Spring 2017

A Markov Decision Process Approach to Adaptive Contact Strategies

Artur Grygorian

A MARKOV DECISION PROCESS APPROACH TO ADAPTIVE CONTACT STRATEGIES

by

ARTUR GRYGORIAN

(Under the Direction of Stephen Carden)

ABSTRACT

In the field of survey methodology, optimizing contact strategies helps organizations increase response rates using their allocated budget. Markov Decision Processes (MDP) are widely used to model decision-making strategies in situations where the outcomes have a random component. In this research we use MDPs and adaptive sampling techniques to construct a strategy that, based on target audience characteristics, suggests the best contact policy. The data we use comes from the First Destination Survey conducted by the Office of Career Services at Georgia Southern University. The constructed model is quite flexible and can be used by other organizations to optimize their contact strategies.

Key Words: Markov Decision Process, MDP, Adaptive Sampling Methods, First Destination Survey(FDS), Policy Iteration, Q-learning

2009 *Mathematics Subject Classification*: 90C40

A MARKOV DECISION PROCESS APPROACH TO ADAPTIVE CONTACT STRATEGIES

by

ARTUR GRYGORIAN

B.S., Odessa I.I. Mechnikov National University, Ukraine, 2011

M.S., Odessa I.I. Mechnikov National University, Ukraine, 2013

M.A., University of Houston, United States, 2014

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment

of the Requirement for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

A MARKOV DECISION PROCESS APPROACH TO ADAPTIVE CONTACT STRATEGIES

by

ARTUR GRYGORIAN

Major Professor: Stephen Carden

Committee: Tharanga Wickramarachchi

Arpita Chatterjee

Emil Iacob

Electronic Version Approved:

May 2017

DEDICATION

I dedicate this thesis to my family, who always stood by me. It is their unconditional love that motivates me to set higher standards and become a better human being.

ACKNOWLEDGMENTS

I would like to express the deepest appreciation to my thesis advisor, Doctor Stephen Carden. He is a tremendous mentor for me. He always encourages my research and helps me to develop my research skills. His advice on both research as well as on my career have been priceless.

A special thank to Career Services Department, particularly to my supervisor and director of Career Services Philip Bruce. They allowed me to use some of the aspects of First Destination Survey data in my thesis, which helps us better construct the model.

My sincere thanks also go to my family. They always support and motivate me to pursue my dream.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF SYMBOLS

- S - State space
- A - Action space
- $R(s_t, a_t)$ - Reward from using action a_t in state s_t
- $P(s_{t+1}|s_t, a_t)$ - Transition probability
- γ - Discount factor
- π - Policy function
- $V^{\pi}(s)$ - Value of a state s under some policy π
- $Q^{\pi}(s, a)$ - Action-value function for policy π
- α - Learning rate

CHAPTER 1

INTRODUCTION

When conducting a survey, the response rate is considered one of the most important issues. Presently, there are a lot of ways to reach an audience, including but not limited to: email, phone, mail, and in person. Understanding the characteristics of the target audience is the key feature to not only increase the response rate but also reduce the costs associated with it. This research uses data collected from the First Destination Survey (FDS) conducted by the office of Career Services at Georgia Southern University (GSU). This data is collected every semester and will be used to successfully create a model that can allow for the adaptation of communication practices to reach the target audience. Additionally, the research seeks to recommend a similar model to organizations in an effort to aid them in effectively reaching their target audience. This model incorporates some ideas from adaptive sampling design and Markov Decision Processes.

1.1 First Destination Survey

Career Services at GSU conducts the FDS among newly graduating students. The collected data captures information about how new college graduates do in their careers within six months of graduation. One issue common to all surveys is increasing the response rate. When conducting the FDS survey, Career Services focuses on increasing student response rates while also being aware of budget constraints. Increased response rates lead to an increase in the amount and quality of data. With more data, we can more clearly answer questions like:

- What kind of jobs are graduate students more likely to get after graduation?
- What is the hiring rate after graduation?

- What is the average salary of graduates from each department?
- How many students were employed locally as opposed to out of state?

Initially, the survey was conducted in several stages. In the first stage, an email was sent to graduates with a link to the survey beginning approximately one month before graduation and continued to be sent every two weeks to all non-respondents. After some period of time, Career Services staff started to contact all nonrespondents by phone to ask them to complete the survey.

1.2 Adaptive Sampling Design

It is an accepted research practice for a researcher to make sampling decisions throughout the process based on real-time observations. One example of the aforementioned practice is adaptive sampling. Adaptive sampling is a sampling method where the population sample may depend on results observed during the survey[1]. Adaptive sampling allows a researcher to take advantage of specific characteristics of a population in an effort to increase sample size and/or reduce cost of the survey. Adaptive sampling was first mentioned by Debabrata Basu in 1969 [2]. He argued that the most efficient sampling designs are the ones with selection probabilities that depend on observed values. Because the target audience for the FDS is recent GSU graduates, it is critical to the success of the survey that enough data is received from students with varying characteristics such as: departments, majors, age groups, ethnicity, gender, and race. Taking inspiration from adaptive sampling methods, this thesis will develop adaptive contact strategies to optimize response rates for each possible cluster of students.

CHAPTER 2

MARKOV DECISION PROCESS (MDP)

2.1 A Markov Decision Process Perspective

Adaptive contact strategies are sequential decision-making problems. For example, after failing to reach a person by phone, you must decide how and whether to continue attempts.

One of the standard techniques that can model this situation is a Markov Decision Process (MDP). An MDP is a discrete time stochastic control process. MDP's are widely used in various areas including ecology, sports, robotics, inventory, manufacturing, biology etc. For example, Kohler [3] applied MDP in a game of darts. Based on the actual score, the player decides where to aim the next shot. The states of the system are the residual required scores to finish the game in a particular round. White [4] used MDP's to model manufacturing decisions of how much production is necessary to meet the target amount given a random number of products will be defective. The new state depends on the defective production of the current state and the decision of how much to produce.

2.2 Framework

An MDP models a system that can be in one of a fixed set of states. At specified points in time, a controller observes the state and chooses an action from a set of actions. As a result of this action, the system transitions to a new state according to a probability distribution determined by both the previous state and the action made. At the next point in time, the system faces a similar situation and the process repeats. While it is possible that the set of permissible actions might be different, we only consider the case with the same set of actions at each point of time. Also,

after every action the controller receives a reward or cost associated with the state and action. The reward/cost may be random variables. The goal of this process is to come up with a function that dictates to the controller which action to use at each specific state that maximizes (minimises) some long-term measure of reward (cost).

2.3 State, Action, Probability, Reward Structure

An MDP is made of 5 components.

1. **The state space**

   Assume that we have a finite set of states denoted by $S = (s_1, s_2..., s_n)$.

2. **The action space**

   Assume that at specified points in time the controller observes the state s_t and chooses an action from a finite set of actions $A = (a_1, a_2, ..., a_m)$.

3. **Reward structure**

   At any given state and chosen action, the controller earns/bears an immediate reward/cost. There is no meaningful difference between reward and cost because maximizing rewards is the same as minimizing costs. These rewards are random variables which are conditionally independent of the history of the process given the present state and chosen action. Denote $R(s_t, a_t)$ as an immediate reward at time t, in a state s_t, with an action a_t.

4. **The transition probabilities**

   Also, let's assume that we are given a transition matrix $P(s_{t+1}|s_t, a_t)$, which for each pair of (s_t, a_t) gives a probability that next state will be s_{t+1}.

5. **Discount factor**

   In this setup we also include a discount factor $\gamma \in [0, 1]$ which represents the

difference in importance between immediate and future rewards. Smaller values of γ represent a greedy or impatient agent, while larger values reflect more patient agent.

So, an MDP is a 5-tuple $(S, A, R(s_t, a_t), P_t(s_{t+1}|s_t, a_t), \gamma)$.

2.4 Expected Total Discounted Reward Criterion

As said previously, the goal is to find an optimal policy, denoted by π^*, which is a function $(S \to A)$ that associates with each state an action which should yield high rewards.

Define the value of a state s under some policy π as

$$V^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t))|s_0 = s\right]. \tag{2.1}$$

$V^{\pi}(s)$ is defined as the expected total discounted reward incurred when the policy π is implemented and the initial state is s. An optimal policy π^* should satisfy $V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s),\ \forall\ s \in S$, which is the best of all policies.

2.5 Bellman's Theorem

For any policy π and for all $s \in S$,

$$\begin{aligned}
V^{\pi}(s) &= E[\ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | s_0 = s\ ] \\
&= E[\ R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t))\ ] \\
&= E[\ R(s, \pi(s)) + \gamma \sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, \pi(s_{t+1}))\ ] \\
&= E[\ R(s, \pi(s))\ ]\ + \gamma E[\ \sum_{t=0}^{\infty} \gamma^t R(s_{t+1}, \pi(s_{t+1}))\ ] \\
&= E[\ R(s, \pi(s))\ ]\ + \gamma \sum_{t=0}^{\infty} \gamma^t E[\ R(s_{t+1}, \pi(s_{t+1}))\ ] \\
&= E[\ R(s, \pi(s))\ ]\ + \gamma \sum_{t=0}^{\infty} \gamma^t \sum_{u \in S} E[\ R(s_{t+1}, \pi(s_{t+1})) | s_1 = u\ ]\ P(u|s, \pi(s)) \\
&= E[\ R(s, \pi(s))\ ]\ + \gamma \sum_{u \in S} [\ \sum_{t=0}^{\infty} \gamma^t E[\ R(s_{t+1}, \pi(s_{t+1})) | s_1 = u\ ]\ ]\ P(u|s, \pi(s)) \\
&= E[\ R(s, \pi(s))\ ]\ + \gamma \sum_{u \in S} V^{\pi}(u) P(u|s, \pi(s)).
\end{aligned}$$

To sum it up:

$$V^{\pi}(s) = E[\ R(s, \pi(s))\ ]\ + \gamma \sum_{u \in S} V^{\pi}(u) P(u|s, \pi(s)), \tag{2.2}$$

which is called Bellman's equation. It shows a relationship between the value of a current state and values for the future states. It incorporates both finite and infinite state cases. For finite case, rewards become negligible starting from some period.

