2018-08-22

Optimal Policy for Blood Inventory Management Problem

Grushevska, Iaryna

UNIVERSITY OF CALGARY

Optimal Policy for Blood Inventory Management Problem

by

Iaryna Grushevska

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN MATHEMATICS AND STATISTICS

CALGARY, ALBERTA
AUGUST, 2018

Abstract

Blood units that are used for transfusion can be stored for a limited amount of time. The blood that is older than 42 days must be discarded. In order not to face shortage usually the oldest blood is used. But the risk of complications after surgery is growing as the age of used blood is growing as well.

In this work we find the optimal policy to use blood for transfusion for two blood types. The main goal is to find the policy that will reduce the shortage and minimize the risk of complications at the same time. For this purpose, we use two methods (Linear Programming and Approximate Dynamic Programming) and compare the results of two approaches.

Acknowledgements

I would like to thank:

Dr. Elena Braverman, Dr. Alireza Sabouri and Dr. Yuriy Zinchenko for being awesome supervisors, for their support, guidance and great patience during my Master's program; for never being busy to meet me and answer my questions; for their help with thesis preparation and advice.

My parents Bohdan and Romana, my brother Taras, and all my friends for their support and belief in me.

Table of Contents

List of Figures and Illustrations

List of Tables

List of Symbols, Abbreviations and Nomenclature

Symbol or abbreviation	Definition
LP	Linear Programming
ADP	Approximate Dynamic Programming
AB+, AB-, A+, A-, B+, B-, O+, O-	Types of blood
MDP	Markov Decision Process
FIFO	First-In First-Out Policy
LIFO	Last-In First-Out Policy

Chapter 1

Introduction

In a blood bank or in a hospital, if not frozen, blood can be stored for 42 days. All the blood units that are older than 42 days must be discarded.

There are two main strategies of choosing the blood for transfusion [5]:

- First-In, First-Out (FIFO) - blood will be used starting from the oldest to the youngest;
- Last-In, First-Out (LIFO) - the youngest blood will be used first.

At this time in the hospitals the oldest blood is usually used for transfusions (FIFO). But as the older blood is used, the risk of complications after surgery is higher [10]. On the other hand if the newest blood would be used many units of older blood will need to be discarded and it is more likely that the hospital will face the shortage of blood.

If a hospital does not have enough blood to satisfy demand for a day, some blood units can be received from a secondary source (blood bank or another hospital). But the cost of getting blood from the secondary source is usually significantly higher than the cost of using a unit of blood that is available in the hospital.

There exist eight blood types: AB+, AB-, A+, A-, B+, B-, O+, O-. Here + and - denote the rhesus factor of the blood. But specific blood for transfusion cannot be substituted by any other type. Blood substitution can be done according to the following diagram (Figure 1.1). There are 27 possible donor-recipient connections.

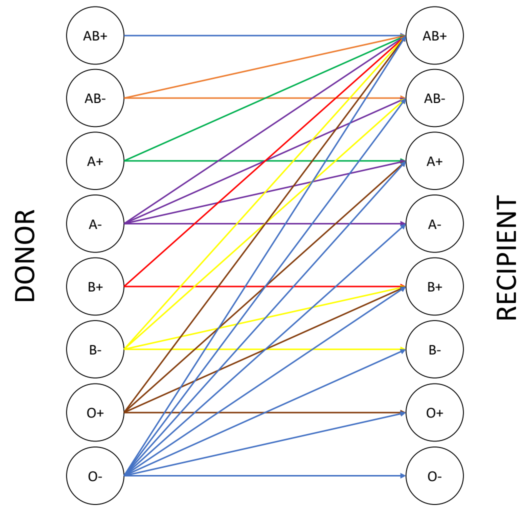

Figure 1.1: Blood substitution scheme

The main goal of this thesis is to solve a blood inventory management problem and to find an optimal way to use the blood for transfusion. We will find the optimal policy using LP approach and approximate optimal policy using the ADP method and compare those two results.

The problem is formulated as a Markov Decision Process. In the first part, we will find the optimal cost of using blood units by solving the Bellman equation. The cost of taking a particular action includes the cost of using a unit of blood that is currently available in a hospital multiplied by the age of the unit that can be also considered as a penalty for possible risk of complications after the surgery. The cost also includes the penalty for getting the unit of blood from the secondary source. Trying to minimize the penalty of getting the blood unit from the secondary source, at the same time we minimize the amount of blood that will be discarded. The Bellman equation will be solved using Linear Programming. Having the optimal cost for each state-action pair we will find the optimal policy again by solving a linear programming problem.

In the second part, we will use the Approximate Dynamic Programming techniques to find an approximate optimal policy for using blood units for transfusion. That will include building an approximate value function, Phase I method and column generation.

If we consider that the recipient can get a blood unit of any age (1 - 42 days) we have the supply vector (that consists of the number of available blood units of every blood type and every age) 8x42=336 long. For this project the problem was reduced, so it can be solved using Linear Programming.

This work is based on research of A. Sabouri [19]. A similar algorithm as developed for one type of blood will be used, but we will extend it to the case of two blood types.

Our problem is closely related to the models of inventory systems of perishable products that we can see in works of Nahmias [13, 12].

Most of the studies of perishable inventory management are focused on finding optimal ordering policies. Issuing policies are assumed to be FIFO (First-In, First-Out) or LIFO (Last-In, First-Out). In particular, FIFO was shown to be an optimal policy by Pierskalla [17]. But the researches Eikelboom [7], Offner [15], Koch [10] show that the use of older blood can cause serious complications after the transfusion such as infections, morbidity and even death. On the other hand, Dzik [5] and Sayers [20] in their studies show that if younger blood are used for transfusions that will cause shortage of blood units that are available in the hospitals.

There is also a threshold policy that was introduced by Atkinson [2]. The idea of a threshold policy is that we use the youngest blood if the age of blood unit is older than the threshold and the oldest blood if the age of blood unit is younger than that threshold. Basically for the blood that is younger than the threshold FIFO policy is working and LIFO policy for the blood that is older than the threshold.

Chapter 2

Problem Formulation

For simplicity, suppose we have only two blood types AB+ (AB positive) and AB- (AB negative). Assume that the blood can be stored for 2 days. Blood that is older than two days will be discarded. Blood can be substituted according to the following diagram.

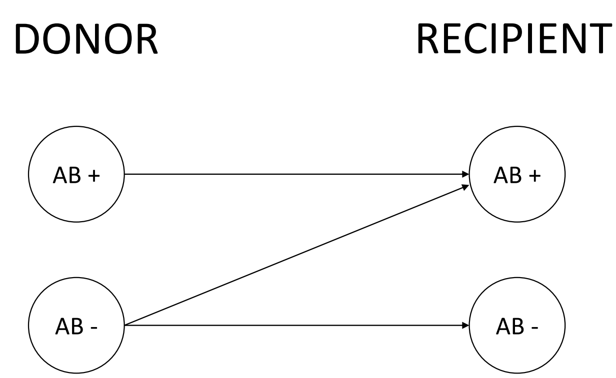

Figure 2.1: Reduced substitution scheme

The following symbols will be used:

$i \in \{1, 2\}$ is the age of blood.

$s_i \in \{s_1^+, s_1^-, s_2^+, s_2^-\}$ - supply of blood of age i at the beginning of the day; plus or minus denotes the rhesus factor of the blood. For simplicity we assume that only one unit of each type of blood might be available every day.

x_i - amount of blood of age i that was used to satisfy a demand. We can use no more blood than it is available, so $x_i \in \{0, 1\}$.

$X = (x_1^{++}, x_1^{--}, x_1^{-+}, x_2^{++}, x_2^{--}, x_2^{-+})$ is an action. x_1^{++} denotes the number of blood units of age 1 that will be transfused from the donor with positive rhesus factor to a recipient with positive rhesus factor. The symbols $++$, $--$, $-+$ denote the rhesus factor of a donor and a recipient, respectively. Similarly, x_2^{++}, x_2^{--}, x_2^{-+} denote the number of blood units of age 2 that will be transfused.

q^+,q^- - amount of blood that arrives at the end of the day $q^+, q^- \in \{0, 1\}$.

We assume that only fresh blood can arrive every day (we do not receive any blood of age 2).

$Q = (q_1^+, q_1^-)$ describes new arrivals, here q_1^+, q_1^- denote the amount of new blood with the rhesus factor $+$ or $-$, respectively. $q_1^+, q_1^- \in \{0, 1\}$

$d = (d^+, d^-)$ - demand. We assume that the maximum demand of each type of blood is not more than 2 units. $d^+, d^- \in \{0, 1, 2\}$

Every day we observe the demand d and decide which action to take (form vector X) in order to satisfy the demand. After a new blood arrives, the new vector of blood supply can be formed:

$(q_1^+, q_1^-, s_1^+ - x_1^{++}, s_1^- - x_1^{--} - x_1^{-+})$

State description:

$S = (s_1^+, s_1^-, s_2^+, s_2^-, d^+, d^-)$

As the first four variables $s_1^+, s_1^-, s_2^+, s_2^-$ take two and the last two d^+, d^- three possible values, the total number of states in the system is equal to $2^4 3^2 = 144$.

The action space is described by the inequalities:

1. $x_i^{++} \leq s_i^+$ means that we cannot use more blood of age i with positive rhesus factor than is available.

2. $x_i^{--} + x_i^{-+} \leq s_i^-$ means that we cannot use more blood of age i with negative rhesus factor than is available.

3. $\sum_i x_i^{--} \leq d^-$ - we won't use more blood than we need to satisfy the demand of blood with negative rhesus factor.

4. $\sum_i x_i^{++} + \sum_i x_i^{-+} \leq d^+$ - we won't use more blood than we need to satisfy the demand of blood with positive rhesus factor.

(0 1 0 0 | 1 0)
(0 1 0 1 | 1 2)
(0 1 1 0 | 0 0)
(1 1 1 1 | 2 2)

Figure 2.2: Samples of state vectors. The vertical line is used to separate the supply and the demand.

Figure (2.2) shows some samples of state vectors. First vector $(0100|10)$ means that we have one unit of blood with negative rhesus factor of age 1 (i.e., the second component $s_1^- = 1$) and the demand is one unit of blood with positive rhesus factor (i.e., the fifth component $d^+ = 1$).

Transition probabilities are introduced as:

$$p(S^{'} \mid S, X) = \begin{cases} P(Q, d^{+'}, d^{-'}), & \text{if } S^{'} = (q_1^+, q_1^-, s_1^+ - x_1^{++}, s_1^- - x_1^{--} - x_1^{-+}, d^{+'}, d^{-'}), \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

where $p(S^{'} \mid S, X)$ describes the probability of transition from S to $S^{'}$ if action X is taken. $P(Q, d^{+'}, d^{-'})$ is the probability that the vector of new arrivals of the next period of time will be Q and the demand of new period will be $(d^{+'}, d^{-'})$.

Number of all possible combinations of vector Q is 4 and for the demand number of all possible combination is 9 so we assume that all the probabilities are equal to $1/(4 * 9)$.

$P(Q, d^{+'}, d^{-'}) = 1/36$

The immediate cost is

$$C(S, X) = \sum_i ic(x_i^{++} + x_i^{-+} + x_i^{--}) + l((d^+ - \sum_i x_i^{++} - \sum_i x_i^{-+}) + (d^- - \sum_i x_i^{--})), \tag{2.2}$$

where ic is the the cost of using a unit of blood available in the current hospital, l is the cost of getting an additional unit of blood from the secondary source (blood bank or another hospital). We multiply cost c by i as a penalty for using older blood. Obviously, as we use older blood the penalty increases.

