ABSTRACT

USING REINFORCEMENT LEARNING IN A SIMULATED INTELLIGENT TUTORING SYSTEM

Manohar Sai Jasti, M.S.
Department of Computer Science
Northern Illinois University, 2019
Dr. Reva Freedman, Director

I used reinforcement learning to investigate which categories of hints are most efficient in an intelligent tutoring system for human anatomy. Efficiency is defined as minimizing the time it takes the student to learn the material. When a student gives a wrong answer, the tutor can give them a text hint, a diagrammatic hint, or a video clip. Each type of hint takes a different amount of time to deliver and takes the student a different amount of time to understand.

I built a simulator for the intelligent tutoring system to collect data from simulated students. I implemented reinforcement learning, in particular two Temporal Difference (TD) Learning techniques on this simulated data to identify the most efficient hint specific to a student and the most efficient hint for the whole student population. I show that the most efficient hint type is a function of the two times listed above.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

AUGUST 2019

USING REINFORCEMENT LEARNING IN A SIMULATED INTELLIGENT TUTORING SYSTEM

BY

MANOHAR SAI JASTI

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

Thesis Director:
Dr. Reva Freedman

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

INTRODUCTION

I investigated why students fail to solve problems speedily when they are using Intelligent Tutoring Software (ITS). I found that one of the main reasons was the lack of hints personalized to their learning style when they fail to answer a question correctly. I thought reinforcement learning (RL), an advanced machine learning technique, would be an apt solution to this hint personalization problem.

By modelling student behavior, explanations given by experts have a nice impact on improving student problem-solving skills. In building an intelligent tutoring system using advanced machine learning techniques, not only can we improve student problem solving skills, but we can increase the productivity of the system. This will save a lot of manual resources and time by automating the things which would otherwise need to be done manually, such as customizing hints for every student. Until now, studies on customizing hints have been mainly limited to texts, but the ITS we are building also includes different types of hints.

Having these benefits in mind as a foundation for this study, I built a simulator for an ITS to collect data from artificial students. Then I built two agents which were responsible for student needs in the simulated environment using reinforcement learning algorithms. The agents learned to provide hints according to the student's learning style. The content formats I considered are text, image and video. Identifying the student's preferred content format in tutoring will help the student to learn the content in a more comfortable way and both master the content as well as solve problems more quickly. I extracted the simulated students' data and performed data segmentation by student. I provided insights by evaluating the reinforcement learning agents' performance.

Reinforcement learning works by optimizing a parameter. The parameter I chose was the sum of hint delivery time and student think time after the hint for each type of hint. I show that the best hint type chosen by each algorithm is sensitive to the sum of these parameters.

Hypothesis

Adding a Reinforcement Learning (RL) component to an Intelligent Tutoring System (ITS) will enable learners to master the content faster.

Scope

This system will deal with written language only (not speech). This particular project deals only with an anatomy and physiology class but could be extended to different fields.

BACKGROUND

When a student fails to understand a particular concept, for assistance they rely on their teachers and parents who have limited free time. Intelligent tutoring systems (ITS) provide the necessary assistance to those learners in this scenario. The goal of an ITS is to provide immediate and tailored instruction, i.e., feedback to learners, without the intervention of a human. The input to the ITS is the student's answer to a particular problem and the output that the ITS delivers is feedback or immediate tailored instruction. If a student is having a hard time solving a problem in the ITS, then the artificial tutor will give hints to encourage the student to solve that problem exactly like a teacher giving hints to students in real life. These hints will be in the sequence developed by the logic builder for the ITS [Martin and Arroyo 2004]. Sometimes the hints given by the ITS will not help the students; this is accepted since every student does not have the same cognitive level when trying to answer a particular problem. Therefore, hints should be personalized according to a particular student's needs. This can be done by modelling the behavior of the student. Concept understanding is the long-term goal because if a student masters the concept then he doesn't need hints to solve a problem and he will be able to solve any type of problem without the intervention of a human.

Some nursing, physical education and physical therapy students have a hard time understanding how the heart functions in their introductory anatomy and physiology classes (BIOS 311 & BIOS 357) at Northern Illinois University. Understanding how the heart functions is the most important thing for any student who wants to pursue a career in the field of medicine, since understanding its function will make it easier for the students to learn about other vital organs like the lungs. An ITS will help the students in understanding those heart functions clearly, which will help them in solving problems about it without the intervention of a human.

In our system the tutor is trying to teach concept ‘X’ by helping students solve a problem ‘Y’. Image, necessary videos and textual explanation should be available to the student to satisfy different learning styles. These should be shown based upon the student’s interest or when the tutor determines that this is the best learning style for the student.

The reinforcement learning agent that I want to insert into the ITS will help learners to successfully master the content faster by giving them the best possible content according to the topic or their learning style. My motivation for the reinforcement learning is AlphaGo, an AI powered system created by Google. It has defeated the human Go world champion. Go is an ancient and complex strategy board game. AlphaGo is designed using reinforcement learning to improve its play [Silver et al. 2018]. It has been automated to help guarantee victory. In the same way the RL agent will improve the overall efficiency of an ITS.

Reinforcement learning is a machine learning technique which has grown rapidly along with deep learning and Bayesian methods in recent times. To understand the importance of reinforcement learning, it helps to understand the drawbacks of supervised machine learning and unsupervised machine learning. Supervised learning uses features along with labels. If we don’t have labels, then supervised learning is not a possible solution. Unsupervised learning identifies patterns or underlying structure present in the data without any labels or supervision. However, unsupervised learning expects to see all the data at once; it is not useful when the data appears one item at a time. Sometimes trial and error with feedback in playing a video game is easier and more likely to reduce error rather than understanding the structure of a video game.

Reinforcement learning will try to find the optimal solution using a trial and error approach with feedback.

In an ITS context, we can apply supervised learning to predict a hint, which will help the student to answer a question in less time. But the drawback remains the same. It assumes we have enough data, but we don't necessarily. In supervised learning it is also important to assume that the training points are developed independent of each other. If the training points are dependent on each other then we should look for the optimal answer to remove the similar training points.

Answers given by a domain expert can be very costly and are not guaranteed to find an optimal solution. We can predict what type of hint approximately to give the student from the data we have but we cannot predict which hint will definitely help them because not every student's cognitive level or thinking ability is the same.

Reinforcement learning is good for sequential decision making when looking for the optimal answer. The RL agent will interact directly with the environment and choose the actions which will yield maximum reward to the agent. The agent learns from past episodes and performs better in the future. In this manner, the student will benefit from a hint that will really help him to master the content faster.

Literature Review on ITSs

According to Evens and Michael [2006], one-on-one tutoring is typically carried out to accomplish one of three different tasks: (a) teaching the student a new subject, (b) remedying a student's failures in understanding something, or (c) helping the student acquire a new skill. Whatever the task to be completed, the tutor must make some decisions about how to reach the intended goal. Expert tutors spend most of their time pushing their students to take a more active role in the learning process by posing problems, asking questions and hinting at answers. Hinting is a powerful tool for promoting active learning. Hints are designed to help students figure out answers for themselves and play a major role in increasing the productivity of an ITS.

According to Nye, Graesser, and Hu [2014] who are continuing to develop the platform, AutoTutor is a natural language based intelligent tutoring system. AutoTutor helps students learn topics in fields such as Newtonian physics, computer literacy, and critical thinking. Human input is represented using either voice or text. It uses computational linguistics algorithms to process the input. AutoTutor simulates the discourse patterns of human tutors.

VanLehn [2006] describes tutoring systems as having two loops. The outer loop is used for executing each task, which here refers to a complex multi-step problem. The remaining loop, called the inner loop, is executed once for each step taken by the student in completing the task. The inner loop can give feedback and hints on each step. This can be used to update a student model, which is used by the outer loop to select new tasks that are appropriate for the student. Andes, a physics tutor created by Gertner, Conati, & VanLehn [1998], constructs a Bayesian network which represents a solution to a physics problem and then addresses the student's difficulties by providing the help they need based on their skills.

Although the systems mentioned above provide mainly text hints, Silva, Costa and Araujo [2019] describe an approach to give multimodal hints to students when solving C/C++ programming problems. They have three kinds of hints: videos, text hints and flow charts. Students may choose a hint of any kind. Then the system creates a list of feedback messages in their preferred modality. Their approach had a positive impact on student learning.

The main purpose of the Magner, Schwonke and Renkl [2011] study was to detect whether pictorial illustrations increased the student's interest or not in solving geometry problems. The results showed that pictorial representations increased emotional interest rather than value related interest. In other words, students saw pictorial representations as decorative instead of informative. This kind of interest increased student willingness to work in a picture filled computer-based learning environment whereas deepening the geometry knowledge of a student depends on the quality of the pictures. Their conclusions are based on self-assessed measurements.

