Computational Pool: A new challenge for game theory pragmatics

Christopher Archibald, Alon Altman, Michael Greenspan and **Yoav Shoham**

Abstract

Computational pool is a relatively recent entrant into the group of games played by computer agents. It features a unique combination of properties that distinguish it from others such games, including continuous action and state spaces, uncertainty in execution, a unique turn-taking structure, and of course an adversarial nature. This article discusses some of the work done to date, focusing on the software side of the pool-playing problem. We discuss in some depth CueCard, the program that won the 2008 computational pool tournament. Research questions and ideas spawned by work on this problem are also discussed. We close by announcing the 2011 computational pool tournament, which will take place in conjunction with the Twenty-Fifth AAAI Conference.

Introduction

Cue sports have been captivating humankind for thousands of years, with written references dating to the first century A.D. They evolved as a branch of modern croquet and golf, as a kind of indoor table version, and much of the modern nomenclature can be traced back to that common root. Cue sports today are vastly popular, and comprise variations such as pool, billiards, carom, snooker, and many other local flavours. In a 2005 U.S. survey, pool ranked as the eighth most popular participation sport in that country, with over 35 million people playing that year. Leagues and tournaments exist in nearly every country worldwide, with strong appeal to both experienced and casual players alike.

A number of robotic cue-players have been developed over the years. The first such system was The Snooker Machine from University of Bristol, U.K. in the late 1980's (Chang 1994). This system comprised an articulated manipulator inverted over a $1/4$-sized snooker table. A single monochrome camera was used to analyze the ball positions, and custom control and strategy software was developed to plan and execute shots. The system was reported to perform moderately well, and could pot simple shots. Since then, there have been a number of other attempts at automating pool, including (Alian et al. 2004; Lin, Yang, and Yang 2004).

Developing a complete robotic system that can be competitive against an accomplished human player is a significant challenge. The most recent, and likely most complete system to date, is Deep Green from Queen's University, Canada (Greenspan et al. 2008), shown in Figure 1. This system uses an industrial gantry robot ceiling-mounted over a full-sized pool table. High resolution firewire cameras are mounted on both the ceiling to identify and localize the balls, and on the robotic wrist to correct for accumulated error and fine tune the cue position prior to a shot. This system has been integrated with the physics simulator used in the computational pool tournaments, and the strategy software developed to plan shots. Complex shots have been effectively planned and executed, including combination shots, and the system plays at an above-amateur level.

In addition to full automation of cue sports, there have been attempts to apply technology to aid in training and analysis, including a vision system to analyze televised snooker games (Denman, Rea, and Kokaram 2003). The Automatic Pool Trainer from the University of Aalborg (Larsen, Jensen, and Vodzi 2002) makes use of a vision system and a steerable laser pointer. Once the static ball positions have been located by the vision system, the laser pointer is used to identify cue aiming positions that would lead to successful shots. A similar system, called "Mixed Reality Pool", is being pursued at Pace University in the U.S. (Hammond 2007).

Pool presents an interesting challenge for AI even absent a robotic component. It features a unique combination of properties:

- infinite state and action spaces
- varying execution skill (i.e., control uncertainty)
- complete observability of play but not of execution skill
- turn taking with state-dependent turn transfer
- and, of course, an adversarial nature.

After a few early investigations, the computational study of strategy in the game of pool was given a major boost with the advent of the first computational pool tournament, held in 2005 as part of the International Computer Olympiad (Greenspan 2005). The tournament was then repeated in 2006 (Greenspan 2006) and 2008.

