

Optimization of a Billiard Player – Tactical Play^{*}

Jean-Pierre Dussault and Jean-François Landry

Département d'Informatique,
Université de Sherbrooke, Sherbrooke (Québec), Canada
{Jean-Pierre.Dussault, Jean-Francois.Landry}@USherbrooke.CA

Abstract. In this paper we explore the tactical aspects needed for the creation of an intelligent computer-pool player. The research results in three modifications to our previous model. An optimization procedure computes the shot parameters and repositions the cue ball on a given target. Moreover, we take a look at possible heuristics to generate a sound selection of targets repositioning. We thus obtain a greedy but rather good billiard player.

1 Introduction

Our work concerns the development and improvement of an automated billiard player modestly named PoolMaster. The billiard game presents mechanical continuous issues (how the balls roll, bounce, etc.), and planning issues (strategic play). Most classical games (chess, checker, go, etc.) deal only with a planning difficulty, monthly addressed by some form of tree search. Billiard requires planning, but also accuracy. Following the main line of our research, we concentrate on the physical accuracy of our player. We identify the point where planning will be unavoidable in order to improve the player, but as we show, the accuracy improvement to be presented yields a much stronger POOLMASTER than its predecessor, that was runner up in the first Computer Pool tournament [7].

In [6], we presented a key computation to develop an optimal billiard player: the skill to sink a ball while achieving good position for the next shot. This is a very basic skill. We then developed an optimization model to compute the actual shot parameters required to sink the ball, and to reach a precise target without addressing the non trivial aspect related to how we choose the target point.

In this paper, we do not discuss the basic optimization methodology developed in [6]. We focus on building some tactical behavior within a player exploiting the aforementioned optimization model. Of course, we will refine the optimization model now and then to represent new items of the positional play unveiled by tactical issues that are investigated.

To begin with, we delineate precisely the context in which the player evolves. We state the basic assumptions on which we rely to construct the player.

Next, we present a model (similar to [4]) to compute the actual difficulty of sinking a given ball, based on the maximum allowable error in the shots'

^{*} This research was partially supported by NSERC grant OGP0005491.

parameters still ensuring the ball is sunk; we consider direct shots as well as kick and bank shots. This basic model allows us to rank the remaining balls from the easiest one to the most difficult one (other approaches have also been developed in [2,3,5,10]).

By identifying the best spot we choose where to put the cue ball on. This will be used as a target, or as the spot on which we put the cue ball when having ball in hand. Actually, we develop an optimization heuristic to build an ordered list of a few good spots. We discuss variants of the evaluation function.

Finally, we develop a generalization of the basic optimization model [6] to take into account three related deficiencies of the original proposition. Sometimes, the cue ball scratches (gets sunk), other times, the object ball does not reach the pocket, and finally, in some situations, it is not required to pin the cue ball at a given target, but to hit some cluster (to break it) with a minimum speed. Those three situations call for incorporating terms in the optimization model that reflect non vanishing speeds of either the cue or the object ball.

Before concluding, we present some limited statistical observations confirming that POOLMASTER is indeed a quite strong billiard player.

2 Context of Computer Pool

In order to produce an automated billiard player, the physics of the game must be understood, modeled, and simulated. For physical modeling we refer to [1,7]. We take as a black box a *simulator* [9] which “executes” the proposed shots. If the game is played using the same simulator, the computed shots and their outcome will match. If the game is played on a real table, the behavior of the shots will certainly differ as no simulator can ever be perfect. The simulator’s input consists of the following five parameters:

- a and b represent the horizontal and vertical offset of the cue tip from the center of the cue ball;
- θ the elevation of the cue stick;
- ϕ the aiming direction;
- V , the speed of the cue stick.

Our player specifies legal values for those five parameters. Given the values, the simulator provides information called the table state, indicating every ball position, or flagging pocketed ones.

