 1
443.            mdp_game()
444.
445.            #Keeping track of reward
446.            if( Lev >= 50):
447.                r += .33
448.                reward.append(r)
449.                reward[0] += 1
450.            else:
451.                r -= .33
452.                reward.append(r)
453.                reward[0] += 1
454.
455.            reward_grid2[4][2] += r
456.
457.        elif( action == ("D") or action == ("d") ):
458.            print( "You have decided to view another Candidate" )
459.
460.            grid2[4][3] += 1
461.    #           choice = int( input("Please Select which candidate you would like to
    examine .\n Press 1 and hit enter.\n Press 2 and hit enter.\n Press 3 and hit enter.\n"))
462.            choice = rm.randint(0,3)
463.            candidates(choice)
464.
465.            #Keeping track of reward
466.            if( Lev >= 50):
467.                r -= .33
468.                reward.append(r)
469.                reward[0] += 1
470.            else:
471.                r += .33
472.                reward.append(r)
473.                reward[0] += 1
474.
475.            reward_grid2[4][3] += r
476.
477.        else:
478.            print("Invalid Action Chosen!")
479.
480.    else:
```

```
481.         print("Invalid choice!")
482.     print("Reward: ", reward)
483.
484.
485.
486. def candidates(choice):
487.
488.     if choice == 1:
489.         print("You have chosen Candidate 1:")
490.         mdp_game()
491.     elif choice == 2:
492.         print("You have chosen Candidate 2:")
493.         mdp_game()
494.     elif choice == 3:
495.         print("You have chosen Candidate 3:")
496.         mdp_game()
497.
498.
499. def check(grid2, val):
500.     return(all(x > val for x in grid2[:][0]))
501.
502.
503. for e in range(int(n_episodes)):
504.     print("--------------------------Episode--------------------------", e)
505.     while True:
506.             print("STARTING GAME")
507.             choice = rm.randint(0,1)
508.         #       choice = int( input("Please Select which candidate you would like to
    examine .\n Press 1 and hit enter.\n Press 2 and hit enter.\n Press 3 and hit enter.\n"))
509.             candidates(choice)
510.
511.             #Automate Restart of game
512.             list1=['N', 'Y']
513.             b=rm.randint(0,1)
514.             restart = list1[b]
515.             print("I HAVE ACCESS", grid2[:][0])
516.             acct = [i[0] for i in grid2]
517.             print(acct)
518.
519.             if restart == 'N':
520.                 print("GAME OVER")
521.                 break
522.             elif restart == 'Y':
523.                 print("NEW GAME")
524.
```

```
525.
526.  print("StatesxActions Grid2: ", grid2)
527.  print("StatesxActions Reward Grid2: ", reward_grid2)
528.  print("reward:", sum(reward))
529.  print("reward Summed Array:", (reward))
530.
```

The below code creates the Q-learning Function.

```
1.  # defines the reward/connection graph
2.  #Accept, decline, continue, other candidate
3.  #Edu, ski, exp, cert, ref
4.  RP = np.array([[100, -.25,  .50, .50],
5.              [ 100, -.25, .66, .5],
6.              [ 100, -.5,  .33, 100],
7.              [.66,  .33,  .5, .5 ],
8.              [ 0, -.33, .5, .5]]).astype("float32")
9.
10. RN = np.array([[0, .33,  .66, .66],
11.             [ 0, .66, .33, 100],
12.             [ -.33, .33,  .33, 100],
13.             [0,  .33,  0, .66 ],
14.             [ -.33, 0, .66, 100]]).astype("float32")
15.
16.
17. Lev = Levenshtein
18.
19.    if (Lev > 50):
20.    r = RP
21.        print("IN RP")
22.    else:
23.    r = RN
24.        print("IN RN")
25.
26.
27. q = np.zeros_like(r)
28.
29. gamma = 0.8
30. alpha = 1.
31. n_episodes = 50
32. n_states = 5
33. n_actions = 4
34. epsilon = 0.05
35. random_state = np.random.RandomState(1999)
36.
```

