Solving an Esoteric Card Game with Reinforcement Learning

Andrew S. Dallas
Stanford University, Stanford, CA, 94305

A learning strategy was developed to learn optimal policies for the card game Somerset and to develop a prediction of what score to expect given an initial hand. Somerset is a trick-taking, imperfect information game similar to the game Euchre. The state and action space of the game is intractably large, so an approximation to the state and action space is used. A Q-learning algorithm is used to learn the policies and impact of using eligibility traces on the learning algorithm's performance is examined. The learning algorithm showed moderate improvement over a random strategy, and the learned Q-values revealed predictions that could be made on the final score based on the opening deal.

I. Introduction

Somerset is a four-player, trick-taking card game similar to Euchre. The game has a variety of decisions that must be made with imperfect information. This work aims to find an ideal decision-making strategy for the game that can be used to inform and estimate how a player might perform once the cards have been dealt.

The rules to the game will be described here briefly to provide context for the methodology employed, and a more detailed set of rules can be found in Ref. [1]. Somerset is played with a custom deck of cards. Instead of traditional suits of hearts and spades, there are numerical suits of 2, 4, 6, 8, 10, and 12. There are n+1 cards of each numerical suit n with unique numerator non-negative integers up to the suit value denominator. As examples, there is a 0/2 card, 1/2 card, and 2/2 card for the 2 suit, there is a 0/4, 1/4, 2/4, 3/4, and 4/4 card for the 4 suit and so on. There are also two special cards that do not follow that pattern. The first is the S/S card which takes on a value of -1/X where X is the trump suit when it is declared. The second is the 0/0 card which is ranked higher than every card except those of the trump suit and other "double" cards (e.g. 2/2, 4/4).

Play of Somerset begins by dealing 12 cards to each of the four players. Two cards are left undealt and unseen for the game in a pile called the Kitty. The players are typically seated in a circle, and players on opposite sides of the circle are teammates but are unable to see the cards of each other or their opponents. Players then guess how many points their team will get based off of the information they have in their hand, and the player with the highest guess is afforded the opportunity to decide on a trump suit that will give himself an advantage. A card in the trump suit will beat cards of any other suit.

After this first phase of the game, the general game play begins. The trump suit-declaring player starts the first trick by placing down an initial card face up from his hand. The player to his left places down another card face up and so on until all four players have played a card. The player which starts the trick can place a card of any suit. Subsequent players must play a card of the same suit as the starting card of the trick if they have one, otherwise they can play any suit. The player with the highest card of the four wins the trick and gets a point for his team, and that player then gets to start the next trick. The highest card is determined as the highest numerator card of the trump suit, or if none of that suit was played, the highest numerator card of the trick's starting suit. This process continues until 12 tricks have been played. A point is earned for each trick won, and 12 more points are available by winning tricks during which one of seven special points cards (each of which has a bonus value of 1, 2, or 3) has been played.

II. Related Work

A variety of artificial intelligence approaches have been applied to trick-taking, imperfect information games. Several different methods for a variety of games have been employed to develop state approximations or decision-making algorithms that lead to more successful agents.

Euchre is the most similar game to Somerset, as it also is a trick-taking card game in which pairs of players are on opposing teams, that has been examined with an artificial intelligence-like approach. A rule-based approach was evaluated by Seelbinder [2]. Interestingly, these approaches were also combined to see how different pairs of strategies, in which one player on a team played according to one set of rules and that player's partner played based on a different set of rules, would impact success. While some strategies were unsuccessful in isolation, they proved to be successful when used as a paired strategy. However, no reinforcement learning was done to improve the rule-based strategies tested.

Hearts is another trick-taking game that is more widely studied, but different from Somerset and Euchre in that each player plays for themselves. Temporal difference (TD) learning has also proven successful for learning strategies to the game of Hearts. In work by Sturtevant and White [3], a TD-learning algorithm was applied to learn strategies, in combination with stochastic linear regression. Sets of state features are combined using a perceptron. 60 different state features are combined in various ways to identify the best patterns. A TD(λ) technique is applied to propagate learning back in time. The algorithm was trained against search-based policies and ultimately proved to be better than those types of policies.