Theorem 2.1. *For an optimal policy* π^*

$$V^{\pi^*}(s) = \max_{a \in A} (E[\ R(s, a)\ ]\ + \gamma \sum_{u \in S} V^{\pi}(u) P(u|s, a)) \tag{2.3}$$

for any $s \in S$.

2.6 Policy Iteration

One of the standard methods to find the optimal policy for MDP is policy iteration, introduced by Ronald Arthur Howard in 1960 [5].

1. The algorithm starts with an arbitrary initial policy π_0, which is sometimes called a "dummy policy".

2. While $\pi_n \neq \pi_{n+1}$ repeat the following steps.

   (a) Solve the system of linear equations for V^{π_n}:

   $$V^{\pi_n}(s) = E[\ R(s, \pi(s))\ ]\ + \gamma \sum_{u \in S} V^{\pi_n}(u) P(u|s, \pi(s)).$$

   (b) Update the policy by setting:

   $$\pi_{n+1}(s) = \arg\max_{a \in A} \{E[\ R(s, a))\ ]\ + \gamma \sum_{u \in S} V^{\pi}(u) P(u|s, a))\}.$$

   Since, it is possible to have several actions that maximize expected return, we adjusted our code to select the first appearance.

3. Once $\pi_{n+1} = \pi_n$, by definition of convergence, V^{π_n} satisfies Bellman's equation, which means V^{π_n} is the optimal value function and we have achieved an optimal policy π^*.

2.7 Q-learning Intuition

Q-learning is considered one of the most important breakthroughs in reinforcement learning [6]. It can be used to find an optimal policy for the finite MDP and it does not need to know the transition probabilities and reward structure. It only requires the set of possible states and actions, and a way to simulate data.

Here we define the notion of action-value function for policy π. We denote it as $Q^{\pi}(s,a)$, which is equal to expected discounted reward if we start at state s, take the action a, and follow policy π:

$$Q^{\pi}(s,a) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) | s_0 = s, a_0 = a\ ] \tag{2.4}$$

Since we would like to quantify the value of each combination of state and action, a_0 is not necessary equal to $\pi(s_0)$. The algorithm starts by arbitrary initializing $Q_0(s,a)$ for all $s \in S, a \in A$ and choosing the initial state. Then at the nth iteration, being in a state s, the controller selects an action a and observes the new state u and reward r associated with this action. Q is updated by the following formula:

$$Q_{n+1}(s,a) \leftarrow Q_n(s,a) + \alpha(r + \gamma \max_{b \in A} Q_n(u,b) - Q_n(s,a)) \tag{2.5}$$

where $\alpha \in [0,1]$ is **the learning rate** that controls the weighting of new and old information. If $\alpha = 0$ the controller learns nothing. If $\alpha = 1$, the controller considers only the new information. Generally, α can be a function of the iteration number and this property is required for convergence. However, for simple examples it can be considered as a constant.

If the next state is a terminal state (the one from which the process cannot continue), the process starts from initial state with updated Q value.

2.8 Example: Gridworld

2.8.1 Problem Setup

Consider a simple 3 by 4 grid. An agent starts in a specified state (indicated as "START" in Figure 2.1) and moves in the grid till it reaches one of the two terminal states. One of them provides a positive reward of 1 and is considered the goal state. The second one gives a negative reward of 1 (indicated as "+1" and "-1", respectively,

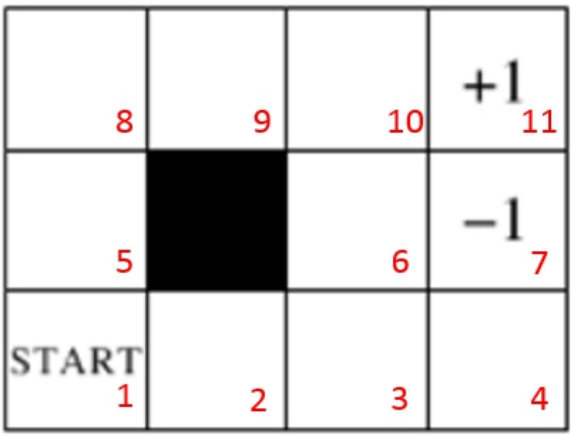

Figure 2.1: 3x4 Gridworld.

in Figure 2.1) and is considered the penalty state. The agent can make one step at a time and is able to go up, down, left, or right. Every move is associated with a movement cost of negative 0.04. The agent moves reliably only 80% of the time. This means that if the agent chooses the desired action (up, down, left, right), 80% of the time he will move to the desired state, but there is a 10% chance to move clockwise of the desired action and 10% to move counter-clockwise of the desired action. Also, if the agent moves to the wall he will bounce back to the state where he was before.

The goal of the agent is to find the optimal policy to reach the goal state. By optimal policy, we mean to find a function that associates every state with an action that maximizes expected discounted total reward.

An online tutorial suggests that the optimal policy is the one in Figure 2.2 [7]. However, the discount factor was not stated, so it is difficult to verify. In the following section we will discuss how the discount factor changes the optimal policy.

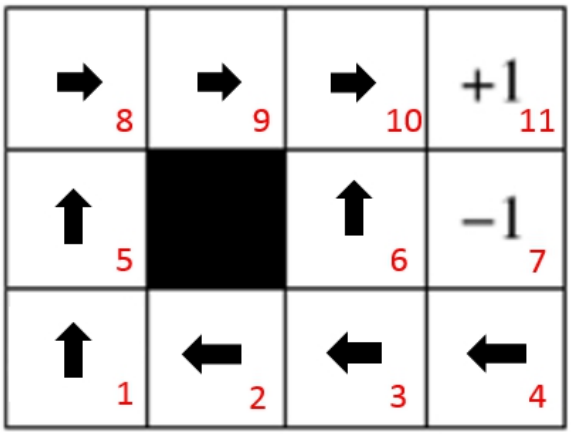

Figure 2.2: Gridworld: Best policy (for unknown γ).

2.8.2 Policy Iteration

Using our own choice of $\gamma = 0.9$, we achieved an optimal policy which is similar to Figure 2.2. Also, we can notice that the results are the same every time we run the simulation, which is one of the good things about policy iteration. An R script is provided in the **Appendix B**.

Changing the discount rate, γ, can reveal some interesting insights. Let us have a look at optimal policies for two discount rates; $\gamma = 0.99$, representing a more patient agent and, $\gamma = 0.9$, representing a less patient agent who seeks more immediate reward. We can notice (Figure 2.3) that the impatient agent ($\gamma = 0.9$) prefers to go

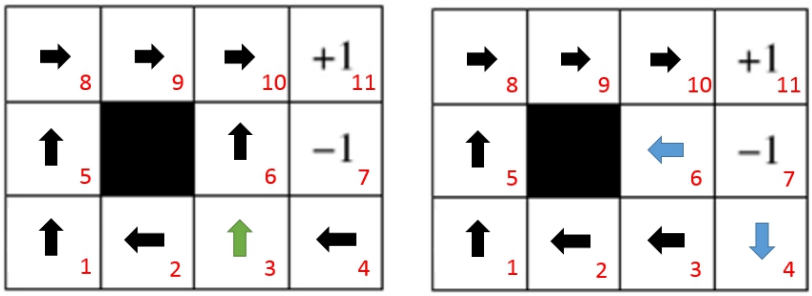

Figure 2.3: Optimal policies for discount rates 0.9 and 0.99 respectively.

up in state 3 even though there is a chance to fall in the pit. On the other hand, the patient agent prefers to go down in state 4 to avoid even a low chance (10%) to fall in the pit. Also in state 6, he prefers to go left with the same reason of avoiding the pit.

2.8.3 Q-learning

To successfully apply q-learning to grid world we need either historical data or a simulation to generate the data. We used simulated data using transition probability matrix and reward structure. An R script can be found in the **Appendix B**.

By running q-learning code several times, we can notice that each time we receive a different answer. This is due to the fact of randomness of the algorithm. Also, we can notice that the states that are close to the terminal state change rarer which is reasonable since in these states the agent is more informed about the future.

Based on the randomness feature of the q-learning result, a logical question arises. How can we choose an optimal policy if every time the optimal policy changes? A possible way to approach this questions is to simulate q-learning, let say 1000 times and for each state determine the most often strategy. This topic is left for future research.

CHAPTER 3

FDS MODEL SETUP

3.1 Data Preparation/Cleaning

It is well known that one of the most important steps in data analysis is to make sure that data values are suitable for analysis. Correctness depends on the variables we are measuring. For example, if we deal with GENDER we would expect to have only two possible values, or if we think about WEIGHT, we should make sure that we are using the same units of measurements in every observation and across the data sets.

I will briefly go through the main problems that we encountered with our data and how we approached them.