In order to find the minimal cost, we are supposed to solve the Bellman equation of the form

$$V(S) = \min_{\forall X} \left\{ C(S, X) + \lambda \sum_{S'} p(S' \mid S, X) V(S') \right\}, \tag{2.3}$$

where V is a value vector, $V(S)$ is the current state, $V(S')$ is the next state, $C(S, X)$ is a cost for being in state S and taking action X, $p(S' \mid S, X)$ is a probability of transition from S to $S^{'}$ if action X is taken, $\lambda \in (0, 1)$ is a discount factor.

The discount factor does not effect the algorithm or the theoretical result. We account for time preferences by including the discount factor [18].

V satisfies the following inequality:

$$V(S) \geq C(S, X) + \lambda \sum_{S'} p(S' \mid S, X) V(S'), \tag{2.4}$$

for all state-action pairs (S, X).

Chapter 3

Solution approach

3.1 Linear Programming approach

We solve the Bellman equation (2.3) using primal linear programming system:

$$\begin{aligned}
&\mathit{maximize} \sum_S \alpha(S) * V(S) \\
&\mathit{subject\ to}\ V(S) - \sum_S \lambda p(S'|S,X) * V(S') \leq C(S,X)
\end{aligned}$$

Objective coefficients $\alpha(s)$ are all positive and $\sum_S \alpha(S) = 1$. We put $\lambda = 0.8$ as far as we are working on the infinite horizon. Otherwise $V(S) = \infty$.

Having optimal value vector $V^*(S)$, we can obtain the optimal policy solving the following equation:

$$d(S) = \min_X \{C(S,X) + \sum_{S'} p(S' \mid S,X) V^*(S)\}, \tag{3.1}$$

where d(S) is a particular action that is optimal to take in state S.

As far as we consider only two blood types and the maximum age is two days, there are 144 variables, because we have 144 states. The number of constraints is 645. The size of the problem is small enough for solving this Linear Programming problem.

But if we consider all the blood types, and the age of blood up to 42 days, our problem suffers from the curse of dimensionality.

3.2 Approximate Dynamic Programming approach

To solve a larger problem (for more blood types and consider the age of blood units up to 42 days) we can use the Approximate Dynamic Programming.

To be able to compare the results of Linear Programming and Approximate Dynamic Programming approaches we solve the problem with the same number of states, as was described in Chapter 2, using Approximate Dynamic Programming.

First we build the approximation to the value function of the form [19]:

$$V(S) \cong \tilde{V}(S) = \theta_0 + \sum_{i=1}^{2} \theta_i^+ u_i^+ + \sum_{i=1}^{2} \theta_i^- u_i^- + \delta_1 d^+ + \delta_2 d^- + \sigma_1 [d^+ - u_2^+]^+ + \sigma_2 [d^- - u_2^-]^+, \quad (3.2)$$

where

$u_i^+ = \sum_{j=1}^{i} s_j^+$ is a total number of available blood units of the age at most i with a positive rhesus factor. Similarly, for the negative rhesus factor $u_i^- = \sum_{j=1}^{i} s_j^-$.

$[d^+ - u_2^+]^+ = \max(0, d^+ - u_2^+)$ denotes the shortage, where d^+ is a demand of the blood with a positive rhesus factor. Similarly, for a negative rhesus factor $[d^- - u_2^-]^+ = \max(0, d^- - u_2^-)$.

θ_i^+ and θ_i^- represent the savings in cost for each additional unit of blood of age at most i.

δ_1 and δ_2 represent the cost of each additional unit of demand.

σ_1 and σ_2 denote the cost of each additional unit of shortage.

Now we will use approximate dynamic programming algorithms to find such coefficients $(\theta_0, \theta_1^+, \theta_1^-, \theta_2^+, \theta_2^-, \delta_1, \delta_2, \sigma_1, \sigma_2)$ that will make the equation (3.2) a good approximation for the exact value function V(S).

3.2.1 Calibrating approximate value function coefficients

To calibrate the coefficients, we use the idea of Schweitzer and Seidmann [21] that is based on linear programming.

We consider the linear programming problem of the form:

maximize $\sum_{S} \alpha(S) * V(S)$

subject to $V(S) - \sum_{S} \lambda p(S'|S, X) * V(S') \geq C(S, X)$

as in Section 3.1.

Now we replace V(S) by our approximation defined in (3.2):

maximize

$$\begin{aligned} \theta_0 + \sum_{i=1}^{2} \mathbb{E}_\alpha[u_i^+]\theta_i^+ + \sum_{i=1}^{2} \mathbb{E}_\alpha[u_i^-]\theta_i^- + \mathbb{E}_\alpha[d^+]\delta_1 + \mathbb{E}_\alpha[d^-]\delta_2 \\ + \mathbb{E}_\alpha[[d^+ - u_2^+]^+]\sigma_1 + \mathbb{E}_\alpha[[d^- - u_2^-]^+]\sigma_2 \end{aligned} \tag{3.3}$$

subject to

$(1-\lambda)\theta_0 + \sum_{i=1}^{2} \Theta_i^+(S, X)\theta_i^+ + \sum_{i=1}^{2} \Theta_i^-(S, X)\theta_i^- + \Delta_1(S)\delta_1 + \Delta_2(S)\delta_2 + \Sigma_1(S, X)\sigma_1 + \Sigma_2(S, X)\sigma_2 \leq C(S, X) \qquad \forall (S, X)$

where

$\mathbb{E}_\alpha[u_i^+] = \sum_{S} \alpha(S) u_i^+(S) \qquad i = 1, 2$

$\mathbb{E}_\alpha[u_i^-] = \sum_{S} \alpha(S) u_i^-(S) \qquad i = 1, 2$

$\mathbb{E}_\alpha[d^+] = \sum_{S} \alpha(S) d^+(S)$

$\mathbb{E}_\alpha[d^-] = \sum_{S} \alpha(S) d^-(S)$

$\mathbb{E}_\alpha[[d^+ - u_2^+]^+] = \sum_{S} \alpha(S)[d^+(S) - u_2^+(S)]^+$

$\mathbb{E}_\alpha[[d^- - u_2^-]^+] = \sum_{S} \alpha(S)[d^-(S) - u_2^-(S)]^+$

and

$\Theta_i^+(S, X) = u_i^+(S) - \lambda \sum_{(Q, d')} Pr(Q, d') u_i^{+\prime}(S, X, Q) \qquad i = 1, 2$

$$\Theta_i^-(S,X) = u_i^-(S) - \lambda \sum_{(Q,d')} Pr(Q,d') u_i^{-\prime}(S,X,Q) \qquad i = 1,2$$
$$\Delta^+(S) = d^+(S) - \lambda \sum_{(Q,d')} Pr(Q,d') d^{+\prime}$$
$$\Delta^-(S) = d^-(S) - \lambda \sum_{(Q,d')} Pr(Q,d') d^{-\prime}$$
$$\Sigma^+(S,X) = [d^+(S) - u_2^+(S)]^+ - \lambda \sum_{(Q,d')} Pr(Q,d')[d^{+\prime}(S) - u_2^{+\prime}(S)]^+$$
$$\Sigma^-(S,X) = [d^-(S) - u_2^-(S)]^+ - \lambda \sum_{(Q,d')} Pr(Q,d')[d^{-\prime}(S) - u_2^{-\prime}(S)]^+,$$

where $d' = (d^{+'}, d^{-'})$ denotes the demand of the next period of time.

There are much fewer variables, however the number of constrains is still large. But to find the optimal solution we can start with the initial set of 9 constraints and add new constrains by finding the most violated ones.

It is suitable to solve the dual problem using column generation.

The dual problem to (3.3) can be stated as follows:

minimize

$$\sum_{(S,X)} C(X,S) W(S,X) \tag{3.4}$$

subject to

$$(1-\lambda) \sum_{(S,X)} W(X,S) = 1$$
$$\sum_{(S,X)} \Theta_i^+(S,X) W(X,S) = \mathbb{E}_\alpha[u_i^+] \qquad i = 1,2$$
$$\sum_{(S,X)} \Theta_i^-(S,X) W(X,S) = \mathbb{E}_\alpha[u_i^-] \qquad i = 1,2$$
$$\sum_{(S,X)} \Delta_1(S) W(X,S) = \mathbb{E}_\alpha[d^+]$$
$$\sum_{(S,X)} \Delta_2(S) W(X,S) = \mathbb{E}_\alpha[d^-]$$
$$\sum_{(S,X)} \Sigma_1(S,X) W(X,S) = \mathbb{E}_\alpha[[d^+ - u_2^+]^+]$$
$$\sum_{(S,X)} \Sigma_2(S,X) W(X,S) = \mathbb{E}_\alpha[[d^- - u_2^-]^+]$$
$$W(S,X) \geq 0 \qquad \forall (S,X).$$

In the dual problem we have a variable for each state-action pair. We can use much less variables to find an optimal value. We will use the Phase I method of linear programming to find the initial set of columns, and then we will use column generation to add the most violated constraints.

Phase I method of linear programming

We start with adding a slack variable to each constraint to the problem (3.4). Then, we minimize the sum of the slack variables.

minimize

$$\sum_{i=1}^{9} y_i \tag{3.5}$$

subject to

$y_1 = 1$

$y_2 = \mathbb{E}_\alpha[u_1^+]$

$y_3 = \mathbb{E}_\alpha[u_2^+]$

$y_4 = \mathbb{E}_\alpha[u_1^-]$

$y_5 = \mathbb{E}_\alpha[u_2^-]$

$y_6 = \mathbb{E}_\alpha[d^+]$

$y_7 = \mathbb{E}_\alpha[d^-]$

$y_8 = \mathbb{E}_\alpha[[d^+ - u_2^+]^+]$

$y_9 = \mathbb{E}_\alpha[[d^- - u_2^-]^+]$

$y_i \geq 0 \qquad \text{for } i = 1..9.$

We add new constraints by solving the following sub-problem:

maximize

$$\begin{aligned}(1-\lambda)\theta_0^* + \sum_{i=1}^{2} \Theta_i^+(S,X)\theta_i^{+*} + \sum_{i=1}^{2} \Theta_i^-(S,X)\theta_i^{-*} + \Delta_1(S)\delta_1^* \\ +\Delta_2(S)\delta_2^* + \Sigma_1(S,X)\sigma_1^* + \Sigma_2(S,X)\sigma_2^*\end{aligned} \tag{3.6}$$

subject to

$$x_1^{++} \leq s_1^+$$

$$x_1^{-+} + x_1^{--} \leq s_1^-$$

$$x_2^{++} \leq s_2^+$$

$$x_2^{-+} + x_2^{--} \leq s_2^-$$

$$\sum_{i=1}^{2} x_i^{++} + \sum_{i=1}^{2} x_i^{-+} \leq d^+$$

$$\sum_{i=1}^{2} x_i^{--} \leq d^-$$

$$x_i, s_i \geq 0, \text{ integer} \qquad i = 1,2$$

$$d^+, d^- \geq 0 \text{ , integer}$$

where

$$\Theta_i^+(S,X) = \sum_{j=1}^{i} s_i^+ - \lambda \sum_{(Q,d')} Pr(Q,d')(\sum_{j=1}^{i} q_i^+ + \sum_{j=1}^{i-1}(s_j^+ - x_j^{++})) \qquad i = 1,2,$$

$Pr(Q, d')$ denotes the probability that the new arrival vector will be Q and new demand will be d'.

$$\Theta_i^-(S,X) = \sum_{j=1}^{i} s_i^- - \lambda \sum_{(Q,d')} Pr(Q,d')(\sum_{j=1}^{i} q_i^- + \sum_{j=1}^{i-1}(s_j^- - x_j^{-+} - x_j^{--})) \qquad i = 1,2$$

$$\Delta_1(S) = d^+ - \lambda \sum_{(Q,d')} Pr(Q,d') d^{+'}$$

$$\Delta_2(S) = d^- - \lambda \sum_{(Q,d')} Pr(Q,d') d^{-'}$$

$$\Sigma_1(S,X) = [d^+ - \sum_{j=1}^{2} s_j^+]^+ - \lambda \sum_{(Q,d')} Pr(Q,d')[d^{+'} - u_2^{+'}]^+$$

$$\Sigma_2(S,X) = [d^- - \sum_{j=1}^{2} s_j^-]^+ - \lambda \sum_{(Q,d')} Pr(Q,d')[d^{-'} - u_2^{-'}]^+$$

$$u_2^{+'} = \sum_{j=1}^{2} q_j^+ + (s_1^+ - x_1^{++})$$

$$u_2^{-'} = \sum_{j=1}^{2} q_j^- + (s_1^- - x_1^{-+} - x_1^{--})$$

and

$(\theta_0^*, \theta_1^{+*}, \theta_1^{-*}, \theta_2^{+*}, \theta_2^{-*}, \delta_1^*, \delta_2^*, \sigma_1^*, \sigma_2^*)$ is the dual solution of the problem (3.5).