Ishii et al [4] formulated the problem of Hearts as a partially observable Markov decision process (POMDP). In this approach, an actor-critic method with function approximation for state and merit values is used. The reinforcement learning agents developed outperformed the rule-based agents and came close to outperforming an expert human player.

Big 2 is a different trick-taking game than Hearts, but with similar objectives. Charlesworth [5] applied deep reinforcement learning to learning strategies for the game of Big 2. An algorithm called Proximal Policy Optimization was used to train a neural network which output a policy based on state information and possible actions. The state space was approximated first using human-developed heuristics and that information was then fed into the neural network. Each of the agents in the game were simulated using the policy that was being learned. The methodology proved successful in that the algorithm is able to outperform human players, but training was relatively computationally intensive as it took 2 days on a 4 core CPU to simulate and train on 3 million games.

Somerset is a relatively esoteric card game, and so no papers on learning strategies for it could be found. However, the techniques applied for the other games could be applicable to this game.

III. Approach

The game of Somerset is modeled as a Markov decision process (MDP). There are effectively two stages in the game. The first phase consists of one step in which the player declares the trump suit. The second phase is when cards are being played by the player. For this study, each phase has a unique state space feature approximation, and unique actions. The model-free reinforcement learning technique Q-learning is used to learn a policy for the starting player. The starting player learns while competing against players which execute a random policy. The code developed can be found at https://github.com/adallas1/somerset.

A. State Space

1. Phase I: State Space of the Starting Hand

The state space of the starting hand for which a player can be dealt is extremely large. With 12 cards to be dealt to a player from a possible 50, a player can receive 121,399,651,100 possible opening hands. Luckily, significant heuristics can be applied to get an idea of how to act based on a given hand. For the Q-learning agent developed, the initial state of the hand is approximated by the following three features:

- The opening hand's most common suit. This feature can take on a value of 2, 4, 6, 8, 10, or 12.
- The number of cards of the hand's most common suit. This feature can take on an integer value of 2 up to the suit value plus 2. The "plus 2" is because the S/S and 0 of the suit are possible.
- Whether the player has the S/S card. This feature can take on a value of yes (1), or no (0).

2. Phase II: State Space During the Play of Cards

The state space while cards are being played is significantly larger than the state space for Phase I. This large size is because to each player, each card can have the state of being in the player's hand, unseen, already

played, or played for the current round by player 1, 2, 3, or 4. As a result, the size of the state space a player can find himself in is roughly $7^{50} = 1.8*10^{42}$. The decisions that need to be made at each phase are based on more complex factors than those generally made to determine the trump suit. For the Q-learning algorithm, the following features are used to approximate the state:

- The trick/round of the game. This feature can take on a value of 1 through 12.
- The number of cards that have been played in the trick. This feature can take on a value of 0 through 3.
- The trump suit value. This feature can take on a value of 2, 4, 6, 8, 10, or 12.
- The number of trump cards that the player has. This feature can take on an integer value of 2 up to the suit value plus 2. As before, the "plus 2" is because the S/S and 0 of the suit are possible.
- If the player's team is winning. This feature takes on a value of 1 if the player's team is winning, a value of 2 if the other team is winning, and a value of 0 if no cards have been played yet.
- If the best card the player has will put him in the lead for the trick. This feature takes on a value of 1 if his best card will put him in the lead, and a value of 0 if not.
- If the player can play a points card. This feature takes on a value of 1 if he can, and a value of 0 if not.

B. Action Space

1. Phase I: State Space of the Starting Hand

The action space to declare trump from the starting hand is relatively simple. As such, no approximations are made. The agent is simply takes the action to choose the trump suit from the suits of 2, 4, 6, 8, 10, or 12.

2. Phase II: State Space During the Play of Cards

There are a large number of actions that can be taken by the player when playing cards. However, the rule of the game that the leading suit must be played if possible, often constrain the number of cards that can be played. Beyond that, the moves that can be made generally fall into one of a few categories. For the learning agent, the following actions are allowed:

- Play the highest card in the player's hand.
- Play the lowest card in the player's hand.
- Play a points card if the player has one legally playable.
- Play the "0/0" if it is legally playable.

When the agent selects one of these actions, it is then converted into the legally playable card that is in the player's hand and is played for the trick.