In our simulation, we have added multimodal hints to Vanlehn's loop based approach.

Literature on Reinforcement Learning in ITSs

There are a few early intelligent tutoring systems that use techniques from machine learning to increase productivity and to reduce the amount of time needed to develop a customized system. Martin and Arroyo [2004] added a RL agent, Agent X, to Wayang Outpost, a web-based intelligent tutoring system for elementary arthmetic. The method they proposed increases the efficiency by customizing the hints that are provided to the student. They clustered the students into different learning levels based on a pretest and re-clustered students depending on performance. Beck, Woolf and Beal [2000] constructed ADVISOR, a two-level machine learning architecture for another tutoring system for arithmetic called Animal Watch. The first agent models student behavior through simulation and the second agent uses reinforcement learning to create a teaching policy that meets the specified educational objective. The hints provided by ADVISOR and AGENT X are in a textual format, which is one of the learning styles that our system will use.

There are a few additional references to RL-based ITSs in more recent years. Malpani, Ravindran and Murthy [2011] presented a personalized ITS which uses reinforcement learning to maximize the time in which a student interacts with the ITS by learning teaching rules. They found that if a student interacts with the system more then he improves his learning more.

Finally, reinforcement learning has become popular again in 2019. Georgila, Core, Nye, Karumbaiah, Auerbach, and Ram [2019] used reinforcement learning in an ITS that teaches interpersonal skills. For learning teaching policies and pruning the state space they used a model-free method called Least-Squares Policy Iteration. An evaluation was done with human participants against a hand crafted ITS. Both systems improved pretest to posttest scores but not significantly. However, their ITS showed a significant gain in higher self-ratings of confidence.

Matsuda and Shimmei [2019] used reinforcement learning to detect flaws in existing courseware by computing a converse policy with the help of Value Iteration, a popular reinforcement learning technique. The best actions in this policy are the least effective instructional materials that can be used in a particular state. They proposed a method called RAFINE (Reinforcement learning Application For INcremental courseware Engineering) which iteratively detects the least effective material and gives it to the courseware developers to improve it. The work they proposed in this paper is a step towards their main goal. Their long-term goal is to create fully autonomous self-improving courseware, where the machine identifies ineffective material and then a human fixes it.

Rowe, Smith, Spain and Lester [2019] used reinforcement learning for automated scenario generation in a simulation-based training environment. They developed the DEEPGEN framework for automated scenario generation and evaluated the degree of novelty for scenarios in the framework by using the Four C Model, which estimates the degree of novelty of a model.

Moore and Stamper [2019] used reinforcement learning of model-based methods for generating hints by following a greedy policy under uncertainty in an adversarial game called Connect Four. In this work they illustrated that with a limited state space, they could provide hints and feedback for a larger percentage of frequently seen states. This work provides some serious suggestions for adversarial game learning environments.

The early work on Agent X and ADVISOR is the closest to ours, although they provide only text hints. In addition, both of these systems are for elementary arithmetic, where the problems are shorter and simpler than ours. This permits the ITS to generate and the student to solve a larger number of problems per session than ours.

Each of the more recent authors has used reinforcement learning for a different purpose in a learning environment. Each of these demonstrates a way that RL can be used in education, but none of them used RL in identifying the student's preferred content format, especially when showing hints.

APPROACHES TO REINFORCEMENT LEARNING

Reinforcement learning is categorized into mainly two categories [Sutton and Barto 2018]:

- Model based methods
- Model free methods

Model Based Methods

Model based methods, as the name suggests, solely depend on the model. Model based methods use Markov Decision Process (MDP) as an environment. A Markov Decision Process is a tuple (S, A, P, R, γ), where S is a finite set of states, A is a finite set of actions, P is a state transition probability matrix, R is a reward function and γ is a discount factor. R depends on an action taken in a state s at a time step t. The total return sums all rewards at all time steps. The value γ is between 0 and 1. If it is close to 0 then we care about an immediate reward and if it is close to 1 then we care about a future reward. Reinforcement learning cares about the total reward, So the MDP is solved when an agent finds a way to maximize the total reward by taking optimized actions in the MDP. There are mainly two methods to solve an MDP: value iteration and **polic**y iteration. These methods use dynamic programming.

<u>Disadvantages of Model Based Methods</u>

The main disadvantages of model based methods are as follows:

1. Total knowledge of MDP is required in order to solve a problem using these methods.
2. In solving MDP for larger problems, dynamic programming suffers from the Bellman curse of dimensionality as the number of states grows exponentially with the number of state variables [Sutton and Barto 2018, p. 87].

Model Free Methods

In model free methods we are going to see how this curse of dimensionality is broken with the help of sampling trajectories. The big advantage of model free methods over model-based methods is that no advanced knowledge of MDP is required, i.e., no transition probability matrix or reward function. The problem I am trying to solve does not contain a transition probability matrix or a pre-existing reward function, so I want to use a model free method.

There are few approaches among model free methods. They are [Sutton and Barto 2018, p. 119]:

1. Monte Carlo Learning
2. Temporal Difference Learning

Monte Carlo Learning

Monte Carlo methods learn from complete episodes. An episode is a sequence of states, actions and rewards under a particular policy. In the Monte Carlo prediction algorithm, we evaluate policies. The idea here is to use the mean return for a value function of a state. The value function gives the long-term value of a state. Basically, it determines how good it is to be in a state. The Monte Carlo control algorithm is used to find the optimal policy. It is done by making the policy greedy respective to the current value function. Hence, a state transition probability matrix and a reward function are not required.

Temporal Difference (TD) Learning

TD learns from incomplete experience using bootstrapping. In Temporal Difference prediction, we evaluate policies. The idea is to update the value of a state towards the estimated return. In the Temporal Difference control algorithm, we optimize the policy by optimizing the value function. That is accomplished by acting greedy to the current value function. Hence, a state transition probability matrix and a reward function are not required. TD can be n-step look ahead. I consider only TD(0), which is one step look ahead.

Difference between TD and Monte Carlo Prediction

TD uses bootstrapping whereas Monte Carlo prediction doesn't use bootstrapping. TD learns before completing episodes whereas Monte Carlo prediction learns after completing episodes, so Monte Carlo works for environments in which episodes terminate.

TD is usually more efficient than Monte Carlo prediction because of low variance. Unlike Monte Carlo prediction, TD is looking at noise in the first step in an episode not the entire trajectory/episode, so the variance will be less.

Although Monte Carlo learning and Temporal Difference learning converge on the same solution, having the above-mentioned benefits of learning in mind, I chose TD over Monte Carlo algorithms to solve my problem.

To explain the difference as an analogy, consider you are travelling from place A to place B and the goal is to estimate at each time step what the remaining time to your destination is. In Monte Carlo learning, it would be as if you didn't trust Google Maps. After completing several episodes, where an episode is travelling from place A to place B, you estimate the remaining time by taking the mean of the distances at each time step for those episodes. In Temporal Difference learning using Google Maps, you start travelling from place A to place B, then Google Maps shows the estimate at each time step. Even if your car breaks down at a particular time step, Temporal Difference learning makes a guess toward your arrival time even without completing an episode. Monte Carlo learning makes this estimate only after completing a particular episode.

Types of Learning in Temporal Difference Control

There are two types of learning:

1. On-policy learning is learning on the job, i.e., learning about a policy π from experience sampled from the same policy π.
2. Off-policy learning is learning about a policy π from experience sampled from behavioral policy μ, where a behavioral policy is the policy that we have from past episodes.

SARSA and Q-LEARNING are two types of TD control. SARSA is an on-policy learning algorithm. Q-LEARNING is an off-policy learning algorithm.

Suppose you have joined a company recently If you learn about the job based on your own experience then that type of learning is on-policy learning. If you learn about the job based on another person's experience or behavior and not by making the mistakes yourself then it is off-policy learning.

On-policy Control with SARSA

In SARSA we care about the state action value pair Q(S, A) rather than the value function of a state. The state action value pair tells us how good an action A in a state S is for the agent.