The solution to the problem (3.6) includes the state $(s_1^{+*}, s_1^{-*}, s_2^{+*}, s_2^{-*}, d^{+*}, d^{-*})$ and the action $(x_1^{++*}, x_1^{-+*}, x_1^{--*}, x_2^{++*}, x_2^{-+*}, x_2^{--*})$. The pair (S^*, X^*) that corresponds to the most violated constraint will be added to the constraints of the problem (3.5).

But $\Sigma(S,X)$ is not a linear function because of the part $[d^- - \sum_{j=1}^{2} s_j^-]^+$ that is a piecewise linear function of the decision variables. We can make this function linear by introducing new integer variables k_0^{++}, k_0^{+-}, $k^{++}(Q,d')$, $k^{+-}(Q,d')$ and b_0^+, $b^+(Q,d')$ that are binary variables. A similar set of variables we introduce for the part, where we consider the units of blood with a negative rhesus factor k_0^{-+}, k_0^{--}, $k^{-+}(Q,d')$, $k^{--}(Q,d')$, b_0^-, $b^-(Q,d')$.

Here $\Sigma_1(S,X) = k_0^{++} - \lambda \sum_{(Q,d')} Pr(Q,d')k^{++}(Q,d')$,
where k_0^{++} satisfies the constraints:

$$k_0^{++} + k_0^{+-} - N = d^+ - \sum_{j=)}^{2} s_j^+ \tag{3.7}$$

$$k_0^{++} \leq N b_0^+ \tag{3.8}$$

$$k_0^{+-} \geq N b_0^+ \tag{3.9}$$

$$k_0^{+-} \leq N \tag{3.10}$$

$$k_0^{++}, k_0^{++} \geq 0, \text{ integer}$$

$$b_0^+ \in \{0,1\},$$

and N is a large integer.

If there is a shortage and $d^+ - \sum_{j=1}^{2} s_j^+ > 0$ then from equations (3.7) and (3.10) we have $k_0^{++} > 0$.

From the inequality (3.8) we have $b_0^+ = 1$.

Then, (3.9) and (3.10) imply $k_0^{+-} = N$.

Finally, from (3.7) we have $k_0^{++} = d^+ - \sum_{j=1}^{2} s_j^+$ (i.e., k_0^{++} denotes the shortage).

If there is no shortage, $d^+ - \sum_{j=1}^{2} s_j^+ \leqslant 0$, then from (3.7) - (3.10) we have $k_0^{++} = 0$.

Next, $k^{++}(Q, d')$, $k^{+-}(Q, d')$ must satisfy the following constraints:

$$k^{++}(Q, d') + k^{+-}(Q, d') - N = d^{+'} - \sum_{j=1}^{2} q_j^+ + (s_1^+ - x_1^{++}) \tag{3.11}$$

$$k^{++}(Q, d') \leq Nb^+(Q, d') \tag{3.12}$$

$$k^{+-}(Q, d') \geq Nb^+(Q, d') \tag{3.13}$$

$$k^{+-}(Q, d') \leq N \tag{3.14}$$

$$k^{++}(Q, d'), k^{+-}(Q, d') \geq 0, \text{ integer}$$

$$b^+(Q, d') \in \{0, 1\}.$$

Now, the problem (3.6) is an integer problem. The solution (S^*, X^*) corresponds to the most violated constraint that we add to the problem (3.5). Once the objective function of (3.5) becomes 0, we stop. Then we remove the slack variables and end up with the initial set of constrains for the problem (3.4).

Column Generation

Now, having the initial set of constraints, we can solve the dual problem

minimize

$$\sum_{(S,X)} C(X, S) W(S, X) \tag{3.15}$$

subject to

$$(1 - \lambda) \sum_{(S,X)} W(X, S) = 1$$

$$\sum_{(S,X)} \Theta_i^+(S,X)W(X,S) = \mathbb{E}_\alpha[u_i^+] \qquad i = 1,2$$
$$\sum_{(S,X)} \Theta_i^-(S,X)W(X,S) = \mathbb{E}_\alpha[u_i^-] \qquad i = 1,2$$
$$\sum_{(S,X)} \Delta_1(S)W(X,S) = \mathbb{E}_\alpha[d^+]$$
$$\sum_{(S,X)} \Delta_2(S)W(X,S) = \mathbb{E}_\alpha[d^-]$$
$$\sum_{(S,X)} \Sigma_1(S,X)W(X,S) = \mathbb{E}_\alpha[[d^+ - u_2^+]^+]$$
$$\sum_{(S,X)} \Sigma_2(S,X)W(X,S) = \mathbb{E}_\alpha[[d^- - u_2^-]^+]$$
$$W(S,X) \geq 0 \qquad \forall (S,X).$$

The solution to this problem does not satisfy all the constraints from the original problem as we consider only the most violated constraints. To find the most violated constraints, we will solve the sub-problem. In this sub-problem, in comparison to the one in Phase I method, we add one more term $C(S,X)$:

maximize

$$\begin{aligned}(1-\lambda)\theta_0^* + \sum_{i=1}^{2} \Theta_i^+(S,X)\theta_i^{+*} + \sum_{i=1}^{2} \Theta_i^-(S,X)\theta_i^{-*} + \Delta_1(S)\delta_1^* \\ +\Delta_2(S)\delta_2^* + \Sigma_1(S,X)\sigma_1^* + \Sigma_2(S,X)\sigma_2^* - C(S,X)\end{aligned} \tag{3.16}$$

subject to

$$x_1^{++} \leq s_1^+$$
$$x_1^{-+} + x_1^{--} \leq s_1^-$$
$$x_2^{++} \leq s_2^+$$
$$x_2^{-+} + x_2^{--} \leq s_2^-$$
$$\sum_{i=1}^{2} x_i^{++} + \sum_{i=1}^{2} x_i^{-+} \leq d^+$$
$$\sum_{i=1}^{2} x_i^{--} \leq d^-$$
$$x_i, s_i \geq 0 \qquad i = 1,2$$
$$d^+, d^- \geq 0.$$

As well as in Phase I method, we have the parts that are not linear $\Sigma_1(S, X)$, $\Sigma_2(S, X)$. Similarly, we introduce the set of variables k_0^{++}, k_0^{+-}, $k^{++}(Q, d')$, $k^{+-}(Q, d')$, b_0^+, $b^+(Q, d')$, k_0^{-+}, k_0^{--}, $k^{-+}(Q, d')$, $k^{--}(Q, d')$, b_0^-, $b^-(Q, d')$ to convert the problem into an integer problem.

We continue to add constraints to the problem (3.15) until the optimality gap is smaller than 0.005.

Values of the parameters θ_0, θ_1^+, θ_2^+, θ_1^-, θ_2^-, δ_1, δ_2, σ_1, σ_2 for two cases $c < l$ and $c \geq l$ are shown in the Table 3.1.

	$c < l$	$c \geq l$
θ_0	0.2	0.2
θ_1^+	0.9778	0.9778
θ_2^+	-0.0222	0.9778
θ_1^-	0.9778	1.9778
θ_1^-	0.9333	1.9778
δ_1	1.9333	1.9333
δ_2	1.9333	1.9333
σ_1	1.0222	0.0889
σ_2	1.0889	0.0889

Table 3.1: Parameters of the approximated value function

3.2.2 Approximate Dynamic Programming - based optimal policy

Now, to find the optimal issuing policy we need to solve the equation:

$$\min_{\forall X}\{C(S, X) + \lambda \sum_{S'} p(S' \mid S, X)\tilde{V}(S')\}, \tag{3.17}$$

where S is an initial state, X denotes an action, $C(S, X)$ is an immediate cost, Q is a vector of new arrivals, d' is a demand in a new period of time, λ is a discount factor.

We substitute $\tilde{V}(S')$ with our approximation function:

$$V(S) \cong \tilde{V}(S) = \theta_0 + \sum_{i=1}^{2} \theta_i^+ u_i^+ + \sum_{i=1}^{2} \theta_i^- u_i^- + \delta_1 d^+ + \delta_2 d^- + \sigma_1 [d^+ - u_2^+]^+ + \sigma_2 [d^- - u_2^-]^+ \tag{3.18}$$

and solve the linear programming problem to find the approximate optimal policy in each period of time:

minimize

$$\sum_i ic(x_i^{++} + x_i^{-+} + x_i^{--}) + l(d^+ - \sum_i x_i^{++} - \sum_i x_i^{-+}) \\ +l(d^- - \sum_i x_i^{--}) + \lambda \sum_{(Q,d')} Pr(Q,d')\tilde{V}(S') \tag{3.19}$$

subject to

$x_1^{++} \leq s_1^+$

$x_1^{-+} + x_1^{--} \leq s_1^-$

$x_2^{++} \leq s_2^+$

$x_2^{-+} + x_2^{--} \leq s_2^-$

$\sum_{i=1}^{2} x_i^{++} + \sum_{i=1}^{2} x_i^{-+} \leq d^+$

$\sum_{i=1}^{2} x_i^{--} \leq d^-$

$x_i, s_i \geq 0$, integer $\forall i$

$d^+, d^- \geq 0$, integer

In (3.19) we again have a part that is not linear, $\sigma_1[d^+ - u_2^+]^+ + \sigma_2[d^- - u_2^-]^+$. In order to linearise those terms, we use the same approach as in Phase I method and Column Generation method.

Chapter 4

Results

Three types of experiments were run:

1. The cost of using one unit of blood is lower than the cost of getting the unit of blood from the secondary source;
2. The cost of using one unit of blood is higher than the cost of getting the unit of blood from the secondary source;
3. Both costs are equal.

4.1 LP-based optimal policy

Let us first consider the results obtained by using the Linear Programming approach.

All the states S and possible actions X were enumerated. There are 144 possible states and 57 possible actions. The transition probabilities between states are all equal to 1/36. On every optimal policy graph x-axes represent the number of a state and y-axes represent the number of an action. On every graph of an optimal cost, y-axes represent the cost.

Figures 4.1 and 4.2 show the optimal cost and the optimal policy for in the case when penalty for using older blood c is smaller than the cost of using the blood from some secondary source l (a blood bank or another hospital).The c/l ratio in this case is 0.1 ($c = 10$,$l = 100$).

Numerical experiments were run for different values c and l but the same optimal policy was obtained for all cases where $0 < c/l < 1$.