The first computer pool tournament was held during the 10th computer Olympiad, a parallel event with ACG 11 in Taipei [7]. The simulator was used by the “computer referee”, and acted as the table as well. In the tournament, some Gaussian noise is added (with [8]) to the prescribed shot parameters to represent the unavoidable inaccuracies of any player, human or robot; this also allows, albeit indirectly, to acknowledge the limitations of the simulator which neglects uneven cloth thickness, imperfect cue stick tip, and so many other real-life details.

Our actual implementation uses the simulator as a black box. Therefore, any improved version of the simulator is likely to improve the realism of our player.

We plan to develop our own simulation engine in order to allow some cooperation between the simulator and the optimizer. In particular, the simulator could be enriched to yield some additional information useful for sensitivity analysis. The optimizer ([11]), which handles non linear constrained least-squares models, relies on some derivative information. Since we now use the simulator as a black box, derivatives are obtained via finite differences.

3 Shot Difficulty

This section provides functions to compute the difficulty of a shot. We are given the cue ball's position, the object ball position, and the aimed pocket. It is based on the margins of the error concept described in [1,4], and the related technical proofs available from the book's WWW page. We also provide extensions to estimate the difficulty of bank and kick shots. We use the estimate of shot difficulty to obtain a priority order, since we consider not only the easiest shot to pocket, but also the repositioning. Therefore, works like [2,3,5,10] are not directly applicable for us. Below, we present a straightforward model to perform the ranking we require.

3.1 Direct Shots

The difficulty of pocketing a given object ball is proportional to the error margin the player has, compared to the perfect shot. The object ball need not reach the center of the pocket. The possible angle θ depends on the distance of the object ball to the pocket and the pocket width. Once this angle is known, we may compute a target window that the cue ball must reach so that the object ball's trajectory remains within the angle θ_p . The maximum angle θ_c allowing the cue ball to reach the target window is a measure of the shot difficulty. So we have

$$\cos \theta_c = \hat{\mathbf{c}}_1 \cdot \hat{\mathbf{c}}_2 \quad (1)$$

with

$$\mathbf{c}_1 = p_{c1} - p_c \quad (2)$$

$$\mathbf{c}_2 = p_{c2} - p_c \quad (3)$$

where p_{c1} , p_{c2} are the two positions where the cue will have to hit the object ball to pocket it at the two extremities of the pocket. As in [6], vectors like \mathbf{c}_1 and \mathbf{c}_2 are denoted in bold, their norm is in math italics c_1 and their normalization inherits a hat: $\hat{\mathbf{c}}_1 = \frac{\mathbf{c}_1}{c_1}$. \mathbf{c}_1 and \mathbf{c}_2 are the two vectors going from the cue ball position (p_c) to the cue ball hit points, as seen in Fig. 2.

However, as can be seen in Fig. 3, this approach may give undesired results in situations where we have a very big cut angle. Further calculations would need to be added to take this into account and that is why we choose to use another method instead, to estimate this difficulty which will require fewer operations and still give a good approximation.

We can easily compute the cosine of the cut angle $\cos \alpha = \hat{\mathbf{v}}_{op} \cdot \hat{\mathbf{v}}_{co}$ where \mathbf{v}_{op} represents the object ball-pocket vector, and \mathbf{v}_{co} the cue ball-object ball vector.

For a corner shot, we have our shot difficulty coefficient κ_c :

$$\kappa_c = \frac{\cos \alpha}{v_{op}v_{co}} \tag{4}$$

For a side shot, we still use a similar method but this time, we add a new parameter γ which will represent the angle between the object ball and the \mathbf{x} -axis of the pocket. This may make sure that we account for the fact that a shot in a direct line with the pocket is much easier than one having an angle. The parameter γ is defined as: $\cos \gamma = \hat{\mathbf{v}}_{op} \cdot \hat{\mathbf{x}}$ with $\hat{\mathbf{x}} = \mathbf{x} = [1, 0, 0]$.