3.1.1 Open-ended Questions

The data collected during previous surveys included a lot of open-ended questions. The biggest problem with open-ended questions is that people may spell things incorrectly. Even questions where the respondent was asked to write a city name had spelling mistakes. If we think about names of universities, which sometimes have three to four words, the room for error becomes even greater. Sometimes, if there is a pattern, we can clean the data using programming tools. But sometimes we should manually clean the data which can take a massive amount of time.

Below is the list of main issues with which we dealt while cleaning the data from open-ended questions.

- **Upper/lower case.** For example, the data values "Statesboro" and "statesboro" are considered to be different. In order to fix this, we changed every value to the same standard with the first upper case and the rest lower case. There are several R packages that can easily do it.

- **Spelling Mistakes.** By default, the data is case sensitive. For example, if we have "Stateboro" and "Statesboro" with missing "s" in the first case. We solved this problem using a dataset of all cities in the US and an R package (function *adist* {*utils*}) that calculates approximate string distance between character vectors. This distance is a generalised Levenshtein distance that estimates the minimal weighted number of insertions, substitutions, or deletions needed to transform one string to another. For example, the distance between "poker" and "stoker" is 2. So, using this distance we constructed a matrix of distances between our data set and the data set of all cities in the US. Then we found a minimum for each row, which is simply the closest distance from the value in our data set to a value from the dataset of all cities in the US. By knowing the closest value, we simply can replace it with a value of our data set. To avoid replacing wrong names we should set up a condition that found minimum should be less than some specified distance. In the FDS data, we used a distance of 2.

  Using this approach we replaced the most similar cases. After that we performed a quick manual check to make sure that we did not miss any important cases.

To avoid this cleaning in the future, we added auto-complete and drop-down features to these questions for next year surveys. By **auto-complete**, we mean that, when respondent of the survey starts typing the answer to the question, the engine behind the question suggests the most similar answers in a list, so the respondent can choose an answer from the proposed ones which come from a stored dataset with correct values. This type of question is good when we have a lot of possible choices (a big data set to choose from).

Sometimes, when there are not a lot of possible values (for example, the US states names), the **drop-down** is better.

3.1.2 Numerical Values

In our survey, we have some questions that asked to provide a numerical answer. One of them is related to salary estimation. This question was open-ended in the previous survey. Below is a list of issues we encounter cleaning this kind of data.

- **Character Values.** The nature of open-ended question makes it possible for responded to include character values along with the number, for example, "10000/". Since there is "/", we can not use this value for any numerical analysis. So the first thing in cleaning numerical data was to eliminate character values and make sure that everything is stored as a numerical value. We used regular expression to search character values and eliminate them.

  To avoid this kind of mistake, we include a condition in the future survey, that only numbers can be typed.

- **Outliers.** To provide more or less robust salary estimation, it is very important to do an outliers analysis. For example, one of the most common things was to put "1" as a salary, which obviously is not appropriate.

  For a lower bound of salary, we considered the lowest reasonable salary that a college graduate can have as a full-time employee. Everything below of that needs to be eliminated. (As a lowest value we picked $ 9600).

  For the upper bound, it is more difficult because some graduates might have high salaries because they have worked in the field before school, and a combination of work experience and degree made it possible for them to get high salary. Regardless, a salary like "1000000" is doubtful. One way to check whether this response is valid is to look at other responses of this individual. Very often those who give a suspicious answer to one question, will also give suspicious

answers to other questions as well. Having several suspicious answers gives us more confidence that this observation should be deleted.

For the future surveys, we included upper bound for possible salaries based on expert opinion from university officials.

Sometimes we can not eliminate suspected outliers because we are not sure whether it is a true salary or not. Leaving these values in the data set can influence our estimation of the average. For this purpose, we have decided to report median rather than mean salary as an estimation of central tendency.

3.2 FDS Model Construction

Below we provide the model that describes the setup of the First Destination Survey (FDS).

Action space:

Based on the Career Service practice, we consider three possible actions: **phone call, email,** and **give up.** Since we constructed a model that can be applied not only by Career Services but also by other institutions (with slight changes in the code), possible actions can be different.

State space:

The state of a possible respondent is described by the following attributes:

- **Cluster:** the FDS data was partitioned by responce rate using a regression tree (regression tree package in R). We came up with 2 clusters based on the response rate. Based on previous data we estimate the response rates of each cluster.

  Parameters in the R code:

  - Number of clusters: $numcl = 2$.

  - Initial response rates: $prob = 0.6$ and 0.7 for cluster 1 and cluster 2 respectively.

- **Contact attempts history:** we have two parameters that identify the number of contact attempts. m is the number of email attempts and n is the number of phone attempts. We require that $m, n \geq 0$, and also $m + n \leq 3$, meaning that we have at most three attempts to contact a potential respondent. For example $(m, n) = (2, 1)$ means that we sent 2 emails and called one time.

  Parameter in the R code:

  - Maximum number of attempts before give up: $numat = 3$.

- **Status:** every possible respondent is associated with a status that can take one of three values:

  - *active* - the person has not responded yet and $m + n < 3$,
  - *responded* - the person responded, so contact attempts have ceased due to received response,
  - *give up* - the person did not respond, so contact attempts have ceased due to giving up or ran out of contact attempts.

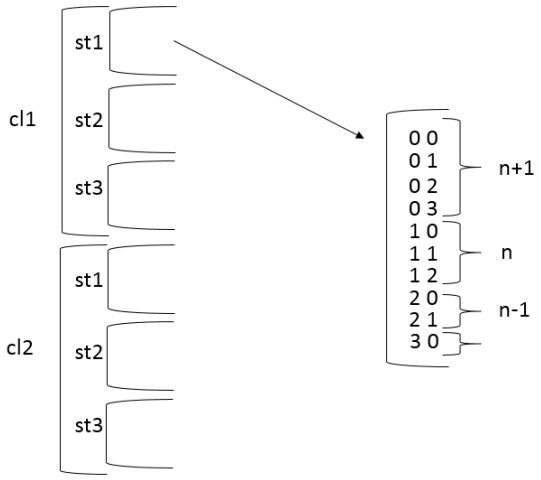

Figure 3.1: Number of states. FDS model.

Figure 3.1 shows the intuition of calculation of number of states. So the general formula for the number of states is:

$$\#ofclusters \cdot \#ofstatuses \cdot \frac{(\#ofactions + 1) \cdot (\#ofactions + 2)}{2} \tag{3.1}$$

Thus the number of states is $2 \cdot 3 \cdot \frac{(3+1) \cdot (3+2)}{2} = 60$. Based on the above construction, we introduced a function that returns a state number based on the following parameters: the cluster value, the status, the number of previous email and phone attempts. The code of this function can be found in the **Appendix A** (the name of the function is "index").

Transition probabilities: In our case, transition probabilities are probabilities to contact a person using either phone or email based the history of contact attempts and cluster. Ideally, the data will give us an estimation of these probabilities, but since we had limited amount of data, we used existing literature to come with these estimations. Based on the available literature about the amount of contacting attempts needed to reach a person, Figure 3.2 [9] shows that about half of the people

are contacted after the first phone call with a decreasing pattern afterwards. There are two main factors that influence how many calls are required to contact a person.

1. Calls during the weekend evenings are on average more efficient than other times.

2. Different populations have different accessibility likelihoods. Thus it is useful to cluster people based on response rate.

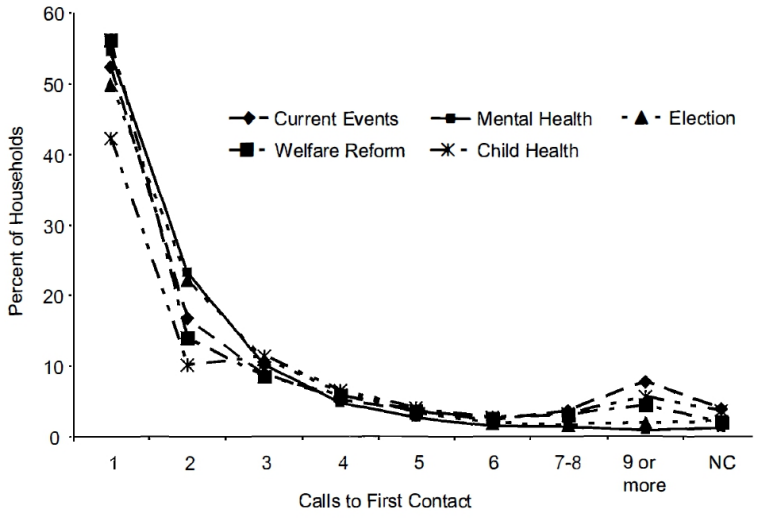

Figure 3.2: Percentage of eligible sample households by calls to first contact for five surveys [9].

The formula below captures the general behavior in Figure 3.2, which calculates the probability of response after action applied.

$$P(m,n) = \begin{cases} \alpha_e^m \alpha_p^n p_i & \text{email;} \\ \alpha_e^m \alpha_p^n p_i + (1 - \alpha_e^m \alpha_p^n p_i)\gamma & \text{phone.} \end{cases} \tag{3.2}$$

Where:

m, n - the number of previous email, phone attempts,

p_i - historical response rate of the cluster ($0 \leq p_i \leq 1, i = 1, 2$),

α_e - measures the "power" of reaching a person via email ($0 < \alpha_e < 1$),

α_p - measures the "power" of reaching a person via phone ($0 < \alpha_p < 1$),

intuitively, α_e and α_p represent how much information we gain from an unsuccessful attempt as to whether the person will ever be contacted.

