CS 182 Final Project
AI Class Scheduler

Raymond Jow

December 9, 2020

1 Introduction

One of the well-known problems in computer science is the scheduling problem, which is a NP-complete problem [4]. In interval scheduling, the problem is to consider a set of tasks which must be completed given disjunctive temporal constraints. The scheduling problem is usually solved using CSP techniques. In my project, I create an automatic scheduler for college students taking classes using a CSP and MDP formulation. The scheduler will utilize certain inputs given by the user such as class difficulty, class times, office hour times, and homework deadlines to determine what action will maximize their productivity.

2 Background and Related Work

In the canonical scheduling problem, a set of jobs $J = j_i, i = 1...n$ consisting of a set of operations $O = o_{mj}$ uses a set of physical resources $M = m_j, j = 1...m$ which must be scheduled according to a routing process [2]. The method for solving this type of problem is to search the solution space by iteratively pruning constraint violations, typically using a backtracking algorithm. A backtracking algorithm traverses the search tree recursively, checking if a solution is consistent with the constraints at each node and choosing to accept or reject a candidate node [5]. Since the domain of possible schedules is very large, the search tree grows exponentially, making the procedure computationally expensive. In order to narrow the set of possible solutions, closure algorithms are used to prune the search space efficiently. These algorithms eliminate local inconsistencies which cannot be part of the global solution. An iterative closure and CSP process is a flexible model that minimizes the number of disjunctive constraints that abides by a set of heuristics. This method is generally called problem reduction, domain filtering, or pruning [2].
In a MDP formulation for production scheduling, several critical factors such as changeover times, production rates, and inventory demand curves are used to determine the value of taking an action. In an optimization open-loop, the goal is to search for a fixed schedule that maximizes average profit over stochastic simulations. However the algorithm is not robust in the sense that it cannot update for variances in real-time production[3].
A more robust formulation of a CSP backtracking algorithm or a MDP Model is to use simulated annealing. Simulated annealing works by initially choosing a random solution, and exploring the state space around it. For the small neighborhood of state spaces surrounding the solution, calculate their utilities. If the neighboring solutions offer a better utility, then the annealing algorithm

will move to it. In doing so, simulated annealing injects randomness to the procedure that allows it to escape local minima or maxima early in the process [1].

3 Problem Specification

I begin by considering the set of jobs, resources, operations, and constraints for a college student taking classes. In my model, there is a set of classes the student is currently taking, labeled $C = c_i, i = 1...n$. For each class, there is homework due on a certain due date t_d^i, which is the task to be completed. The set of operations is the set of actions a student can take $a_i \in \{a_{class}^i, a_{oh}^i, a_{work}^i\}$, which I define as a binary tuple corresponding to going to class, going to office hours, or working on the homework, respectively.
For each operation, there is a certain utility to choosing that action, otherwise known as the efficiency of production. In addition, there are three constraints set by the user: the student's weekly work schedule, the specified total amount of work done in a day and the specified hours of break in between work hours. The weekly schedule is divided into 1 hour time slots. The goal is to choose a weekly schedule that maximizes the efficiency of production by considering all valid actions at each 1 hour interval while satisfying all constraints specified by the user.

4 Approach 1

With the problem specification outlined above, I define a productivity function that determines the efficiency of production of each action. The utility function I define is two-fold:

$$E_i[s_i] = d_i \cdot (c_1 \cdot e^{-t_{class}^i} \cdot c_2 \cdot e^{-t_{oh}^i}) + d_i \tag{1}$$

$$W_i[s_i] = (1 + \sqrt{c_3 \cdot t_{class}^i} + \sqrt{c_4 \cdot t_{oh}^i}) \cdot t_{work}^i \tag{2}$$

The first equation models the expected time to complete the homework, given a state s_i, a class difficulty d_i in the range of 1-10, and constants c_1 and c_2. The second equation models the work done on the homework given a state a_i and constants c_3 and c_4. The state s_i is defined as a length 3 tuple $s_i \in \{t_{class}^i, t_{oh}^i, t_{work}^i\}$ representing the total amount of hours spent in class, the total amount of hours spent in office hours, and the total amount of hours spent working on the homework, respectively.

The intuition behind the equations is as follows. There is a diminishing rate of return for attending class or office hours, modeled by the inverse exponential and square root functions. The factor $1 + \sqrt{c_3 \cdot t_{class}^i} + \sqrt{c_4 \cdot t_{oh}^i}$ in equation 2 is the efficiency of work factor. If no office hours or class are attended, the efficiency of work is 1, meaning that 1 hour of work contributes to 1 hour of work done on the homework assignment. With an increase in time spent in class or time spent in office hours, the productivity of work factor increases, meaning that 1 hour of work contributes to greater than 1 hour of work done on the homework assignment. Once $W_i \geq E_i$, the efficiency of production becomes zero, because the homework is complete.

I define a TimeSlot class for each 1 hour time slot. The weekly schedule is a list of lists where each day is a list of TimeSlots. The TimeSlot class calculates the utility of all actions using the following equation.

$$V[s_i] = \max_{a_i}(w_i \cdot e^{-|t_d^i - t|/168} \cdot (\delta E_i[a_i] + \delta W_i[a_i])) \tag{3}$$

w_i represents the weight of each class, and $\delta E_i[a_i]$ and $\delta W_i[a_i]$ represent the change in $E_i[s_i]$ and $W_i[s_i]$ for taking an action a_i.