We now arrive at the side shot difficulty with:

$$\kappa_s = \frac{\cos \alpha \cos \gamma}{v_{op}v_{co}} \tag{5}$$

Here we will explicitly notice the fact that a side shot often needs more precision than a corner shot.

It is also important to note that a cut-off is used for γ to make sure that in the case of a very small angle, which would result in an impossible shot, we ignore that shot. We remark that in ideal cases this should never happen since a good detection of possible shots is to be foreseen in the future. In our case, with the help of empirical results, we defined the cut-off value at 0.25.

Our κ is an intuitive measure of the shot difficulty. We present some statistical observations in Subsect. 7.1.

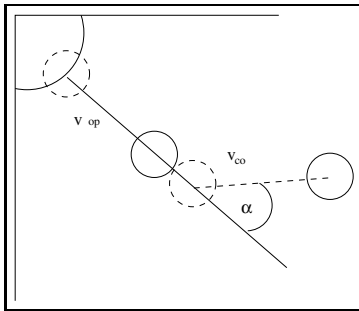


Fig. 1. Difficulty coefficient estimated with cut angle and distance

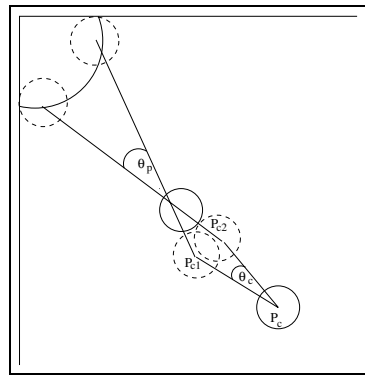


Fig. 2. Difficulty coefficient represented by alpha

3.2 Kick/Bank Shots

In some cases, often when there is a very big cut angle, it is possible that the easiest of the direct shots is still harder to perform than a bank or kick shot. To assess the difficulty of these indirect shots, we create a mirror image of the

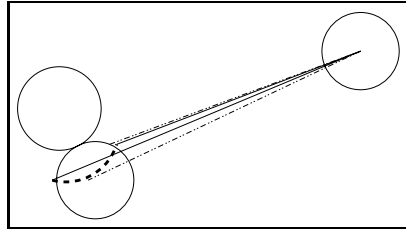


Fig. 3. Problems with the exact difficulty coefficient

current table on the corresponding side. In the case of a kick shot, we mirror the position of every ball on the table except the cue ball. For a bank shot we do the same except for the object ball. This enables us to calculate the difficulty of the shot as if it was a direct shot. We do so by taking the coordinates and aiming for the pockets of the mirror table. Of course, the coefficient calculated are only valid if we do a standard shot with no spin added. However, this still gives us a good idea of how hard the shot is. The mirror approach is incorporated in our optimization model and can be very helpful when there are no possible direct shots. Fig. 4 shows an example of a kick shot.

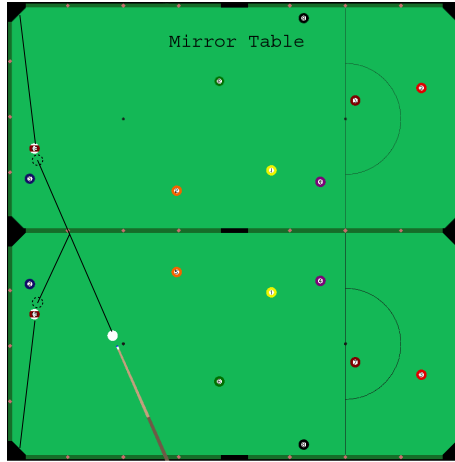


Fig. 4. Since in this case there is no direct shot possible (stripes), we will do a kick shot using the north rail by computing a mirror image of the table

We use the mirror table to estimate the shot’s difficulty. When actually computing the shot, the optimization process benefits from an initial guess obtained by the mirror paradigm; moreover, the actual optimized shot will take into account the rebound properties of the rail. Since the rebound properties of the rail certainly affect the shot difficulty, we do not use it in our estimate.