γ - is chosen to give the function the approximate shape (Figure 3.2), as the one suggested in the book [9].

The R code for this function can be found in the **Appendix A** (The name of the function is "probresp").

Reward structure: Every time we try to contact a person, we incur a cost depending on the method of contact. Also, when we succeed we receive a positive value. To incorporate this, we came up with the function that calculates expected reward based on the set of parameters.

$$r = -\sigma_j + P(m, n) * \phi_i \tag{3.3}$$

where:

r - reward associated with successfully contacting the person,

σ_j - cost associated with email($j = 1$), phone($j = 2$),

ϕ_i - value associated with contacting a person from cluster i (i=1,2).

The cost of one call is estimated based on hourly rate of graduate students (approximately \$9), who actually make the calls and average time of the call (which is approximately 10 min.). Thus $\sigma_2 = 1.5$. The cost of one email is hard to calculate, because it involves the work of several departments. Since it is logical to assume that the cost of one call is higher then a cost of one email, a reasonable estimate of the cost of email is $\sigma_1 = 0.55$.

Since it is important to receive responses from different clusters, we will value a response from a cluster with lower response rate higher then the one from a cluster

with higher response rate. The exact values are chosen arbitrarily for the sake of the model. The following are the values that we used in our model $(7, 3)$.

The code for this function can be found in the **Appendix A** (The name of the function is "reward").

3.3 Results

3.3.1 Optimal Contacting Strategies

To come with the optimal contact strategy we used policy iteration with discount factor equals to one (finite horizon case).

Table 3.1 contains a short summary of the parameters that we used in our model to advise the optimal strategy for Career Services.

Name of a parameter	Description	Value
numcl	number of clusters	2
numstat	number of statuses (active, responded, give up)	3
numat	maximum number of attempts	3
prob (p_i)	initial clusters' response rate (cl1, cl2)	(0.6, 0.7)
gamma (γ)	approximation factor	0.6
alpha_e (α_e)	the reaching "power" of email	0.5
alpha_p (α_p)	the reaching "power" of phone	0.9
cl_value (ϕ_i)	the value of cluster (cl1, cl2)	(7, 3)
cost (σ_j)	the cost (email, phone)	(0.55, 1.5)

Table 3.1: Summary of parameters used in the FDS model.

The final policy for both clusters indicating the best-contact strategy in every feasible state is provided in Table 3.2. For example, the first row of the table says that, given the model setup, the best way to start contacting a person from a cluster one or two is to send an email. At the same time, if we look at the second row, which says that if we have not reached the person yet (status is active) but tried to call once before, the best way to contact is to call again if the person is from cluster one,

or to send an email if the person is from cluster two. One of the reasons that we have different contact strategies for each cluster is because we set up different cluster values. In our case, cluster one has the value of 7 while cluster two has 3, as a result in some cases it is more beneficial to try to call a person from cluster one even though it is more expensive.

History of attempts (email, phone)	Cluster 1	Cluster 2
(0, 0)	email	email
(0, 1)	phone	email
(0, 2)	phone	email
(0, 3)	give up	give up
(1, 0)	phone	phone
(1, 1)	phone	phone
(1, 2)	give up	give up
(2, 0)	phone	phone
(2, 1)	give up	give up
(3, 0)	give up	give up

Table 3.2: Optimal contacting strategies for both clusters. Case with cluster values (7,3).

Based on the Table 3.2, the recommendation to a Career Services team might be to use the following contacting strategies. For the cluster one, first of all send an email, if the person has not responded then try to call him/her, if the person still has not responded, call him/her again. Notice, that the contacting strategy is the same for cluster 2. But if we take a look at all feasible states, we can notice that in some

of the states we have different actions depending on the cluster. For example, for the cluster two, the first recommendation is to send an email, it means that we will never transition to a state (0,1), where the recommendation is to email again, which different than in cluster one.

While estimating some of the initial parameters, we used an expert opinion or a data-driven approximation that may be inaccurate. For example, based on the response rates from the clusters, we claimed that, since we received less responses from cluster one and it is important to have enough data from each cluster we gave more value to the cluster with less response rate. However, it is hard to tell what exact number we should assign with each cluster, whether it should be 2, 3 or 4 we are not sure. Let's see what happens with the policy if the clusters' values are (7,2) instead of (7,3).

Table 3.3 shows that by slightly changing the value for cluster two from 3 to 2, it changed the optimal policy in the following way. The first attempt is email which is the same as before, however, the second attempt becomes email, which is different from the previous case, which was phone. The interesting this is that, after the second attempt which is email, there is no third attempt which means that the possible value of contacting a person is less then cost of sending email.

The determination of how to best assign value to each cluster is beyond the scope of this paper. We leave value assignment problem to future research.

History of attempts (email, phone)	Cluster 1	Cluster 2
(0, 0)	email	email
(0, 1)	phone	email
(0, 2)	phone	email
(0, 3)	give up	give up
(1, 0)	phone	email
(1, 1)	phone	email
(1, 2)	give up	give up
(2, 0)	phone	give up
(2, 1)	give up	give up
(3, 0)	give up	give up

Table 3.3: Optimal contacting strategies for both clusters. Case with cluster values (7,2).

3.3.2 Sensitivity Analysis

One of the assumptions behind the model is the estimation of so-called "power" of the contact strategy ($0 \leq \alpha_e, \alpha_p \leq 1$). For example, in the FDS model, we assumed that α_e is 0.5, while α_p is 0.9. The logic behind this is simply that the chance to contact a person via phone is higher than via email which is supported by existing literature of survey methodology [9]. However, it is informative to test a model for different combinations of (α_e, α_p). For this purpose, we created a metric that for every combination of (α_e, α_p) counts how many times the policy suggests to contact via email or phone.

In Figure 3.3, the upper graph represents the metric for phone, while the lower graph represents the metric for email. Each dot has two attributes, colour and size, both of which represent the value of the metric. For example, the green dot, which is medium sized, in the upper graph means that the policy for the specific combination of (α_e, α_p) has two calls in it. The same logic applies to second graph, but with regard to the amount of emails.

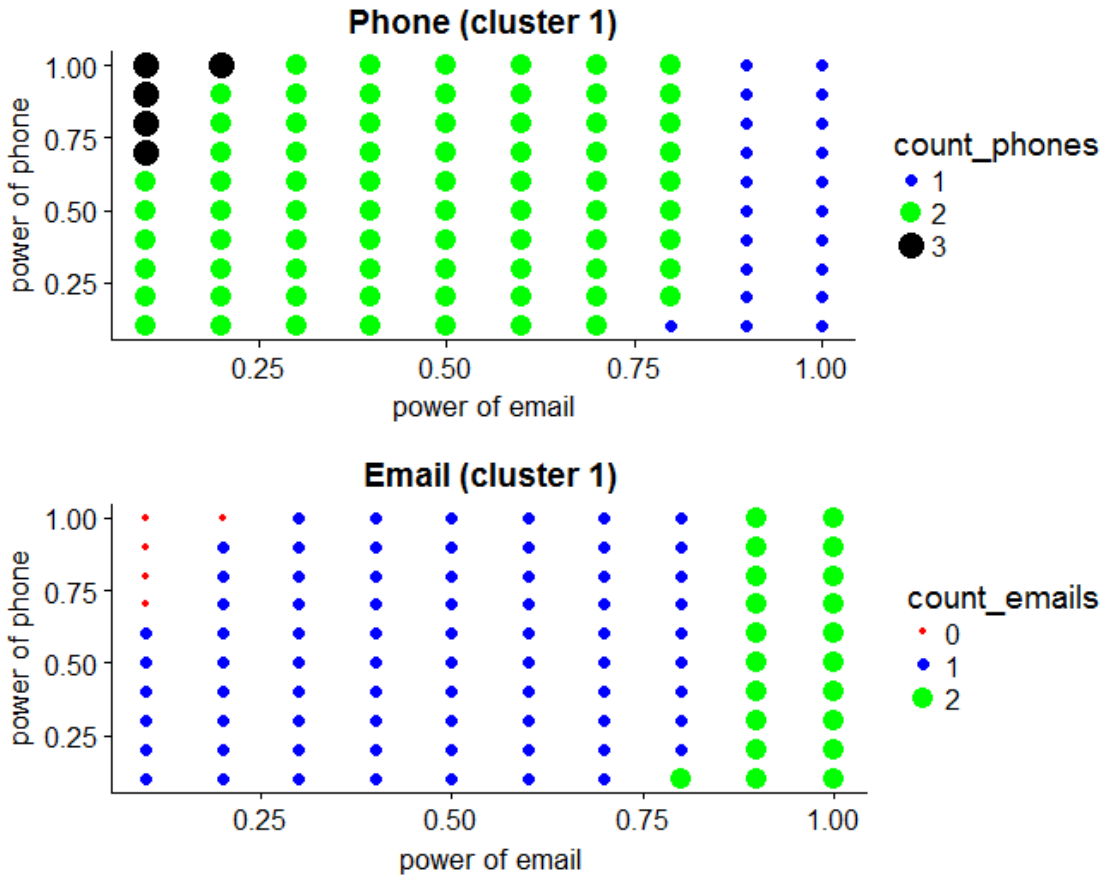

Figure 3.3: Sensitivity analysis for cluster one based on different values of α_e, α_p.