The game played in all computer pool competitions to date has been 8-ball, based on the official rules of the Billiards Congress of America (1992). 8-ball is played on a

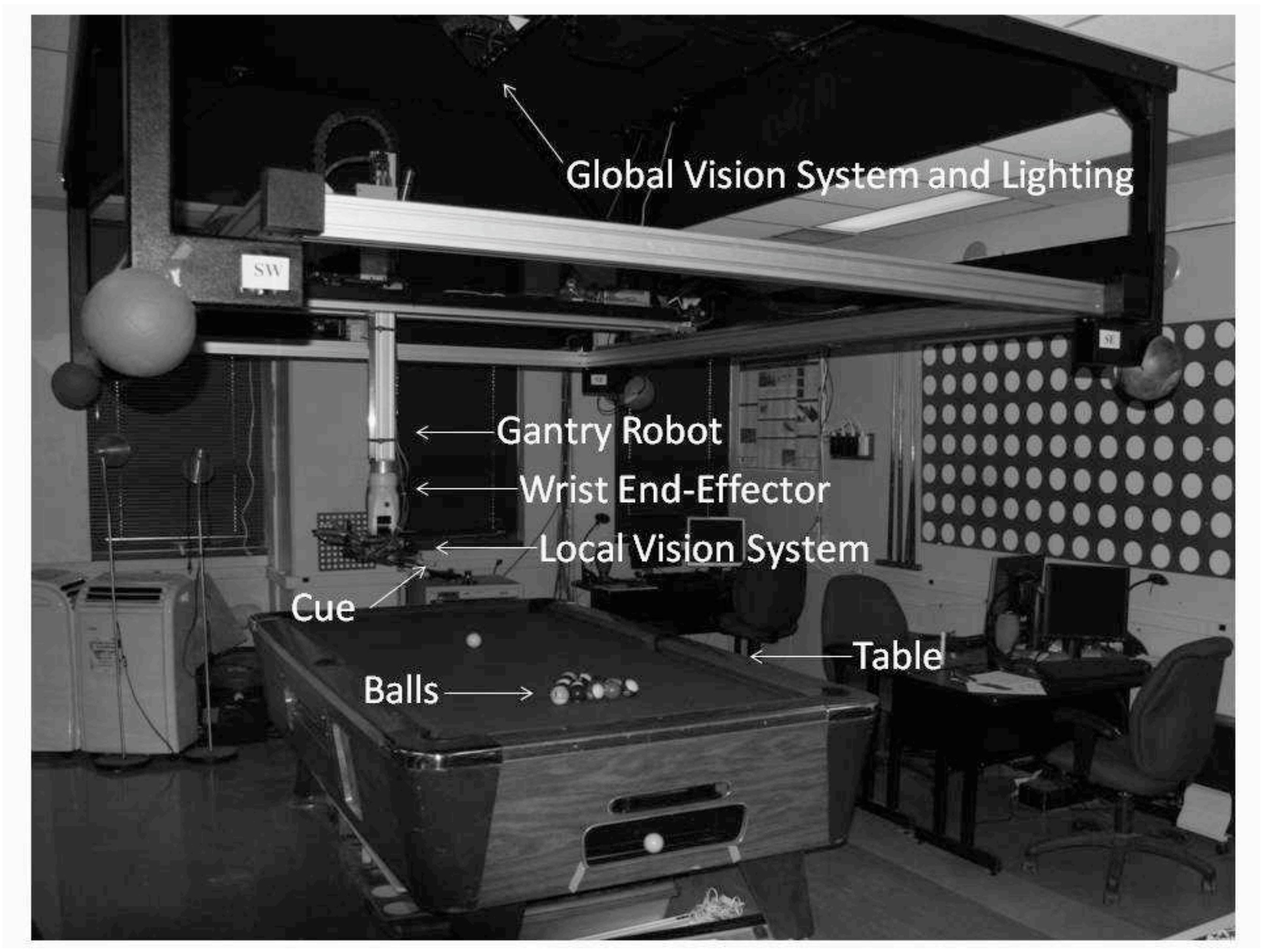


Figure 1: The Deep Green pool-playing robot

rectangular pool table with six pockets which is initially racked with 15 object balls (7 solids, 7 stripes, and one 8-ball), and a cue ball (see figure 2). Play begins with one player's break shot. If a ball is sunk on the break shot, then the breaking player keeps his or her turn, and must shoot again. Otherwise, the other player gets a chance to shoot.

