An Empirical Evaluation of Policy Rollout for Clue®

Eric Marshall
Oregon State University
M.S. Final Project
marshaer@oregonstate.edu
Adviser: Professor Alan Fern

Abstract

We model the popular board game of Clue as an MDP and evaluate Monte-Carlo policy rollout in a simulated environment pitting different agents and policies against each other. We describe the choices we made in the representation, along with some of the problems we encountered along the way. We find that even a simple heuristic policy can dominate a random policy, and that single-stage rollout can be used to incrementally improve existing policies, and confirm that multi-stage rollout is not practical for this domain.

1 Introduction

The classic murder mystery board game Clue offers an interesting domain for evaluating automated planning algorithms for several reasons. First, the game consists of multiple, competing agents taking actions in their environment—actions that can affect other agents—and racing to acquire enough information to solve the mystery that is at the core of each game. Second, the agents obtain both certain and uncertain knowledge as they take actions and observe the behavior of other agents. This uncertain knowledge in particular makes the game interesting from a probabilistic and planning standpoint.

Third, the game has simple, well-defined rules, is easy to learn, and is straightforward to simulate with a computer. These factors combine to make Clue an interesting problem to study, and make it well-suited for many automated planning algorithms.

1.1 Domain

The game of Clue consists of 21 game cards of three types: six suspects, six weapons, and nine rooms. At the beginning of the game, the cards are separated into three decks, and one card is drawn from each and placed into the Case File. The premise of the game is that the cards in the Case File describe a fictitious murder that occurred, and the object of the game is to discover who committed the murder, what weapon was used, and where the murder occurred; that is, to determine the contents of the Case File.

The remaining cards are then shuffled and distributed as evenly as possible among the players. Play begins with the first player taking their turn, where a turn consists of two stages, (1) a suggestion phase, where a suspect, weapon, and room are announced as the suggestion, and (2) an optional accusation phase, where a suspect, weapon, and room are announced and the game is either won or lost by the accusing player. (Note that the official board game rules also require a
“move” stage for each turn, however these rules are somewhat more complicated to describe, and were not represented in our simulator and as such, will not be discussed.)

After a suggestion is made, each player in turn is required to refute the suggestion, if possible, by showing the player making the suggestion a card involved in the suggestion to prove it is not in the case file and thus the suggestion is false. Note that the card is only shown to the player that made the suggestion, and the other players only observe that some card was shown, but which card is kept hidden. As soon as any player refutes a suggestion, the suggestion phase ends and the accusation phase begins, or the turn ends if the player waives their right to make an accusation.

An accusation, like a suggestion, consists of a suspect, weapon, and room. When an accusation is made, the player immediately checks the case file for the accused cards, and wins if the accusation was correct, or immediately loses if it was not correct.

We relaxed the problem to remove the requirement that a player’s pawn must be located in the room that they name in their suggestion. Instead, players can name any room at any time. This was done in order to remove one random element from the game and focus on the core problem of optimizing the effectiveness of each suggestion.

2 MDP Representation

We modeled the game as a Markov Decision Process (MDP) defined by the set of states, actions, rewards, and transitions S, A, R, T. Before we describe the MDP in detail, we must first introduce the two types of knowledge a Clue agent acquires during the course of a game: certain and uncertain knowledge.

2.1 Knowledge Representation

We use the term certain knowledge to describe all information the agent has obtained about a player holding or not holding a particular card. For example, if some player \( p \) shows the agent some card \( c \) in their hand to refute a suggestion, we change the state to represent that \( p \) has \( c \). Likewise, if some other player \( p' \) states that they cannot refute a suggestion consisting of suspect \( s \), weapon \( w \), and room \( r \), we update our state to indicate the \( p' \) can’t have \( s \), \( w \), or \( r \).

Uncertain knowledge is obtained by observing refutations to other player’s suggestions. For example, if a player \( p \) refutes a suggestion made by some other player, \( p' \), consisting of suspect \( s \), weapon \( w \), and room \( r \), by showing \( p' \) an unknown card, we update our state to indicate that \( p \) must have \( s \), \( w \), or \( r \).

While it is relatively simple to reason over and propagate implications deriving from certain knowledge, effectively exploiting this uncertain knowledge is one of the difficult challenges a Clue agent faces.