```
    state_grid = [[0 for i in range(n_states)] for i in range(n_actions)] #2States x 3 Actions
    reward_grid = [[0 for i in range(n_states)] for i in range(n_actions)] #2States x 3 Actions


    def state_space(s):
        if(s == 0):
            print("In State Education")
        elif(s == 1):
            print("In State Skills")
        elif(s == 2):
            print("In State Experience")
        elif(s == 3):
            print("In State Certifications")
        elif(s == 4):
            print("In State Referrals")
        return


    def action_space(a):
        if(a == 0):
            print("Action: Accept Qualifications")
        elif(a == 1):
            print("Action: Decline Qualifications")
        elif(a == 2):
            print("Action: Review Qualifications")
        elif(a == 3):
            print("Action: Examine Other Candidates")
        return


    def update_q(state, new_state, action):

    q[state, action] = q[state, action] + alpha * (r[state, action] + gamma * max(q[new_state, :]) - q[state, action])
        return r[state, action]


    def mdp():
        for e in range(int(n_episodes)):
            print("-------------------------episode-------------------------", e)

        states = list(range(n_states))

```

```
79.         print("states", states)
80.
81.     random_state.shuffle(states) #shuffles order of states
82.     curr_state = states[0] #assigns current state as a random state
83.
84.     goal = False
85.
86.         print("current_state", curr_state)
87.
88.     prob = rm.uniform(0, 1)
89.         print(prob)
90.
91.         if (prob > .2):
92.        state = curr_state
93.             print("In STATE", state)
94.         else:
95.             if(prob > .7):
96.          curr_state = rm.randint(3,5)
97.                 print("In STATE", state)
98.             else:
99.          curr_state = rm.randint(1,5)
100.                print("In STATE", state)
101.
102.            print("current_state", curr_state)
103.
104.        if (curr_state == 0):
105.            print("STATE 1")
106.
107.            while not goal:
108.                # epsilon greedy
109.                valid_moves = r[curr_state] >= 0
110.                print("r[current_state]", r[curr_state])
111.
112.                if rm.uniform(0, 1) < epsilon: #Explore
113.                    print("EXPLORING")
114.                    actions = np.array(list(range(n_actions)))
115.                    random_state.shuffle(actions)
116.                    action = actions[0]
117.                    print("EXPLORING ACTION PERFORMED", action)
118.                    new_state = action #choose random action
119.
120.                    print("next_state1", new_state)
121.                    state_space(new_state)
122.
123.                else: #Exploit Q table
```

```
124.                print("EXPLOITING")
125.                if np.sum(q[curr_state]) > 0:
126.                    action = np.argmax(q[curr_state]) #action is the max of the q-val in the
    current state
127.                else:
128.                    # Don't allow invalid moves at the start
129.                    # Just take a random move
130.                    actions = np.array(list(range(n_actions)))
131.                    random_state.shuffle(actions)
132.                    action = actions[0]
133.                    print("EXPLOITING ACTION PERFORMED", action)
134.
135.
136.                new_state = action
137.                state_space(new_state)
138.
139.            value = update_q(curr_state, new_state, action)
140.            print("value", value)
141.
142.            # Goal state has reward 100
143.            if value > 1:
144.                goal = True
145.            curr_state = new_state
146.
147.
148.        if (curr_state == 1):
149.            print("STATE 2")
150.
151.            while not goal:
152.                # epsilon greedy
153.                valid_moves = r[curr_state] >= 0
154.                print("r[current_state]", r[curr_state])
155.
156.                if rm.uniform(0, 1) < epsilon: #Explore
157.                    print("EXPLORING")
158.                    actions = np.array(list(range(n_actions)))
159.                    random_state.shuffle(actions)
160.                    action = actions[0]
161.                    print("EXPLORING ACTION PERFORMED", action)
162.                    new_state = action #choose random action
163.
164.                    print("next_state1", new_state)
165.                    state_space(new_state)
166.
167.                else: #Exploit Q table
```