3.3 Combinations

At times, it may be possible that even a hard shot like a combination is the best shot to perform. To determine correctly how hard this shot will be, we may use the same approach as for direct shots, but we need to add two more parameters since we now have two cut angles and three distances. The formula (5) now becomes (for a corner shot):

$$\kappa = \frac{\cos \alpha_1 \cos \alpha_2}{v_{o_2 p} v_{c o_1} v_{o_1 o_2}} \quad (6)$$

where the notation $\mathbf{v}_{p_1 p_2}$ stands for the vector joining the positions p_1 and p_2 , and $v_{p_1 p_2}$ is its norm; o_1 is the position of the first object ball, o_2 the second, c the cue ball, and p the pocket; α_1, α_2 are the two cut angles. Of course, for a side shot, we must scale κ by $\cos \gamma$.

4 Global Table Difficulty

Below, we present variants of a global table difficulty coefficient. We are given the balls' positions, and we provide a way to combine the individual object balls' difficulties into a global measure. This ability is quite important, especially at the beginning of the game if the table is still open as it will let the player choose the best balls to aim for (low or high) based on how each ball is positioned. It will also be quite favorable when the player has a ball-in-hand as we will see in Section 5.

Since we already have a coefficient of difficulty for any particular shot from Sect. 3, we can use this as part of our calculations. We consider two strategies and combine individual shot difficulty coefficients into a global table difficulty.

A naive approach would be to list and sum up all the shot difficulty coefficients, but as we were able to find out, this would seem an approach that is too general and will not lead to the appropriate positioning of one particular ball.

Instead, we suggest to use the average and the maximum of the shot difficulty coefficients, which will provide a more appropriate value if the positioning is closer to one ball. We can see the best cue ball positions as outlined by the contours in Fig. 6. The main difference between the sum formula and the average lies in the fact that the number of considered balls increases the sum, but not the average. Therefore, the average focuses slightly more on the best candidates. This can be seen in the Figs. 6 and 5. The same comparison can be made with the maximum formula, but to a higher degree since we now only reposition for one ball and forget the other possible ones, thus we have a more aggressive repositioning. Those represent the level sets of the function. In Fig. 5, the sum clearly shows a preferred position while in Fig. 6, the average measure shows two other rather good spots. If we look at the max in Fig. 7, its also very straightforward to see how we reposition for one ball only, which would definitely of use if we want to look a few shots in advance.

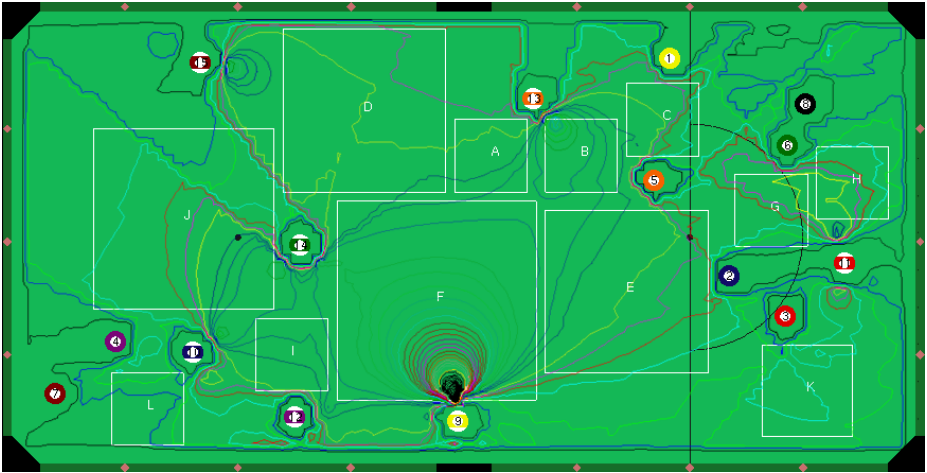


Fig. 5. Outline of the best regions for the cue ball positioning (aiming for the stripes) using the sum measure. In this case, it is obvious that the best area is close to the 9 ball as it will be an easy direct shot and also has all of the other stripes in direct view. Other good spot where fewer balls are in direct view are simply not competitive.