For a post-break shot to be legal, the first ball struck by the cue ball must be of the shooting player's side, and the cue ball must not enter a pocket itself. The first ball legally pocketed after the break determines the side (solids or stripes) of each player. Until this occurs, the table is open, meaning that both solid balls and striped balls are legal targets for either player. Players retain their turn as long as they call (in advance) an object ball of their side and a pocket, and proceed to legally sink the called ball into the called pocket. Following an illegal shot, the opponent gets *ball-in-hand*. This means that they can place the cue ball anywhere on the table prior to their shot. After all object balls of the active player's side have been sunk, that player must then attempt to sink the 8-ball. At this point, calling and legally sinking

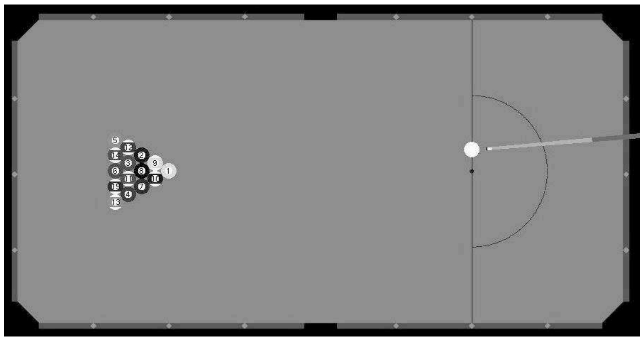


Figure 2: Computational pool table with balls racked for 8-ball

the 8-ball wins the game.

The computational pool tournaments are based on a client-server model where a server maintains the state of a virtual pool table and executes shots sent by client software agents on the PoolFiz physics simulator (Greenspan 2006). Each agent has a 10 minute time limit per game to choose shots.

A shot is represented by five real numbers: v, φ, θ, a, and b, which are depicted graphically in figure 3. v represents the cue velocity upon striking the cue ball, φ represents the cue orientation, θ represents the angle of the cue stick above the table, and a and b designate the horizontal and vertical offsets of the cue impact point from the center of the cue ball. The location of this point, where the cue stick stikes the cue ball, plays a big role in imparting spin, or "english", to the cue ball. φ and θ are measured in degrees, v in m/s, and a and b are measured in millimeters. Since the physics simulator is deterministic, and in order to simulate less than perfect skill, Gaussian noise is added to the shot parameters on the server side. The result of the noisy shot is then communicated back to the clients. The ICGA tournament's noise model was a zero-mean Gaussian distribution with standard deviations of $\sigma_\theta = 0.1$, $\sigma_\varphi = 0.125$, $\sigma_V = 0.075$, $\sigma_a = 0.5$, $\sigma_b = 0.5$.

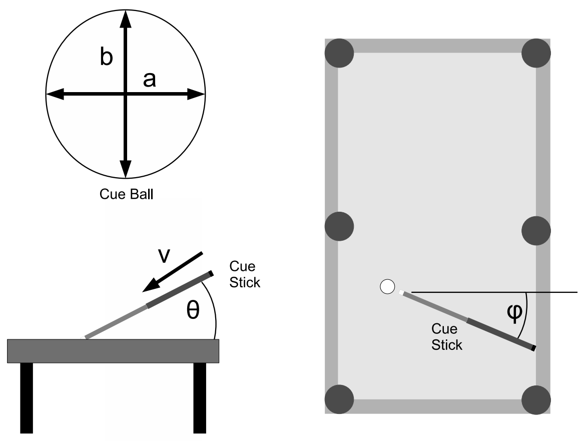


Figure 3: Computational pool shot parameters

The initial approaches explored for the computational pool tournaments varied significantly among competitors, including, for example, an optimization-based method that simulated human position play (Landry and Dussault 2007) a knowledge-based approach, and Monte Carlo search-tree methods (Leckie and Greenspan 2007). The technique that dominated the first two tournaments was based upon a variation of a search-tree method that performed look-ahead by three shots on a discretized version of the search space (Smith 2006). The following section describes in detail the program that won the most recent competition, in 2008, and various research it spawned. The final section includes a discussion of the future of computational pool and a Call for Participation in the next computational pool championships, slated to take place as part of AAAI in 2011.

