Conquering Sequence:
How I learned to stop worrying and love the abstracted state space
Presentation link: https://youtu.be/nD0Kii5JU14

1. Problem formation

The field of Artificial Intelligence (AI) has achieved recent success in perfect information games, achieving super-human level performance in Chess and Go. [1] However, these approaches have struggled with real world decision making, which occurs with imperfect information and vast state spaces. [1] Such problems are more similar to Sequence, a multiplayer turn based board game.

Sequence has *partial state observability,* with visible information – your hand, the board, draft pile, discard pile - and hidden information – each other players hand, and the deck. The effect of actions are *uncertain*, given three agents take turns before the players next turn. This results in both the board and draft cards being drastically different at the players next turn. It can be naturally modelled as *partially observable Markov decision process* (POMDP).

A fundamental challenge of Sequence is the utter size of the state space, with approximately 6.36e+45 possible board states multiplied by a further 3.07e+24 hand combinations. Further, Sequence also has a similar problem to other classical games with minimal rewards, with the formation of a sequence occurring at most twice, and an overall victory at most once. The scale of Sequence demands a problem transformation. Historically, agents have attempted to abstract the state space creating a "many-to-one mapping" to a smaller tractable problem. [2] However, such abstractions while reducing complexity, can also result in ineffective agent behaviour. [2]

In this vein, we created multiple abstracted models to derive agent actions. The first model, '*Classical Sequence',* creates a classical planning problem by ignoring all hidden information to turn a set of belief states into a single state, and ignoring all other players actions to create a deterministic transition function. Another model, '*MDP Sequence',* attempts to reintroduce the effect of other agents by creating a probabilistic transition function to simulate the moves of other player. The models were derived solely using the provided Sequence simulator and manipulating the hand and board attributes. While, this was easy to implement, it proved computationally onerous requiring multiple copies of the game state. This limited the number of nodes expanded by a search algorithm in one turn to approximately 400 nodes before timing out. Given the high branching factor which, on average is approximately 50, this limited expansion proved inadequate to implement many search strategies for the tournament, and was a continuous challenge in agent development.

2. Discarded approaches

Even *Classical Sequence* model has a state space of approximately 6.36e+45 possible board states. There is a sparsity of rewards, with a low chance of encountering identical game states. Any AI approach that used a table to store Q-values was immediately excluded including value iteration, and model free approaches such as Q-learning. While, the state space does reduce once the game has begun, it was the team's impression there would be insufficient time for reasonable convergence with such a strategy. A PDDL representation of the *Classical Sequence* was excluded due to the complexity of representation.

3. Pursued approaches

3.1 Value approximation

3.1.1 Logical state value approximation

We created a logic based state approximation function founded on the insight that the *Sequence* board is composed of a set of overlapping groups of 5 of spaces that each represent a possible sequence. The more pieces a player had in a potential set, the more exponentially valuable it is. However, if the set has even a single opponents piece it is no

longer a potential sequence and is worthless as shown in *Figure 1*. The centre pieces are a special extension of this rule as a potential sequence with 4 spaces worth twice the value.

	*The full row*
	*A group of five that is a possible sequence for red*
	*A group of five that cannot be a sequence for either player*
Figure 1. Decomposition of a row in Sequence	

This valuation naturally prizes spaces in a more central location as they are part of more potential sequences. A score derived from iterating over all possible sequence is normalised over the total score for both teams, returning a value between 0 and 1, believed to be indicative of board favourability. While the state value approximation is of polynomial complexity, to be used as a Q-value, a state would need be expanded with the action, and the subsequent board evaluated, which is computationally expensive.

3.1.2 Deep-Q valuation

A key component of the AlphaGo agent was the approximate state value function derived using a deep neural network (DNN) from a set of self-play games. [1] This approach was thought unfeasible given our limitations in compute and training instances. However, after feature extraction for linear approximation failed as we were unable to derive any higher dimensional representation, we resorted to creating a DeepQ value approximation function.

We used 200,000 training instances generated using self-play between our best agents, omitting random play as unnecessary noise. [3] The instances were represented by a feature vector of dimension (10,10,3) to represent the board and a play or remove action, and labelled with the outcome of the game. The model was trained with a standard image analysis neural network RESNET using Pytorch. [4, 5] In testing the model achieved a surprising 82% accuracy to the predict match outcome given a state-action pair. Unfortunately, the team was unable to implement this Q-approximation function due to time.

3.2 Classical planning model agents

3.2.1 Greedy-Q agent.

The most obvious application of our rule based approach was an agent that considered all actions and greedily selected highest Q-value. However, the action space frequently exceed 400 and the agent would timeout. Further, as the draft card selected did not immediately affect the board, an irrelevant draft card was routinely selected. This was improved by separating the card played and the card drafted. The agent greedily selecting the card that resulted in the best board state, then selected the best draft card given that subsequent board state. A possible weakness of this approach is the inability to see how cards relate to each other.