In the problem reduced version of my backtracking algorithm, instead of considering all possible schedules that satisfy constraints and choosing the one that maximizes total utility, I iteratively choose the TimeSlot with the maximum utility, and reject candidate TimeSlots that do not satisfy constraints. Although this algorithm does not satisfy completeness, it narrows down the domain of possible solutions considerably and arrives at a reasonable solution.

Algorithm 1 Backtracking Algorithm

```
procedure BACKTRACK($workSchedule, constraints$)
    $breakHours, totalWorkHours = constraints$
    if len($workSchedule$) == 0 then
        return []
    else
        $maxTimeslot$ = MAXIMIZEUTILITY($workSchedule$)
        UPDATE($maxTimeslot$)
        REMOVE($workSchedule, maxTimeslot$)
        $workScheduleCopy \leftarrow$ COPY($workSchedule$)
        for $timeSlot$ in $workScheduleCopy$ do
            if $|timeSlot.time - maxTimeslot.time| \leq breakHours$ then
                REMOVE($workSchedule, timeSlot$)
            end if
        end for
    end if
return ([maxTimeslot] + BACKTRACK(workSchedule, constraints))[:$totalWorkHours$]
end procedure
```

5 Approach 2

In the second iteration of the AI Scheduler, I reconstruct a student's work schedule as purely a Markov Decision Process (MDP). In the MDP Model, a student's state in a class is defined as $S_{ijk} = \{f_i, p_j, w_k\}, i = 0...10, j = 0...10 \cdot d_i, k = 0...100$ where $i, j, k \in \mathbb{Z}$ represent the fatigue level, the proficiency level, and the percent work done as a discrete set of states, respectively. The possible actions are $a_i \in \{a^i_{class}, a^i_{oh}, a^i_{work}, a^i_{rest}\}$. The transition function $T(s, a, s')$ is defined as the following:

$$T(s_i, a_{class}, s'_j) = \begin{cases} f'_j = min(10, f_i + c_1) \\ p'_j = min(10 \cdot d_i, p_i + c_2) \\ w'_j = w_i \end{cases}$$

$$T(s, a_{oh}, s') = \begin{cases} f'_j = min(10, f_i + c_3) \\ p'_j = min(10 \cdot d_i, p_i + c_4) \\ w'_j = w_i \end{cases}$$

$$T(s, a_{work}, s') = \begin{cases} f'_j = min(10, f_i + c_5) \\ p'_j = min(10 \cdot d_i, p_i + c_6) \\ w'_j = min(100, w_i + \lfloor c_7 \cdot \frac{p_i}{10 \cdot d_i} \cdot \frac{10 - f_i}{10} \rfloor) \end{cases}$$

$$T(s, a_{rest}, s') = \begin{cases} f'_j = max(0, f_i - c_8) \\ p'_j = p_i \\ w'_j = w_i \end{cases}$$

Additionally, a Reward function $R(s, a)$ is defined as:

$$R(s, a^i) = \Delta w_i = w'_i - w_i \tag{4}$$

Once value iteration is completed, the timeSlot class calculates the utility of all valid actions using the following equation.

$$V[s_i] = \max_{a_i}(w_i \cdot (R(s, a_i) + \frac{|t_d^i - t|}{168} \cdot Q[s'])) \tag{5}$$

The factor $\frac{|t_d^i - t|}{168}$ is equivalent to the gamma value γ in the Markov Chain Model, and signifies the discount rate of future rewards. The closer the timeSlot is to the deadline, the less value is given to future rewards.

Algorithm 2 Value Iteration

```
procedure ValueIteration($Q, actions, gamma, epsilon$)
    while True do
        $oldQ \leftarrow \text{Copy}(Q)$
        $maxUpdate = 0$
        $stateSpace = [(i, j, k)$ for $i = 0...10, j = 0...10 \cdot d_i, k = 0...100]$
        for $s$ in $stateSpace$ do
            $V \leftarrow -\infty$
            for $a$ in $actions$ do
                $s' \leftarrow \mathrm{T}(s, a)$
                $Q \leftarrow \mathrm{R}(s, a) + gamma \cdot oldQ[s']$
                if $Q > V$ then
                    $V \leftarrow Q$
                end if
                $Q[s] \leftarrow V$
                $maxUpdate \leftarrow \text{max}(maxUpdate, |oldQ[s'] - V|)$
            end for
            if $maxUpdate < epsilon$ then
                return Q
            end if
        end for
    end while
end procedure
```