The story of a program

In this section we will explore in more depth the setting of computational pool. We will do this by telling the story of the Stanford Computational Billiards group, our experience with computational pool, and by sharing some insights we have gained throughout this process.

Models and beginnings

Our first exposure to computational pool came at the AAAI conference in 2006. Michael Smith presented a paper (Smith 2006) describing the setting of computational pool and the agent PickPocket, which had won the 2005 computational pool tournament. His presentation caught our interest and sparked a desire to explore this intriguing new domain and to compete in the next tournament.

One of our initial questions concerned the value of game-theoretic reasoning to a computational pool agent's design. Most tournaments involving computer agents end up not using game-theoretic analysis. A few exceptions are the game of chess, if alpha-beta pruning and other similar techniques are considered as game-theoretic, and poker, where the equilibrium of an abstract version of the game is explicitly computed. Certainly a prerequisite for determining whether such analysis can prove fruitful in a new domain is to have a model to analyze, and finding such a model became our first goal.

Investigation quickly revealed that there was no existing model which cleanly captured the game of pool. Other game models existing in the literature had some similar qualities, but also were either missing critical features or contained extraneous features which needlessly complicated analysis. In order to reason about pool games, we introduced a model specifically tailored to representing pool games (Archibald and Shoham 2009). With some reasonable assumptions about the game components, we showed the existence of an equilibrium. This equilibrium is a *stationary* equilibrium, which means that players' equilibrium strategies do not depend on the history of the game, only the current table state.

Given this preliminary game-theoretic understanding of pool, our focus switched to the practical problem of designing a software agent capable of winning the next computational pool tournament.

Design Challenges

One of the most striking features of cue sports is their continuous action space. Most other games that have been the subject of much study and research, such as chess, checkers, go, etc., feature finite action spaces. Because of the continuous nature of the available actions, there are an infinite number of actions that can be attempted from any given table state. The game also features actions taken in sequence, which seems to reward planning ahead. One of the fundamental design challenges we faced was trading off depth and breadth of search. Is it better to spend more time experimenting with shots from the initial table state, or should we search and plan ahead in our attempt to run the table?

Another design question we faced involved dealing with the opponent. Specifically, how much should our agent reason about or notice its opponent? In theory, it seems clear that such considerations could make a big difference to the performance of an agent in actual matches. In practice, though, at the noise levels of the tournament, our agent was able to win nearly 70% of the time off the break (i.e. run the table). These wins occur regardless of who the opponent is. Thus, at the noise levels of the 2008 ICGA tournament, there was relatively little to be gained be adding extensive consideration of the opponent. This led us to only consider the opponent when no suitable shot can be found from a given state, at which point defensive strategies are invoked.

Computational pool agent design

In this section we describe our computational pool agent, CueCard, and discuss how it selects a shot for execution given a typical table state. We will also briefly mention what is done in atypical situations.

Given a table state, our agent proceeds as shown in Figure 5. First, interesting aiming directions (φ values) are calculated. This is done geometrically, and each direction is chosen in order to accomplish one of several goals. Directions for straight-in shots (where the object ball goes directly into the pocket), more complex shots with multiple ball-ball or ball-rail collisions, and special shots designed to disperse clusters of balls are all calculated. A shot is simulated without noise in each of these directions. If this shot successfully pockets a ball, then it is kept for further processing. For each of these successful shots, we discretize v and randomly vary the other parameters (a, b, θ) to generate a variant shot in the desired direction. Each of these variant shots, if successful without noise, is simulated n times with noise added. The resulting state of each noisy simulation is scored using an evaluation function. This function uses a look up table to estimate the difficulty of the remaining straight-in shot opportunities in the resulting state for the current player. The value of a specific state is equal to $1 * p_1 + 0.33 * p_2 + 0.15 * p_3$, where p_i is the success probability of the i-th most probable straight-in shot. The value of a specific shot is the average value of the states which resulted from the n noisy simulations of the shot. This process of generating, simulating, and evaluating variant shots continues until a predetermined amount of time has elapsed.