Observation of the games yielded insight that allowed logic rules to guide greedy approaches. Greedy-Q agents played a wild card the instant it was the best option, despite a non-wild card potentially offering a comparable move. We compared wild card and non-wild cards move and defined a threshold that must be exceeded for the wild card to be played.

3.2.2 Breadth First Search Agent.

To consider how cards in one players hand may impact each other, a breadth first search (BFS) algorithm was implemented, with the agent playing cards in their hand successively and returning the path with the highest board valuation at the terminal node. Initially draft cards were included, but due to the increased complexity and the observation that draft

cards are highly changeable between turns, they were dropped. This agent was unable to explore any depth beyond 1 within the time limit and was not viable for the tournament.

3.2.3 A* search algorithm

In a further attempt to reduce the search space, an A* search agent used a simple admissible goal-aware heuristic based on the minimum number of pieces a player was from a win. Given most hands would not contain sufficient cards to reach this state, this approach was also terminated at a certain depth, returning the furthest node explored.

3.3 MDP

3.3.1 Monte Carlo Tree Search Agent

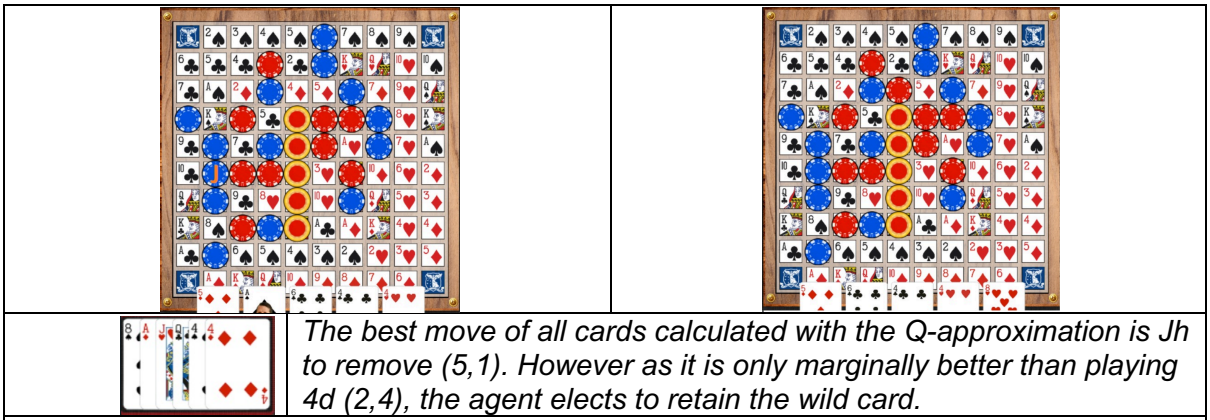
All agents thus far have not explicitly considered the action of any opponent. In order to approximate an opponent's action a probabilistic transition function was defined. All possible opponent moves were generated, ranked with logic Q-value function and the top 5 moves were returned with uniform probability. While this generated realistic moves, the process was computationally intensive.

To traverse a wide state space Monte Carlo Tree Search was used. To improve convergence the probability of the child selected was initialised to the normalised approximate state value. The inefficiency of the probabilistic action function was immediately felt with the agent taking a long time to even complete even 5 iterations. Without a meaningful number of iterations, the simulations proved largely useless, with different multi-arm bandit methodology predictably having no effect. This method was abandoned even as a bench mark due to its extraordinary lengthy iterations.

4. Agent Selection

The choice of agent to submit for the final tournament was limited severely by the time limit of each turn. Of the agents we developed only the Greedy-Q and A* were able to return an action in an appropriate time. Our results are the average of 200 games, with agents switching colour midway, to mitigate the advantage of playing first.

The results of A* were poor, with a win rate of less 20% against all variations of the Greedy-Q agents, reflecting the first terminal state reached with the heuristic was in no way optimal. The first Greedy-Q agent that didn't consider the draft card, was systematically beaten approximately 98% of the time by a similar agent that did greedily choose the draft card. This subsequent agent was then beaten 60% of time by a Greedy-Q agent that would hold their wild card until a move was at least 0.01 better than a non-wild card move as judged by the logical Q approximation function, demonstrated in *Figure 2*. These internal results were validated in tournament submission, with each subsequent iteration of the Greedy-Q agent improving from 71.7%, 73.7%, and finally to 86.7% in general tournament play.



The best move of all cards calculated with the Q-approximation is Jh to remove (5,1). However as it is only marginally better than playing 4d (2,4), the agent elects to retain the wild card.