It is also worth to notice that for the cluster one the phone call is more common then sending email. Only when the power of email is more than 0.8 we use more emails in the policy then phone calls. However, by looking at the cluster two (Figure

3.4), we can notice that sending email is more common then phone calls. Only when the power of email is less than 0.5 we have more phone calls then emails.

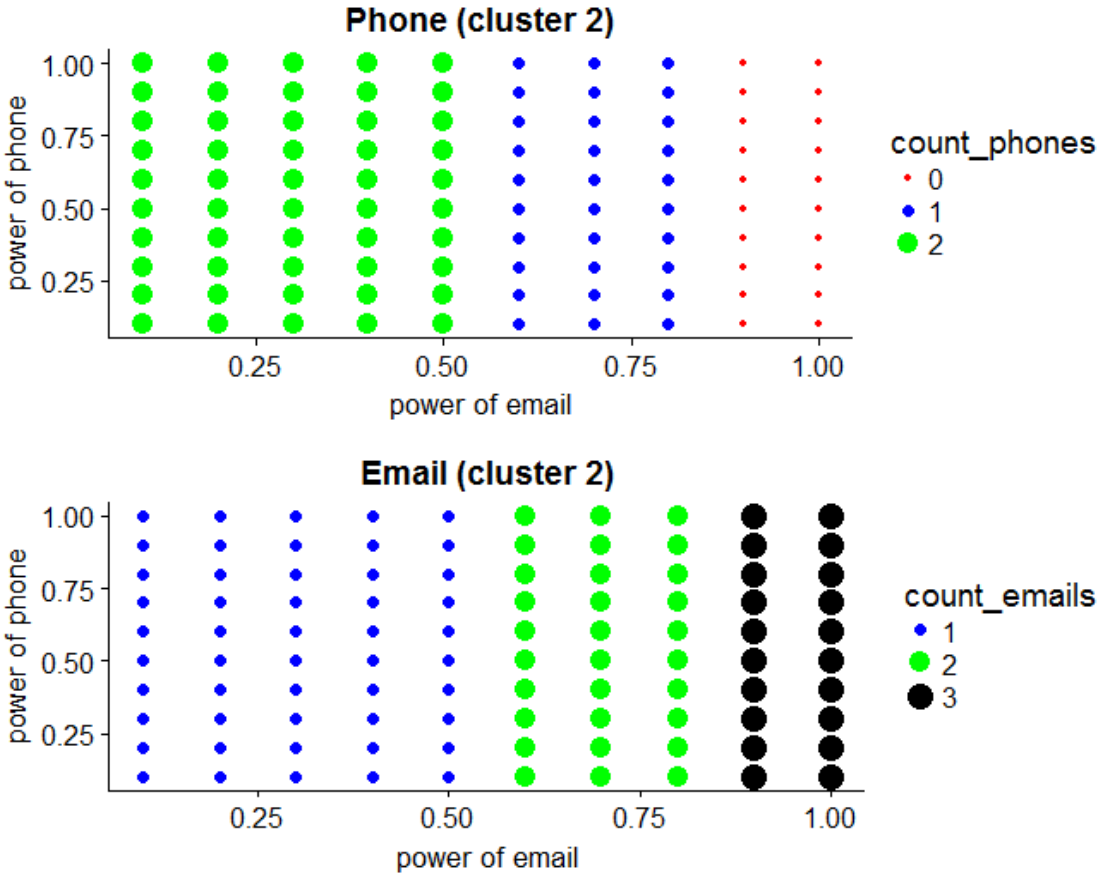

Figure 3.4: Sensitivity analysis for cluster two based on different values of α_e, α_p.

CHAPTER 4

CONCLUSION AND FURTHER RESEARCH

In this thesis we constructed a model that can be of use for the Career Services office at Georgia Southern University. Moreover, our model can be a starting point for other organizations that want to optimize their contact strategies. Our theoretical model, as well as the R script provided in the Appendix A, can be adjusted to take into account specific needs of particular organizations.

By applying our model to the First Destination Survey, we came up with the same strategy to contact respondents from both clusters. Our strategy is (email, call, call), which means email first, if the person have not responded then call, if the person have not responded call him again. This strategy seems to be logical, taking into account the cost and reaching power of email and call. But since, we have created a theoretical framework, our model can be applied to more complex and not obvious scenarios.

Future research. It is worth to note that the concept of MDP have not been applied in optimizing contact strategies before and this topic is quite open. One logical continuation of this paper is adjusting and applying our model to more general and complex cases, such as ones that have more contact methods and a more diverse target population. Also it would be useful to take into account a budget allocated to a survey, this might be applied via including it as an additional parameter categorizing the state space.

REFERENCES

[1] S. K. Thompson, *Adaptive Cluster Sampling*, Journal of the American Statistical Association Vol. 85, No. 412 (Dec., 1990), pp.1050-1059.

[2] D. Basu, *Role of the Sufficiency and Likelihood Principles in Sample Survey Theory,* Sankhya (1969), Ser. A, 31, 441-454.

[3] D. Kohler, *Optimal strategies for the game of darts.* J. Opl Res. Soc. (1982) 33, 871-884.

[4] D. J. White, *Dynamic programming and systems of uncertain duration.* Mgtnt Sci. (1965) 19, 37-67.

[5] R. Howard, *Dynamic Programming and Markov Processes,* MIT Press(1960), Cambridge, MA.

[6] Watkins, C.J.C.H. *Learning from Delayed Rewards.* Ph.D. thesis, Cambridge University. (1989).

[7] J. Schrum, *Reinforcement Learning 2 - Grid World,* Youtube (2015) (https://www.youtube.com/watch?v=bHeeaXgqVig).

[8] R. Sutton, A. Barto, *Reinforcement Learning. An Introduction.* MIT Press/A Bradford Book, Cambridge (2012), USA.

[9] Robert M. Groves, *Survey methodology.* Hoboken, NJ: J. Wiley (2004).

Appendix A

FIRST APPENDIX

A.1 R Scripts. Model For the First Destination Survey

The model consists of three scripts:

1. **main_v5.R** - consists of main functions that create transitions probability matrix as well as reward matrix.

2. **policy_iteration_v2.R** - calculating the optimal policy using policy iteration.

3. **plot_v2.R** - creates sensitivity plot.