6 Experiments

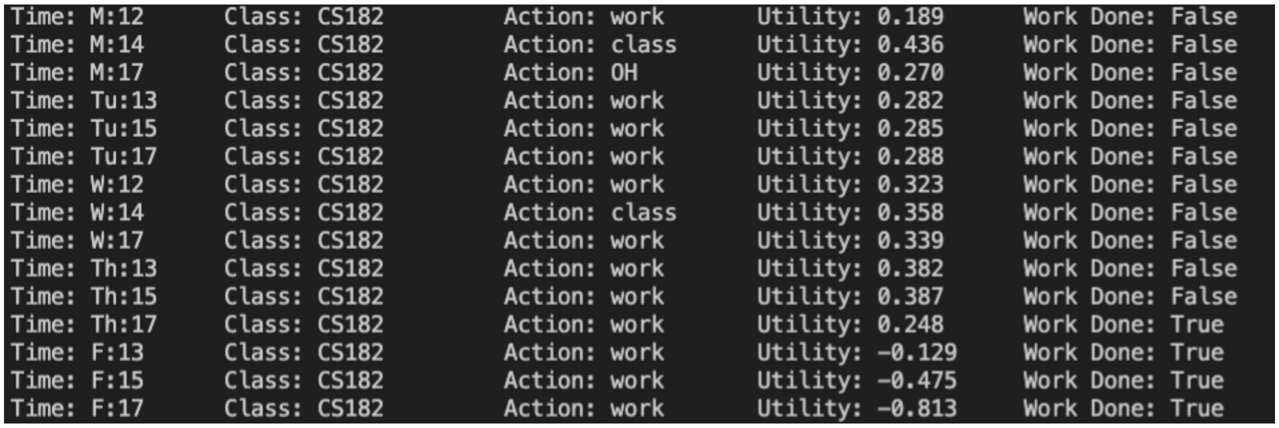
The Scheduler demonstrated in approach 1, the CSP Formulation, utilizes backtracking search with a problem reduction. By iteratively selecting the maximum utility action for a timeslot and removing candidate solutions that do not satisfy the constraints, I implement a limited-complete closure process that selectively removes constraint violations. By doing so, the optimal solution in which the total utility of a schedule is maximized is not guaranteed. This is due to the fact that the backtracking search algorithm only considers the max utility at a particular time, and does not consider future rewards as a result of the max action. However, the soundness of the backtracking procedure guarantees that the solution found will satisfy the break hours, total work hours, and work schedule constraints set by the user.

The Scheduler demonstrated in approach 2 aims to reconstruct the break hour constraint set by the user into a MDP model. This factor is introduced using the fatigue state, a discrete value, and the introduction of the action rest in the action set. The level of fatigue diminishes the reward of working, introducing a trade off between immediate reward and future reward. Additionally, by introducing a MDP model, the state space spans the entire domain. Therefore, by performing value iteration and determining the utility of each state, the solution given by the algorithm is the optimal solution, given the parameters. The tradeoff of the MDP Model is that value iteration is computationally expensive, growing polynomially and with the maximum iterations bounded by $\frac{1}{1-\gamma}$.

Without data about a student's work schedule and a reinforcement learning model, it is impossible to empirically evaluate the algorithms presented in approach 1 and approach 2. However, test cases for a student's work schedule can be used to determine if the algorithm's generated output makes logical sense, based on human intuition. I consider the basic case of one class, a second case with the same class but with different inputs, and finally three classes with varying inputs, comparing the output of both algorithms. These examples will illustrate the decision process of each algorithm.

6.1 Results

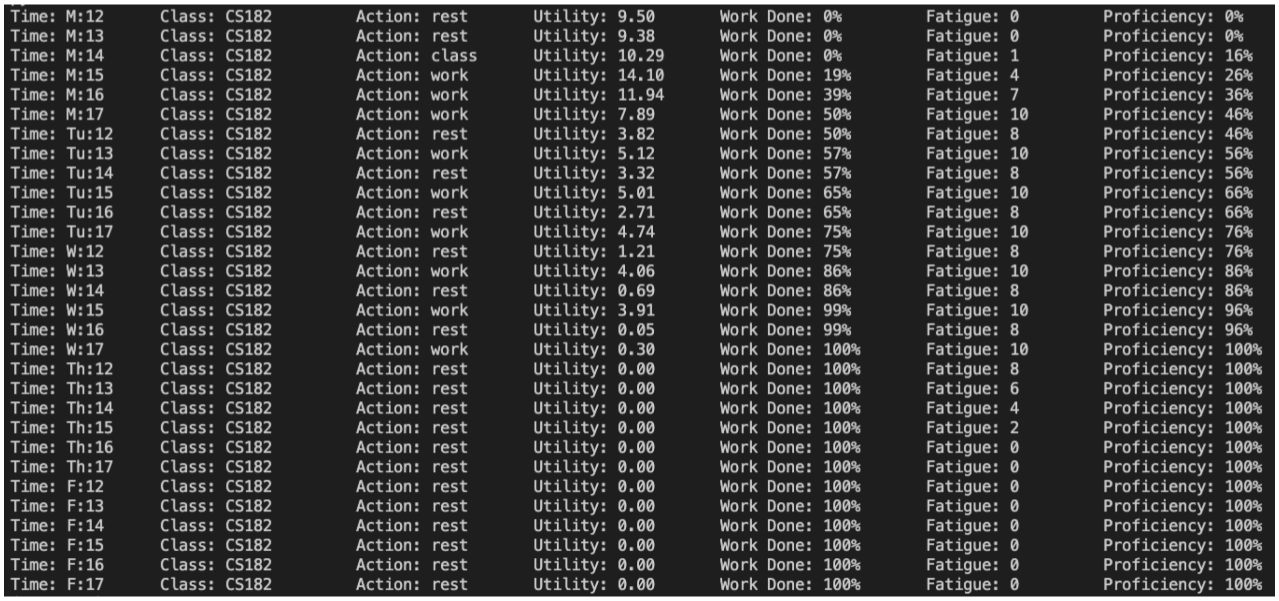
In the first test case, I consider a single class with class hours Monday, Wednesday, and Friday from 2-3 pm, office hours Monday and Wednesday from 3-7 pm, the homework deadline at Thursday 6 pm, and a homework weight of 0.3 given by ['CS182','MWF:14-15','MW:15-19',5,'Th:18',0.30]. Additionally, for the CSP, there is a hard constraint of 1 break hour between work and less than a total of 5 hours of work per day. The schedule returned by the CSP is shown below.



```
Time: M:12      Class: CS182        Action: work      Utility: 0.189      Work Done: False
Time: M:14      Class: CS182        Action: class     Utility: 0.436      Work Done: False
Time: M:17      Class: CS182        Action: OH        Utility: 0.270      Work Done: False
Time: Tu:13     Class: CS182        Action: work      Utility: 0.282      Work Done: False
Time: Tu:15     Class: CS182        Action: work      Utility: 0.285      Work Done: False
Time: Tu:17     Class: CS182        Action: work      Utility: 0.288      Work Done: False
Time: W:12      Class: CS182        Action: work      Utility: 0.323      Work Done: False
Time: W:14      Class: CS182        Action: class     Utility: 0.358      Work Done: False
Time: W:17      Class: CS182        Action: work      Utility: 0.339      Work Done: False
Time: Th:13     Class: CS182        Action: work      Utility: 0.382      Work Done: False
Time: Th:15     Class: CS182        Action: work      Utility: 0.387      Work Done: False
Time: Th:17     Class: CS182        Action: work      Utility: 0.248      Work Done: True
Time: F:13      Class: CS182        Action: work      Utility: -0.129     Work Done: True
Time: F:15      Class: CS182        Action: work      Utility: -0.475     Work Done: True
Time: F:17      Class: CS182        Action: work      Utility: -0.813     Work Done: True
```