Figure 2. *Holding the wild card for a better play*

An attempt to optimise the logic based state approximation function and the wild card threshold using self-play with only *Greedy-Q* agents returned largely equivocal results. Any result was questioned due to the poor representation of opposing strategies creating a non-representative sample. This limited pool of play styles was a challenge for all testing.
A strength of this agent is its continuous priority on sequence formation, using a greedy approach to make quick effective decisions in unseen states in rapid time. Its performance is independent of previous exposure. Further due the normalised scoring state approximation function, the agent is able to naturally balance both offence and defence.

A potential weakness of the agent is ignoring combinations of card. However, interestingly a BFS agent that does consider combinations performs worse as its depth increases, with its win rate plummeting from 98% to 48% to 2% with a depth of 1,2,3 respectively against the Greedy-Q agent. This sharp drop can be attributed to the danger of ignoring other agents impact on the board. In Figure 3, the BFS agent suggests a path assuming a board state that is substantially different at that turn. This illustrates the point that while state abstraction is useful, it can also provide faulty assumptions that lead to suboptimal behaviour. [2]

Greedy-Q Agent: plays 8h (6,3), BFS Agent path: Kh (5,6), 2. 9c (6,2), 3. 8h (6,3)

Board state at agents next position. Qh (6,6) played by ally has opened a sequence

Greedy-Q agent makes immediate reward play and captures a sequence

Figure 3. Questionable utility of projecting into future.

Given, the POMDP nature of the true problem with a large state space and high transitional volatility, the results suggest considering each board state independently, accepting future uncertainty making a high impact play is a prudent strategy.

5. Future directions

We can improve our current Greedy-Q agent by testing it against a wider variety of playing styles. We are also curious to implement the Deep-Q model that was not implemented due to time constraints. However ultimately, the team felt the lack of an efficient simulator severely limited viable search strategies even in a state abstracted problem, and rectifying this would be a high priority. Following this, we would like to reattempt a MCTS experimenting using a MDP that returns a random or policy based action, to improve the number of iterations.

Our results echo the literature that shows many machine learning model struggle with the complexity of POMDMP. [1] We would also like to consider Sequence from game theoretic perspective, where it can be seen as imperfect information extensive-form game. There has been success using expert knowledge to produce high level feature state representation from which a Nash equilibrium strategy can be derived. [1] Further, there has been a recent demonstration combining self-play with neural network function approximation to derive a Nash equilibrium strategy without any expert feature extraction as a general solution. [1] This approach intrigues us as we were able to leverage no expert knowledge.

References

1. Heinrich, J., & Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*.
2. Johanson, M., Burch, N., Valenzano, R., & Bowling, M. (2013, May). Evaluating state-space abstractions in extensive-form games. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems* (pp. 271-278).
3. Brown, N., Bakhtin, A., Lerer, A., & Gong, Q. (2020). Combining deep reinforcement learning and search for imperfect-information games. *arXiv preprint arXiv:2007.13544*.
4. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
5. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

The reflection of Paari Palaniswami
Student ID: 846338

1. **What did I learn about working in a team?**

I am from a non-computer science background and this was my first group project involving coding with a group. Initially, it was interesting to hear how other people formulated the problem, and their prior beliefs about was techniques would work. Our group all had different intuitions and there was some natural tension as to what to pursue. This led to some interesting discussions, and allowed me to clarify my own reasoning.

A definite advantage I felt working with others was pair programming. A team member's insight into how the simulator worked proved pivotal to standing up any agents. Further, with pair programming we were able to identify problems more effectively in the myriad of helper functions we developed. Certainly, there was more communication about code structure, but the emotional stress of being "stuck" was greatly reduced working in a group. Our agents naturally followed on from each other, and we all contributed to each agent, which gave us a shared sense of comradery. I strongly believe, the breadth of perspective offered by the team lead to a better formulation of the problem, and stronger implementation. Previously, I had preferred to work individually on problems, but I can much more see the advantage of group work in this setting.

2. **What did I learn about artificial intelligence?**

To me this project highlighted the importance of context when considering artificial intelligence, here defined by rational behaviour. The behaviour of the agent was dictated largely by the state space, and how it was modified. This problem was difficult, and many techniques couldn't be applied, and that helped me to more appreciate the difficulty of creating agents to operate on real world problems, which is my long term goal. Whilst working on this project, I considered other problems I am interested in, and immediately started thinking about the state space, and possible abstractions.

Following on, the project helped to consider problem abstraction or reduction in general. Understanding, what techniques I had to use, such as classical search algorithms, and their specific requirements, helped guide the state abstraction. It was interesting to read about the difficulty with POMDP as open research challenge with large potential impact on real life problems.