Additional actions in real life are possible. An example would be to "play the second highest/lowest card," possible, or "play the lowest card of a specific suit," but these actions require significant knowledge of the state space, and the approximations used for the state space likely would not allow for the complexity needed to learn those type of strategies. These actions are very rarely used in practice as well, and due to the rules of the game, it is infrequent that these more detailed actions get to be employed.

C. Learning Algorithm

Q-learning is used for the agent to learn optimal actions. Two versions of Q-learning are used: the "vanilla" Q-learning algorithm as presented in Kochenderfer [6], and a version that uses eligibility traces. Since what matters at the end of the game is maximizing the points won, it was hypothesized that eligibility traces would allow for the rewards gained at later stages to influence learning at earlier stages much more effectively. Ref. [7] describes a Q(λ) algorithm which propagates as far back as the exploitation steps have been taken. However, that would greatly limit the length of eligibility traces and so a Q(λ) algorithm is used in which rewards are propagated backwards for every state-action pair seen in the current game. The algorithm used to incorporate eligibility trace learning for the agent is shown in Figure 1.

To balance exploration and exploitation of the algorithm, an "ε-decreasing" strategy is employed, in which ε starts off large to encourage more exploration and then as more games are played, ε decreases to encourage more exploitation. The equation used for ε can be seen in (1). The ε used in a given game was consistent between the trump-selection and card-selection action phases.

$$\varepsilon = \frac{c}{c + \#\ of\ games\ simulated} \quad (1)$$

Algorithm: Q(λ) Learning

```
Q ← 0s
γ ← 1.0
for n from 0 to number of training games:
  ε ← c/(c+n)
  N ←0s
  SeenStateActionPairs ← {}
  state ← initialState()
  loop through game states until the game is over:
    if (random number from 0.0 to 1.0) < ε
      choose random action
    else
      choose action which maximizes Q from state
    N(state,action) = 1
    Add (state,action) to SeenStateActionPairs
    Get nextState and reward from (state,action)
    maxNextQ ← max Q value possible from nextState
    δ ← reward + γ* maxNextQ – Q(state,action)
    for (state,action) in SeenStateActionPairs:
      Q(state,action) ← Q(state,action) + α ∗ δ ∗ N(state,action)
      N(state,action) ← γ ∗ λ ∗ N(state,action)
    state ← nextState
```

Figure 1: Q(λ) algorithm pseudocode.

IV. Results

A. Performance before Learning

Before running the Q-learning algorithm, the baseline performance of the competing teams was established. 10,000 random games were simulated, with all of the agents taking random actions. Over this period, due to the advantage of being able to play the leading card in the first round, the team that went first in the opening round (Team 1) averaged 12.2 points, and the other team averaged 11.3 points. Note that, while a maximum of 24 points are possible, because 2 cards are left in the Kitty which could mean some of the special points cards are not used in play, the number of points scored in a round is always less than or equal to 24.

B. Learning and Algorithm Comparison

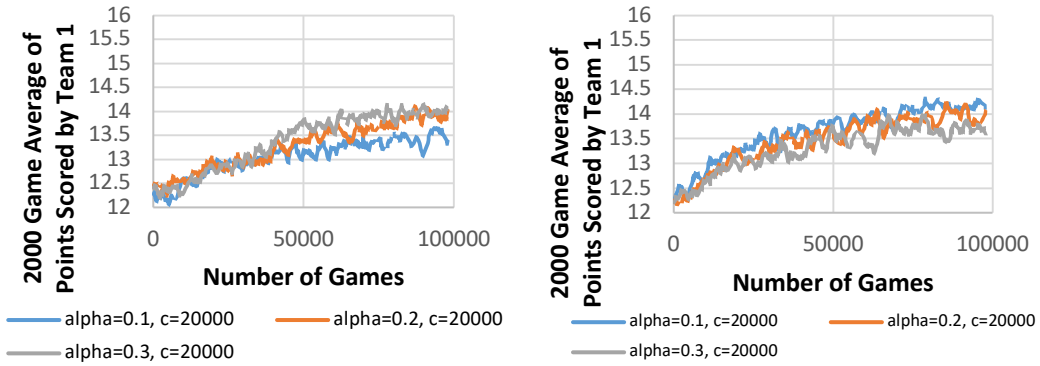

Figure 2: Comparison of the vanilla Q-learning (left) and Q(λ) (right) algorithms for different learning rates (α).