The top k of all shots that were evaluated are retained for further examination. To refine the evaluation of these k shots, another level of search is performed beginning from the states which resulted from the n noisy simulations of each of the k shots. Shot directions are again generated, and random variations of successful shots are simulated with noise. The average resulting state evaluation of the best shot from a given state is used as the new score for that state. After this second level of search is completed, we then recalculate the score for each of the k shots, using the new values generated from the second level of search. The shot with the highest overall score is selected for execution.

If the score for the selected shot does not exceed some threshold, then CueCard instead generates a defensive last resort shot. For this shot we consider only straight-in shot directions from the original table position. These shots are now evaluated based on the resulting position for us *and* the position for the opponent. The goal is to leave the opponent in a bad position if we miss, and us in a good position if we make the shot.

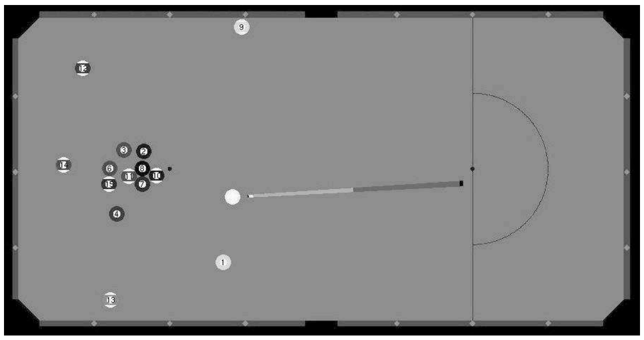


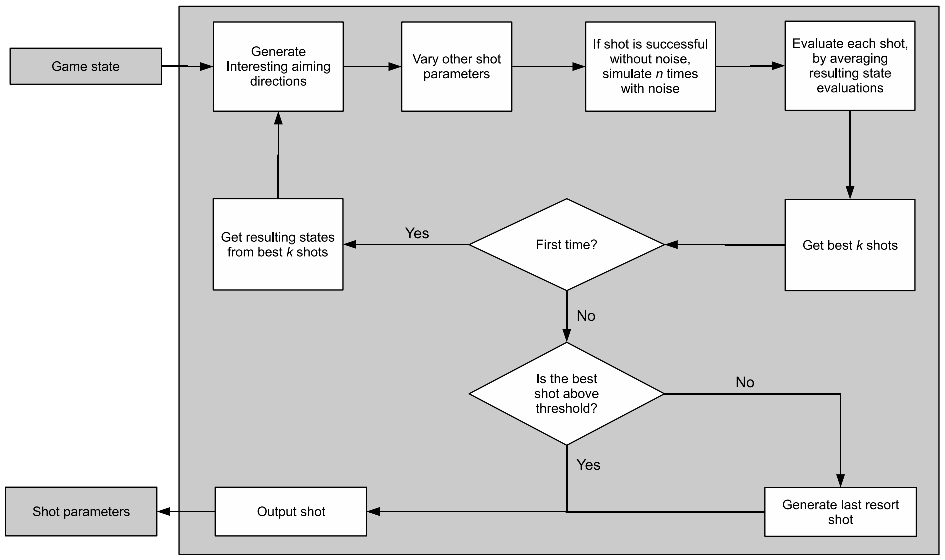
Figure 4: A typical post break table state

The break shot is not generated in this same manner, but instead is always the same and was the result of extensive offline search. The goal of this offline search was to find a break shot which kept the turn by sinking a ball, and also spread the balls out fairly evenly over the table. The shots which did best at each of these tasks were very poor at the other. For example, a break shot which sunk a ball 98% of the time also left the other balls in their racked position. A break shot which spread the balls out completely only succeeded in sinking a ball 64% of the time. These two goals are very interrelated. For example, if CueCard retains the shot but is then forced to break up a cluster of balls on its second shot, it has a high chance of losing the turn after the second shot. On the other hand, if the break shot spreads out the balls and gives up the turn, the opponent has been given the turn and a very easy table state. We found a compromise break shot which succeeded 92% of the time and also spread about half of the balls away from the intitial cluster. A typical state resulting from CueCard's break is shown in figure 4.