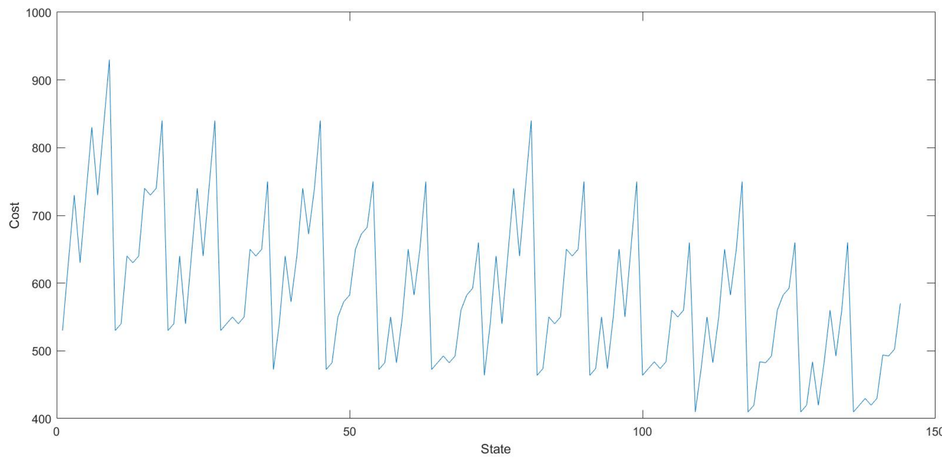

Figure 4.1: Optimal cost for the case $c < l$ ($c = 10$,$l = 100$)

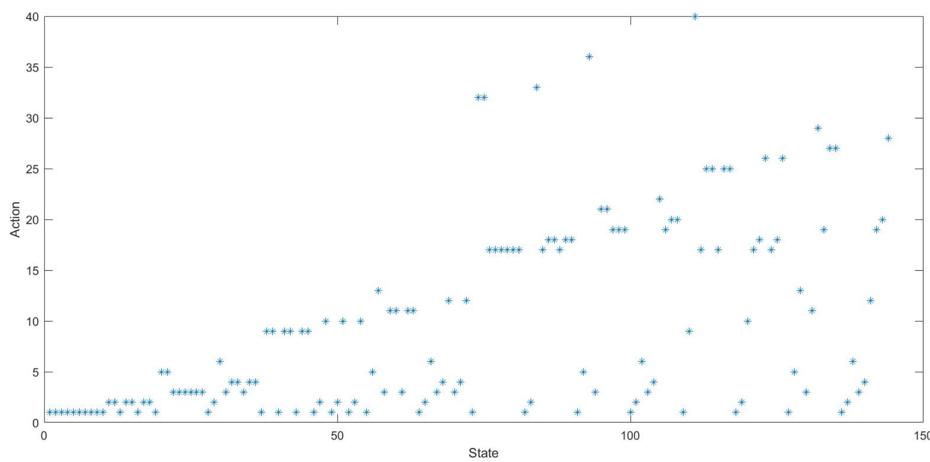

Figure 4.2: Optimal policy for the case $c < l$($c = 10$,$l = 100$)

On the figures 4.3 and 4.4 we can see the optimal cost and optimal policy for the second case (i.e., the cost of using one unit of blood c is higher than the cost of getting the unit of blood from the secondary source l). The c/l ratio in this case is equal to 10 ($c = 1000$,$l = 100$). The same result was obtained for the third case where the cost of using one unit of blood c is equal to the cost of getting the unit of blood from the secondary source l ($c = l = 100$).

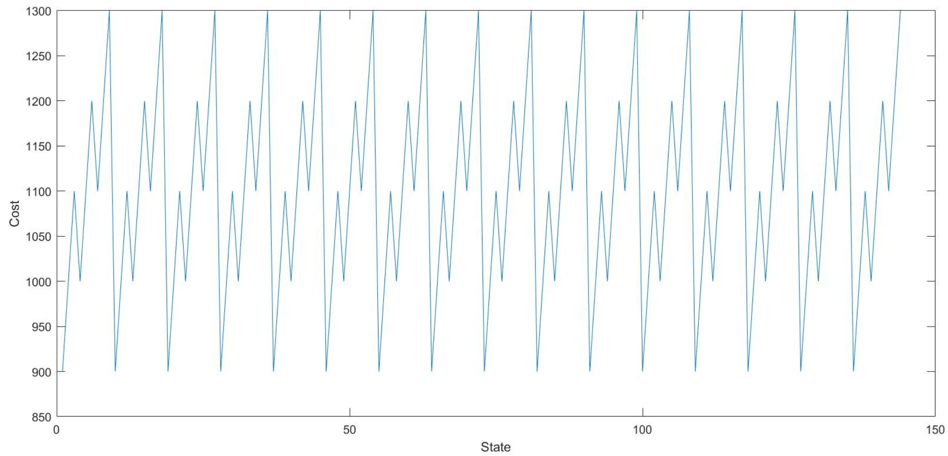

Figure 4.3: Optimal cost for the case $c \geq l(c = 1000, l = 100), (c = l = 100)$

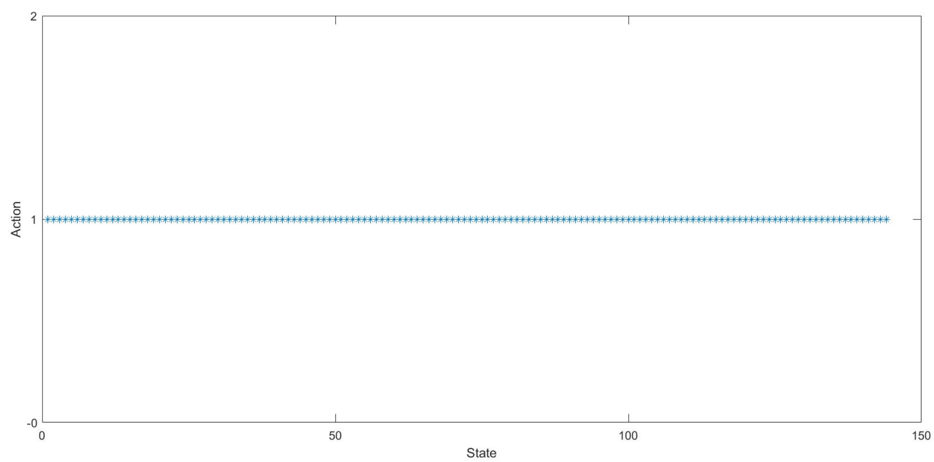

Figure 4.4: Optimal policy for the case $c \geq l(c = 1000, l = 100), (c = l = 100)$

4.2 ADP-based optimal policy

Finally, we show the results obtained by using Approximate Dynamical Programming approach.

As in the previous section (4.1), all the states S and possible actions X were enumerated. There are as well 144 possible states and 57 possible actions. On every optimal policy graph x-axes represent the number of a state and y-axes represent the number of an action.

Figures 4.5 and 4.6 show the approximate optimal policy and optimal cost in the case when the cost of using one unit of blood is lower than the cost of getting the unit of blood from the secondary source.

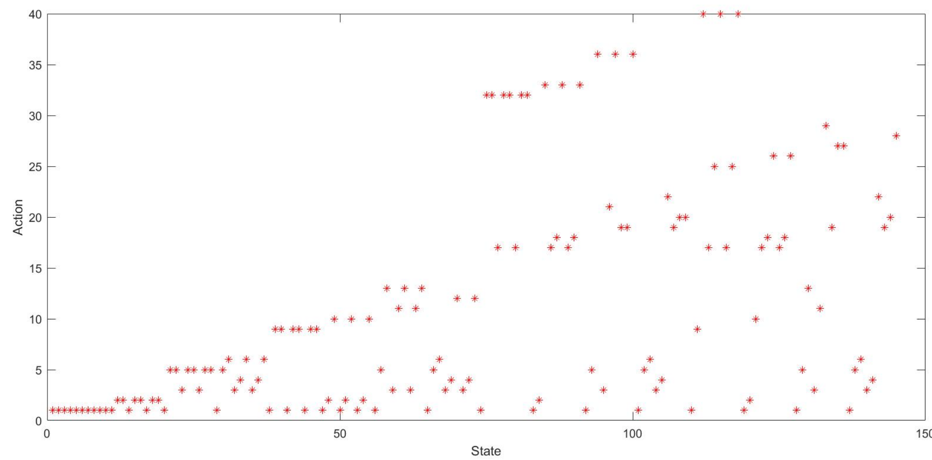

Figure 4.5: Approximate optimal policy for the case $c < l(c = 10, l = 100)$

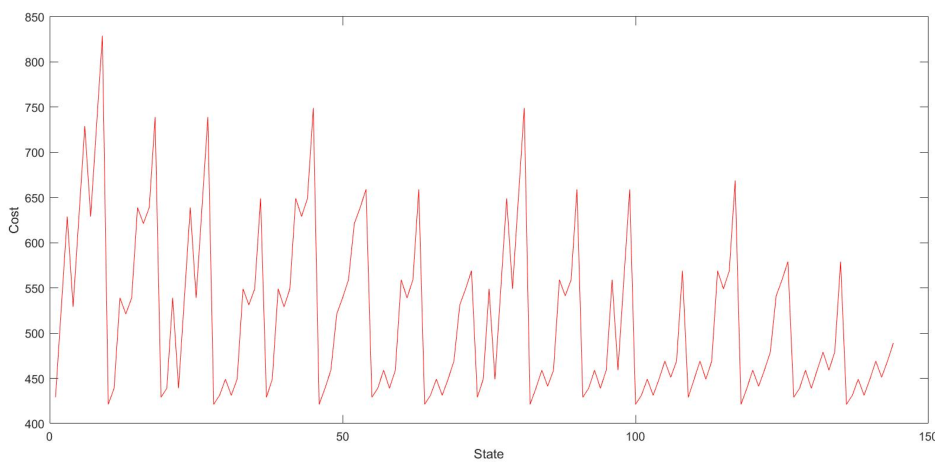

Figure 4.6: Optimal cost for the case $c < l(c = 10, l = 100)$

Figures 4.7 and 4.8 show the approximate optimal policy and optimal cost for the case when using one available unit of blood is more expensive than getting the unit of blood from the secondary source. Again, the same result is for the case when two costs are equal.