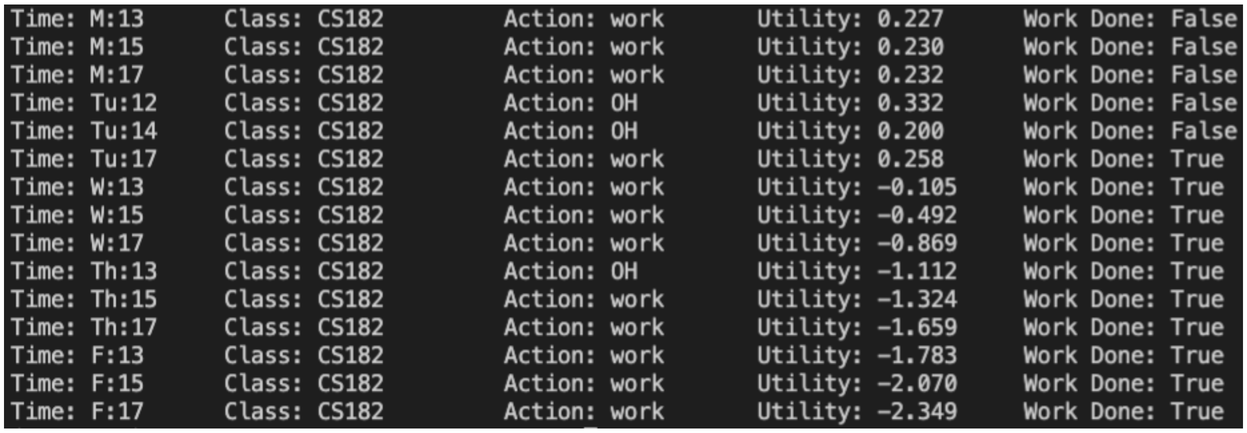
The important features of the CSP schedule are the following. The hard constraints set by the user is always satisfied, because the backtracking algorithm eliminates candidate solutions that do not meet the constraints. Furthermore, there is spike in utility of work after class and office hours, as mentioned earlier as the efficiency factor in the work equation $W_i[s_i]$. Once the work done switches to true, the utility becomes negative, which is intuitive in the sense that a student would want to switch to working on other assignments once one is completed.