2.2 States

The states of our MDP are simply the certain and uncertain knowledge the agent has acquired at any particular point. The certain knowledge is represented as an \( n \times m \) matrix, where \( n = \) the number of players, and \( m = \) the number of cards. Each cell in the matrix can take one of three values indicating that a player has, can’t have, or can have a particular card. The matrix is initialized by setting each value to can have, and updating these values to has or can’t have as certain knowledge is acquired throughout the course of the game.

The uncertain knowledge was represented by a list of four-tuples containing the player making a refutation and the three cards involved in the corresponding suggestion.

The number of total unique card dealings in Clue is extremely large. We believe there are over 5 billion possible configurations of a three-player game, as shown in the following equation, and over 44 trillion possible configurations of a six-player game. For the purposes of this project, we only consider three player games.

\[
\binom{18}{6} \times \binom{12}{6} = 5,557,616,064
\]

2.3 Actions

One of the benefits of relaxing the problem to ignore player locations was the simplification of our action space. By ignoring locations, we only need to represent one type of action: the suggestion. We did not include accusations in the action space because we instead chose to implement a rule that triggered victory if some player determines that a
suspect, weapon, and room were known to be held by none of the players, and therefore must be in the Case File.

2.4 Rewards

Our reward function is similar to our victory condition, in that it returns a value of 1 if a suspect, weapon, and room are known to be held by none of the players, and otherwise returns 0. We also experimented with shaping reward functions that gave a higher reward depending on how much certain knowledge had been accumulated, but these were abandoned during development in favor of the simpler reward function described above.

2.5 Transitions

The transition function was perhaps the trickiest part of the MDP to implement, as our implementation required several iterations before sufficiently representing the mechanics of the domain and being useful for simulation.

We believe the main reason that the transition function for this problem was challenging to implement was because the domain is effectively only partially observable, and instead of using a proper POMDP representation, we were essentially trying to force a partially observable domain into a fully observable representation. In our representation, the transition function had to account for the partial observability and produce a next state that was consistent with our set of observations (specifically, the uncertain knowledge).

Our first, naïve implementation of the transition function chose a card at random from the suggestion and assigned it to a randomly chosen player, if the assignment was consistent with the certain and uncertain information the agent had observed. This approach was insufficient because it did not properly represent the game’s mechanics, specifically the possibility that a suggestion might not be refuted by a player, or that it might not be refuted at all.

We tried a few other variations on the transition function, each more sophisticated than the last, until we achieved satisfactory results. The resulting function assigns a random card \( c \) from the suggestion to the next player \( p \), according to the game’s natural turn order, with an approximation of the probability that \( p \) has \( c \). We approximated that probability using the following function:

\[
pr(p \ has \ c) = 1 - \prod_{i=1\ldots k} \frac{m - (i + 1)}{m - i},
\]

where \( m \) is the number of cards that \( p \) can have and \( k \) is the number of remaining cards that \( p \) was dealt, i.e. the number of cards of \( p \) that are known to the agent.

This transition function correctly represents the mechanics of Clue, and approximates the probability that a suggestion will be refuted.

2.6 Transitions Using CSPs

The primary drawback of the transition function described above is that it only leverages the certain knowledge acquired by the agent, and largely ignores the uncertain knowledge obtained by observing refutations of other players’ suggestions. We attempted to bridge this gap by modeling the game state as a constraint satisfaction problem (CSP) and using a 3rd-party CSP solver (Prud'homme et al.) to find solutions to the current state. Once a solution is found, the same logic can be used in the transition function, except now it is operating on a fully solved state instead of a partially observed state, so it is no longer necessary to estimate probabilities of a player having a certain card. Instead, we can sample from this distribution directly by finding many possible solutions to the CSP.

To model the state as a CSP, we defined \((|p| + 1) * (|s| + |w| + |r|)\) Boolean variables, which represented whether each player held a particular card. Strictly speaking, the representation could have been done with fewer variables by removing the variables associated with the Case File, however that would require more complex constraints and we felt this was not justified. We then added the basic constraints over those variables to satisfy the rules of the game, such as each card being held by only one player, and the case file must hold exactly one of each card type.