It is recommended to run **plot_v2.R**, and check the policy in the matrix "mat".

```
# initializing transition probability and reward matrices to populate them further
int = function(piconfig) {
  # calculating the number of states
  numstates = piconfig$numcl*((piconfig$numat+1)*(piconfig$numat+2)/2)*piconfig$
      numstat

  # initializing zero matrices for email, phone, give up accordingly
  p_m = matrix(0, nrow = numstates, ncol = numstates )
  p_p = matrix(0, nrow = numstates, ncol = numstates )
  p_g = matrix(0, nrow = numstates, ncol = numstates )

  # initializing reward vectors
  r_m = matrix(0, nrow = numstates, ncol = 1)
  r_p = matrix(0, nrow = numstates, ncol = 1)
  r_g = matrix(0, nrow = numstates, ncol = 1)
  list = list("numstates"=numstates,"p_m"=p_m,"p_p"=p_p,"p_g"=p_g,"r_m"=r_m,"r_p"=r_p
      ,"r_g"=r_g)
  return(list)
}

# index function calculates indexes for the matrices
```

```
index <- function(c_m,c_p,cl,status,numstates) {
  if ((piconfig$numat-c_m) <= (piconfig$numat-1)) {
    ind = (cl-1)*(numstates/piconfig$numcl) +
          (status-1)*(numstates/(piconfig$numcl*piconfig$numstat)) +
           sum((piconfig$numat+2-c_m):(piconfig$numat+2-1)) +
          (c_p+1)
  } else {
    ind = (cl-1)*(numstates/piconfig$numcl) +
          (status-1)*(numstates/(piconfig$numcl*piconfig$numstat)) +
          (c_p+1)
  }
  return(ind)
}

# calculates prob of status happening
probresp <- function(a, c_m, c_p, cl) {
  if (a == 1) {
    resp = piconfig$alpha_e^(c_m)*piconfig$alpha_p^(c_p)*piconfig$prob[cl]
  } else {
    resp = piconfig$alpha_e^(c_m)*piconfig$alpha_p^(c_p)*piconfig$prob[cl] +
          (1-piconfig$alpha_e^(c_m)*piconfig$alpha_p^(c_p)*piconfig$prob[cl])*
              piconfig$gamma
  }
  return(resp)
}

# calculates rewards
reward <- function(cl_value, respob, cost){
  exp_value = -cost + respob*cl_value
  return(exp_value)
}

# populates the matrices from the int function
building_sys_dynamics <- function(piconfig) {

# Actions coding
  # (a == 1) Sending email
  # (a == 2) Sending phone
  # (a == 3) Give up
# Statuses coding
```

```
59  # (s == 1) Active status
60  # (s == 2) Responded status
61  # (s == 3) Give up status
62
63  # looping through clusters
64  for(cl in 1:piconfig$numcl) {
65    # looping through statuses
66    for(s in 1:piconfig$numstat) {
67      # looping through history of email attempts
68      for(c_m in 0:piconfig$numat) {
69        # looping through history of phone attempts
70        for(c_p in 0:(piconfig$numat-c_m)) {
71          # if person is active (s==1) and the number
72          # of previous attempts is less that maximum of possible
73          if ((c_m + c_p < piconfig$numat) & (s == 1)) {
74            # looping through possible actions (1 - email, 2 - phone
75            # 3 - give up)
76            for (a in 1:piconfig$numac) {
77              # if email
78              if (a == 1) {
79                status = s
80                c_m_temp = c_m
81                c_p_temp = c_p
82                row_index = index(c_m,c_p,cl,status,numstates)
83                c_m_temp = c_m_temp + 1
84                # if person responds
85                status = 2
86                column_index = index(c_m_temp,c_p_temp,cl,status,numstates)
87                p_m[row_index, column_index] = probresp(a, c_m, c_p, cl)
88                r_m[row_index, 1] = reward(piconfig$cl_value[cl],
89                                           p_m[row_index, column_index], piconfig$
                                               cost[a])
90                if (c_m + c_p < piconfig$numat - 1) {
91                  # when person didn't responded
92                  status = 1
93                  column_index = index(c_m_temp,c_p_temp,cl,status,numstates)
94                  p_m[row_index, column_index] = 1 - probresp(a, c_m, c_p, cl)
95                } else if (c_m + c_p == piconfig$numat-1) {
96                  status = 3
97                  column_index = index(c_m_temp,c_p_temp,cl,status,numstates)
```

```
98                p_m[row_index, column_index] = 1 - probresp(a, c_m, c_p, cl)
99              }
100           } else if (a == 2) {
101             status = s
102             c_m_temp = c_m
103             c_p_temp = c_p
104             row_index = index(c_m,c_p,cl,status,numstates)
105             c_p_temp = c_p_temp+1
106             status = 2
107             column_index = index(c_m_temp,c_p_temp,cl,status,numstates)
108             p_p[row_index, column_index] = probresp(a, c_m, c_p, cl)
109             r_p[row_index, 1] = reward(piconfig$cl_value[cl],
110                                        p_p[row_index, column_index], piconfig$
                                               cost[a])
111             if (c_m + c_p < piconfig$numat-1) {
112               status = 1
113               column_index = index(c_m_temp,c_p_temp,cl,status,numstates)
114               p_p[row_index, column_index] = 1 - probresp(a, c_m, c_p, cl)
115             } else if (c_m + c_p == piconfig$numat-1) {
116               status = 3
117               column_index = index(c_m_temp,c_p_temp,cl,status,numstates)
118               p_p[row_index, column_index] = 1 - probresp(a, c_m, c_p, cl)
119             }
120           }
121         }
122       } else if (s == 2 ) {
123         # The person is already responded
124         row_index = index(c_m,c_p,cl,s,numstates)
125         # do nothing
126         p_m[row_index,row_index] = 1
127         p_p[row_index,row_index] = 1
128         p_g[row_index,row_index] = 1
129       } else if (s==3) {
130         # The person is unreachable
131         row_index = index(c_m,c_p,cl,s,numstates)
132         # do nothing
133         p_m[row_index,row_index] = 1
134         p_p[row_index,row_index] = 1
135         p_g[row_index,row_index] = 1
136       } else if ((c_m + c_p == piconfig$numat) & (s==1)) {
```

```
            row_index = index(c_m,c_p,cl,s,numstates)
            status = 3
            column_index = index(c_m,c_p,cl,status,numstates)
            p_m[row_index, column_index] = 1
            p_p[row_index, column_index] = 1
          }
        }
      }
    }
  }
  s = list("p_m"=p_m,"p_p"=p_p, "p_g" = p_g, "r_m"=r_m, "r_p"=r_p, "r_g" = r_g)
  return(s)
}
```

main_v5.R

```
policy = function (gam = 0.9) {
  n_s = 60

  #dummy policy
  pi <- rep(3,n_s)
  # 1 - Mail
  # 2 - Phone
  # 3 - Give up

  #Initial policy's value
  v =  solve((diag(n_s) - gam*p_g)) %*% r_g
  diag(n_s)

  #Storing policy and its values for comparison later
  pi_old <- pi
  v_old <- v

  #Build a new, better policy
  for (i in 1:n_s) {
    best_action_value = v[i]
    c_m = r_m[i] + gam*p_m[i,] %*% v
    c_p = r_p[i] + gam*p_p[i,] %*% v
    c_g = r_g[i] + gam*p_g[i,] %*% v
    if (c_m > best_action_value) {
```

```
25        pi[i] = 1
26        best_action_value = c_m
27      }
28      if (c_p > best_action_value) {
29        pi[i] = 2
30        best_action_value = c_p
31      }
32      if (c_g > best_action_value) {
33        pi[i] = 3
34        best_action_value = c_g
35      }
36    }
37
38    difference = sum ((pi - pi_old)^2)
39    pi_old = pi
40
41    while (difference !=0) {
42      r <- matrix(0, nrow = n_s, ncol = 1)
43      p <- matrix(0, nrow = n_s, ncol = n_s)
44      for (i in 1:n_s) {
45        if (pi[i] == 1) {
46          r[i] = r_m[i]
47          p[i, ] = p_m[i, ]
48        }
49        else if (pi[i] == 2) {
50          r[i] = r_p[i]
51          p[i, ] = p_p[i, ]
52        }
53        else if (pi[i] == 3) {
54          r[i] = r_g[i]
55          p[i, ] = p_g[i, ]
56        }
57      }
58      # Find the value of a new policy
59      v = solve( (diag(n_s) - gam*p) ) %*% r
60      for (i in 1:n_s) {
61        best_action_value = v[i]
62        c_m = r_m[i] + gam*p_m[i,] %*% v
63        c_p = r_p[i] + gam*p_p[i,] %*% v
64        c_g = r_g[i] + gam*p_g[i,] %*% v
```

```
      if (c_m > best_action_value) {
        pi[i] = 1
        best_action_value = c_m
      }
      if (c_p > best_action_value) {
        pi[i] = 2
        best_action_value = c_p
      }
      if (c_g > best_action_value) {
        pi[i] = 3
        best_action_value = c_g
      }
    }
    # to see if new policy is different
    difference = sum ((pi - pi_old)^2)
    pi_old = pi
  }
  return(list("policy" = pi,"value"=sum(v)))
}
```