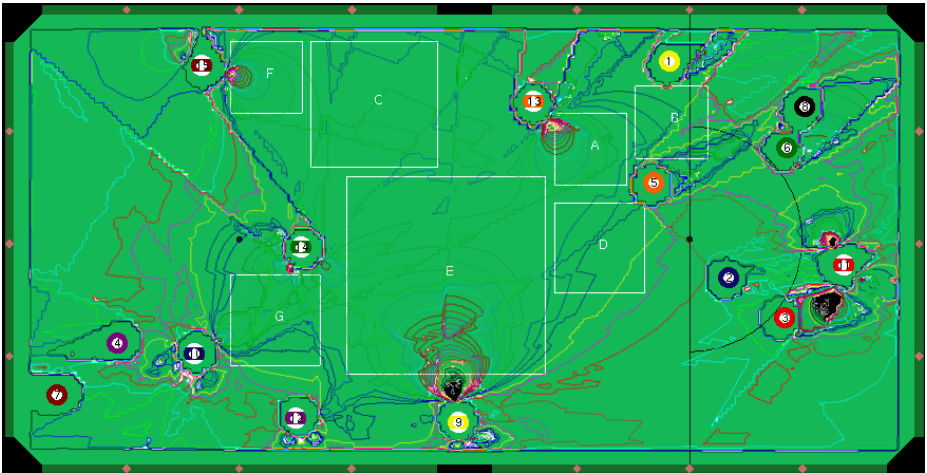


Fig. 6. Outline of the best regions for the cue ball positioning (aiming for the stripes) using the average measure. In this case, the number of relatively easy balls increases the measure. Apart from the spot close to the 9, two other rather good spot appear: close to the 11, where the average considers only the 11 since all other balls are hidden, and close to the 13. Observe that the spot close to the 11 ball is small, since positions where another ball becomes visible but more difficult because of the distance affect the average.

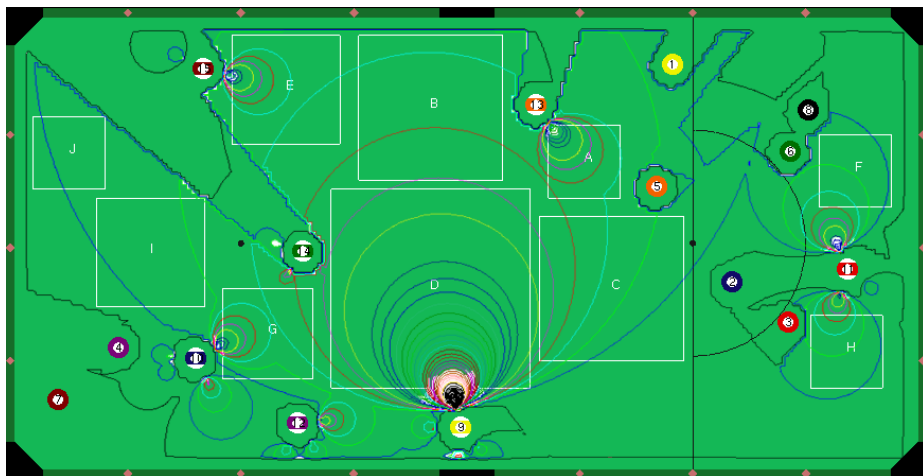


Fig. 7. Outline of the best regions for the cue ball positioning (aiming for the stripes) using the maximum measure. In this case, we immediately see each zone is focused on one ball for one pocket and nothing else. It is a much more aggressive play but one that a more talented player will probably use since he will rarely miss his shots.