Ball-in-hand shots, which occur after the opponent fouls, differ from typical shots in that CueCard must first place the cue ball on the table and then execute a shot. To accomplish this CueCard generates interesting places to set the ball, typically close to an object ball and in position for a straight-in shot. For each of these locations a shortened single level search is performed. The shot with the highest overall score is selected for execution, and the cue ball is placed in the corresponding location.

Addressing Design Challenges

We return briefly to the design challenges mentioned earlier. To resolve the issue of trading off depth and breadth of search, we decided to have two levels of search, similar to previous competitors, and also to increase the amount of time we were able to spend exploring shots at each level. This was done by distributing the shot generation, simulation and evaluation over 20 computers and also by reimplementing and optimizing the physics. The reengineered simulator ran approximately five times faster than the previous version. As stated earlier, to address the opponent we



Game state
Generate Interesting aiming directions
Vary other shot parameters
If shot is successful without noise, simulate n times with noise
Evaluate each shot, by averaging resulting state evaluations
Get best k shots
First time?
Yes
Get resulting states from best k shots
No
Is the best shot above threshold?
No
Yes
Generate last resort shot
Output shot
Shot parameters

Figure 5: How CueCard chooses a shot in typical situations

decided that for the tournament noise level we would only consider the opponent in cases where no suitable shot existed. This situation occurred on less than 2% of shots.

Results

In the 2008 ICGA computational pool tournament CueCard played against two opponents who have participated in previous competitions: PickPocket, written by Michael Smith, (Smith 2007) and Elix, written by Marc Goddard. PickPocket was the champion of all previous competitions, and was thus the state of the art as we began designing CueCard. In the tournament, each agent played a 39 game match against each other agent. CueCard won the tournament with 64 wins (in 78 games), compared to only 34 wins total by PickPocket.

This was not enough games to statistically establish the fact that CueCard was the superior agent, and so, after the tournament, we ran more games between CueCard and PickPocket. In this later match, CueCard won 492-144 (77.4%), clearly establishing that it is the better player.

Analyis of performance

After the success of the tournament, we were naturally faced with many questions. Most notable was determining which of our improvements helped the most towards our victory. As we ran experiments to answer this question, we were surprised by the results (Archibald, Altman, and Shoham 2009). Most surprising was that the additional computers and reworked physics engine helped, but not very much. For example, the 19 extra CPUs only helped CueCard to a 350-282 (55.4%) victory in a match between the 20-CPU version of CueCard and a single CPU version of CueCard. Game specific tweaks, like the break shot, helped, but also didn't tell the whole story. This insight into how individual improvements did or did not contribute to CueCard's success is extremely valuable, and the results of these experiments are still impacting the next generation of billiards agent.

Generalization

Another natural question was whether any general theory or interesting observations could be extracted from the billiards environment? Pool gives us a unique set of characteristics, some of which are more clearly delineated than ever before. This provides an opportunity to study the effects of some of these characteristics on the interactions of agents within this framework. The hope is that some generally applicable principles can be gleaned from what is learned in the domain of pool.

The first such characteristic that has been investigated as a result of our foray into computational pool is that of execution precision. Adding noise to an agent's shot in a game of computational pool gives us a clear notion of a player's execution precision. We can modify this execution precision, making an agent more precise or less so. The first question we addressed was: How do changes in agents' execution precision impact the game? For example, could holding the tournament with a different level of noise have lead to a different agent winning? Is there a "best" noise level at which to hold the tournament?

We investigated these questions experimentally (Archibald, Altman, and Shoham 2010b) using redesigned client-server software. Using CueCard, as well as some other agents that were designed to be of differing strategic complexity, we ran many simulations to gain insight into the way that different noise levels affected the agents. We specifically examined how variations in the noise applied to agents' shots, and how the amount of time given the agents