The algorithm for implementing SARSA is as follows [Sutton and Barto 2018, p. 130]:

```
Initialize Q(S, A) for all states and for all actions with any value
For all terminal states and any action, initialize Q(S,A) to 0
For each episode
    Initialize state S
    Choose action A from state S using policy $\pi$ derived from Q
    // policy derived from Q can be any policy such as ε-greedy
    For each step of episode
            Take action A and see reward R, next state S'
            Choose next action A' from S' using policy $\pi$ derived from Q
            //policy derived from Q can be any policy such as ε-greedy
            Update $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
            Assign state S to next state S', and assign action to next action A'
    Until S is terminal
```

Off-policy Control with Q-LEARNING

Q-LEARNING also focuses on Q(S, A). We can improve both target and behavior policies. Target policy π is greedy with respect to Q(S, A) and behavior policy μ is ε-greedy with respect to Q(S, A). The idea in an ε-greedy strategy is to choose an epsilon probability. Every time you want to pick an action, choose a random probability. If the epsilon probability is greater than your chosen random probability, then select a random action from the action space, otherwise pick the greedy action. This strategy ensures policy improvement in every episode [Sutton and Barto 2018, p. 101].

The algorithm for implementing Q-LEARNING is as follows [Sutton and Barto 2018, p. 131]:

```
Initialize Q(S, A) for all states and for all actions with any value
For all terminal states and any action, initialize Q(S,A) to 0
For each episode:
        Initialize state S
        For each step of episode
            Choose action A from state S using policy $\pi$ derived from Q
            // policy derived from Q can be any policy such as ε-greedy
            Take action A and see reward R, next state S'
            Update $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
            Assign state S to next state S'
        Until S is terminal
```

Expected Value SARSA

Expected Value SARSA is also a TD control algorithm. Expected Value SARSA performs slightly better than SARSA given the same amount of experience [Sutton and Barto 2018, p. 131]. Expected Value SARSA can be an on-policy or an off-policy learning algorithm, depending on the problem. When solving a particular problem, the algorithm might use a different policy from the target policy to generate behavior or it might learn about its own target policy and strive to improve it. Expected Value SARSA also cares about Q(S, A). Instead of maximizing over the next state action pairs like Q-LEARNING, Expected Value SARSA uses the expected values to update Q(S, A).

The algorithm to implement Expected Value SARSA is as follows [Sutton and Barto 2018, p. 133]:

```
Initialize Q(S, A) for all states and for all actions with any value
For all terminal states and any action, initialize Q(S,A) to 0
For each episode:
        Initialize state S
        For each step of episode
            Choose action A from S using policy $\pi$ derived from Q
            // policy derived from Q can be any policy such as ε-greedy
            Take action A and see reward R, next state S'
            Update $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \sum_a \pi(S', a) Q(S', a) - Q(S, A)]$
            Assign state S to next state S'
        Until S is terminal
```

Justification for Approach

I want to implement Q-LEARNING and off-policy Expected Value SARSA for my hypothesis because they are guaranteed to converge in a model free setting [Sutton and Barto 2018, p. 124] and they can learn about a target policy while following a behavior policy on the move.

METHODOLOGY

Structure of the Anatomy ITS

Figure 1 shows the structure of the ITS for anatomy we are building which is based on a common structure of ITSs. The student interacts with the web interface, which interacts with the tutor model, which takes the place of the live teacher. The tutor model asks the student a question about a particular scenario. The scenarios and the questions are extracted from the content DB and sent to the student. The student replies back with an answer. The tutor model evaluates whether the answer is correct or not and then stores the student progress into the student model. All the information about student progress is accumulated in the Student DB. The tutor model replies back to the student with either a hint or a question or another scenario based on how well the student is doing.

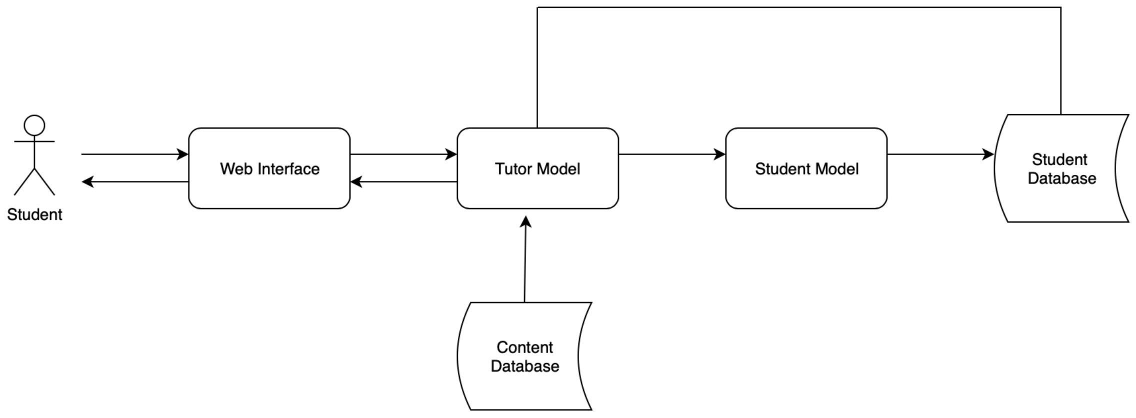


Figure 1: General structure of the anatomy ITS.

Here is a sample scenario and an associated question [Jasti and Freedman 2019].

> Scenario:
>
> > Mr. C. is walking to the bus stop when he sees the bus. He starts running to catch it, but feels short of breath and has to stop and rest.
>
> Question: *(multiple choice)*
>
> > What is the first thing that happened in Mr. C's physiology that caused him to feel short of breath?

The data flow in Figure 1 is as follows:

1. Student to web interface and vice versa
   a. Student replies with his answers to web interface.
   b. Web interface replies to student with Tutor model information.
2. Web interface to tutor model and vice versa
   a. Web interface sends the student's answer to the tutor model.
   b. Tutor model replies to the web interface with the information which includes scenarios or questions or hints that are shown to the student through it.
3. Tutor model to student model
   a. Tutor model writes the student progress information into the student model.
4. Student model to student DB
   a. Student model stores the students' progress information into the student DB.
5. Student DB to tutor model
   a. Tutor model reads the student progress information from the student DB.

6. Content DB to tutor model
    a. Tutor model extracts the following information from the content DB whenever possible.
        i. Anatomy information such as concept map.
        ii. Scenario information such as descriptions and questions.

Justification for Simulation

I used simulation because I want to design a prototype to experiment with different sets of parameters for a more efficient anatomy ITS design. The three kinds of parameters are hint types, time to deliver each type of hint, and time it takes the student to understand. This gives the developers an opportunity to concentrate on the most efficient types of hints.

Justification for the Use of Reinforcement Learning (RL)

In our system, if there were six scenarios, and for every scenario there were six questions, and for every question there were three hints (one video, one text, one diagram) there would be 108 possible hints. It is difficult to get accurate information for so many hints. Faculty don't necessarily explicitly know what hints to give and they also don't know each student's learning style. So I want to develop an algorithm to choose the best hints. The best hints are those which help students learn the most in the least amount of time. I believe that reinforcement learning would be perfect for this task because it figures out optimal solutions, i.e., the best hints to give to a specific student using a trial and error approach with feedback.

General Structure of My Methodology

I wrote Python code to do the following things:

1. Carry out the simulation of the intelligent tutoring system.
2. Store the required simulated data in a CSV file.
3. Extract, analyze and train the data using reinforcement learning temporal difference learning algorithms.
4. Generate hint type recommendations for the whole student population and specific to individual students.
5. Evaluate the agent mean reward for both cases and store the best agent recommendations in a pickle file (a Python persistent database format). The pickle file recommendations are used when the simulator runs again. When the simulator runs again the hint type is shown according to the pickled recommendations. Figure 5 shows the general structure of my methodology.

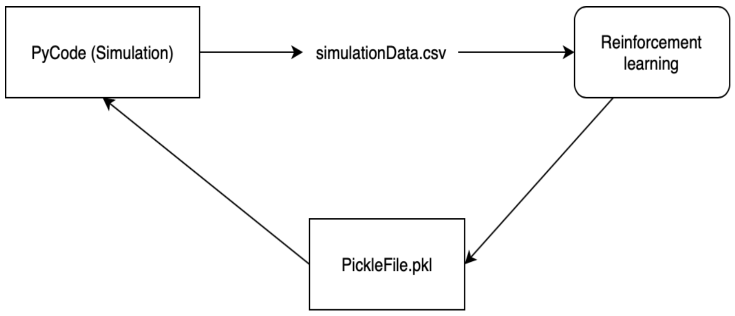


Figure 2: General structure of my methodology.

PyCode (Simulation)

I wrote the Python code to carry out a simulation of the anatomy ITS. The ITS teaches learners about how the heart functions with textual, video and image hints and also helps learners in solving the most critical real-life problems that occur in the functioning of the heart. Figure 3 shows the concept map used in the system [Kluga, Jasti, Freedman and Naples 2019].