policy_iteration_v2.R

```
# setting up initial parameters
piconfig = list(numcl=2, numstat=3, numat=3, numac=3, prob = c(0.6, 0.7),
                gamma = .6, alpha_e = .5, alpha_p = .9, cl_value = c(7,3), cost = c
                    (.55,1.5) )

# numcl - number of clusters
# numstat - number of states
# numat - number of possible attempts
# prob - response rate from clusters (based on the previous data)
# gamma - the coefficient in the probresp function that helps to approximate it to
    the graph
# alpha_e - the reaching "power" of email
# alpha_p - the reaching "power" of phone
# cl_value - the value of reaching to the respondent from a particular cluster
# cost - the cost of contacting by email, phone

o = 1  # maximum value for each alpha
h = 10 # amount of increments
```

```
17
18 mat = matrix(0, nrow=h^2, ncol = 66) # has columns for alpha1, alpha2, the entire
       policy,
19 # and summaries of actions for the policy for each cluster
20
21 # loops over alpha_e
22 for (l in 1:h) {
23   piconfig$alpha_e = l*(o/h)
24   # loops over alpha_p
25   for (j in 1:h) {
26     piconfig$alpha_p = j*(o/h)
27
28     # these lines store alpha1 and alpha2 in the matrix
29     mat[(l-1)*h + j,1] = piconfig$alpha_e
30     mat[(l-1)*h + j,2] = piconfig$alpha_p
31
32     # running the main_v5 script that creates functions that we
33     # will use further
34     source("main_v5.R")
35
36     # setup the empty vectors/matrices/etc for use by "building_sys_dynamics"
37     numstates = int(piconfig)$numstates
38     p_m = int(piconfig)$p_m
39     p_p = int(piconfig)$p_p
40     p_g = int(piconfig)$p_g
41     r_m = int(piconfig)$r_m
42     r_p = int(piconfig)$r_p
43     r_g = int(piconfig)$r_g
44
45     # get the "real" dynamics from the function.  Populates the vectors/matrices
46     s = building_sys_dynamics(piconfig)
47     p_m = s$p_m
48     p_p = s$p_p
49     p_g = s$p_g
50     r_m = s$r_m
51     r_p = s$r_p
52     r_g = s$r_g
53
54     # running the policy_iteration script
55     source("policy_iteration_v2.R")
```

```
56
57    # stores the policy in the matrix
58    mat[(l-1)*h + j,3:62]= policy()$policy
59
60    # candidate location for changing policy
61    quitflag = FALSE
62    stateindex_cl1 = 3
63    stateindex_cl2 = 33
64    actionsum = c(0,0,0,0)
65    # for cluster one
66    while (!quitflag) {
67      if (mat[(l-1)*h + j,stateindex_cl1] == 1) {
68        actionsum[1] = actionsum[1] + 1
69        if (stateindex_cl1 %in% c(3,4,5)) {
70          stateindex_cl1 = stateindex_cl1 + 4
71        }
72        else if (stateindex_cl1 %in% c(7,8)) {
73          stateindex_cl1 = stateindex_cl1 + 3
74        }
75        else if (stateindex_cl1 %in% c(10)) {
76          stateindex_cl1 = stateindex_cl1 + 2
77        }
78        else if (stateindex_cl1 %in% c(6,9,11,12)) {
79          quitflag = TRUE
80        }
81      }
82      if (mat[(l-1)*h + j,stateindex_cl1] == 2) {
83        actionsum[2] = actionsum[2] + 1
84        if (stateindex_cl1 %in% c(3,4,5,7,8,10)) {
85          stateindex_cl1 = stateindex_cl1 + 1
86        }
87        else if (stateindex_cl1 %in% c(6,9,11,12)) {
88          quitflag = TRUE
89        }
90      }
91      if (mat[(l-1)*h + j,stateindex_cl1] == 3) {
92        quitflag = TRUE
93      }
94    }
95
```

```
    quitflag = FALSE
    # for cluster two
    while (!quitflag) {
      if (mat[(l-1)*h + j,stateindex_cl2] == 1) {
        actionsum[3] = actionsum[3] + 1
        if (stateindex_cl2 %in% c(33,34,35)) {
          stateindex_cl2 = stateindex_cl2 + 4
        }
        else if (stateindex_cl2 %in% c(37,38)) {
          stateindex_cl2 = stateindex_cl2 + 3
        }
        else if (stateindex_cl2 %in% c(40)) {
          stateindex_cl2 = stateindex_cl2 + 2
        }
        else if (stateindex_cl2 %in% c(36,39,41,42)) {
          quitflag = TRUE
        }
      }
      if (mat[(l-1)*h + j,stateindex_cl2] == 2) {
        actionsum[4] = actionsum[4] + 1
        if (stateindex_cl2 %in% c(33,34,35,37,38,40)) {
          stateindex_cl2 = stateindex_cl2 + 1
        }
        else if (stateindex_cl2 %in% c(36,39,41,42)) {
          quitflag = TRUE
        }
      }
      if (mat[(l-1)*h + j,stateindex_cl2] == 3) {
        quitflag = TRUE
      }
    }
    # stores the metrics in the appropriate columns in the matrix
    mat[(l-1)*h +j,63] = actionsum[1]
    mat[(l-1)*h +j,64] = actionsum[2]
    mat[(l-1)*h +j,65] = actionsum[3]
    mat[(l-1)*h +j,66] = actionsum[4]
  }
}
# loading packages for ploting
require(ggplot2)
```

```
require(cowplot)

# creating a data frame
m = data.frame(mat)

# creating the Sensitivity graph for Cluster 1

count_emails = factor(m[,63])
a1 = ggplot(m, aes(x = X1, y = X2, color = count_emails, size =count_emails)) +
  geom_point() +
  labs(
    x = "power of email",
    y = "power of phone",
    title = "Email (cluster 1)"
  ) +
  scale_colour_manual(
    values = c("0" = "red","1" = "blue","2" = "green", "3" = "black")
  ) +
  scale_size_manual(
    values=c("0"=1, "1" = 2, "2"= 4, "3" = 5)
  ) +
  theme(
    axis.title=element_text(size=13)
  )

count_phones = factor(m[,64])
a2 = ggplot(m, aes(x = X1, y = X2, color = count_phones, size = count_phones)) +
  geom_point() +
  labs(
    x = "power of email",
    y = "power of phone",
    title = "Phone (cluster 1)"
  ) +
  scale_colour_manual(
    values = c("0" = "red","1" = "blue","2" = "green", "3" = "black")
  ) +
  scale_size_manual(
    values=c("0"=1, "1" = 2, "2"= 4, "3"=5)
  ) +
  theme(
```

```
    axis.title=element_text(size=13)
  )

# ploting a1 and a2 in the same graph
plot_grid(a2, a1, ncol = 1, nrow = 2)



count_emails = factor(m[,65])
a3 = ggplot(m, aes(x = X1, y = X2, color = count_emails, size =count_emails)) +
  geom_point() +
  labs(
    x = "power of email",
    y = "power of phone",
    title = "Email (cluster 2)"
  ) +
  scale_colour_manual(
    values = c("0" = "red","1" = "blue","2" = "green", "3" = "black")
  ) +
  scale_size_manual(
    values=c("0"=1, "1" = 2, "2"= 4, "3" = 5)
  ) +
  theme(
    axis.title=element_text(size=13)
  )

count_phones = factor(m[,66])
a4 = ggplot(m, aes(x = X1, y = X2, color = count_phones, size = count_phones)) +
  geom_point() +
  labs(
    x = "power of email",
    y = "power of phone",
    title = "Phone (cluster 2)"
  ) +
  scale_colour_manual(
    values = c("0" = "red","1" = "blue","2" = "green", "3" = "black")
  ) +
  scale_size_manual(
    values=c("0"=1, "1" = 2, "2"= 4, "3"=5)
  ) +
```

```
216   theme(
217     axis.title=element_text(size=13)
218   )
219
220 # ploting a1 and a2 in the same graph
221 plot_grid(a4, a3, ncol = 1, nrow = 2)
```

plot_v2.R

Appendix B

SECOND APPENDIX

B.1 Grid World

B.1.1 Policy Iteration R Script

The policy iteration script for grid world consists of two parts. The first part is the function that gives a best policy based on the discount factor. The second part is a sensitivity analysis based on different discount factors.

```
# Policy function that gives an optimal policy for the given discount factor
policy <- function (gamma = 0.9) {

  # gamma - discount factor

  m_down <- c(.9, .1,  0,  0,  0,  0,  0,  0,  0,  0,  0,
              .1, .8, .1,  0,  0,  0,  0,  0,  0,  0,  0,
               0, .1, .8, .1,  0,  0,  0,  0,  0,  0,  0,
               0,  0, .1, .9,  0,  0,  0,  0,  0,  0,  0,
              .8,  0,  0,  0, .2,  0,  0,  0,  0,  0,  0,
               0,  0, .8,  0,  0, .1, .1,  0,  0,  0,  0,
               0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
               0,  0,  0,  0, .8,  0,  0, .1, .1,  0,  0,
               0,  0,  0,  0,  0,  0,  0, .1, .8, .1,  0,
               0,  0,  0,  0,  0, .8,  0,  0, .1,  0, .1,
               0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

  m_left <- c(.9,  0,  0,  0, .1,  0,  0,  0,  0,  0,  0,
              .8, .2,  0,  0,  0,  0,  0,  0,  0,  0,  0,
               0, .8, .1,  0,  0, .1,  0,  0,  0,  0,  0,
               0,  0, .8, .1,  0,  0, .1,  0,  0,  0,  0,
              .1,  0,  0,  0, .8,  0,  0, .1,  0,  0,  0,
               0,  0, .1,  0,  0, .8,  0,  0,  0, .1,  0,
               0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
               0,  0,  0,  0, .1,  0,  0, .9,  0,  0,  0,
               0,  0,  0,  0,  0,  0,  0, .8, .2,  0,  0,
```

```
              0,  0,  0,  0,  0, .1,  0,  0, .8, .1,  0,
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

m_right <- c(.1, .8,  0,  0, .1,  0,  0,  0,  0,  0,  0,
              0, .2, .8,  0,  0,  0,  0,  0,  0,  0,  0,
              0,  0, .1, .8,  0, .1,  0,  0,  0,  0,  0,
              0,  0,  0, .9,  0,  0, .1,  0,  0,  0,  0,
             .1,  0,  0,  0, .8,  0,  0, .1,  0,  0,  0,
              0,  0, .1,  0,  0,  0, .8,  0,  0, .1,  0,
              0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
              0,  0,  0,  0, .1,  0,  0, .1, .8,  0,  0,
              0,  0,  0,  0,  0,  0,  0,  0, .2, .8,  0,
              0,  0,  0,  0,  0, .1,  0,  0,  0, .1, .8,
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

 m_up <- c(  .1, .1,  0,  0, .8,  0,  0,  0,  0,  0,  0,
             .1, .8, .1,  0,  0,  0,  0,  0,  0,  0,  0,
              0, .1,  0, .1,  0, .8,  0,  0,  0,  0,  0,
              0,  0, .1, .1,  0,  0, .8,  0,  0,  0,  0,
              0,  0,  0,  0, .2,  0,  0, .8,  0,  0,  0,
              0,  0,  0,  0,  0,  0, .1, .1,  0, .8,  0,
              0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
              0,  0,  0,  0,  0,  0,  0, .9, .1,  0,  0,
              0,  0,  0,  0,  0,  0,  0, .1, .8, .1,  0,
              0,  0,  0,  0,  0,  0,  0,  0, .1, .8, .1,
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1  )

 p_down <- matrix(m_down, 11, 11, byrow=TRUE)
 p_left <- matrix(m_left, 11, 11, byrow=TRUE)
 p_right <- matrix(m_right, 11, 11, byrow=TRUE)
 p_up <- matrix(m_up, 11, 11, byrow=TRUE)


 r_down <-  c(0,  0,  0,   0,  0, -.1,  0,  0,  0, .1, 0) - 0.04
 r_left <-  c(0,  0,  0, -.1,  0,   0,  0,  0,  0,  0, 0) - 0.04
 r_right<-  c(0,  0,  0, -.1,  0, -.8,  0,  0,  0, .8, 0) - 0.04
 r_up <-    c(0,  0,  0, -.8,  0, -.1,  0,  0,  0, .1, 0) - 0.04

 #dummy policy
 pi <- c(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2) # all left
```

```
# 1 - down
# 2 - left
# 3 - right
# 4 - up

#Initial policy's value
v = solve( (diag(11) - gamma*p_left) ) %*% r_left

#Storing policy and its values for comparison later
pi_old = pi
v_old = v

#Build a new, better policy
for (i in 1:11) {

  best_action_value = v[i]
  c_d = r_down[i] + gamma*p_down[i, ] %*% v
  c_l = r_left[i] + gamma*p_left[i, ] %*% v
  c_r = r_right[i] + gamma*p_right[i, ] %*% v
  c_u = r_up[i] + gamma*p_up[i, ] %*% v

  if (c_d > best_action_value) {
    pi[i] = 1;
    best_action_value = c_d
  }
  if (c_l > best_action_value) {
    pi[i] = 2;
    best_action_value = c_l
  }
  if (c_r > best_action_value) {
    pi[i] = 3;
    best_action_value = c_r
  }
  if (c_u > best_action_value) {
    pi[i] = 4;
    best_action_value = c_u
  }
}
difference = sum ((pi - pi_old)^2)
pi_old = pi
```

```
107  while (difference !=0) {
108    r <- matrix(0, nrow = 11, ncol = 1 )
109    p <- matrix(0, nrow = 11, ncol = 11 )
110    for ( i in 1:11 ) {
111      if (pi[i] == 1) {
112        r[i] = r_down[i]
113        p[i, ] = p_down[i, ]
114      }
115      else if ( pi[i] == 2 ){
116        r[i] = r_left[i]
117        p[i, ] = p_left[i, ]
118      }
119      else if (pi[i] == 3 ) {
120        r[i] = r_right[i]
121        p[i, ] = p_right[i, ]
122      }
123      else if (pi[i] == 4 ) {
124        r[i] = r_up[i]
125        p[i, ] = p_up[i, ]
126      }
127    }
128
129    # Find the value of a new policy
130    v = solve( (diag(11) - gamma*p) ) %*% r
131    v
132
133    for (i in 1:11) {
134
135      best_action_value = v[i]
136      c_d = r_down[i] + gamma*p_down[i,] %*% v
137      c_l = r_left[i] + gamma*p_left[i,] %*% v
138      c_r = r_right[i] + gamma*p_right[i,] %*% v
139      c_u = r_up[i] + gamma*p_up[i,] %*% v
140
141      if (c_d > best_action_value) {
142        pi[i] = 1
143        best_action_value = c_d
144      }
145      if (c_l > best_action_value) {
146        pi[i] = 2
```

```
147         best_action_value = c_l
148       }
149       if (c_r > best_action_value) {
150         pi[i] = 3
151         best_action_value = c_r
152       }
153       if (c_u > best_action_value) {
154         pi[i] = 4
155         best_action_value = c_u
156       }
157     }
158
159     pi # to see if new policy is different
160     difference = sum ((pi - pi_old)^2)
161     pi_old = pi
162   }
163   pi
164 }
165
166 # for gamma sensitivity analysis
167 m = matrix(0, nrow = 10, ncol = 12 )
168 for (i in 1:10) {
169   m[i,] = c(0.89 + i*0.01, policy(gamma = 0.89 + i*0.01))
170 }
```