We also added constraints to match the certain knowledge obtained by the agent (e.g. some card \( c \) is held by some player \( p \)), and constraints to model the uncertain knowledge (e.g. some player \( p \) must have one or more of cards \( c_1, c_2, \) and \( c_3 \)). Once all of the variables and constraints were set, we could ask the CSP solver to generate many solutions and sample from the possible solution space.
3 Experiments

3.1 System Description

We implemented the MDP and Clue simulator in Java, and ran our tests on a system with a quad-core CPU running Windows 10 with Java 1.8.0_31. Since the policy rollout algorithm is well-suited for parallelism, we implemented the algorithm using a fixed-size thread pool, where each thread is responsible for running \( w \) simulations of a particular action. This multi-threaded approach allowed us to fully saturate all CPU cores available to us, and greatly improve the speed of the algorithm.

3.2 Complexity and Performance

The complexity of the policy rollout algorithm is known to be \( O(kwh) \), where \( k \) is the number of actions, \( w \) is the number of simulations to run for each action, and \( h \) is the horizon. Using this formula, we calculated the number of “computational units” required to run policy rollout using \( k=6\times6\times9=324 \), \( w=100 \), and \( h=25 \), so \( h\times k \times w = 324 \times 100 \times 25 = 810,000 \). Our multi-threaded system took 15-20 seconds to do this computation.

We calculated the cost of doing two-stage rollout using the same units, \((324 \times 100 \times 25)^2 > 650 \) billion. We estimated this would take roughly 4.5 months to compute, which would be necessary for each turn. Thus, we conclude that multi-stage policy rollout is not feasible for this domain.

3.3 Experimental Setup

We implemented three baseline agents to compare the policy rollout algorithm against: a random stateless agent, a random stateful agent, and a heuristic agent. The random stateless agent did not record any observed information. Its actions were chosen by randomly selecting any card that it did not itself hold. Because this agent kept no state, it was not able to determine when it had solved the game, so we added a special provision to the simulator to automatically end the game in the event that an agent suggests the cards contained in the case file. This way, separate accusation actions were not required.

The random stateful agent kept track of all observations from the game and randomly selected cards that were not known to be held by any player.

The heuristic agent was built around a simple heuristic policy that chooses a card that minimizes the total number of players that can have that card, if the holder of the card is not known. Like the previous agents, the heuristic agent never chooses its own cards.

Our simulation environment allowed three agents to play \( n \) games six times: once for each \( 3! \) possible ordering of players. Each ordering was kept consistent so that, for example, the first player would always get the same set of cards regardless of which agent was actually occupying that position, etc. This allowed us to evaluate not only which agent performed the best, but also which turn order was best for each agent.

We evaluated policy rollout by sampling uniformly across all \( 6\times6\times9=324 \) actions using an \( h \)-horizon Q-value estimation function. Except where noted, we configured the policy rollout with the following parameters:

\[
\begin{align*}
\text{horizon} & = 10 \\
\beta & = 0.9
\end{align*}
\]

Where \( w \) is the number of times we simulate each action, horizon is the number of trajectories we follow the base policy, and \( \beta \) is the reward decay constant used to discount future rewards.

3.4 Results

We ran several experiments consisting of the same agent in all six configurations over a total of 100 games in order to isolate each agent’s performance. These results are shown in Table 1.

<table>
<thead>
<tr>
<th>Agent</th>
<th># Turns</th>
<th>Win %</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random (stateless)</td>
<td>111.0</td>
<td>33.7%</td>
<td>33.4%</td>
<td>32.8%</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>15.8</td>
<td>39.0%</td>
<td>29.7%</td>
<td>31.3%</td>
<td></td>
</tr>
<tr>
<td>Random rollout</td>
<td>12.9</td>
<td>38.7%</td>
<td>30.2%</td>
<td>31.2%</td>
<td></td>
</tr>
<tr>
<td>Heuristic</td>
<td>11.7</td>
<td>35.2%</td>
<td>29.7%</td>
<td>35.1%</td>
<td></td>
</tr>
<tr>
<td>Heuristic rollout</td>
<td>13.0</td>
<td>40.7%</td>
<td>26.8%</td>
<td>32.5%</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Average turns and win percentage by turn order for homogenous game configurations.
We found that the random stateless agent unsurprisingly performed the worst of the group, requiring 111.0 turns on average to find the solution. This figure nicely matches our expectations of randomly guessing the solution, since on average a player will receive two of each card type, leaving $4^3 \times 7 = 112$ combinations. We can also see that this agent performed slightly better the earlier in the turn order that it went. This may be unsurprising, but it is interesting because no other agent followed the same pattern.