The MDP model provides the following schedule.


```
Time: M:12     Class: CS182     Action: rest     Utility: 9.50     Work Done: 0%       Fatigue: 0      Proficiency: 0%
Time: M:13     Class: CS182     Action: rest     Utility: 9.38     Work Done: 0%       Fatigue: 0      Proficiency: 0%
Time: M:14     Class: CS182     Action: class    Utility: 10.29    Work Done: 0%       Fatigue: 1      Proficiency: 16%
Time: M:15     Class: CS182     Action: work     Utility: 14.10    Work Done: 19%      Fatigue: 4      Proficiency: 26%
Time: M:16     Class: CS182     Action: work     Utility: 11.94    Work Done: 39%      Fatigue: 7      Proficiency: 36%
Time: M:17     Class: CS182     Action: work     Utility: 7.89     Work Done: 50%      Fatigue: 10     Proficiency: 46%
Time: Tu:12    Class: CS182     Action: rest     Utility: 3.82     Work Done: 50%      Fatigue: 8      Proficiency: 46%
Time: Tu:13    Class: CS182     Action: work     Utility: 5.12     Work Done: 57%      Fatigue: 10     Proficiency: 56%
Time: Tu:14    Class: CS182     Action: rest     Utility: 3.32     Work Done: 57%      Fatigue: 8      Proficiency: 56%
Time: Tu:15    Class: CS182     Action: work     Utility: 5.01     Work Done: 65%      Fatigue: 10     Proficiency: 66%
Time: Tu:16    Class: CS182     Action: rest     Utility: 2.71     Work Done: 65%      Fatigue: 8      Proficiency: 66%
Time: Tu:17    Class: CS182     Action: work     Utility: 4.74     Work Done: 75%      Fatigue: 10     Proficiency: 76%
Time: W:12     Class: CS182     Action: rest     Utility: 1.21     Work Done: 75%      Fatigue: 8      Proficiency: 76%
Time: W:13     Class: CS182     Action: work     Utility: 4.06     Work Done: 86%      Fatigue: 10     Proficiency: 86%
Time: W:14     Class: CS182     Action: rest     Utility: 0.69     Work Done: 86%      Fatigue: 8      Proficiency: 86%
Time: W:15     Class: CS182     Action: work     Utility: 3.91     Work Done: 99%      Fatigue: 10     Proficiency: 96%
Time: W:16     Class: CS182     Action: rest     Utility: 0.05     Work Done: 99%      Fatigue: 8      Proficiency: 96%
Time: W:17     Class: CS182     Action: work     Utility: 0.30     Work Done: 100%     Fatigue: 10     Proficiency: 100%
Time: Th:12    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 8      Proficiency: 100%
Time: Th:13    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 6      Proficiency: 100%
Time: Th:14    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 4      Proficiency: 100%
Time: Th:15    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 2      Proficiency: 100%
Time: Th:16    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: Th:17    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: F:12     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: F:13     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: F:14     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: F:15     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: F:16     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
Time: F:17     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%     Fatigue: 0      Proficiency: 100%
```

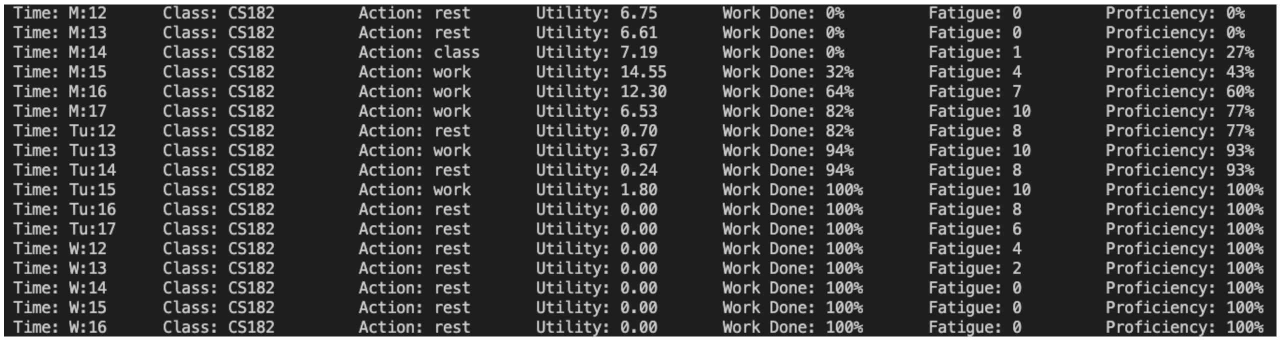
As shown by the MDP Model, instead of a hard constraint that eliminates consecutive hours of work, the MDP Scheduler has a fatigue state that determines if the student should rest or work. With the constants I have entered in the model, the MDP Scheduler decides that the maximum utility is to work until the maximum level of fatigue is reached (10), and then alternates between rest and work, giving the same sequence of actions as the CSP Scheduler. However, the MDP model is taking into account the future utility of decreasing fatigue to maximize productivity due to value iteration, while the CSP is simply eliminating the timeslot. This makes the MDP model more robust in deciding which action to take.

In the second test case, I consider the same class with some changes. I change office hours to Tuesday and Thursday from 12-3 pm, change the homework deadline to Wednesday at 12 pm, and decrease the difficulty of the class to 3, which corresponds with ['CS182','MWF:14-15','TuTh:12-15',3,'W:12',0.30]. After making these changes, the CSP produced to following output.


```
Time: M:13     Class: CS182     Action: work     Utility: 0.227     Work Done: False
Time: M:15     Class: CS182     Action: work     Utility: 0.230     Work Done: False
Time: M:17     Class: CS182     Action: work     Utility: 0.232     Work Done: False
Time: Tu:12    Class: CS182     Action: OH       Utility: 0.332     Work Done: False
Time: Tu:14    Class: CS182     Action: OH       Utility: 0.200     Work Done: False
Time: Tu:17    Class: CS182     Action: work     Utility: 0.258     Work Done: True
Time: W:13     Class: CS182     Action: work     Utility: -0.105    Work Done: True
Time: W:15     Class: CS182     Action: work     Utility: -0.492    Work Done: True
Time: W:17     Class: CS182     Action: work     Utility: -0.869    Work Done: True
Time: Th:13    Class: CS182     Action: OH       Utility: -1.112    Work Done: True
Time: Th:15    Class: CS182     Action: work     Utility: -1.324    Work Done: True
Time: Th:17    Class: CS182     Action: work     Utility: -1.659    Work Done: True
Time: F:13     Class: CS182     Action: work     Utility: -1.783    Work Done: True
Time: F:15     Class: CS182     Action: work     Utility: -2.070    Work Done: True
Time: F:17     Class: CS182     Action: work     Utility: -2.349    Work Done: True
```

The interesting observation here is that when the class difficulty is low, the CSP determines that it is not worth going to class and instead starts to work on the assignment immediately. The action of office hours is taken, which leads to an increase in utility, but by Tuesday at 5 pm, the CS182 assignment is completed.