gridworld_R.R

B.1.2 Q-learning R Scripts

Q-learning R script consist of three parts. First part creates a transition reward matrix. The second part creates q-learning function that calculates an optimal policy. The final part is the sensitivity analysis based on different parameters of the model.

```
1 #transition reward function creates transition reward matrix
2 transition_reward <- function (state, action) {
3
```

```
# state will be a number from 1-11.
# action will be a number from 1-4.
# 1 is down
# 2 is left
# 3 is right
# 4 is up

n = 11;  # Number of states
k = 1;   # Size of sample to take

m_down <- c(.9, .1,  0,  0,  0,  0,  0,  0,  0,  0,  0,
            .1, .8, .1,  0,  0,  0,  0,  0,  0,  0,  0,
             0, .1, .8, .1,  0,  0,  0,  0,  0,  0,  0,
             0,  0, .1, .9,  0,  0,  0,  0,  0,  0,  0,
            .8,  0,  0,  0, .2,  0,  0,  0,  0,  0,  0,
             0,  0, .8,  0,  0, .1, .1,  0,  0,  0,  0,
             0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
             0,  0,  0,  0, .8,  0,  0, .1, .1,  0,  0,
             0,  0,  0,  0,  0,  0,  0, .1, .8, .1,  0,
             0,  0,  0,  0,  0, .8,  0,  0, .1,  0, .1,
             0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

m_left <- c(.9,  0,  0,  0, .1,  0,  0,  0,  0,  0,  0,
            .8, .2,  0,  0,  0,  0,  0,  0,  0,  0,  0,
             0, .8, .1,  0,  0, .1,  0,  0,  0,  0,  0,
             0,  0, .8, .1,  0,  0, .1,  0,  0,  0,  0,
            .1,  0,  0,  0, .8,  0,  0, .1,  0,  0,  0,
             0,  0, .1,  0,  0, .8,  0,  0,  0, .1,  0,
             0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
             0,  0,  0,  0, .1,  0,  0, .9,  0,  0,  0,
             0,  0,  0,  0,  0,  0,  0, .8, .2,  0,  0,
             0,  0,  0,  0,  0, .1,  0,  0, .8, .1,  0,
             0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

m_right <- c(.1, .8, 0,  0, .1,  0,  0,  0,  0,  0,  0,
             0, .2, .8,  0,  0,  0,  0,  0,  0,  0,  0,
             0,  0, .1, .8,  0, .1,  0,  0,  0,  0,  0,
             0,  0,  0, .9,  0,  0, .1,  0,  0,  0,  0,
            .1,  0,  0,  0, .8,  0,  0, .1,  0,  0,  0,
             0,  0, .1,  0,  0,  0, .8,  0,  0, .1,  0,
```

```
              0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
              0,  0,  0,  0, .1,  0,  0, .1, .8,  0,  0,
              0,  0,  0,  0,  0,  0,  0,  0, .2, .8,  0,
              0,  0,  0,  0,  0, .1,  0,  0,  0, .1, .8,
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

m_up <-   c(.1, .1,  0,  0, .8,  0,  0,  0,  0,  0,  0,
            .1, .8, .1,  0,  0,  0,  0,  0,  0,  0,  0,
              0, .1,  0, .1,  0, .8,  0,  0,  0,  0,  0,
              0,  0, .1, .1,  0,  0, .8,  0,  0,  0,  0,
              0,  0,  0,  0, .2,  0,  0, .8,  0,  0,  0,
              0,  0,  0,  0,  0,  0, .1, .1,  0, .8,  0,
              0,  0,  0,  0,  0,  0,  1,  0,  0,  0,  0,
              0,  0,  0,  0,  0,  0,  0, .9, .1,  0,  0,
              0,  0,  0,  0,  0,  0,  0, .1, .8, .1,  0,
              0,  0,  0,  0,  0,  0,  0,  0, .1, .8, .1,
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1 )

p_down <- matrix( m_down, 11, 11, byrow = TRUE )
p_left <- matrix( m_left, 11, 11, byrow = TRUE )
p_right <- matrix( m_right, 11, 11, byrow = TRUE )
p_up <- matrix( m_up, 11, 11, byrow = TRUE )

if ( action ==1 ) {
  next_state <- sample( 1:n, k, replace = TRUE, prob = p_down[state, ] )
}
else if ( action == 2 ) {
  next_state <- sample( 1:n, k, replace = TRUE, prob = p_left[state, ] )
}
else if ( action == 3 ) {
  next_state <- sample( 1:n, k, replace = TRUE, prob = p_right[state, ] )
}
else {
  next_state <- sample( 1:n, k, replace = TRUE, prob = p_up[state, ] )
}

if ( next_state == 7 ) {
  reward = -1 - 0.04
}
else if ( next_state == 11 ) {
```

```
    reward = 1- 0.04
  }
  else {
    reward = -0.04
  }

  # a <- data.frame(next_state = next_state, reward = reward)
  # a
  c(next_state, reward)
}
```

transition_reward_R.R

```
# To find an optimal policy run q_learning function
q_learning <- function (n = 1000, gamma =0.8, epsilon = 0.2, alpha = 0.1) {

  # n - Iterations
  # gamma - Discount factor
  # epsilon - Exploration coefficient
  # alpha - Learning parameter

  q_values = matrix(0, nrow = 11, ncol = 4 )

  state = 1
  action = sample( 1:4, 1 )

  source("C:/Users/artur-grygorian/Google Drive/thesis/Q-learning grid world/
      transition_reward_R.R")

  for ( i in 1:n ) {
    next_state = transition_reward( state, action )[1]
    reward = transition_reward( state, action )[2]

    q_values[state, action] = (1-alpha)*q_values[state,action] + alpha*(reward +
        gamma*max(q_values[next_state, ]))

    if (is.element(next_state, c(7,11))) {
      state = 1
    }
    else {
```

```
      state = next_state
    }
    if ( runif(1) < epsilon ) {
      action = sample ( 1:4, 1)
    }
    else {
      action = which.max(q_values[state, ])
    }
  }

  policy = matrix(0, nrow = 1, ncol = 11 )

  for (k in 1:11) {
    action = which.max(q_values[k,])
    policy[k] = action
  }
  return(policy)

}

m = matrix(0, nrow = 10, ncol = 12 )


  # Sensitivity analysis based on different values
# for alpha
for (i in 1:10) {
  m[i,] = c(i*0.1, q_learning(alpha = i*0.1))
}

# for gamma
for (i in 1:10) {
  m[i,] = c(i*0.1, q_learning(gamma = i*0.1))
}

# for number of iterations
for (i in 1:10) {
  m[i,] = c(i*1000, q_learning(n = i*1000))
}

# for epsilon
```

```
for (i in 1:10) {
  m[i,] = c(i*0.1, q_learning(epsilon = i*0.1))
}
```

q_learning_R.R