After the baseline performance of the two teams was established, the learning agent on Team 1 was trained with the Q-learning and Q(λ) algorithms. To ensure that performance of the learning algorithms was maximized, a study on the impact of hyperparameters in the algorithms was done. First, the learning rate (α) was modified. Figure 3 shows the impact of different learning rates on the Q and Q(λ) algorithms. After 100,000 games, differences in the learning rate from 0.1 to 0.3 resulted in about 0.5 points difference in performance for the learning agent's team. Interestingly, the vanilla Q-learning algorithm benefitted from higher learning rates and the Q(λ) algorithm benefitted from lower learning rates.

Additionally, the exploration/exploitation balance was examined as well. The value of c in Eq. (1) was varied to see how that value impacted performance of the learning agent. Figure 3 shows the impact that different rates of exploration and exploitation have on the algorithms. From this figure, it can be clearly seen that at a learning rate of 0.1, Q(λ) significantly outperforms the vanilla Q-learning algorithm for algorithms that use a lower value of c. Lower values of c increase exploration early in the algorithm's learning and exploitation later on. The value of c has limited impact on how well the vanilla Q-learning algorithm does, but the Q(λ) algorithm improves with lowered c values.

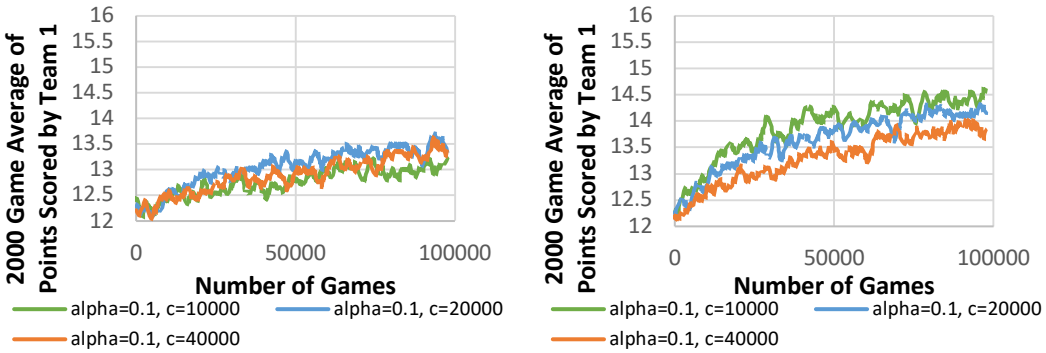

Figure 3: Comparison of the vanilla Q-learning (left) and Q(λ) (right) algorithms for different exploration parameters (c).

C. Final Algorithm Performance and Learned Q-Values

The comparison of algorithms and tuning of learning rate and exploration/exploitation balance above were used to determine a strategy for getting the best performance possible from Q-learning. A learning rate of 0.1 and c-value of 5,000 were used in the Q(λ) algorithm, and the algorithm was trained on an extended number of games. Figure 4 shows the algorithms performance over time. The learning agent's team reaches a fairly stable plateau of an average of around 15 points per game after 300,000 games. If the learning algorithm was trained on more games, it is possible that it could improve performance further, but it is not expected to change the result considerably. The performance improvement of 3 points is a significant improvement over a random strategy.

It should be noted that it is unknown how well the optimum policy could do, because a significant portion of the game is due to the initial random distribution of cards, and so there will be cases where the cards dealt to the learning agent's team lead to inferior outcomes. Additionally, the learning agent's teammate also followed a random strategy, meaning the teammate could not be counted on to provide a performance benefit to make up for bad cards being dealt to the learning agent.