Storing and Accessing the Concept Map

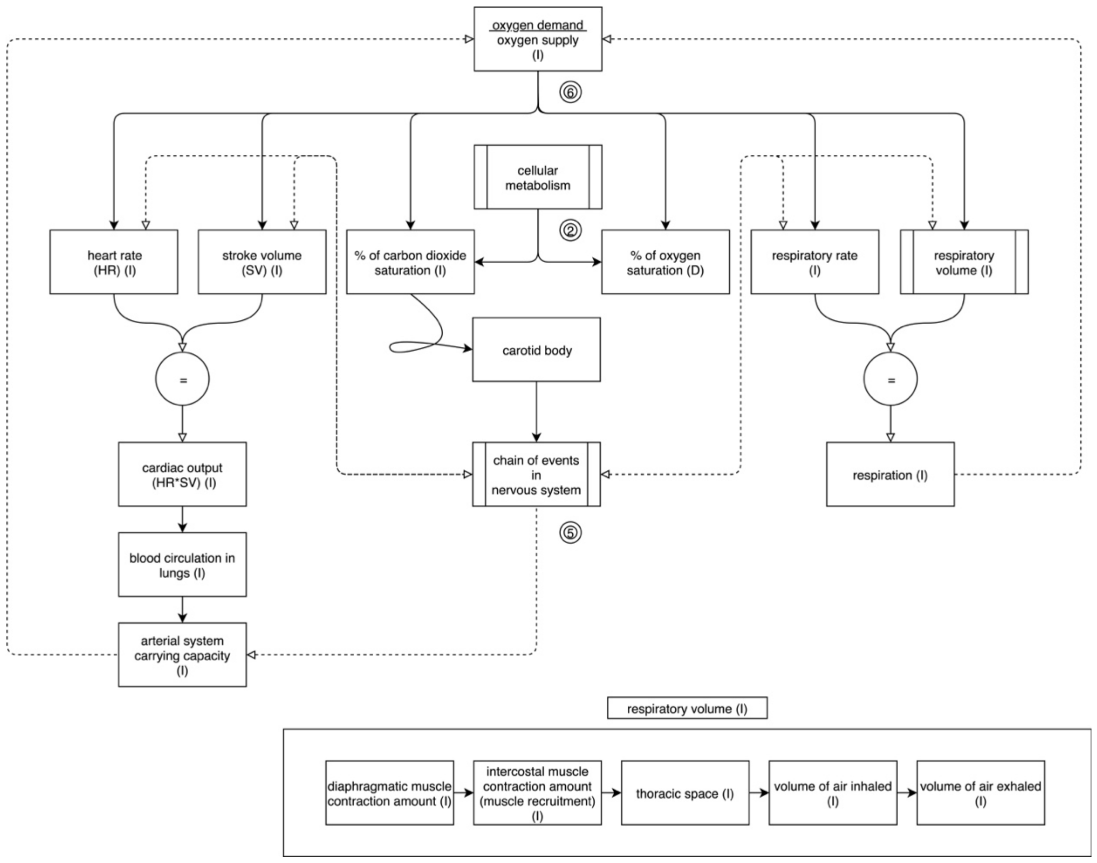
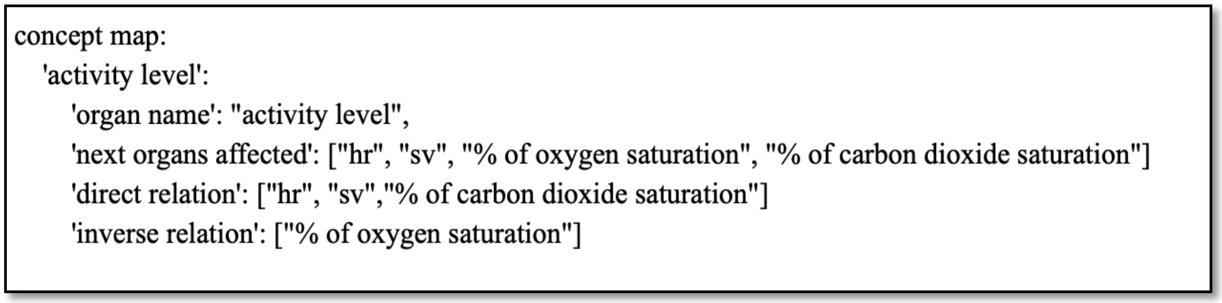

oxygen demand
oxygen supply
(I)
⑥
cellular metabolism
②
heart rate (HR) (I)
stroke volume (SV) (I)
% of carbon dioxide saturation (I)
% of oxygen saturation (D)
respiratory rate (I)
respiratory volume (I)
carotid body
=
cardiac output (HR*SV) (I)
chain of events in nervous system
respiration (I)
⑤
blood circulation in lungs (I)
arterial system carrying capacity (I)
respiratory volume (I)
diaphragmatic muscle contraction amount (I)
intercostal muscle contraction amount (muscle recruitment) (I)
thoracic space (I)
volume of air inhaled (I)
volume of air exhaled (I)

Figure 3: Concept map used in the ITS [Kluga et al. 2019].

I stored the concept map in a nested dictionary format, which includes for each parameter, the following: the parameters affected when that parameter is affected, and whether those parameters are directly or inversely affected. Figure 4 shows an example from the concept map stored in a nested dictionary format.



```
concept map:
  'activity level':
     'organ name': "activity level",
     'next organs affected': ["hr", "sv", "% of oxygen saturation", "% of carbon dioxide saturation"]
     'direct relation': ["hr", "sv","% of carbon dioxide saturation"]
     'inverse relation': ["% of oxygen saturation"]
```

Figure 4: An observation of a concept map stored in a nested dictionary format.

When activity level increases, heart rate (HR) increases, stroke volume (SV) increases, and % of carbon dioxide saturation increases, but % of oxygen saturation decreases.

A problem can be solved by tracing through the concept map from the first component affected until no more changes are found.

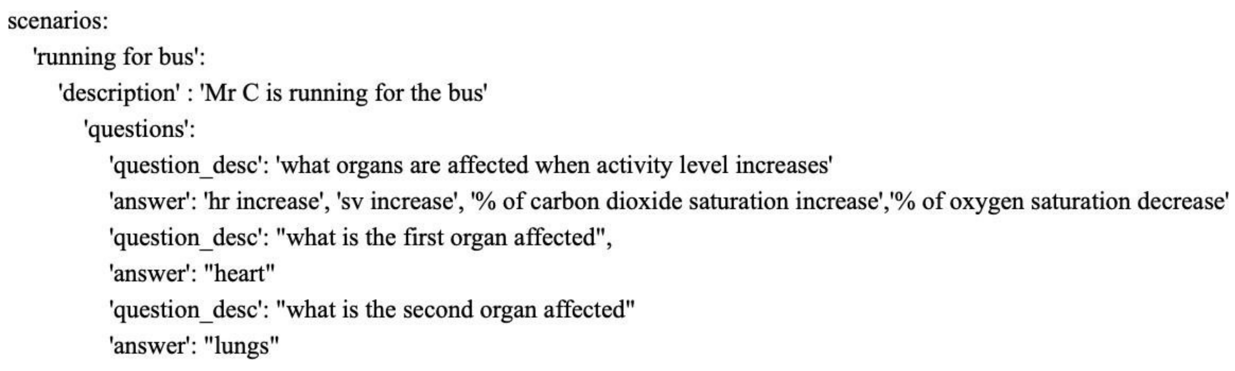
I implemented functions to retrieve and manipulate the information from the nested dictionary concept map. Table 1 shows a sample for retrieving and manipulating the information by giving an input to the “get next organs state” function. This function follows an arrow from one component to the next

Table 1: An example of getting the manipulated information from the concept map

Function	Input	Output
Get next organs state ()	[‘activity level’, ‘increase’]	[‘hr increase’, ‘sv increase’, ‘% of carbon dioxide saturation increase’], [‘% of oxygen saturation decrease’]

Storing and Accessing Scenarios

For showing the problems under different scenarios to a student I stored scenario information in a nested dictionary format, which includes a description of a scenario. The scenario also contains questions along with correct answers and hints if the student gives a wrong answer. Figure 5 shows a sample of a scenario including a description and three sample questions.



```
scenarios:
  'running for bus':
    'description' : 'Mr C is running for the bus'
      'questions':
        'question_desc': 'what organs are affected when activity level increases'
        'answer': 'hr increase', 'sv increase', '% of carbon dioxide saturation increase','% of oxygen saturation decrease'
        'question_desc': "what is the first organ affected",
        'answer': "heart"
        'question_desc': "what is the second organ affected"
        'answer': "lungs"
```

Figure 5: An example of the scenario nested dictionary.