```
168.                print("EXPLOITING")
169.                if np.sum(q[curr_state]) > 0:
170.                    action = np.argmax(q[curr_state]) #action is the max of the q-val in the
    current state
171.                else:
172.                    # Don't allow invalid moves at the start
173.                    # Just take a random move
174.                    actions = np.array(list(range(n_actions)))
175.                    random_state.shuffle(actions)
176.                    action = actions[0]
177.                    print("EXPLOITING ACTION PERFORMED", action)
178.
179.
180.                new_state = action
181.                state_space(new_state)
182.
183.            value = update_q(curr_state, new_state, action)
184.            print("value", value)
185.
186.            # Goal state has reward 100
187.            if value > 1:
188.                goal = True
189.            curr_state = new_state
190.
191.        if (curr_state == 2):
192.            print("STATE 3")
193.
194.            while not goal:
195.                # epsilon greedy
196.                valid_moves = r[curr_state] >= 0
197.                print("r[current_state]", r[curr_state])
198.
199.                if rm.uniform(0, 1) < epsilon: #Explore
200.                    print("EXPLORING")
201.                    actions = np.array(list(range(n_actions)))
202.                    random_state.shuffle(actions)
203.                    action = actions[0]
204.                    print("EXPLORING ACTION PERFORMED", action)
205.                    new_state = action #choose random action
206.
207.                    print("next_state1", new_state)
208.                    state_space(new_state)
209.
210.                else: #Exploit Q table
211.                    print("EXPLOITING")
```

```
212.                 if np.sum(q[curr_state]) > 0:
213.                     action = np.argmax(q[curr_state]) #action is the max of the q-val in the
    current state
214.                 else:
215.                     # Don't allow invalid moves at the start
216.                     # Just take a random move
217.                     actions = np.array(list(range(n_actions)))
218.                     random_state.shuffle(actions)
219.                     action = actions[0]
220.                     print("EXPLOITING ACTION PERFORMED", action)
221.
222.
223.                 new_state = action
224.                 state_space(new_state)
225.
226.             value = update_q(curr_state, new_state, action)
227.             print("value", value)
228.
229.             # Goal state has reward 100
230.             if value > 1:
231.                 goal = True
232.             curr_state = new_state
233.
234.         if (curr_state == 3):
235.             print("STATE 4")
236.
237.             while not goal:
238.                 # epsilon greedy
239.                 valid_moves = r[curr_state] >= 0
240.                 print("r[current_state]", r[curr_state])
241.
242.                 if rm.uniform(0, 1) < epsilon: #Explore
243.                     print("EXPLORING")
244.                     actions = np.array(list(range(n_actions)))
245.                     random_state.shuffle(actions)
246.                     action = actions[0]
247.                     print("EXPLORING ACTION PERFORMED", action)
248.                     new_state = action #choose random action
249.
250.                     print("next_state1", new_state)
251.                     state_space(new_state)
252.
253.                 else: #Exploit Q table
254.                     print("EXPLOITING")
255.                     if np.sum(q[curr_state]) > 0:
```

```
                action = np.argmax(q[curr_state]) #action is the max of the q-val in the current state
            else:
                # Don't allow invalid moves at the start
                # Just take a random move
                actions = np.array(list(range(n_actions)))
                random_state.shuffle(actions)
                action = actions[0]
                print("EXPLOITING ACTION PERFORMED", action)


            new_state = action
            state_space(new_state)

        value = update_q(curr_state, new_state, action)
        print("value", value)