3. **What was the best aspect of my performance in my team?**

I believe the best aspect on my performance was leading the creation of models. I believed I was accurately able to appreciate what abstraction need to occur, and what the technical challenges involved with that were. I think I was able to generate good ideas, and move on to different approaches. I think I had a good attention for detail. I think was able to communicate my vision to team mates.

4. **What is the area that I need to improve the most?**

I think I could improve in my listening. I think occasionally I can become fixated on my own idea particularly if I feel emotionally attached to it, and I can become resistant. I can improve by showing more malleability, and welcoming competing ideas as adding robustness. I believe also the my attention for detail can wear on my team mates, and I have a desire for perfectionism. I think this could also be improved with increasing my patience, and considering the goals of others. Technically, I believe I can improve in my skills coding with the group. I found using the GitHub repository challenging, and found managing different branches confusing. I think greater proficiency in this aspect would have improved our efficiency, and ability to work separately.

The reflection of Akil Munusamy Pitchandi
Student ID: 1147875

1. **What did I learn about working in a team?**

This group project allowed me to work in a team and I believe I have gained a great deal of knowledge and experience in the process. To begin with, each of us had a different view and approach to deal with every aspect of this project which gave me a broader perspective and helped me understand the subject and the task even more. In addition, differences in opinion caused heated discussion which helped us iteratively improve our agent's performance. Our team's work mirrored that of an agile team that picks up tasks, completes them, discusses the outcomes, and iteratively improves on them. This experience gave me a different outlook on working as a team and how to efficiently get work done.

2. **What did I learn about artificial intelligence?**

Initially beginning with this project, I found it tough to understand the interface and the framework provided to us. But with every failed attempt, I was able to learn to work around the given framework, which I believe an AI agent must do effectively. Although the framework was far challenging than the previous year's projects or our first Pacman assignment, this assignment helped me grasp some of the major techniques discussed in the lectures. I also learned that there is no tailor-made AI agent that gives the best result in every application. We as a team analyzed the size of the state space of this project to evaluate the techniques mentioned in the lectures and picked the ones that we thought can be implemented.

3. **What was the best aspect of my performance in my team?**

I believe the best aspect of my contributions mainly focused on understanding the given interface. I spent quite a few hours trying to implement the simplest of techniques and failing. Hence, I had to go through every code multiple times to understand every aspect of it to get the agent to work. This way I was able to first implement a poorly performing agent. Using this understanding of the Game I was able to help my teammates come up with a better agent. Apart from this, I also completed tasks like developing heuristic, etc.

4. **What is the area that I need to improve the most?**

I believe I need to gain a wider perspective on any solution so that I don't get stuck on the wrong path for a long time. I need to be able to acknowledge when I am at a dead-end and need to look for other ways to get the task done.

The refection of Pawan Malhotra
Student ID: 1131927

1. **What did I learn about working in a team?**

This group project was very challenging, but that made it very interesting at the same time. I am glad that I had a very good team, where every member was highly motivated and had a problem-solving approach. I believe I have gained immense knowledge from my peers in terms of their approach to solving problems. I learnt how stressful environments created from failed experiments can cause a difference in opinion which might lead to fall of the team, but we were able to look past that and move forward towards our common goal of creating a working AI agent to compete in the tournament. Other than this, I gained experience in planning (scheduling of tasks/meetings) and communication.

2. **What did I learn about artificial intelligence?**

I learnt a great amount of information about different AI techniques being used to play a range of games from perfect information games such as Chess, Tic-Tac-Toe and GO, to imperfect information or partially hidden information games such as Poker and Bridge. It took me a good amount of time to actually understand the dynamics of the game as well as the code provided to implement the game. The game posed a challenge of huge action-state space, therefore limiting the number of techniques which could be successfully applied. With luck and some good understanding, I was able to implement different techniques though only few were feasible. Although, we were not able to implement the Deep-Q approximation technique due to time constraints, I really enjoyed the training and experimentation phases of the process.

3. **What was the best aspect of my performance in my team?**

I spent a lot of time understanding the interface of the game and how the game is played. It gave our team a good understanding how the code works and important classes/methods that are needed. I worked together with my team in building AI techniques. I specifically worked on Monte Carlo Tree Search an A*. I extensively worked on generating data for our experiments (tournament between our agents) and also created a dataset of about 200,000 instances for our Deep-Q approximation experiment.

4. **What is the area that I need to improve the most?**

I believe I struggled with the abstraction of the game and was not able to properly abstract the features of the problem, therefore was struggling to apply the different AI techniques. I overcame this issue with the help of my team members. Also, I need to improve on time management, as I was constantly working on multiple projects due to multiple university commitments.