In the event of the cue ball not having a clear path to the object ball, we still have to take it into account by adding a small value to our coefficient, since the path might clear out after the first shot is made. When that happens we will evaluate the shot difficulty as if the path was clear, but we add only half of that value to our computation.

5 Best and Worst Spots on the Table

In this section, we provide an heuristic optimization procedure to identify the best spots to aim for on the table. Moreover, we discuss the simplifying assumptions on which our optimization relies.

Spacially (seen on the table) the table difficulty is a non-linear complex function. We are interested in approximating good spots, corresponding to good local maxima. There might be several good spots. Clearly, we are not only interested in the global maximum of this function. Therefore, we propose to search (using an ascent method) local maxima starting from points on a grid. Several starting points usually lead to the same local maximum, and we are left with a short list of good spots. The relevant function reads

$$\max_{x,y} f(x,y). \quad (7)$$

with $f(x,y)$ representing our evaluation function.

We use this technique to find the best spot on the table: in case it is not possible to reach one of these spots, we can use the inverse of the function to find the worst spot and put the opponent in a tough position with a safe shot.

There are many other ways to explore a 2D function over a space described by a rectangle. We may evaluate f on a coarse grid, and perform a similar search on refined promising sub-grids (displayed by the square zones on Figures 6, 5, 7). However, even a 100×50 grid yields an accuracy of only $2cm$. We prefer the technique described above, since it takes advantage of the continuous nature of the function. Further enhancements will include heuristics to restrict the search and to facilitate attaining preferred regions.

6 Optimization with Ball Speed Considerations

Having solved the problem of finding a good target to reach with the cue ball after sinking an object ball, we may concentrate on assembling a much more complete objective function. We will use this section to refine our previous optimization model and solve some important issues to create a really solid player.

6.1 Sinking Object Ball with Minimum Speed

We start by addressing one of the most important matters, which consists in making sure that the object ball is actually sunk in a pocket during the optimization of the cue ball spin and velocity. Since our optimization method will try to do everything in its power to get the cue ball at the designed target, it is possible that in doing so it will reduce its velocity which in turn might cause the object ball not to be sunk. In our previous model, we had the objective function:

$$\min_{\mathbf{v}_o, \omega_o, \mathbf{c}, \bar{t}, \tilde{t}} (\|\mathbf{p}(\bar{t}) - \mathbf{c}\|^2 - 4R^2)^2 + \|\bar{\mathbf{p}}(\bar{\tau}_f) - \mathbf{s}\|^2 + \|\tilde{\mathbf{p}}(\tilde{t}) - \mathbf{b}\|^2 \tag{8}$$

which we will simplify (for the sake of this article) to :

$$\min_{\mathbf{v}_o, \omega_o} \frac{1}{2} (\|\mathbf{p}_c(\tau_f) - \mathbf{c}\|^2 + \|\mathbf{p}_o(\tau_f) - \mathbf{p}\|^2). \tag{9}$$

\mathbf{p}_c , \mathbf{p}_o are the positions of the cue and object balls at their final resting time, \mathbf{c} the cue ball target, and \mathbf{p} the pocket aimed.

By adding a new component to this objective function to penalize the object ball speed, we are able to ensure it will reach the aimed pocket with the desired minimum speed. Let $\mathbf{v}_o(\tau_f)$ be the speed of the object ball at its final time τ_f and \mathbf{v}_{min} the minimum speed. We can rewrite equation (9) into

$$\min_{\mathbf{v}_o, \omega_o} \frac{1}{2} (\|\mathbf{p}_c(\tau_f) - \mathbf{c}\|^2 + \|\mathbf{p}_o(\tau_f) - \mathbf{p}\|^2 + \max(v_{min} - v_o(\tau_f), 0)^2). \tag{10}$$