The MDP Scheduler outputs a slightly different schedule.

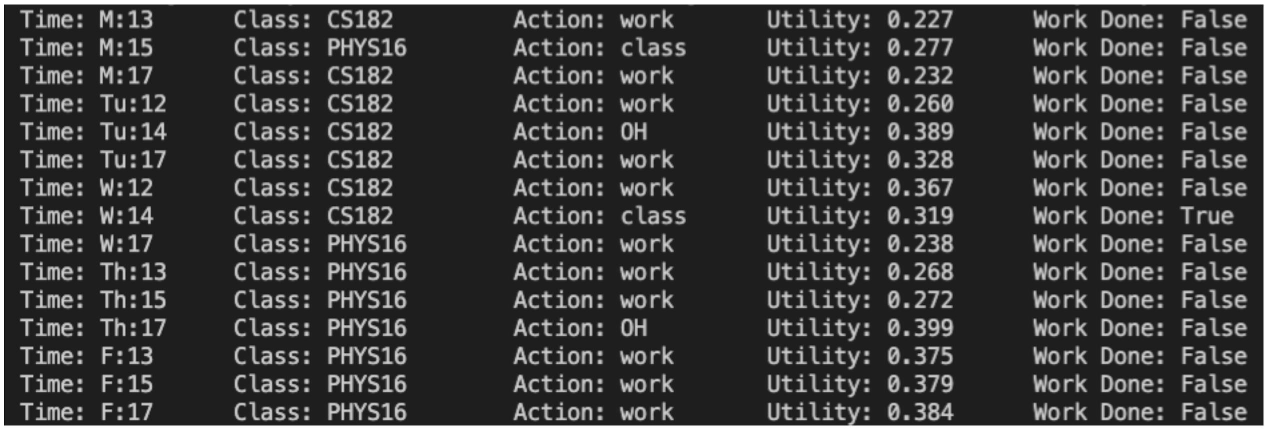


```
Time: M:12     Class: CS182     Action: rest     Utility: 6.75     Work Done: 0%      Fatigue: 0      Proficiency: 0%
Time: M:13     Class: CS182     Action: rest     Utility: 6.61     Work Done: 0%      Fatigue: 0      Proficiency: 0%
Time: M:14     Class: CS182     Action: class    Utility: 7.19     Work Done: 0%      Fatigue: 1      Proficiency: 27%
Time: M:15     Class: CS182     Action: work     Utility: 14.55    Work Done: 32%     Fatigue: 4      Proficiency: 43%
Time: M:16     Class: CS182     Action: work     Utility: 12.30    Work Done: 64%     Fatigue: 7      Proficiency: 60%
Time: M:17     Class: CS182     Action: work     Utility: 6.53     Work Done: 82%     Fatigue: 10     Proficiency: 77%
Time: Tu:12    Class: CS182     Action: rest     Utility: 0.70     Work Done: 82%     Fatigue: 8      Proficiency: 77%
Time: Tu:13    Class: CS182     Action: work     Utility: 3.67     Work Done: 94%     Fatigue: 10     Proficiency: 93%
Time: Tu:14    Class: CS182     Action: rest     Utility: 0.24     Work Done: 94%     Fatigue: 8      Proficiency: 93%
Time: Tu:15    Class: CS182     Action: work     Utility: 1.80     Work Done: 100%    Fatigue: 10     Proficiency: 100%
Time: Tu:16    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 8      Proficiency: 100%
Time: Tu:17    Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 6      Proficiency: 100%
Time: W:12     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 4      Proficiency: 100%
Time: W:13     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 2      Proficiency: 100%
Time: W:14     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 0      Proficiency: 100%
Time: W:15     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 0      Proficiency: 100%
Time: W:16     Class: CS182     Action: rest     Utility: 0.00     Work Done: 100%    Fatigue: 0      Proficiency: 100%
```

Initially, the MDP Scheduler chooses to take the action class, which leads to a large jump in proficiency. Since proficiency increases the reward for work, the MDP Scheduler recommends to go to class and then work. The future reward of going to class is shown by the large spike in utility of the actions work following the action class.

In the third test case, I consider 3 classes to show a more complex example of each Scheduler's algorithm. The class data is the following:
class1 = ['PHYS16','MWF:12-16','MWTh:13-18',6,'F:18',0.30]
class2 = ['CS182','MWF:14-15','TuTh:12-15',4,'W:12',0.30]
class3 = ['GENED1023','TuTh:12-14','MW:12-13',1,'W:18',0.15]