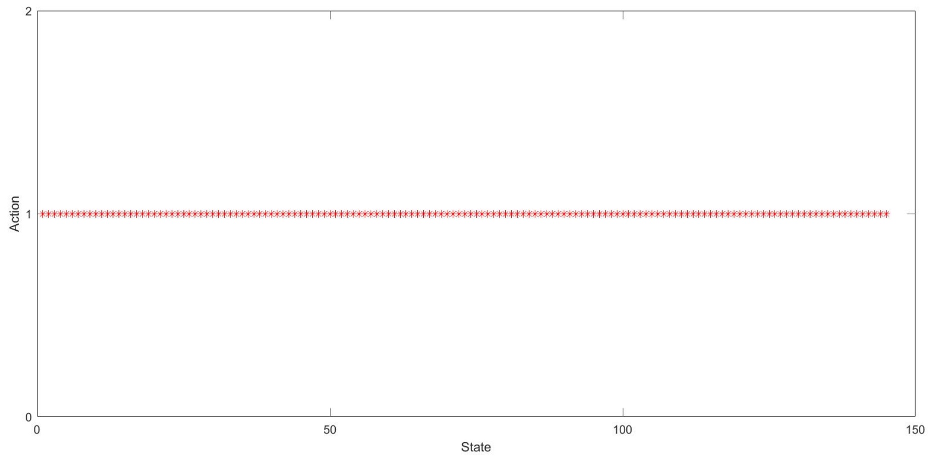

Figure 4.7: Approximate optimal policy for the case $c \geq l(c = 1000, l = 100), (c = l = 100)$

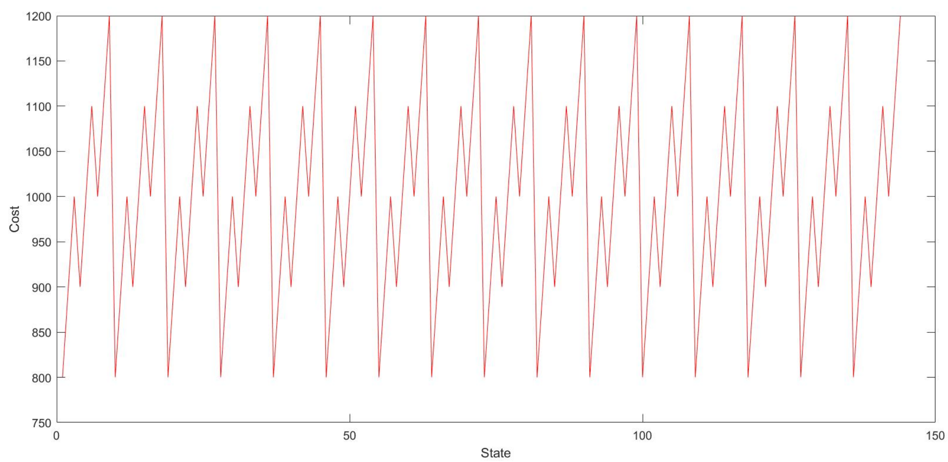

Figure 4.8: Optimal cost for the case $c \geq l(c = 1000, l = 100), (c = l = 100)$

4.3 Comparison of the results

In this section, we compare the results obtained by two approaches we considered. On each figure there are two graphs. The blue graph represents the result obtained by LP approach, and the red one shows the result for ADP.

So we can see that our LP-based and ADP-based optimal policies do not match for all states in the system. This is happening because in Approximate Dynamic Programming algorithm we use not the original value function but the approximated one. The difference is

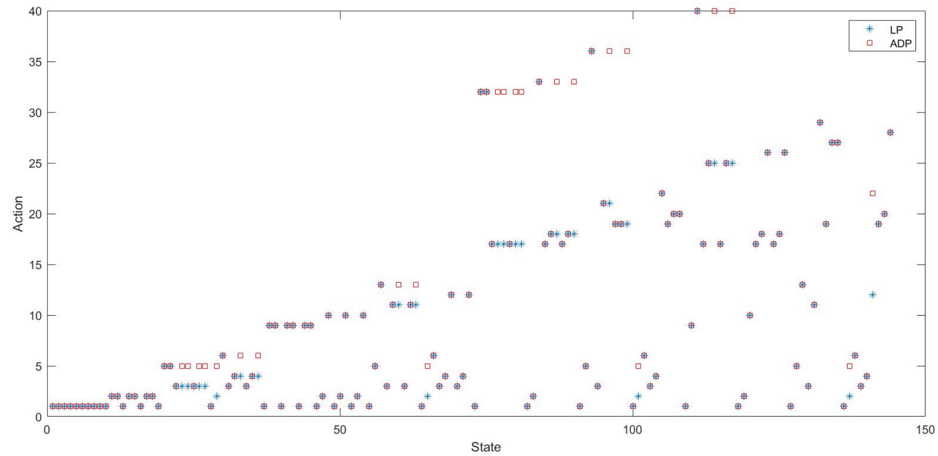

Figure 4.9: Comparison of the results for the case $c < l (c = 10, l = 100)$

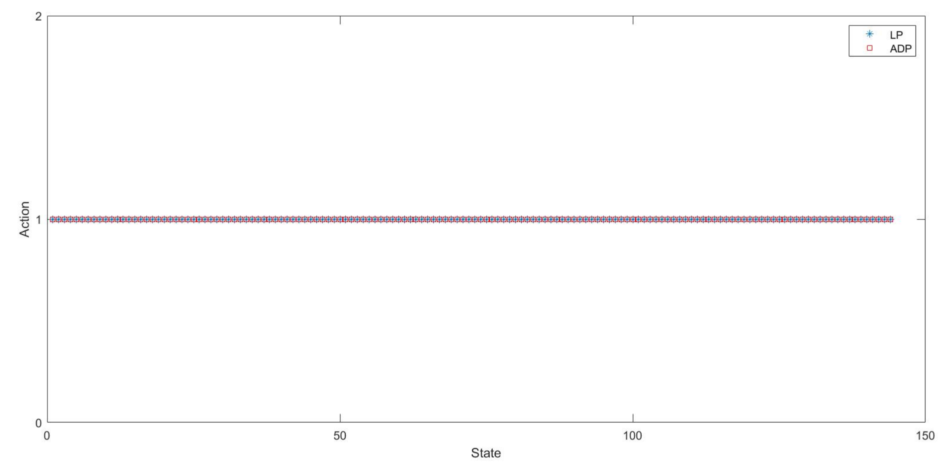

Figure 4.10: Comparison of the results for the case $c \geq l (c = 1000, l = 100), (c = l = 100)$

in which demand to satisfy first. ADP-based policy usually suggests to satisfy the demand of positive blood first. LP-based policy more often assigns the blood units equally for the demand of blood with positive and negative rhesus factor.

Both policies suggest to use younger blood first (i.e., to use LIFO policy) even if there is no younger blood with the same rhesus factor (i.e., it is suggested to use younger blood of the type AB- rather than older blood of the type AB+ for the patient with AB+ blood type). The fresher blood is suggested to use first in this small instance of the problem mostly because we face shortage only in 17% of the time so discarding an older blood does not effect the supply much.

Also we can observe the comparison of cost functions obtained by two methods. We can see that the value of the cost function that was calculated using LP is higher in both cases for all states.

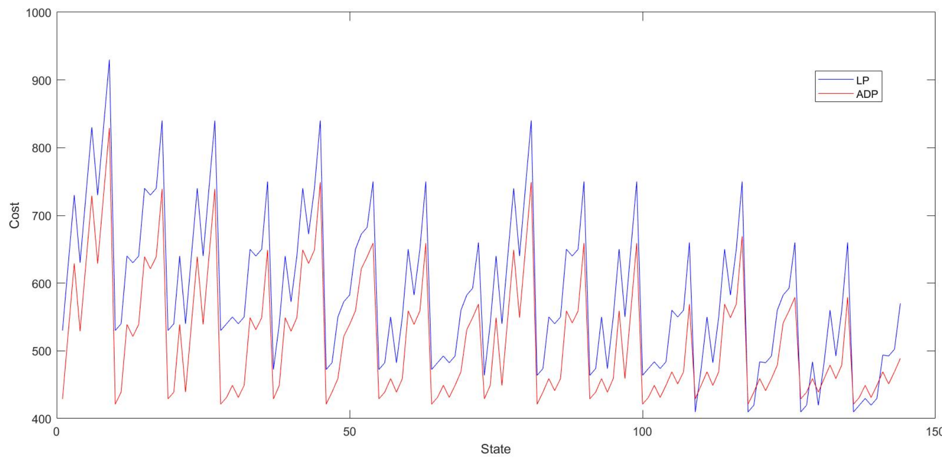

Figure 4.11: Comparison of the cost functions for the case $c < l(c = 10,l = 100)$

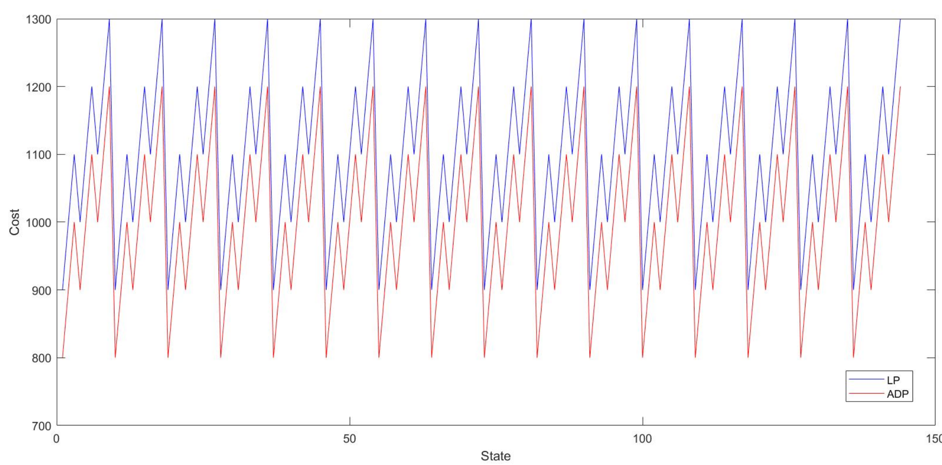

Figure 4.12: Comparison of the cost functions for the case $c \geq l(c = 1000,l = 100),(c = l = 100)$

On the following Figures 4.13 and 4.14 we can see the comparison of value functions as parameters c and l change.

The first graph 4.13 shows how the average cost over all states changes as c increase from 1 to 100, $l = 100$ remains constant. The average cost obtained by ADP method grows much slower. As ratio c/l is getting bigger the difference between two values obtained by different approaches grows as well.

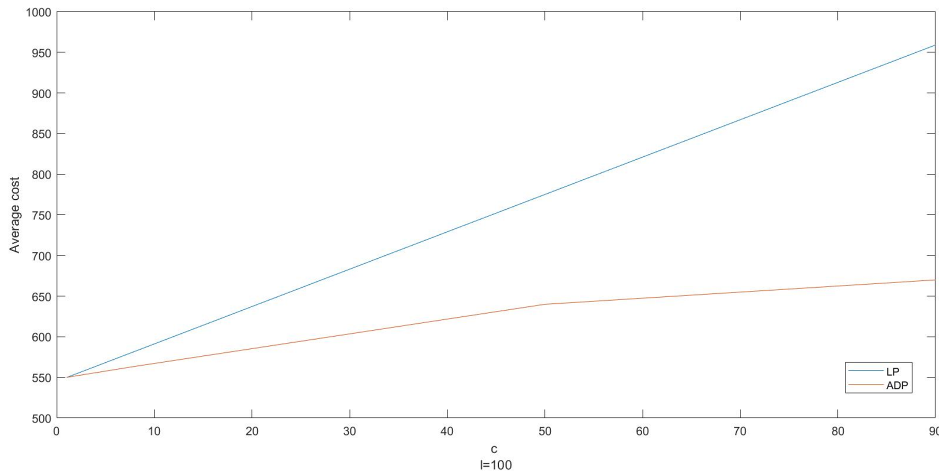

Figure 4.13: Comparison of the average cost functions for growing ratio $0 < c/l < 1$

Next Figure 4.14 shows the comparison of average values as c increases from 1 to 100 and l decreases from 100 to 1 simultaneously. We can see the same tendency for the part where $0 < c/l < 1$. After the point where $c = l$ average cost decreases for both methods as the policy to use the blood from the secondary source starts to work. And as the price for getting each additional unit of blood from the secondary source decreases the average cost is getting smaller respectively.

All the results presented in this thesis are the results of the reduced problem. In a full-size problem where we consider the blood units of any age up to 42 days and all the blood types the supply vector is 336 long, the vector of an action is 1,134 long, vector of new arrivals is 336 long that make the problem impossible to solve using Linear Programming. Also the computational time for using Approximate Dynamic Programming increases significantly in comparison to 14.5 seconds computational time for reduced problem.