I created a Scenario class to retrieve the information from a scenario nested dictionary. Table 2 shows an example of retrieving the question descriptions and answers from a scenario nested dictionary using the Scenario class.

Table 2: An example of retrieving information from the scenarios dictionary

Scenarios	Description	Answer
‘running for bus’	what organs are affected when activity level increases?	hr, sv, % of oxygen saturation, % of carbon dioxide saturation
	what is the first organ affected?	heart
	what is the second organ affected?	lungs

<u>Storing and Accessing Student’s Basic Information</u>

I stored student information in a dictionary format which includes student ID, student name and student section. Figure 6 shows an observation from the student dictionary.

```
Students:
        Sid: Z12
        Sname: s1
        Ssec: 1
```

Figure 6: An observation from the student dictionary.

I created a Student class to retrieve the information from the student dictionary.

Table 3: An example of retrieving information from the student dictionary using student ID

Szid	Sname	Ssec
Z12	S1	1

Building the Simulator for the ITS

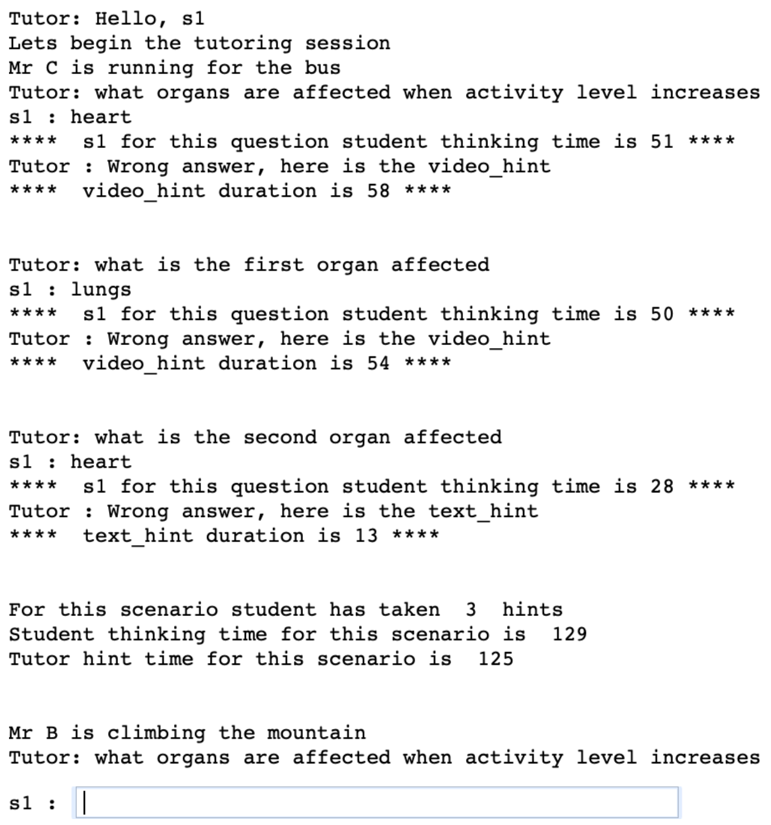
The artificial tutor uses the Scenarios class and the Student class for retrieving information for the main system and displaying it whenever needed. The artificial tutor interacts with students one at a time. Figure 7 shows the three nested loops with which the artificial tutor operates: showing different scenarios; for each scenario, displaying a set of questions one by one: for every incorrect answer showing a hint one by one.

Figure 8 shows the output of a sample interaction between a student and a tutor in the simulator. The input given is evaluated by the artificial tutor using the information in the concept map. If the student's answer matches the correct answer then the next question is displayed, otherwise the artificial tutor displays a hint. In the first iteration the probability of showing a hint from three different hint types is 1/3 = 0.33.

The simulator contains three loops: the student loop, the scenario loop and the question loop. The question loop under a particular scenario ends when the student completes answering all the questions. The scenario loop ends when the student completes answering all the questions in all the scenarios. The student loop ends when all students have completed answering all the questions in all the scenarios. The pseudo code is illustrated in Figure 7.

```
for student in students:
   print (Tutor: Hello student)
   for scenario in scenarios:
      print(scenario description)
         for question in questions:
            print(question)
            read the input from the student
            evaluate the input
            if student input is correct
                  goto next question
            else
                  show a hint based on probabilities
                  record the hint time
                  record the student thinking time
```

Figure 7: The pseduocode for the ITS simulator.



```
Tutor: Hello, s1
Lets begin the tutoring session
Mr C is running for the bus
Tutor: what organs are affected when activity level increases
s1 : heart
****  s1 for this question student thinking time is 51 ****
Tutor : Wrong answer, here is the video_hint
****  video_hint duration is 58 ****


Tutor: what is the first organ affected
s1 : lungs
****  s1 for this question student thinking time is 50 ****
Tutor : Wrong answer, here is the video_hint
****  video_hint duration is 54 ****


Tutor: what is the second organ affected
s1 : heart
****  s1 for this question student thinking time is 28 ****
Tutor : Wrong answer, here is the text_hint
****  text_hint duration is 13 ****


For this scenario student has taken  3  hints
Student thinking time for this scenario is  129
Tutor hint time for this scenario is  125


Mr B is climbing the mountain
Tutor: what organs are affected when activity level increases

s1 :
```

Figure 8: Sample output of the simulated ITS.

Storing the Required Simulated Data in a CSV file

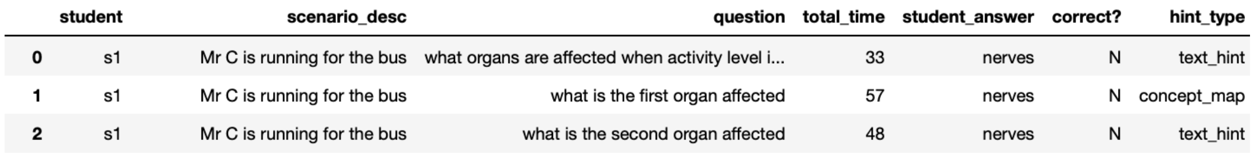
In Figure 7 the student think time and the tutor hint time are recorded and displayed for each question, scenario and student. At the end of the tutoring session the total student think time and tutor hint time are calculated, recorded and displayed.

In this interaction process between a student and the artificial tutor, the required data features are stored for every student. Table 4 shows the list of features and their descriptions of the simulated data.

Table 4: List of features and their descriptions of the simulated data

Feature name	Description
Time stamp	Timestamp is stored when the student has entered his/her answer
Student	Name of the student
Scenario description	The description of a scenario where the student is answering a question
Question	The question which the student is solving
Student question thinking time	End of the tutor's question to beginning of student's answer or End of tutor's hint to beginning of student's answer
Student answer	The answer that the student has given for a question under a scenario.
Correct?	This feature indicates whether an answer is right or wrong
Hint	The hint number
Hint type	The type of hint; whether a hint given is in video, text or diagrammatic format
Hint time	The time needed by an average student to read/view a hint

All the stored data points for the different features considered are stored into a CSV file which is later used for analysis and input to the RL agent training. A partial example of the simulated data CSV file is shown in Figure 9.



	student	scenario_desc	question	total_time	student_answer	correct?	hint_type
0	s1	Mr C is running for the bus	what organs are affected when activity level i...	33	nerves	N	text_hint
1	s1	Mr C is running for the bus	what is the first organ affected	57	nerves	N	concept_map
2	s1	Mr C is running for the bus	what is the second organ affected	48	nerves	N	text_hint

Figure 9: Partial rows from the simulated data CSV file.

Using RL to Implement a Solution to the Multi-Armed Bandit Problem

In general, a slot machine has a single arm. Consider a slot machine which has more than one arm. When using this kind of slot machine, the goal of a gambler is to try each and every arm. After k trials, he can figure out the arm that provides the maximum payout.

I want to use TD(0) algorithms in my simulation of a real-world problem where state doesn't matter. In other words, I want to experiment with different rewards for different categories of hints without taking into account the order in which the test cases come.

That problem is equivalent to a multi-armed bandit with 3 arms. It asks which category of hints provides the best rewards. The three arms are as follows:

1. Text hint
2. Video hint
3. Diagrammatic hint

Reinforcement learning permits doing a more complex type of experiment that takes sequences into account. In other words, the RL algorithms that I used optimize the sequence of hints given to the student. For example, after two long video hints in a row, a shorter diagrammatic or text hint that summarizes the content might be more efficient even if in general video hints work best.