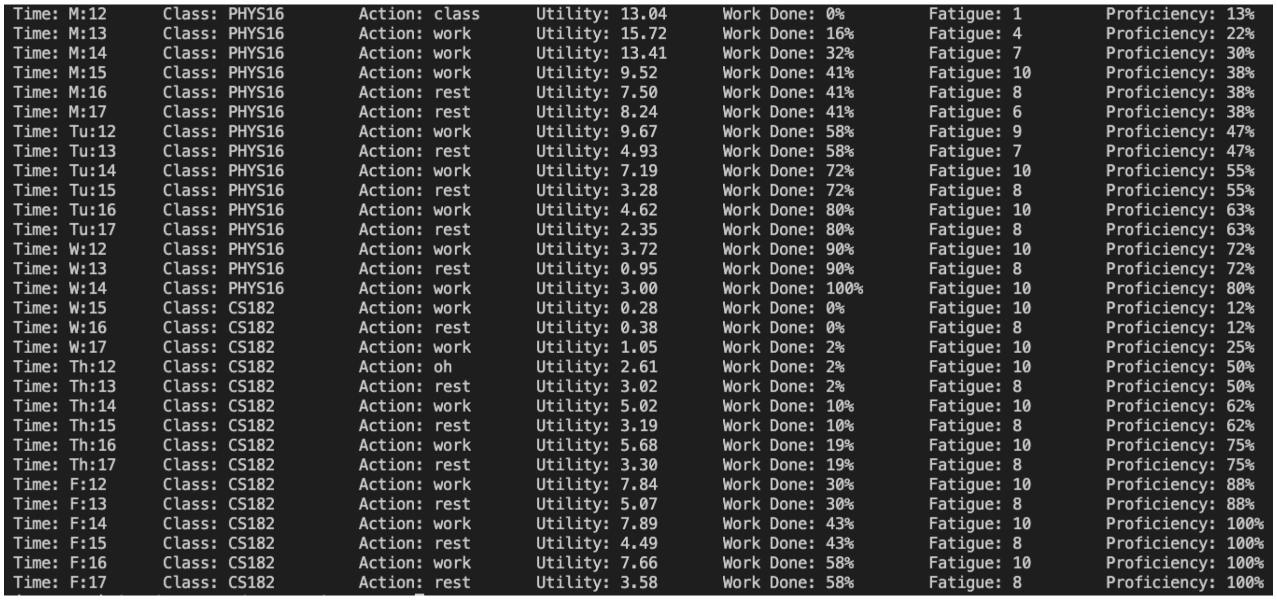
The CSP Scheduler returns the following schedule.



```
Time: M:13      Class: CS182      Action: work      Utility: 0.227      Work Done: False
Time: M:15      Class: PHYS16     Action: class     Utility: 0.277      Work Done: False
Time: M:17      Class: CS182      Action: work      Utility: 0.232      Work Done: False
Time: Tu:12     Class: CS182      Action: work      Utility: 0.260      Work Done: False
Time: Tu:14     Class: CS182      Action: OH        Utility: 0.389      Work Done: False
Time: Tu:17     Class: CS182      Action: work      Utility: 0.328      Work Done: False
Time: W:12      Class: CS182      Action: work      Utility: 0.367      Work Done: False
Time: W:14      Class: CS182      Action: class     Utility: 0.319      Work Done: True
Time: W:17      Class: PHYS16     Action: work      Utility: 0.238      Work Done: False
Time: Th:13     Class: PHYS16     Action: work      Utility: 0.268      Work Done: False
Time: Th:15     Class: PHYS16     Action: work      Utility: 0.272      Work Done: False
Time: Th:17     Class: PHYS16     Action: OH        Utility: 0.399      Work Done: False
Time: F:13      Class: PHYS16     Action: work      Utility: 0.375      Work Done: False
Time: F:15      Class: PHYS16     Action: work      Utility: 0.379      Work Done: False
Time: F:17      Class: PHYS16     Action: work      Utility: 0.384      Work Done: False
```

In this scenario, there is no time to complete all assignments by the deadline, so the CSP scheduler chooses to work on the PHYS16 and CS182 homework first. This makes intuitive sense given that the homework weight is greater for these two classes. Another point of observation is that the work done switches to true after the action class is taken on Wednesday 2 pm. This is inconsistent with a real life model, because a student wouldn't be able to complete homework by attending class, but since the efficiency factor in $W_i[s_i]$ is multiplied by all preceding hours of work, an increase in class time also increases the work done.