        # Goal state has reward 100
        if value > 1:
            goal = True
        curr_state = new_state

    if (curr_state == 4):
        print("STATE 5")

        while not goal:
            # epsilon greedy
            valid_moves = r[curr_state] >= 0
            print("r[current_state]", r[curr_state])

            if rm.uniform(0, 1) < epsilon: #Explore
                print("EXPLORING")
                actions = np.array(list(range(n_actions)))
                random_state.shuffle(actions)
                action = actions[0]
                print("EXPLORING ACTION PERFORMED", action)
                new_state = action #choose random action

                print("next_state1", new_state)
                state_space(new_state)

            else: #Exploit Q table
                print("EXPLOITING")
                if np.sum(q[curr_state]) > 0:
```

```
                action = np.argmax(q[curr_state]) #action is the max of the q-val in the current state
            else:
                # Don't allow invalid moves at the start
                # Just take a random move
                actions = np.array(list(range(n_actions)))
                random_state.shuffle(actions)
                action = actions[0]
                print("EXPLOITING ACTION PERFORMED", action)


            new_state = action
            state_space(new_state)

        value = update_q(curr_state, new_state, action)
        print("value", value)

        # Goal state has reward 100
        if value > 1:
            goal = True
        curr_state = new_state


    print(q)
mdp()
```

The Below Code is responsible for the visualization of the Heat Maps and Donut Chart

```
#Heat Map
data = q.tolist()
length = max(map(len, data))
y=np.array([xi+[None]*(length-len(xi)) for xi in data])

df = pd.DataFrame(data=data,columns=['Accept','Reject','Continue', 'Candidate'],
index=['Education','Skills', 'Experience', 'Certifications', 'Referrals'])
print(df)

fig = plt.figure(num=None, figsize=(10, 10), dpi=80, facecolor='w', edgecolor='k')

cmap = sns.cubehelix_palette(light=1, as_cmap=True)
sns.set(font_scale=1.4)
 res = sns.heatmap(df, annot=True, vmin=0.0, vmax=300.0,
                fmt='.2f', cmap=cmap, cbar_kws={"shrink": .82},
                linewidths=0.1, linecolor='gray')

```

```
res.invert_yaxis()

plt.title('Stochastic Q-Learning MDP Hiring Modeling')
plt.savefig('Stochastic_Q_Learning_Hiring_Modeling.png', bbox_inches='tight')
plt.show()
```

```
#Donut Chart

fig, ax = plt.subplots()

size = 1.75
vals = np.array(data)
print(data)

topic = ['Education','Skills','Experience','Certifications','Referrals']
action = ['Accept', 'Decline', 'Continue', 'Candidate']

outer_label = list(topic)
inner_label = list(action)

cmap = plt.get_cmap("tab20c")
outer_colors = cmap(np.arange(6)*4)
inner_colors = cmap([1, 2, 5, 6, 9, 10])

explode = (0.2,0.2,0.2,0.2,0.2)

colors = ['#ffcf41', '#022c5c', '#0094b6', '#e44813', '#ba1615']
colors_actions = ['#cec5b1','#99a210', '#64a5bb','#ffd8d1']

ax.pie(vals.sum(axis=1), labels = outer_label, radius=3, colors=colors,
       wedgeprops=dict(width=size, edgecolor='w'), autopct='%1.f%%',
textprops={'fontsize': 20})

ax.pie(vals.flatten(), radius=2, colors=colors_actions)

centre_circle = plt.Circle((0,0),1.5,color='black', fc='white',linewidth=0)
# fig = plt.gcf()
fig.gca().add_artist(centre_circle)

ax.set(aspect="auto", title='Deterministic Q-Learning Policy Model')

plt.axis('equal')
plt.tight_layout()

#Create the legend
ax.legend(outer_label,
          title ="MDP Groups",
          loc ="center left",
          bbox_to_anchor =(.75, 0, 0.5, 1))

fig.set_size_inches(18.5, 10.5)

plt.savefig('Deterministic_Q_Learning_Hiring_Modeling_Pie.png', )

plt.show()
```