Reinforcement Learning Pipeline for Best Hint Type for Whole Student Population

The Reinforcement Learning pipeline consists of several stages which include the following: preparing the inputs and creating a reward function for training the agents on these inputs, selecting the best agent based on the mean reward, and storing the recommendations of the best hint type in a pickle file. The pipeline to complete this process for identifying the best hint type for the whole student population is shown in Figure 10.

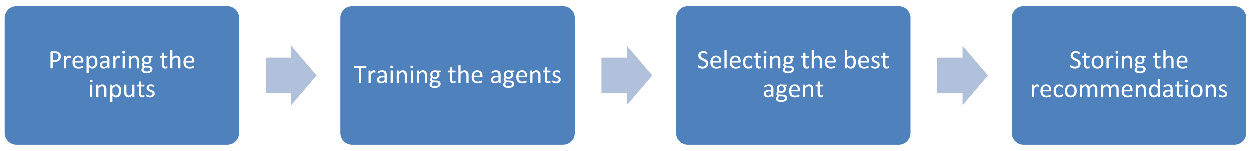


Figure 10: Pipeline to find out best hint type for the whole student population.

Important inputs for the RL agent are:

1. Hint delivery time
2. Student thinking time after hint

Hint delivery time is the amount of time that it took the student to view the hint. Student thinking time after the hint is the amount of time that it took the student to answer the question after viewing the hint.

I extracted the minimum and maximum time from the total time grouped by hint type (see Table 5 for some sample times).

Table 5: Example showing range of student question thinking times by hint type

Student time per hint	Concept map	Text hint	Video hint
Min time (in seconds)	21	22	20
Max time (in seconds)	59	57	58

I created two agents using TD(0) control algorithms:

1. Q-LEARNING
2. Expected Value SARSA

Each agent has three actions to take in the simulation. The three actions are mapped to integers for readability and simplicity. Table 6 shows the actions performed by an agent and its mappings.

Table 6: Actions performed by an agent and its mappings

Index	Action
0	Concept map
1	Video hint
2	Text hint

Agents play and train with the simulation environment by taking actions. Actions are particular kind of hints. For every action, the reward is calculated by the reward function. The reward function for each RL agent is inversely proportional to student question thinking time (total time). This total time is the sum of hint delivery time and student thinking time after hint [Beck, Woolf and Beal 2000]. Agents knowledge is updated for that particular action. Updating the agent's knowledge differs by agent because they are two different algorithms of TD(0) control.

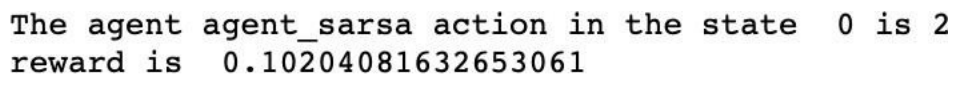
Agents take actions using the epsilon greedy policy. A random probability and desired hyperparameter epsilon are chosen. Table 7 illustrates what the epsilon hyperparameter is. If epsilon is greater than random probability, the agent chooses a random action. Otherwise it chooses the best action available.

Table 7: Agents' hyperparameters and their descriptions

Hyperparameter	Description
Epsilon	Epsilon belongs to [0, 1]. If it is close to 1 then the probability of agent exploration is very high. If it is close to 0 then the probability of agent exploration is very low.
Discount (gamma)	γ belongs to [0,1]. If it is close to 0 then we care about the reward now and if it is close to 1 then we care about the reward later.
Learning rate (alpha)	Learning rate is the speed at which the agent's knowledge is updated. In other words, learning rate controls how quickly or slowly an agent model learns to solve a problem.

I implemented the Q-LEARNING and Expected Value SARSA algorithms that I wrote based on specifications in the assignment on model-free learning in the Practical Reinforcement Learning course in Coursera[1]. I used the same code for creating the agents for my problem with the three hyperparameters learning rate, epsilon and discount (see Table 7).

For training, I wrote a play and train method in which an agent goes through a state S and takes an action A when the student answer is wrong. Student question thinking time for an answer is calculated randomly based on a uniform distribution between the minimum and maximum time for the particular action that the agent took. Then the reward is calculated from the reward function. In the example shown in Figure 11, the Expected Value SARSA agent takes an action of 2 in state 0 and gets a reward of 0.102. Total reward is updated for every observation. This loop will complete when the agent completes playing with all the observations. This function returns the total reward in the end.



```
The agent agent_sarsa action in the state  0 is 2
reward is  0.10204081632653061
```

Figure 11: Sample output of reinforcement learning agent taking an action and getting a reward.

After showing a hint, a new student answer is chosen randomly based on the estimated percentage of answer correctness for the hint type (see Table 8). My results do not take these values into account; this code is present for future enhancement.

[1] https://www.coursera.org/learn/practical-rl.

Table 8: Estimated percentage of answer correctness by hint type

Hint type	Percentage of answer correctness
Concept map	75
Video hint	90
Text hint	80

I wrote a plot mean reward method in which I trained Q-LEARNING and Expected Value SARSA for 1000 episodes with changing hyperparameters using the play and train function.

For every episode in the training I am doing the following:

1. I am storing the action counts for each agent and the rewards.
2. I am calculating the mean reward of both the agents.

Table 9 shows an example of storing the action counts of the agents after 1000 episodes of training.

Table 9: An example of action counts of agents for all simulated student population

Agent	Concept map	Video hint	Text hint
Q-LEARNING	28190	6875	9935
Expected Value SARSA	32558	3426	9016

For every 100 episodes in the training I plot the exponential mean reward of both agents. I chose exponential mean reward because I want to give more weight to the recent total rewards. Figure 12 shows a plot of the moving average of the exponential mean rewards over 1000 episodes.

```
EVSARSA mean reward = 8.15103336093
QLEARNING mean reward = 8.14914295143
```

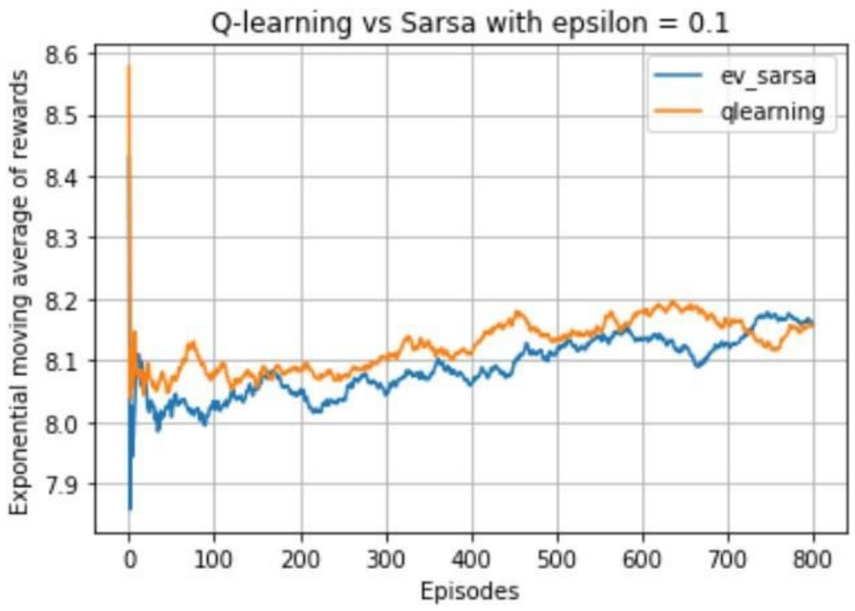


Figure 12: Sample plot of exponential moving average rewards over 1000 episodes.

I selected the best agent based on the highest mean reward. For example, in Figure 12, Expected Value SARSA has a slight edge over Q-LEARNING. So, I will select Expected Value SARSA as the best agent. I selected the best hint type based on the action count of the best agent.

I calculated the probabilities of hint types from the best agent action counts. Table 10 shows some sample probabilities for each hint. After 1000 episodes, these will become the recommendation probabilities for each hint. I stored the best hint type and recommendation probabilities for each individual hint into a pickle file. These recommendations were used in the ITS simulator for the next run.

Table 10: A sample of recommendations from the best agent for the whole student population

Hint type	Recommendation Probability
Concept map	0.723511
Video hint	0.200355
Text hint	0.076133

Reinforcement Learning Pipeline for Best Hint Type Specific to a Student

The pipeline to figure out the best hint type specific to a student is shown in Figure 13. First, I segmented the student data and then I followed the same pipeline that I used to find the best hint type for the whole student population.