6.2 Cue Ball Scratching

A second problem is the cue ball scratching. More often than not, if the cue ball is pocketed after sinking the object ball, it will stay that way because the objective function will then hit a stationary area while minimising its position

and will consider it as a local minimum. To help the function get out of this bad situation, we introduce a new term $v_c(\tau_f) - v_0$ which represents the velocity of the cue ball at its final resting time (v_0 representing a null speed). If the cue ball is immobilised on the table at its final time, then this component will not affect the rest of the function. However, if the cue ball is pocketed, its final velocity will not be null and the optimization will continue to find a better minimum and climb out of the pocket. We slightly complicate our previous equation (10) to

$$\min_{\mathbf{v}_0, \omega_0} \frac{1}{2} (\|\mathbf{p}_c(\tau_f) - \mathbf{c}\|^2 + \|\mathbf{p}_o(\tau_f) - \mathbf{p}\|^2 + \max(v_{min} - v_o(\tau_f), 0)^2 + (v_c(\tau_f) - v_0)^2). \quad (11)$$

6.3 Breaking Clusters

A small trick will help us to solve a third problem, i.e., breaking clusters of balls. Sometimes in a game it may happen that there is simply no way to reposition the cue ball correctly. When that problem arises, it may be beneficial to try and break a cluster containing one or more of our balls to plan ahead for future shots. We have most of the necessary components available, and all we need to add is a time of impact τ_i to our minimization:

$$\min_{\mathbf{v}_0, \omega_0, \tau_i} \frac{1}{2} (\|\mathbf{p}_c(\tau_i) - \mathbf{c}\|^2 + \|\mathbf{p}_o(\tau_f) - \mathbf{p}\|^2 + \max(v_{min} - v_o(\tau_f), 0)^2 + (v_c(\tau_i) - v_{cmin})^2). \quad (12)$$

We can now easily define a minimum speed at which the cue ball should hit the aimed cluster with v_{cmin} .

Actually, POOLMASTER does not wait until being without a good position shot to attempt breaking the clusters; we use a straightforward heuristic to trigger the choice of breaking the annoying cluster instead of aiming for a reposition. This strategic issue is one that will benefit from a planification approach, perhaps using a game tree.

7 Improved PoolMaster

Now it is the time to look at the actual benefits obtained by incorporating the features described above into our player. We will formulate observations to assess the gain in strength, and also will document some of our choices by comparisons. We present in Subsect. 7.1 empirical qualitative comparisons between the actual success rate and our κ difficulty coefficient. Thereafter, all our results will be assessed by mini-tournaments between competing variants. The results were obtained on a local server with a referee created to the exact image of the one used for the official tournaments (exact same rules and noise). As mentioned before,

the server adds some additive noise to the shot parameters. The higher the noise level, the closer the different competitors. This makes sense, since a very high noise level makes the probability to miss a shot important, and so buries the expected reward of the refined player. We thus present two classes of results. Small noise to simulate a professional player, and higher noise to simulate a good amateur player.

Adding noise evaluates the robustness of the strategy used. We believe that no line of play will have a significant success under sufficiently high noise levels, but some form of stochastic optimization will certainly improve under small noise levels.

7.1 Shot Difficulty

As discussed in Sect. 3, the difficulty of a given shot depends on the error margin of the player when executing the shot. This is a complex function of all the shot parameters. Moreover, given a noise specification on the shot's parameters, we focus on the probability of success.

At the time to select a shot, we wish to rank the different options. In order to do so, let us illustrate that the measure we propose, $\frac{\cos \alpha}{p_{op} p_{co}}$ is qualitatively reasonable. In Figs. 8 and 9, we compare the actual success rate of several cases labeled $(d_1 - d_2)$, with d_1 the distance between the object ball and the pocked, and d_2 from the object ball to the cue ball. We may conclude that our formula is a very good predictor for ranking purposes. As seen on the graphs, $\sqrt{\cos \alpha}$ would seem to fit slightly better to the observations, but we still prefer the plain \cos , having an intuitive physical interpretation (reduction of a window with increasing angle). We remark that the fit cannot be perfect: for sufficiently small values of the noise, the success may well be 100% for all but the most severe cut angles. This does not mean that the difficulty is constant, but that the difficulty lies within the player's accuracy.