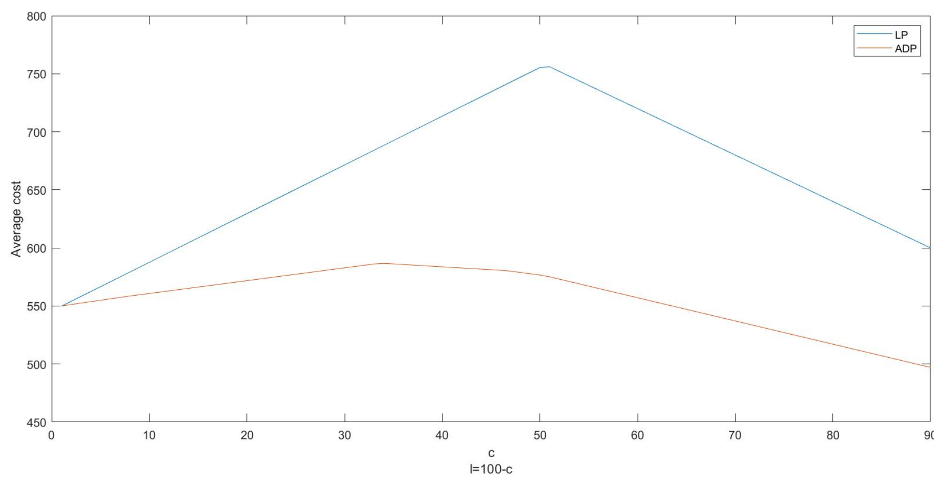

Figure 4.14: Comparison of the average cost functions for growing ratio $0 < c/l < 10$

Chapter 5

Conclusions

In this work, we studied the problem of finding the optimal policy for using blood units for transfusion. The main goal was to reduce shortage and the number of blood units that should be discarded (i.e., the number of outdated units of blood). Also we tried to reduce the risk of complication by assigning a penalty for using older blood. We used two different approaches to find the solution and compared the obtained results.

In the first part we used the Linear Programming (LP) approach to find the optimal policy. As far as the size of the problem was very big, in order to be able to solve the problem using LP, we considered only two blood types and assumed that blood can be stored only for 2 days.

In the second part of the work, in order to find the solution to the problem we used Approximate Dynamic Programming techniques. First we approximated the value function, and then we used the Phase I method and column generation to solve the linear programming form of dynamic programming.

We ran three types of experiments: for the penalty cost of using older blood bigger than using the unit of blood from the secondary source (a blood bank or another hospital), for the penalty cost of using older blood smaller than using the unit of blood from the secondary source, and for two costs being equal. So we saw that the approximated optimal policy obtained by ADP approach and the one we obtained by using LP are similar for some states.

The main goal for future is to expand the problem for all blood types and consider the full period of possible blood storing of 42 days. It is highly likely that the dependence between the age of blood and complications that appear after transfusion is not linear. So there is a need to study that dependence and find a function that describes it.

Bibliography

[1] H. Abouee-Mehrizi, O.Baron, O.Berman, V. Sarhangian, *Allocation Policies in Blood Transfusion*. University of Waterloo.

[2] M.P. Atkinson, M.J. Fontaine, L.T. Goodnough, *A novel allocation strategy for blood transfusions: investigating the tradeoff between the age and availability of transfused blood*. Transfusion. 2012.

[3] D.P. De Farias and B. Van Roy, *The Linear programming Approach to Approximate Dynamical Programming*,Operation Research, Vol.51, No. 6, pp.850-865, 2003.

[4] J. Desrosiers, M.E. Lubbecke, M.M. Solomon, *Column Generation*. Springer, 2005.

[5] W.H. Dzik, N. Beckman, M.F. Murphy, M. Delaney, *Factors affecting red blood cell storage age at the time of transfusion*. Transfusion. 2013.

[6] M. Ehrgott, J. Tind, *Column Generation in Integer Programming with Applications in Multicriteria Optimization*. University of Auckland, University of Copenhagen, 2007.

[7] J.W. Eikelboom, R.J. Cook, Y. Liu, N.M. Heddle, *Duration of red cell storage before transfusion and in-hospital mortality*. American Heart Journal, 2010.

[8] R.Haijema, J. van der Wal, N.M van Dijk, *Blood platelet production: Optimization byy dynamic programming and simulation*,Computer and Operation Research, Vol.34, pp.760-779, 2007.

[9] I.Z. Karaesmen, A. Scheller-Wolf, B. Deniz, *Managing perishable and ageing inventories: review and future research directions. Planning production and inventories in the extended enterprise*. pp.393-436, Springer, 2011.

[10] C.G. Koch, L. Li, D.I. Sessler, P. Figueroa, G.A. Hoeltge, T. Mihaljevic, E.H. Blackstone, *Duration of Red-Cell Storage and Complications after Cardiac Surgery*. The New England Journal of Medicine, 312, 2008.

[11] M.S.Maxwell, M.Restrepo, S.G. Henderson, H.Topaloglu, *Approximate Dynamic Programming for Ambulance Redeployment*. Informs Journal on Computing, Vol. 22, No.2, pp.266-281, 2010.

[12] S. Nahmias, *Perishable Inventory Systems*. Vol. 160, Springer, 2011.

[13] S. Nahmias, *Perishable inventory: A review*. Operation research, Vol. 30(4), pp.680-708, 1982.

[14] S. Nahmias, W.P. Pierskalla, *Optimal ordering policies for a product that perishes in two periods subject to stochastic demand*. Naval Research Logistics Quarterly, 1973.

[15] P.J. Offner, E.E. Moore, W.L. Biffl, J.L. Johnson, *Increased Rate of Infection Associated With Transfusion of Old Blood After Severe Injury*. Arch Surg. 2002.

[16] P.Q. Pan, *Linear Programming Computation*. Springer, 2014.

[17] W.P. Pierskalla, C.D. Roach, *Optimal issuing policies for perishable inventory*. Management Science, Vol. 18, pp.603-614, 1972.

[18] M. L. Puterman, *Markov Decision Processes: discrete stochastic dynamic programming*. Wiley-Interscience, New York, 1994.

[19] A. Sabouri, *Applications of Stochastic Optimization Models in Patient Screening and Blood Inventory Management*. PhD Thesis, University of British Columbia, Vancouver, 2014.

[20] M. Sayers, J. Centilli, *What if shelf life becomes a consideration in ordering red blood cells?* Transfusion. 2012.

[21] P.J.Schweitzer and A.Seidmann, *Generalized polynomial approximations in markovian decision processes*. Journal of mathematical analysis and applications, 110(2):568-582, 1985.

[22] Warren B. Powell, *Approximate Dynamical Programming: Solving the Curse of Dimensionality*,Wiley, 2011.

[23] G. Zallen, P.J. Offner, E.E. Moore, J. Blackwell, D.J. Ciesla, J. Gabriel, C. Denny C.C. Silliman, *Age of Transfused Blood is an Independent Risk Factor for Postinjury Multiple Organ Failure*,The American Journal of Surgery, Vol.178, 1999.

[24] D. Zhang, D. Adelman, *An Approximate Dynamic Programming Approach to Network Revenue Management with Customer Choice*, Transportation Science, Vol. 43, No.3, pp.381-394, 2009.

Appendix A

Matlab code for LP approach

```
n = (0:15)';
S = de2bi(n); % matrix of all the possible supply vectors

n=(0:63)';
X= de2bi(n); % matrix of all the possible actions
A=sum(X,2);
%---------------------- deleting all the actions with  the sum >4
        for k=1:59
             if A(k)>4
                  X(k,:)=[];
                  A=sum(X,2);
                  k=k-1;
             end
        end
        X(58,:)=[];

X; % matrix where each row is a possible action. total number of
```

```
    actions in the system is 57
 %------------------------
[x, y] = ndgrid ([0, 1, 2]);
D=[x(:), y(:)]; % matrix of all the possible demands

%----------------------creating matrix of all states
ST=zeros(144,7);
n=1;
for i=1:16
    for j=1:9
     T= [n S(i,:) D(j,:)];
     ST(n,:)=T(:);
     n=n+1;
    end
end
ST;
%--------------------
%--------------------checking every action if it satisfies the
    constraints and
%if yes, calculate the cost
c=1000;
l=100;
n=1;
C=zeros(1,3);
for j=1:57
  for i=1:144
    if X(j,1)<=ST(i,2)
```

```
            if X(j,4)<=ST(i,4)
                 if (X(j,2)+X(j,3))<=ST(i,3)
                     if (X(j,5)+X(j,6))<=ST(i,5)
                         if X(j,1)+X(j,3)+X(j,4)+X(j,6)<=ST(i,6)
                             if (X(j,2)+X(j,5))<=ST(i,7)
                                 cost=c*(X(j,1)+X(j,2)+X(j,3))+c*(X(j
                                    ,4)+X(j,5)+X(j,6))+l*((ST(i,6)-X(j
                                    ,1)-X(j,3)-X(j,4)-X(j,6))+(ST(i,7)
                                    -X(j,2)-X(j,5)));
                                 C1=[i j cost];
                                 C(n,:)=C1(:);
                                 n=n+1;
                             end
                         end
                     end
                 end
            end
        end
    end
end
C; %matrix of state-action-cost
%-------------------------------
%------------------------------Creating a transition matrix
F=zeros(645,144);
F1=zeros(645,144);
for i=1:645
    s=C(i,1);
```

```
        a=C(i,2);
        d1=ST(s,4)-X(a,4);
        d2=ST(s,5)-X(a,5)-X(a,6);
         for j=1:144
                 if ST(j,4)==d1
                         if ST(j,5)==d2
                                F(i,j)=(-1/36)*0.8;
                                F1(i,j)=1/36;
                         end
                 end
         end
         F(i,s)=F(i,s)+1;
end
F;
% filename='matixF.xlsx';
% xlswrite(filename,F,1,'A1');
%--------------------------------------------------------------
%-------------------------solving LP to find optimal cost
n=144;
cvx_begin
        variable v(n)
        maximize sum(v)
        subject to
              F*v<=C(:,3);
cvx_end
v;
plot(v);
```

```
%------------------------

%%------------------------ solving Belman equation to find an optimal policy
d=zeros(144,1);
for i=1:144
    z=99999999;
    for j=1:645
        if C(j,1)==i
            c=C(j,3)+F1(j,:)*v;
            if c<z
                z=c;
                d(i)=C(j,2);
            end
        end
    end
end
d;
plot(d, '*')
hold on
plot (g(2:end,2),'squarer')
```