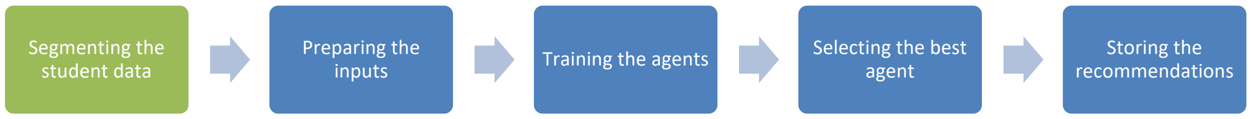
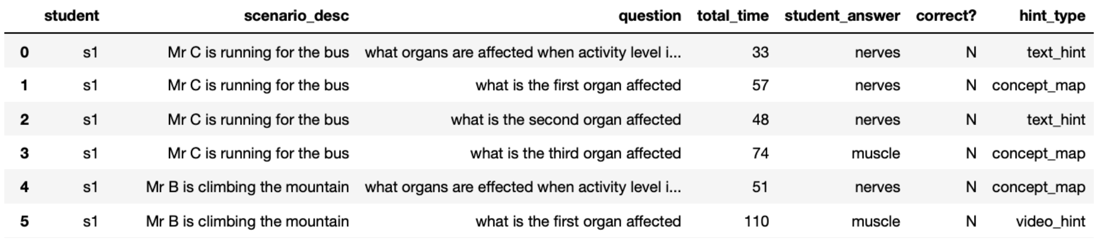


Figure 13: Reinforcement pipeline to find the best hint type specific to a student.

I segmented the students based on student name. Figure 14 shows partial records of student for the student s1 data frame.



	student	scenario_desc	question	total_time	student_answer	correct?	hint_type
0	s1	Mr C is running for the bus	what organs are affected when activity level i...	33	nerves	N	text_hint
1	s1	Mr C is running for the bus	what is the first organ affected	57	nerves	N	concept_map
2	s1	Mr C is running for the bus	what is the second organ affected	48	nerves	N	text_hint
3	s1	Mr C is running for the bus	what is the third organ affected	74	muscle	N	concept_map
4	s1	Mr B is climbing the mountain	what organs are effected when activity level i...	51	nerves	N	concept_map
5	s1	Mr B is climbing the mountain	what is the first organ affected	110	muscle	N	video_hint

Figure 14: Partial records of student 's1' from the simulated data.

For every student, I processed this data through the pipeline shown in Figure 13 to find the best hint type for that student. After completing the training, an example of the agents' action counts is shown in Table 11.

Table 11: An example of action counts of agents for individual students

Student	Best Agent	Concept map	Video hint	Text hint
S1	Q-LEARNING	3163	5137	700
S2	Expected value SARSA	8312	373	315

From these action counts, I calculate the recommendation probabilities for the best agent and store them in a pickle file. Table 12 shows the sample recommendation probabilities for each student using the best agent. I will use these recommendations to show a hint specific to a student in the next run of the ITS simulator.

Table 12: An example of a recommendation suggested by best agent for individual students

Student	Concept map	Video hint	Text hint
S1	0.351	0.571	0.077
S2	0.920	0.041	0.035

In addition to storing the recommendation probabilities for the best agent, I also store the best hint type into another pickle file. Table 13 shows an example of the best hint type specific to a student.

Table 13: Best hint type for individual students based on agent's action count

Student	Best hint type
S1	Video hint
S2	Concept map

RESULTS

Plotting the Distributions of Hint Types from the Simulated Data

In the simulation, before acquiring the recommendation probabilities or the best hint type, I gave a hint type randomly whenever the student failed to answer a question correctly. The hint types are given with an equal probability of 1/3. After completing 1000 episodes, I looked at the data to see how well the hint types were distributed. Figure 15 shows a sample distribution of hint types after the first episode. The distribution of hint types is not the same every time due to randomization.

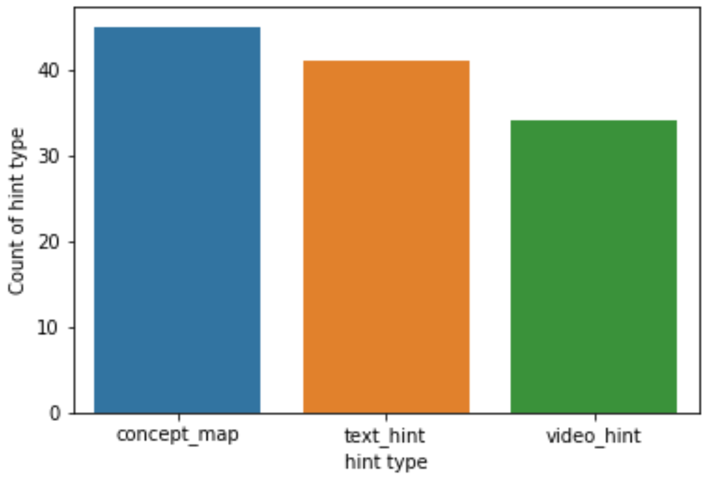


Figure 15: Distribution of hint types in the simulated data.

The Advantage of Expected Value SARSA over Q-LEARNING

To decide which agent is better I did the following:

1. Experimented with both changing epsilon and with a constant epsilon to see the convergence rate of the agents and the mean reward.
2. The following are the values for other hyperparameters that I hand tuned and considered during the training:
    a. Discount = 1
    b. Learning rate = 0.25

<u>Expected Value SARSA vs Q-LEARNING with Constant Epsilon</u>

Figure 16 shows the performance of Expected Value SARSA and Q-LEARNING agents over 1000 episodes with epsilon = 0.1. Although Expected Value SARSA maintains a mean reward greater than Q-LEARNING in the initial stages, Q-LEARNING has a slight edge over Expected Value SARSA. Overall, Expected Value SARSA and Q-LEARNING achieved similar mean rewards and both algorithms converged very quickly after only 200 episodes.

Both the algorithms, Q-LEARNING and Expected Value SARSA, learned to prefer one action over others. In fact, they both preferred showing the concept map hint type over the other hint types.

```
EVSARSA mean reward = 12.0869638374
QLEARNING mean reward = 12.0880605477
```

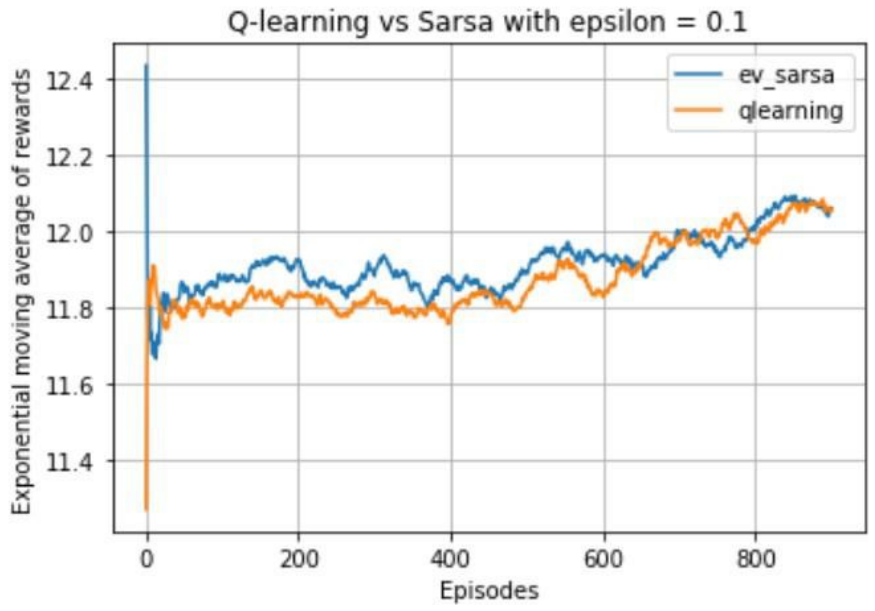


Figure 16: Expected Value SARSA vs Q-LEARNING over 1000 episodes with constant epsilon.

There are 120 observations in the simulated data. I am running 1000 episodes on each of these agents. Table 14 shows the counts of hint types for each agent.

Table 14: Final action counts of agents with constant epsilon for whole student population

Agent	Concept map	Video hint	Text hint
Q-LEARNING	87188	9919	22893
Expected Value SARSA	96643	7718	15639

Before examining why both the agents preferred the concept map, we need to look at the range of student question thinking times categorized by hint types, because this is the vital input feature for determining the preferred hint type. These are shown in Table 15. The values for the concept map and text hints are quite similar. The range for video hints is larger.