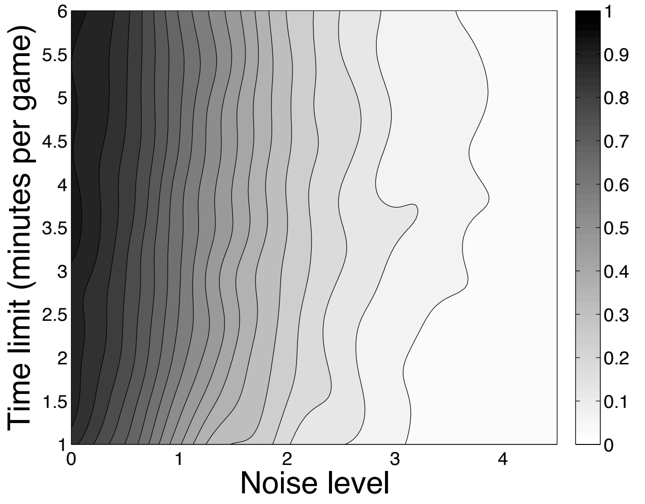


Figure 6: The impact of time and noise level on CueCard's performance

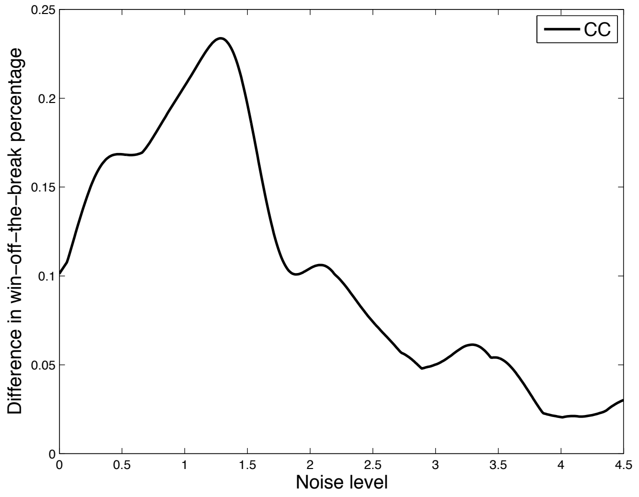


Figure 7: Difference time makes to CueCard at different noise levels

for computation impacted the agents' win-off-the-break percentages. Around 20,000 games were simulated for each agent at randomly chosen noise values and time limits. The noise was Gaussian with standard deviations varying between zero and five times the 2008 tournament values. Time limits were varied between one and six minutes per agent per game. For each simulated game, we recorded whether or not the agent won off the break. This raw binary data was then smoothed to provide an estimate of the win-off-the-break percentage for each agent at any combination of noise level and time limit. A contour plot showing the resulting win-off-the-break probability estimates for CueCard is shown in figure 6.

One of the aspects we investigated was how much having extra time helped the agents at each different noise level. For CueCard, extra time always helped, but the amount that it helped depended a lot upon the noise level. In figure 7 we show a plot of the difference in win-off-the-break percentage between CueCard with six minutes and CueCard with one minute for each noise level. The plot peaks at around 1.4 times the 2008 tournament noise level. If we consider an agent with more time to be a more strategic and intelligent agent, given that the agents are using the same high-level strategy, this suggests that the presence of noise might increase the chance of the most strategic agent winning.

This investigation naturally led to questions about the impact that execution precision has on agent interaction in general game settings. We proposed that execution precision be modeled as a restriction on the set of distributions over actions that an agent can utilize in a game (Archibald, Altman, and Shoham 2010a). We showed that an agent's safety level in a game, or the amount that it can guarantee itself, regardless of opponent, cannot decrease as its execution precision increases. However, when we look at the payoff that an agent receives in a Nash equilibrium of the game, there are cases where an agent would prefer to be have less execution precision.

The future of computational pool

The work so far on computational pool has barely scratched the surface of what this exciting new area has to offer, both in terms of practical experience with software agents and theoretical understanding and insight into unique features of agent interaction.

To increase our understanding on the practical side of things, future computational pool tournaments will branch out from the format used so far, which has been 8-ball at a single noise level. The first change we wish to make is to have each tournament feature competitions at different noise levels. As mentioned earlier, experiments have shown that noise affects each agent differently. Holding competitions at multiple noise levels would encourage exploration of new agent strategies and approaches to computational pool. In addition, there are many other settings to explore, including asymmetric settings, where the agents have different noise levels, settings where the exact noise level is only learned at run time, or settings where the exact noise level is never explicitly learned at all. We can also expand to other cue sports, such as 9-ball, one-pocket and snooker. While the underlying physics is identical in all variations, the different rules for each game create different incentives for the agents and could increase the value of modeling the opponent or searching ahead in the game.