Appendix B

Matlab code for ADP approach

```
n = (0:15)';
S = de2bi(n); % matrix of all the possible supply vectors

n=(0:63)';
X= de2bi(n); % matrix of all the possible actions
A=sum(X,2);
%---------------------------- deleting all the actions with  the sum >4
        for k=1:59
            if A(k)>4
                X(k,:)=[];
                A=sum(X,2);
                k=k-1;
            end
        end
        X(58,:)=[];

X; % matrix where each row is a possible action. total number of
```

```
    actions in the system is 57
 %--------------------------------
[x, y] = ndgrid ([0, 1, 2]);
D=[x(:), y(:)]; % matrix of all the possible demands

%------------------------------creating matrix of all states
ST=zeros(144,7);
n=1;
for i=1:16
      for j=1:9
       T= [n S(i,:) D(j,:)];
       ST(n,:)=T(:);
       n=n+1;
      end
end
ST;
%--------------------------
%--------------------------checking every action if it satisfies the
    constraints and
%if yes, calculate the cost
c=1000;
l=100;
n=1;
C=zeros(1,3);
for j=1:57
   for i=1:144
      if X(j,1)<=ST(i,2)
```

```
            if X(j,4)<=ST(i,4)
                if (X(j,2)+X(j,3))<=ST(i,3)
                    if (X(j,5)+X(j,6))<=ST(i,5)
                        if X(j,1)+X(j,3)+X(j,4)+X(j,6)<=ST(i,6)
                            if (X(j,2)+X(j,5))<=ST(i,7)
                                cost=c*(X(j,1)+X(j,2)+X(j,3))+c*(X(j
                                    ,4)+X(j,5)+X(j,6))+l*((ST(i,6)-X(j
                                    ,1)-X(j,3)-X(j,4)-X(j,6))+(ST(i,7)
                                    -X(j,2)-X(j,5)));
                                C1=[i j cost];
                                C(n,:)=C1(:);
                                n=n+1;
                            end
                        end
                    end
                end
            end
        end
    end
end
C; %matrix of state-action-cost
%------------------------------
%------------------------------Creating a transition matrix
F=zeros(645,144);
F1=zeros(645,144);
for i=1:645
    s=C(i,1);
```

```
        a=C(i,2);
        d1=ST(s,4)-X(a,4);
        d2=ST(s,5)-X(a,5)-X(a,6);
         for j=1:144
                if ST(j,4)==d1
                    if ST(j,5)==d2
                        F(i,j)=(-1/36)*0.8;
                        F1(i,j)=1/36;
                    end
                end
         end
         F(i,s)=F(i,s)+1;
end
F;
%---forming the right hand side of the original dual problem
b=zeros(9,1);
b(1,1)=1;
for i=1:144
        b(2,1)=b(2,1)+ST(i,2)*(1/144);
        b(3,1)=b(3,1)+ST(i,3)*(1/144);
        b(4,1)=b(4,1)+ST(i,2)*(1/144)+ST(i,4)*(1/144);
        b(5,1)=b(5,1)+ST(i,3)*(1/144)+ST(i,5)*(1/144);
        b(6,1)=b(6,1)+ST(i,6)*(1/144);
        b(7,1)=b(7,1)+ST(i,7)*(1/144);

        d1=ST(i,6)-ST(i,2)-ST(i,4);
        if d1<0
```

```
            d1=0;
        end
        b(8,1)=b(8,1)+d1*(1/144);

        d2=ST(i,7)-ST(i,3)-ST(i,3);
          if d2<0
            d2=0;
          end
        b(9,1)=b(8,1)+d2*(1/144);
end
%------------------------------------------------------------

%---------------solving the sub-problem of the Phase 1 method

I=eye(9);
actions=[0; 0; 0; 0];
D1=[0; 1; 2; 0; 1; 2];
Q1=[0; 0; 0; 1; 1; 1];

n=1;

                cvx_begin
                cvx_solver gurobi;
                variables y(9);
                dual variables t; %thetas, deltas, gammas
                minimize sum(y);
                subject to
```

```
      t: I*y==b;
        y>=0;

    cvx_end

N=10;

   cvx_begin
     cvx_solver gurobi;
     integer variables x(6) s(6) k11 K121 K122 K123 K124 K125
        K126 K221 K222 K223 K224 K225 K226 K111 K112 K113 K114
         K115 K116 k21 K211 K212 K213 K214 K215 K216 k12 k22;
         %s,x - state-action
     binary variables b01 b02 b11 b12 b13 b14 b15 b16 b21 b22
        b23 b24 b25 b26;
     maximize ((1-0.8)*t(1)+t(2)*(s(1)-0.8*1/12)+t(3)*(s(2)
        -0.8*1/12)+t(4)*(s(1)+s(3)-0.8*(1\6)*(s(1)-x(1))
        +0.8*(1/12))+t(5)*(s(2)+s(4)-0.8*(1/12+(1/6)*(s(2)-x
        (2)-x(3))))+t(6)*(s(5)-0.8*1/6)+t(7)*(s(6)-0.8*1/6)+t
        (8)*(k11-0.8*(K111/36+K112/36+K113/36+K114/36+K115/36+
        K116/36))+t(9)*(k21-0.8*(K211/36+K212/36+K213/36+K214
        /36+K215/36+K216/36)));

     subject to
     x(1)<=s(1);
     x(2)+x(3)<=s(2);
     x(4)<=s(3);
```

```
x(5)+x(6)<=s(4);
x(1)+x(2)+x(4)+x(5)<=s(5);
x(3)+x(6)<=s(6);
x>=0;
s>=0;
k11+k12-N==s(5)-s(1)-s(3);
k21+k22-N==s(6)-s(2)-s(4);
k11<=N*b01;
k21<=N*b02;
k12>=N*b01;
k22>=N*b02;
k12<=N;
k22<=N;
k11>=0;
k12>=0;
k21>=0;
k22>=0;

K111+K121-N==s(1)-x(1);
K112+K122-N==1+s(1)-x(1);
K113+K123-N==2+s(1)-x(1);
K114+K124-N==s(1)-x(1)-1;
K115+K125-N==s(1)-x(1);
K116+K126-N==1+s(1)-x(1);

K211+K221-N==s(2)-x(2)-x(3);
K212+K222-N==1+s(2)-x(2)-x(3);
```

```
K213+K223-N==2+s(2)-x(2)-x(3);
K214+K224-N==s(2)-x(2)-x(3)-1;
K215+K225-N==s(2)-x(2)-x(3);
K216+K226-N==1+s(2)-x(2)-x(3);

K111<=N*b11;
K112<=N*b12;
K113<=N*b13;
K114<=N*b14;
K115<=N*b15;
K116<=N*b16;

K211<=N*b21;
K212<=N*b22;
K213<=N*b23;
K214<=N*b24;
K215<=N*b25;
K216<=N*b26;

K121>=N*b11;
K122>=N*b12;
K123>=N*b13;
K124>=N*b14;
K125>=N*b15;
K126>=N*b16;

K221>=N*b21;
```

```
K222>=N*b22 ;
K223>=N*b23 ;
K224>=N*b24 ;
K225>=N*b25 ;
K226>=N*b26 ;

K121<=N;
K122<=N;
K123<=N;
K124<=N;
K125<=N;
K126<=N;

K221<=N;
K222<=N;
K223<=N;
K224<=N;
K225<=N;
K226<=N;

K111>=0;
K112>=0;
K113>=0;
K114>=0;
K115>=0;
K116>=0;

```

```
K211>=0;
K212>=0;
K213>=0;
K214>=0;
K215>=0;
K216>=0;

K121>=0;
K122>=0;
K123>=0;
K124>=0;
K125>=0;
K126>=0;

K221>=0;
K222>=0;
K223>=0;
K224>=0;
K225>=0;
K226>=0;
 0<=s(5)<=2;
 0<=s(6)<=2;
 0<=s(1)<=1;
 0<=s(2)<=1;
 0<=s(3)<=1;
 0<=s(4)<=1;

```

```
                0<=x(1)<=1;
                0<=x(2)<=1;
                0<=x(3)<=1;
                0<=x(4)<=1;
                0<=x(5)<=1;
                0<=x(6)<=1;
            cvx_end
    actions=[actions; s];
O1=s(1)-0.8*(1/36);
O2=s(2)-0.8*(1/36);
O3=s(1)+s(3)-0.8*((1/36)*(s(3)-x(4))+(1/36)*(1+s(3)-x(4)));
O4=s(2)+s(4)-0.8*((1/36)*(s(4)-x(5)-x(6))+(1/36)*(1+s(4)-x(5)-x(6)
    ));
D1=s(5)-0.8*1/12;
D2=s(6)-0.8*1/12;

e1=s(5)-s(1)-s(3);
if e1<0
    e1=0;
end
e2=s(6)-s(2)-s(4);
if e2<0
    e2=0;
end
e11=0-(s(1)-x(1));
if e11<0
    e11=0;
```

```
end
e12=1-(s(1)-x(1));
if e12<0
    e12=0;
end
e13=2-(s(1)-x(1));
if e13<0
    e13=0;
end
e14=0-(1+s(1)-x(1));
if e14<0
    e14=0;
end
e15=1-(1+s(1)-x(1));
if e15<0
    e15=0;
end
e16=2-(1+s(1)-x(1));
if e16<0
    e16=0;
end

e21=0-(s(2)-x(2)-x(3));
if e21<0
    e21=0;
end
e22=1-(s(2)-x(2)-x(3));
```

```
if e22<0
    e22=0;
end
e23=2-(s(2)-x(2)-x(3));
if e23<0
    e23=0;
end
e24=0-(1+s(2)-x(2)-x(3));
if e24<0
    e24=0;
end
e25=1-(1+s(2)-x(2)-x(3));
if e25<0
    e25=0;
end
e26=2-(1+s(2)-x(2)-x(3));
if e26<0
    e26=0;
end
E1=e1+0.8*(e11*1/36+e12*1/36+e13*1/36+e14*1/36+e15*1/36+e16*1/36);
E2=e2+0.8*(e21*1/36+e22*1/36+e23*1/36+e24*1/36+e25*1/36+e26*1/36);

a=[0.2;O1;O2;O3;O4;D1;D2;E1;E2];
A=[a];

Y=sum(y);
B=[s',x'];
```

```
while n<=12
        cvx_begin
        cvx_solver gurobi;
        variables y(9) w(n);
        dual variables t; %thetas, deltas, gammas
        minimize sum(y);
        subject to
         t: A*w+I*y==b;
           y>=0;
            w>=0;
       cvx_end

    N=10;

      cvx_begin
        cvx_solver gurobi;
        integer variables x(6) s(6) k11 K121 K122 K123 K124 K125
            K126 K221 K222 K223 K224 K225 K226 K111 K112 K113 K114
             K115 K116 k21 K211 K212 K213 K214 K215 K216 k12 k22;
             %s,x - state-action
        binary variables b01 b02 b11 b12 b13 b14 b15 b16 b21 b22
            b23 b24 b25 b26;
        maximize ((1-0.8)*t(1)+t(2)*(s(1)-0.8*1/12)+t(3)*(s(2)
            -0.8*1/12)+t(4)*(s(1)+s(3)-0.8*(1\6)*(s(1)-x(1))
            +0.8*(1/12))+t(5)*(s(2)+s(4)-0.8*(1/12+(1/6)*(s(2)-x
            (2)-x(3))))+t(6)*(s(5)-0.8*1/6)+t(7)*(s(6)-0.8*1/6)+t
```

```
(8)*(k11-0.8*(K111/36+K112/36+K113/36+K114/36+K115/36+
K116/36))+t(9)*(k21-0.8*(K211/36+K212/36+K213/36+K214
/36+K215/36+K216/36)));

    subject to
    x(1)<=s(1);
    x(2)+x(3)<=s(2);
    x(4)<=s(3);
    x(5)+x(6)<=s(4);
    x(1)+x(2)+x(4)+x(5)<=s(5);
    x(3)+x(6)<=s(6);
    x>=0;
    s>=0;
    k11+k12-N==s(5)-s(1)-s(3);
    k21+k22-N==s(6)-s(2)-s(4);
    k11<=N*b01;
    k21<=N*b02;
    k12>=N*b01;
    k22>=N*b02;
    k12<=N;
    k22<=N;
    k11>=0;
    k12>=0;
    k21>=0;
    k22>=0;

    K111+K121-N==s(1)-x(1);
```

```
K112+K122-N==1+s(1)-x(1);
K113+K123-N==2+s(1)-x(1);
K114+K124-N==s(1)-x(1)-1;
K115+K125-N==s(1)-x(1);
K116+K126-N==1+s(1)-x(1);

K211+K221-N==s(2)-x(2)-x(3);
K212+K222-N==1+s(2)-x(2)-x(3);
K213+K223-N==2+s(2)-x(2)-x(3);
K214+K224-N==s(2)-x(2)-x(3)-1;
K215+K225-N==s(2)-x(2)-x(3);
K216+K226-N==1+s(2)-x(2)-x(3);

K111<=N*b11;
K112<=N*b12;
K113<=N*b13;
K114<=N*b14;
K115<=N*b15;
K116<=N*b16;

K211<=N*b21;
K212<=N*b22;
K213<=N*b23;
K214<=N*b24;
K215<=N*b25;
K216<=N*b26;
```

```
K121>=N*b11 ;
K122>=N*b12 ;
K123>=N*b13 ;
K124>=N*b14 ;
K125>=N*b15 ;
K126>=N*b16 ;

K221>=N*b21 ;
K222>=N*b22 ;
K223>=N*b23 ;
K224>=N*b24 ;
K225>=N*b25 ;
K226>=N*b26 ;

K121<=N;
K122<=N;
K123<=N;
K124<=N;
K125<=N;
K126<=N;

K221<=N;
K222<=N;
K223<=N;
K224<=N;
K225<=N;
K226<=N;
```

```
K111>=0;
K112>=0;
K113>=0;
K114>=0;
K115>=0;
K116>=0;

K211>=0;
K212>=0;
K213>=0;
K214>=0;
K215>=0;
K216>=0;

K121>=0;
K122>=0;
K123>=0;
K124>=0;
K125>=0;
K126>=0;

K221>=0;
K222>=0;
K223>=0;
K224>=0;
K225>=0;
```

```
            K226>=0;
             0<=s(5)<=2;
             0<=s(6)<=2;
             0<=s(1)<=1;
             0<=s(2)<=1;
             0<=s(3)<=1;
             0<=s(4)<=1;

             0<=x(1)<=1;
             0<=x(2)<=1;
             0<=x(3)<=1;
             0<=x(4)<=1;
             0<=x(5)<=1;
             0<=x(6)<=1;
            cvx_end
        actions=[actions;s];
O1=s(1)-0.8*(1/36);
O2=s(2)-0.8*(1/36);
O3=s(1)+s(3)-0.8*((1/36)*(s(3)-x(4))+(1/36)*(1+s(3)-x(4)));
O4=s(2)+s(4)-0.8*((1/36)*(s(4)-x(5)-x(6))+(1/36)*(1+s(4)-x(5)-x(6)
    ));
D1=s(5)-0.8*1/12;
D2=s(6)-0.8*1/12;

g=[s' x'];
B=[B;g];