The MDP Scheduler returns the following schedule.



```
Time: M:12     Class: PHYS16     Action: class     Utility: 13.04     Work Done: 0%       Fatigue: 1      Proficiency: 13%
Time: M:13     Class: PHYS16     Action: work      Utility: 15.72     Work Done: 16%      Fatigue: 4      Proficiency: 22%
Time: M:14     Class: PHYS16     Action: work      Utility: 13.41     Work Done: 32%      Fatigue: 7      Proficiency: 30%
Time: M:15     Class: PHYS16     Action: work      Utility: 9.52      Work Done: 41%      Fatigue: 10     Proficiency: 38%
Time: M:16     Class: PHYS16     Action: rest      Utility: 7.50      Work Done: 41%      Fatigue: 8      Proficiency: 38%
Time: M:17     Class: PHYS16     Action: rest      Utility: 8.24      Work Done: 41%      Fatigue: 6      Proficiency: 38%
Time: Tu:12    Class: PHYS16     Action: work      Utility: 9.67      Work Done: 58%      Fatigue: 9      Proficiency: 47%
Time: Tu:13    Class: PHYS16     Action: rest      Utility: 4.93      Work Done: 58%      Fatigue: 7      Proficiency: 47%
Time: Tu:14    Class: PHYS16     Action: work      Utility: 7.19      Work Done: 72%      Fatigue: 10     Proficiency: 55%
Time: Tu:15    Class: PHYS16     Action: rest      Utility: 3.28      Work Done: 72%      Fatigue: 8      Proficiency: 55%
Time: Tu:16    Class: PHYS16     Action: work      Utility: 4.62      Work Done: 80%      Fatigue: 10     Proficiency: 63%
Time: Tu:17    Class: PHYS16     Action: rest      Utility: 2.35      Work Done: 80%      Fatigue: 8      Proficiency: 63%
Time: W:12     Class: PHYS16     Action: work      Utility: 3.72      Work Done: 90%      Fatigue: 10     Proficiency: 72%
Time: W:13     Class: PHYS16     Action: rest      Utility: 0.95      Work Done: 90%      Fatigue: 8      Proficiency: 72%
Time: W:14     Class: PHYS16     Action: work      Utility: 3.00      Work Done: 100%     Fatigue: 10     Proficiency: 80%
Time: W:15     Class: CS182      Action: work      Utility: 0.28      Work Done: 0%       Fatigue: 10     Proficiency: 12%
Time: W:16     Class: CS182      Action: rest      Utility: 0.38      Work Done: 0%       Fatigue: 8      Proficiency: 12%
Time: W:17     Class: CS182      Action: work      Utility: 1.05      Work Done: 2%       Fatigue: 10     Proficiency: 25%
Time: Th:12    Class: CS182      Action: oh        Utility: 2.61      Work Done: 2%       Fatigue: 10     Proficiency: 50%
Time: Th:13    Class: CS182      Action: rest      Utility: 3.02      Work Done: 2%       Fatigue: 8      Proficiency: 50%
Time: Th:14    Class: CS182      Action: work      Utility: 5.02      Work Done: 10%      Fatigue: 10     Proficiency: 62%
Time: Th:15    Class: CS182      Action: rest      Utility: 3.19      Work Done: 10%      Fatigue: 8      Proficiency: 62%
Time: Th:16    Class: CS182      Action: work      Utility: 5.68      Work Done: 19%      Fatigue: 10     Proficiency: 75%
Time: Th:17    Class: CS182      Action: rest      Utility: 3.30      Work Done: 19%      Fatigue: 8      Proficiency: 75%
Time: F:12     Class: CS182      Action: work      Utility: 7.84      Work Done: 30%      Fatigue: 10     Proficiency: 88%
Time: F:13     Class: CS182      Action: rest      Utility: 5.07      Work Done: 30%      Fatigue: 8      Proficiency: 88%
Time: F:14     Class: CS182      Action: work      Utility: 7.89      Work Done: 43%      Fatigue: 10     Proficiency: 100%
Time: F:15     Class: CS182      Action: rest      Utility: 4.49      Work Done: 43%      Fatigue: 8      Proficiency: 100%
Time: F:16     Class: CS182      Action: work      Utility: 7.66      Work Done: 58%      Fatigue: 10     Proficiency: 100%
Time: F:17     Class: CS182      Action: rest      Utility: 3.58      Work Done: 58%      Fatigue: 8      Proficiency: 100%
```

As previously seen by the CSP Schedule, the MDP Schedule also determines that there is no time to complete all assignments by the deadline, so the MDP Scheduler chooses to work on PHYS16 homework and then CS182 homework. The reason for the MDP Scheduler's decision is that the class difficulty greatly contributes to the proficiency and in turn the reward. The MDP Model uses the factor $\frac{|t_d^i - t|}{168}$ as the urgency factor, which does not alter the utility as much as the $e^{-|t_d^i - t|/168}$ factor in the CSP model, so the MDP Model incorrectly chooses to continue working on PHYS16, greatly diminishing the utility of actions later on.

7 Discussion

From analyzing the three test cases, there are decision-making qualities of each approach that make it a sensible model. However, without an empirical metric of a good schedule versus a bad one, a better scheduler cannot be determined. Nonetheless, it can be concluded that the Markov Decision Model provides a better framework for introducing reinforcement learning, since the state space of fatigue, proficiency, and work done values are observables that can be tracked and corrected by a mean squared error or quadratic loss function using user data on their work schedule and homework grades. The natural step towards a reinforcement learning model would be to set the constants of the transition function and the reward function as weights in a neural network and implement stochastic gradient descent. The homework grades and the percent of the problem set completed naturally correlate to the proficiency and work done in a class, so optimization is feasible. However, another issue is the low volume of data available. For example, a student typically has around 5 to 10 homework assignments for a single class per semester, so the machine learning algorithm would only have sparse data points to train on. Other future work would include extending the complexity of the Markov Decision Model to include urgency as part of the state space. Another approach is to incorporate simulated annealing to the CSP model and MDP model, giving it the robustness to escape local maxima.

A System Description

Appendix 1 – Detailed instructions on how to run the program are provided in the README.md file on my github. Github link: https://github.com/raymondj80/AI-Class-Scheduler

B Group Makeup

Appendix 2 – I worked on the project alone.

References

[1] Esra Aycan and Tolga Ayav. Solving the course scheduling problem using simulated annealing. In *2009 IEEE International Advance Computing Conference*, pages 462–466. IEEE, 2009.

[2] María Isabel Alfonso Galipienso and Federico Barber Sanchís. A mixed closure-csp method to solve scheduling problems. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 559–570. Springer, 2001.

[3] Jeff G Schneider, Justin A Boyan, and Andrew W Moore. Value function based production scheduling. In *ICML*, pages 522–530, 1998.

[4] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[5] Peter Van Beek. Backtracking search algorithms. In *Foundations of artificial intelligence*, volume 2, pages 85–134. Elsevier, 2006.