To facilitate easier administration of future tournaments and to enable experimentation, a significant effort has been made to redesign the physics simulator and the server and client architecture to make the entire system more robust and usable. The new server has a web interface which enables easy administration of games, matches, and tournaments. All of this code is available for download. The goal is to lower the barrier to entry for new teams, allowing them to quickly implement and test any ideas they have about how

2011 International Computational Billiards Championships

Entry deadline: May 31, 2011

The 4th International Computational Billiards Championships will be held during August 2011, in conjunction with the 25th AAAI Conference. Software agents will face off in 8-ball in a variety of ***new*** formats. Join us!

visit

http://billiards.stanford.edu

for details

to design effective billiards players.

The next computational pool championships are planned for August 2011, to be held in conjunction with the Twenty-Fifth AAAI Conference. The championships will feature separate competitions at different noise levels, allowing for innovation and new ideas, since new strategies may be most effective at the new noise levels. We invite you to visit our website for details (`http://billiards.stanford.edu`), and join us on this exciting journey.

References

Alian, M. E.; Shouraki, S.; Shalmani, M.; Karimian, P.; and Sabzmeydani, P. 2004. Roboshark: A gantry pool player robot. In *Proceedings of ISR 2004*.

Archibald, C.; Altman, A.; and Shoham, Y. 2009. Analysis of a winning computational billiards player. In *Proceedings of IJCAI 2009*, 1377–1382.

Archibald, C.; Altman, A.; and Shoham, Y. 2010a. Games of skill. Manuscript.

Archibald, C.; Altman, A.; and Shoham, Y. 2010b. Success, strategy and skill: an experimental study. In *Proceedings of AAMAS 2010*.

Archibald, C., and Shoham, Y. 2009. Modeling billiards games. In *Proceedings of AAMAS 2009*, 193–199.

Billiards Congress of America. 1992. *Billiards: The Official Rules and Records Book*. New York, New York: The Lyons Press.

Chang, S. W. S. 1994. *Automating Skills Using a Robot Snooker Player*. Ph.D. Dissertation, Bristol University.

Denman, H.; Rea, N.; and Kokaram, A. 2003. Content-based analysis for video from snooker broadcasts. *Computer Vision and Image Understanding* 92(2/3):176–195.

Greenspan, M.; Lam, J.; Leckie, W.; Godard, M.; Zaidi, I.; Anderson, K.; Dupuis, D.; and Jordan, S. 2008. Toward a competitive pool playing robot. *IEEE Computer Magazine* 41(1):46–53.

Greenspan, M. 2005. UofA wins the pool tournament. *International Computer Games Association Journal* 28:191–193.

Greenspan, M. 2006. Pickpocket wins pool tournament. *International Computer Games Association Journal* 29(3):153–156.

Hammond, B. 2007. A computer vision tangible user interface for mixed reality billiards. Master's thesis, Pace University.

Landry, J.-F., and Dussault, J.-P. 2007. Ai optimization of a billiard player. *Journal of Intelligent and Robotic Systems* 50(4):399–417.

Larsen, L.; Jensen, M.; and Vodzi, W. 2002. Multi modal user interaction in an automatic pool trainer. In *Proceedings of ICMI 2002*, 361–366.

Leckie, W., and Greenspan, M. 2007. Monte carlo methods in pool strategy game trees. In *5th International Conference on Computers and Games*, volume 4630 of *Lecture Notes on Computer Science*, 244–255.

Lin, Z.; Yang, J.; and Yang, C. 2004. Grey decision-making for a billiard robot. In *Proceedings of SMC 2004*, 5350–5355.

Smith, M. 2006. Running the table: An AI for computer billiards. In *Proceedings of AAAI 2006*, 994–999.

Smith, M. 2007. PickPocket: A computer billiards shark. *Artificial Intelligence* 171:1069–1091.