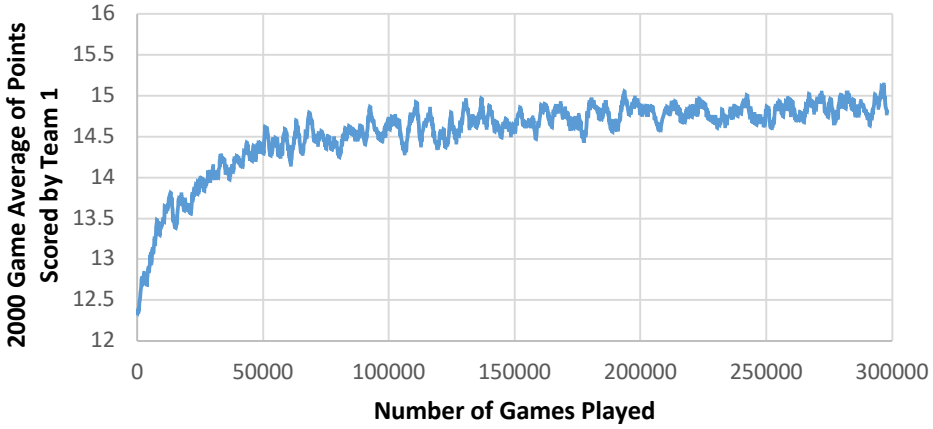

Figure 4: Learning performance of the Q(λ) algorithm for with a learning rate of 0.1, and an exploration parameter of 5,000.

The resulting Q values for the most common initial state-action pairs are displayed in Table 1. Several strategic insights can be learned from these results alone. As expected, opening hands with higher number of cards of the most common suit perform better than hands with more diverse hands. For example (12, 5, 0) and (10, 5, 0) perform better than (8, 3, 0). Additionally, for the same number of cards of the hand's most common suit, it is better for that most common suit to be lower. As examples, (10, 5, 0) performs better than (12, 5, 0) and it is also the case that (6, 3, 0) performs better than (8, 3, 0).

Table 1: Q values and occurrences for the most common opening state-action pairs.

State			Action	Q Value	Occurrences over 300,000 games
Hand's Most Common Suit	Number of Cards of the Hand's Most Common Suit	Have S/S Card			
10	4	0	10	16.2053982	29721
12	4	0	12	15.4512755	27549
12	5	0	12	16.9655895	24172
8	4	0	8	16.1141916	20204
6	3	0	6	15.9370535	15489
8	3	0	8	12.6025239	13615
10	5	0	10	17.7324335	13404

V. Conclusions

The use of Q-learning is an effective means of developing an agent to play team-based, imperfect information games. Even without training both teammates, significant performance improvement can be realized. The effectiveness of the learning algorithm is fairly dependent on the hyperparameters that are used. Additionally, Q(λ), which takes into account the whole history of a single hand of the game, is able to learn significantly better than the vanilla Q-learning algorithm. Future work could include training against more intelligent agents, training both teammates, and comparing the learning agent's decision-making strategy against experienced human players.

VI. References

[1] "Rules for Double Somerset," URL: https://www.somersetgame.com/double-somerset-rules.html [retrieved 1 December 2019]
[2] Seelbinder, B. E., "Cooperative Artificial Intelligence in the Euchre Card Game," Master's Thesis, University of Nevada, Reno, URL: https://pdfs.semanticscholar.org/2967/a8f4e649a5d69db275d3d90cb29510bba711.pdf [retrieved 1 December 2019]
[3] Sturtevant, N. R., and White, A. M., "Feature Construction for Reinforcement Learning in Hearts," *Computers and Games: 5th International* Conference, 2006, URL: https://sites.ualberta.ca/~amw8/hearts.pdf [retrieved 1 December 2019]
[4] Ishii, S., Fujita, H., Mitsutake, M., Yamazaki, T., Matsuda, J., and Matsuno, Y., "A Reinforcement Learning Scheme for a Partially Observable Multi-Agent Game," *Machine Learning*, Vol. 59, 2005, pp. 31-54. URL: https://link.springer.com/content/pdf/10.1007%2Fs10994-005-0461-8.pdf [retrieved 1 December 2019]
[5] Charlesworth, H., "Application of Self-Play Reinforcement Learning to a Four-Player Game of Imperfect Information," URL: https://arxiv.org/pdf/1808.10442.pdf [retrieved 1 December 2019]
[6] Kochenderfer, M. J., *Decision Making Under Uncertainty: Theory and Application*, The MIT Press, Cambridge, 2015.
[7] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, 2nd ed., The MIT Press, Cambridge, 2015.