Table 15: Range of student question thinking times by hint type from the simulated data

Student time per hint	Concept map	Text hint	Video hint
Min time (in seconds)	30	27	35
Max time (in seconds)	77	78	112

So whenever an agent, either Q-LEARNING or Expected Value SARSA, took the video hint, their reward for that state is less because our reward function is inversely proportional to the time. I believe the reason that agents prefer the concept map over text hints is due to the constant epsilon = 0.1. In other words, the agent explored actions other than the concept map only 10% of the time; it took the concept map most of the time. Even when it explored a random action, for example if it chose the text type, the reward will be similar to the concept map reward. So in order to test this theory I decreased epsilon over 100 episodes for both agents.

Expected Value SARSA vs Q-LEARNING with Decreasing Epsilon

I started epsilon with the largest possibility of chosing a hint type randomly, i.e., 1.0 for both agents, and then for every 100 episodes I decreased it by 0.1. Figure 17 shows the performance of Q-LEARNING and Expected Value SARSA with decreasing epsilon. After 1000 episodes epsilon will be 0. It is noted that both agents constantly strive to increase their mean reward in every episode. Overall, Expected Value SARSA performed better than Q-LEARNING but Q-LEARNING improved with decreasing epsilon.

```
EVSARSA mean reward = 12.2461726674
QLEARNING mean reward = 12.1460382043
```

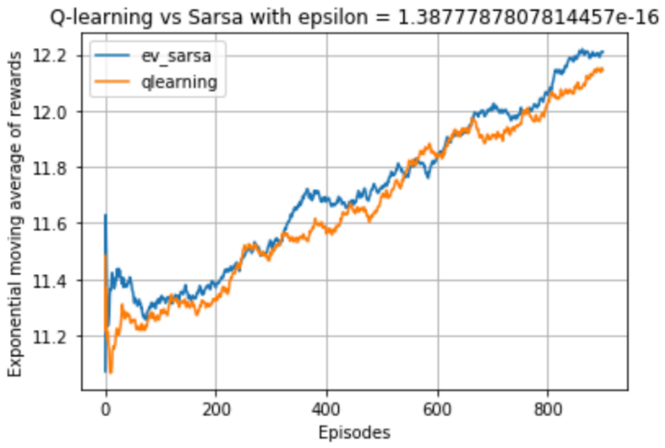


Figure 17: Expected Value SARSA vs Q-LEARNING over 1000 episodes with decreasing epsilon.

Table 16 shows the distribution of hint types as actions taken by Q-LEARNING and Expected Value SARSA with decreasing epsilon.

Table 16: Final action counts of agents with decreasing epsilon for whole student population

Agent	Concept map	Video hint	Text hint
Q-LEARNING	68678	14019	29579
Expected Value SARSA	74602	10604	40718

Best Hint Type for the Whole Class

Expected Value SARSA took text hints more than Q-LEARNING took text hints over 1000 episodes with decreasing epsilon. This made the mean reward of Expected Value SARSA surpass Q-LEARNING by a little bit at the end. So the concept map is the best hint type and this is pickled. The final recommendations probabilities of hint types are shown in Table 17.

Table 17: Final recommendation probabilities suggested by the best agent

Best Agent	Concept map	Video hint	Text hint
Expected Value SARSA	0.572	0.080	0.339

Best Hint Type Specific to a Student

I followed the same approach of decreasing epsilon to choose the better of Q-LEARNING and Expected value SARSA for making recommendations specific to a student. Table 18 shows the action counts of best agent for partial students.

Table 18: Final action counts of best agent for sample of individual students

Student	Best agent	Concept map	Video hint	Text hint
s1	Q-learning	1281	242	10477
s2	Q-learning	6423	253	5324
s3	Q-learning	3627	7168	1205

I believe that Q-learning scores the highest mean reward because it has few observations and it is acting greedier when updating Q(S, A) as compared to Expected Value SARSA. So based on the action count, Table 19 shows thebest hint type for partial students.

Table 19: Best hint type for a sample of individual students suggested by the best agent

Student	Best Hint Type
s1	Text hint
s2	Concept map
s3	Video hint

The final recommendation probabilities for best hint type specific to a student are shown in Table 20.

Table 20: Final recommendation probabilities of showing a hint type for a sample of students

Student	Best agent	Concept map	Video hint	Text hint
s1	Q-learning	0.100	0.020	0.873
s2	Q-learning	0.530	0.021	0.440
s3	Q-learning	0.300	0.597	0.100

CONCLUSIONS AND FUTURE WORK

I built a simulated system for an ITS to try out different teaching policies to model student behavior and give them personalized tutoring in the most productive way, i.e., using the least amount of student time. My work showed to what degree reinforcement learning can speed up the development of an ITS by automating the process of giving hints. Each simulated student was given three types of hints, namely text, video and diagrammatic.

I ran the simulation with two Temporal Difference Learning algorithms, namely Q-LEARNING and Expected Value SARSA. Although there were slight differences, both algorithms converged satisfactorily.

I analyzed the output at a global level and also at a per-student level. In each case, the hint type preferred by the algorithm was a function of the range of the two random variables chosen for input, namely the hint delivery type and the student think time after the hint for each hint type.

I am looking forward to additional experiments using more complex reward formulas to better model the tutoring situation, and to testing the system with students taking human anatomy and physiology courses.

REFERENCES

Beck, J.E., Woolf, B.P. and Beal, C.R. 2000. ADVISOR: A Machine Learning Architecture for Intelligent Tutor Construction. In *Proceedings of the 17th National Conference on Artificial Intelligence.*

Evens, M.W. and Michael, J.A. 2006. *One-on-One Tutoring by Humans and Computers.* Psychology Press.

Georgila, K., Core, M.G., Nye, B.D., Karumbaiah, S., Auerbach, D. and Ram, M. 2019. Using Reinforcement Learning to Optimize the Policies of an Intelligent Tutoring System for Interpersonal Skills Training. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*.

Gertner, A.S., Conati, C. and VanLehn, K. 1998. Procedural Help in Andes: Generating Hints Using a Bayesian Network Student Model. In *Proceedings of the 15th National Conference on Artificial Intelligence*.

Jasti, M.S. and Freedman, R. 2019. Using a Simulator to Choose the Best Hints in a Reinforcement Learning-Based Multimodal ITS. In *Proceedings of the 15th International Conference on Intelligent Tutoring Systems.*

Kluga, B., Jasti, M.S., Freedman, R. and Naples, V. 2019. Using a Causal Concept Map Based ITS to Add Intelligence to a Textbook for Human Anatomy. In *First Workshop on Intelligent Textbooks at the 20th International Conference on Artificial Intelligence in Education*.

Magner, U., Schwonke, R. and Renkl, A. 2010. Pictorial Illustrations in Intelligent Tutoring Systems: Do they Distract or Elicit Interest and Engagement? In *Proceedings of the 9th International Conference of the Learning Sciences.*

Malpani, A., Ravindran, B. and Murthy, H. 2011. Personalized Intelligent Tutoring System Using Reinforcement Learning. In *Proceedings of the 24th International FLAIRS Conference*.

Martin, K.N. and Arroyo, I. 2004. AgentX: Using Reinforcement Learning to Improve the Effectiveness of Intelligent Tutoring Systems. In *Proceedings of the 7th International Conference on Intelligent Tutoring Systems.*

Matsuda, N. and Shimmei, M. 2019. Application of Reinforcement Learning For Automated Contents Validation Towards Self-Improving Online Courseware. In *Proceedings of the 7th Annual GIFT Users Symposium.*

Moore, S. and Stamper, J. 2019. Decision Support for an Adversarial Game Environment Using Automatic Hint Generation. In *Proceedings of the 15th International Conference on Intelligent Tutoring Systems.*

Nye, B.D., Graesser, A.C. and Hu, X. 2014. AutoTutor and Family: A Review of 17 Years of Natural Language Tutoring. *International Journal of Artificial Intelligence in Education*, vol. 24, pp. 427-469.

Rowe, J., Smith, A., Spain, R. and Lester, J. 2019. Understanding Novelty in Reinforcement Learning-Based Automated Scenario Generation. In *Proceedings of the 7th Annual GIFT Users Symposium.*

Silva, P., Costa, E. and Araujo, J.R. 2019. An Adaptive Approach to Provide Feedback for Students in Programming Problem Solving. In *Proceedings of the 15th International Conference on Intelligent Tutoring Systems.*

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L. Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D. 2018. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-play. *American Association for the Advancement of Science,* vol. 364, pp. 1140-1144.

Sutton, R.S. and Barto, A.G. 2018. *Reinforcement Learning: An Introduction,* 2/e. MIT Press.

VanLehn, K. 2006. The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education*, vol. 16, pp. 227-265.