```

```
e1=s(5)-s(1)-s(3);
if e1<0
    e1=0;
end
e2=s(6)-s(2)-s(4);
if e2<0
    e2=0;
end
e11=0-(s(1)-x(1));
if e11<0
    e11=0;
end
e12=1-(s(1)-x(1));
if e12<0
    e12=0;
end
e13=2-(s(1)-x(1));
if e13<0
    e13=0;
end
e14=0-(1+s(1)-x(1));
if e14<0
    e14=0;
end
e15=1-(1+s(1)-x(1));
if e15<0
    e15=0;
```

```
end
e16=2-(1+s(1)-x(1));
if e16<0
        e16=0;
end

e21=0-(s(2)-x(2)-x(3));
if e21<0
        e21=0;
end
e22=1-(s(2)-x(2)-x(3));
if e22<0
        e22=0;
end
e23=2-(s(2)-x(2)-x(3));
if e23<0
        e23=0;
end
e24=0-(1+s(2)-x(2)-x(3));
if e24<0
        e24=0;
end
e25=1-(1+s(2)-x(2)-x(3));
if e25<0
        e25=0;
end
e26=2-(1+s(2)-x(2)-x(3));
```

```
if e26<0
     e26=0;
end
E1=e1+0.8*(e11*1/36+e12*1/36+e13*1/36+e14*1/36+e15*1/36+e16*1/36);
E2=e2+0.8*(e21*1/36+e22*1/36+e23*1/36+e24*1/36+e25*1/36+e26*1/36);

a=[0.2;O1;O2;O3;O4;D1;D2;E1;E2];
A=[A,a];
n=n+1;
%Y=y(1)+y(2)+y(3)+y(4)+y(5)+y(6)+y(7)+y(8)+y(9);
 end
 %B = B(2:end,:);

 for i=1:13
 cost(i)=c*(B(i,7)+B(i,8)+B(i,9)+2*B(i,7)+2*B(i,7)+2*B(i,7))+l*(B(
     i,5)-B(i,7)-B(i,8)-B(i,10)-B(i,11)+B(i,6)-B(i,9)-B(i,12));
 end
 n=13;
 j=2;
 Opt=[0 100];
 cr=1;
 while cr>0.0005
 cvx_begin
           cvx_solver gurobi;
           variables w(n);
           dual variables t; %thetas, deltas, gammas
           minimize sum(cost*w);
```

```
            subject to
             t: A*w==b;
                w>=0;
cvx_end

 cvx_begin
            cvx_solver gurobi;
            integer variables x(6) s(6) k11 K121 K122 K123 K124 K125
                K126 K221 K222 K223 K224 K225 K226 K111 K112 K113 K114
                 K115 K116 k21 K211 K212 K213 K214 K215 K216 k12 k22;
                 %s,x - state-action
            binary variables b01 b02 b11 b12 b13 b14 b15 b16 b21 b22
                b23 b24 b25 b26;
            maximize ((1-0.8)*t(1)+t(2)*(s(1)-0.8*1/12)+t(3)*(s(2)
                -0.8*1/12)+t(4)*(s(1)+s(3)-0.8*(1\6)*(s(1)-x(1))
                +0.8*(1/12))+t(5)*(s(2)+s(4)-0.8*(1/12+(1/6)*(s(2)-x
                (2)-x(3))))+t(6)*(s(5)-0.8*1/6)+t(7)*(s(6)-0.8*1/6)+t
                (8)*(k11-0.8*(K111/36+K112/36+K113/36+K114/36+K115/36+
                K116/36))+t(9)*(k21-0.8*(K211/36+K212/36+K213/36+K214
                /36+K215/36+K216/36))+(c*(x(1)+x(2)+x(3)+2*x(4)+2*x(5)
                +2*x(6))+l*(s(5)-x(1)-x(2)-x(4)-x(5)+s(6)-x(3)-x(6))))
                ;

            subject to
            x(1)<=s(1);
            x(2)+x(3)<=s(2);
            x(4)<=s(3);
```

```
x(5)+x(6)<=s(4);
x(1)+x(2)+x(4)+x(5)<=s(5);
x(3)+x(6)<=s(6);
x>=0;
s>=0;
k11+k12-N==s(5)-s(1)-s(3);
k21+k22-N==s(6)-s(2)-s(4);
k11<=N*b01;
k21<=N*b02;
k12>=N*b01;
k22>=N*b02;
k12<=N;
k22<=N;
k11>=0;
k12>=0;
k21>=0;
k22>=0;

K111+K121-N==s(1)-x(1);
K112+K122-N==1+s(1)-x(1);
K113+K123-N==2+s(1)-x(1);
K114+K124-N==s(1)-x(1)-1;
K115+K125-N==s(1)-x(1);
K116+K126-N==1+s(1)-x(1);

K211+K221-N==s(2)-x(2)-x(3);
K212+K222-N==1+s(2)-x(2)-x(3);
```

```
K213+K223-N==2+s(2)-x(2)-x(3);
K214+K224-N==s(2)-x(2)-x(3)-1;
K215+K225-N==s(2)-x(2)-x(3);
K216+K226-N==1+s(2)-x(2)-x(3);

K111<=N*b11;
K112<=N*b12;
K113<=N*b13;
K114<=N*b14;
K115<=N*b15;
K116<=N*b16;

K211<=N*b21;
K212<=N*b22;
K213<=N*b23;
K214<=N*b24;
K215<=N*b25;
K216<=N*b26;

K121>=N*b11;
K122>=N*b12;
K123>=N*b13;
K124>=N*b14;
K125>=N*b15;
K126>=N*b16;

K221>=N*b21;
```

```
K222>=N*b22;
K223>=N*b23;
K224>=N*b24;
K225>=N*b25;
K226>=N*b26;

K121<=N;
K122<=N;
K123<=N;
K124<=N;
K125<=N;
K126<=N;

K221<=N;
K222<=N;
K223<=N;
K224<=N;
K225<=N;
K226<=N;

K111>=0;
K112>=0;
K113>=0;
K114>=0;
K115>=0;
K116>=0;
```

```
K211>=0;
K212>=0;
K213>=0;
K214>=0;
K215>=0;
K216>=0;

K121>=0;
K122>=0;
K123>=0;
K124>=0;
K125>=0;
K126>=0;

K221>=0;
K222>=0;
K223>=0;
K224>=0;
K225>=0;
K226>=0;
 0<=s(5)<=2;
 0<=s(6)<=2;
 0<=s(1)<=1;
 0<=s(2)<=1;
 0<=s(3)<=1;
 0<=s(4)<=1;
```

```
                0<=x(1)<=1;
                0<=x(2)<=1;
                0<=x(3)<=1;
                0<=x(4)<=1;
                0<=x(5)<=1;
                0<=x(6)<=1;
            cvx_end
      actions=[actions;s];
O1=s(1) -0.8*(1/36);
O2=s(2) -0.8*(1/36);
O3=s(1)+s(3) -0.8*((1/36)*(s(3)-x(4))+(1/36)*(1+s(3)-x(4)));
O4=s(2)+s(4) -0.8*((1/36)*(s(4)-x(5)-x(6))+(1/36)*(1+s(4)-x(5)-x(6)
    ));
D1=s(5) -0.8*1/12;
D2=s(6) -0.8*1/12;

g=[s' x'];
B=[B;g];
 i=i+1;
 cost(i)=c*(B(i,7)+B(i,8)+B(i,9)+2*B(i,7)+2*B(i,7)+2*B(i,7))+l*(B(
     i,5)-B(i,7)-B(i,8)-B(i,10)-B(i,11)+B(i,6)-B(i,9)-B(i,12));

e1=s(5)-s(1)-s(3);
if e1<0
     e1=0;
end
e2=s(6)-s(2)-s(4);
```

```
if  e2<0
        e2=0;
end
e11=0-(s(1)-x(1));
if  e11<0
        e11=0;
end
e12=1-(s(1)-x(1));
if  e12<0
        e12=0;
end
e13=2-(s(1)-x(1));
if  e13<0
        e13=0;
end
e14=0-(1+s(1)-x(1));
if  e14<0
        e14=0;
end
e15=1-(1+s(1)-x(1));
if  e15<0
        e15=0;
end
e16=2-(1+s(1)-x(1));
if  e16<0
        e16=0;
end
```

```
e21=0-(s(2)-x(2)-x(3));
if e21<0
    e21=0;
end
e22=1-(s(2)-x(2)-x(3));
if e22<0
    e22=0;
end
e23=2-(s(2)-x(2)-x(3));
if e23<0
    e23=0;
end
e24=0-(1+s(2)-x(2)-x(3));
if e24<0
    e24=0;
end
e25=1-(1+s(2)-x(2)-x(3));
if e25<0
    e25=0;
end
e26=2-(1+s(2)-x(2)-x(3));
if e26<0
    e26=0;
end
E1=e1+0.8*(e11*1/36+e12*1/36+e13*1/36+e14*1/36+e15*1/36+e16*1/36);
E2=e2+0.8*(e21*1/36+e22*1/36+e23*1/36+e24*1/36+e25*1/36+e26*1/36);
```

```
a=[0.2;O1;O2;O3;O4;D1;D2;E1;E2];
A=[A,a];
n=n+1;
j=j+1;
Opt(j)=cvx_optval;
cr=abs(Opt(j)-Opt(j-1));
  end
```