Since the only difference between the stateless and stateful random agents is the tracking of state, we can conclude that this alone accounts for the large performance improvement from 111.0 turns per win to 15.8.

The policy rollout agent was able to improve upon the base random policy by nearly 3 turns on average to 12.9, but surprisingly increased the number of turns required when used with the heuristic policy by 1.3 turns. This result is puzzling, however we suspect that this is an unfortunate interaction between the CSP solver finding solutions in the same neighborhood and the heuristic acting in a greedy way. We think that if either the CSP solver traversed solutions in a random order, or if we increased $w$ to approach the number of possible solutions, this effect would be removed. Indeed, we see evidence of this in Figure 5, which we discuss later.

The turn order results, displayed as a chart in Figure 2, show a clear pattern across all stateful agents that the ideal position in the turn order is: 1st, 3rd, and finally 2nd. This was an unexpected result for which we do not have a satisfying explanation.

We also ran some experiments pitting two instances of the baseline agent against policy rollout using the same base policy used by the other two agents. We used this configuration to experiment with the policy rollout parameters $w$ and horizon. These results are shown in Figures 3 and 4 relating to the random stateful base policy, and Figures 5 and 6 relating to the heuristic base policy.

![Figure 2: Win percentage by turn order for homogenous game configurations.](image)

![Figure 3: Policy rollout versus two random stateful agents with fixed horizon.](image)

The results in Figure 3 show that increasing $w$ had a positive effect on win rate and a negative effect on turns, i.e. policy rollout is improving upon the base policy. Figure 4 shows the results of changing the horizon with a fixed $w$, but these results are not as clear. Although there is a positive correlation between horizon and win rate, the number of turns per win fluctuates. We expect the fluctuation could be decreased by running more simulations or by fixing $w$ to a larger value, such as 100 or 1,000.

![Figure 4: Policy rollout versus two random stateful agents with fixed $w$.](image)
Figure 5 shows the results of running policy rollout with a heuristic baseline policy against two heuristic agents. Once again we see a positive correlation between $w$ and win rate and a general negative trend for turns. Compared to policy rollout with a random base policy, we see here that rollout with a heuristic base policy requires a much larger $w$ before we begin to see improvement. In this case, rollout does not improve upon the base policy until $w > 1,000$. As noted earlier, this is likely due to the CSP solver only exploring a small neighborhood of possible solutions coupled with the heuristic policies greedy tendencies, so more sampling is required to diversify the set of solutions explored.

One noteworthy result from this figure is the configuration where $w = 3,000$, which resulted in the lowest observed individual turn average for policy rollout of 10.7 turns per win.

Finally, in Figure 6, we show the results of increasing horizon with $w$ fixed at 10. Once again, these results are a bit mixed, and we attribute that to the $w$ parameter being set too low, as we suggested regarding Figure 4. We also acknowledge that increasing the horizon far beyond the average game length may not lead to improvement, and it is possible that is what we are observing for horizons $> \text{approximately 10-20}$.

4 Conclusions

In summary, we found that Clue can be represented as an MDP, although we need to be careful in crafting the transition function to represent not only the game’s mechanics, but also the uncertainties present in our observations. We also found that representing the game as a CSP can be a useful method to utilize uncertain knowledge.

We also found that policy rollout can be used to improve upon a simple random policy in this domain, and that our heuristic policy could also be improved, although it required more resources before any improvement was gained, and the improvement was much smaller. This may have been due to an unfortunate interaction between the CSP solver and the heuristic only exploring a small neighborhood of solutions. We also found that multi-stage rollout is not feasible using the current representation.

Our results also indicate that the parameter $w$ is more sensitive with respect to win rate than the horizon parameter.

For future work, we believe that applying Monte-Carlo tree search algorithms, such as UCT (Kocsis and Szepesvari) or POMCP (Silver and Veness) may improve our results. We would also like to incorporate player locations into the state space, and add moves, refutations, and accusations into the action space.

References