To fine tune defensive play, i.e., to decide when to attempt a shot, and when to use rather a defensive line, we need a refined estimate. A greedy approach is simulating the shot N times, recording the successes, and using this as an estimate.

7.2 Choice of the Global Table Coefficient

In Tables 1 and 2, we compare the three table coefficients introduced in Sect. 4, using both the optimisation method and a 2d grid zone scan method. Our simulated tournaments indicate that all target solution variants are more or less equivalent for the moment being. We also tested one match of Optim vs Zones to see which would be the real winner but the results weren't enough descriptive to do every matchup which would've taken 36 tournaments of 100 games. It is quite possible that the differences are so slight that they are covered by other flaws of the player or simply by the noise added to the shots.

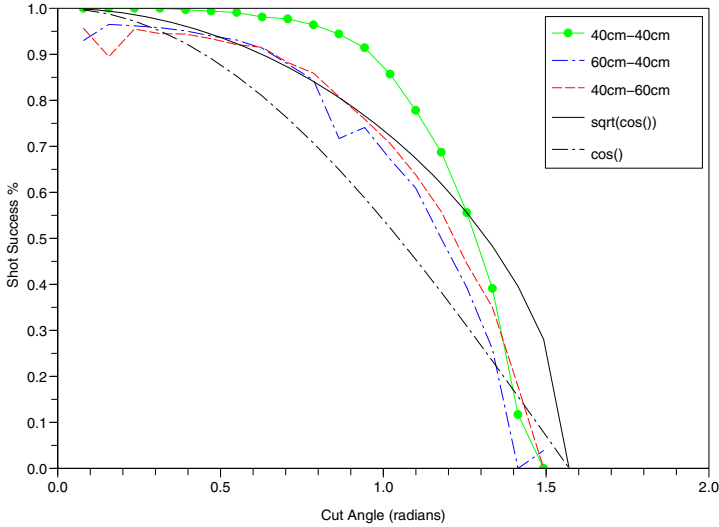


Fig. 8. Shot percentage for (40–40), (40–60) and (60–40). We compare the actual percentage of success for the tournament noise level. The qualitative fit with $\sqrt{\cos(\alpha)}$ is good. In abscissa, we sample a quarter of a circle.

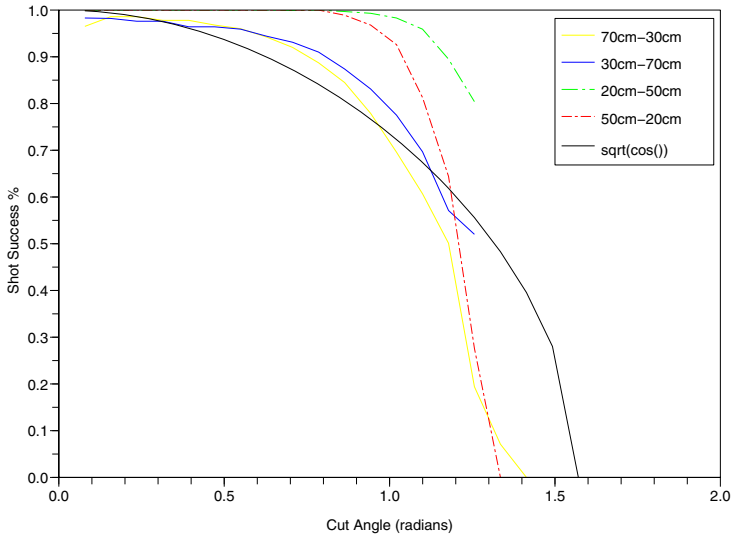


Fig. 9. Shot percentage for (20–50) and (30–70). Here, we consider easier shots involving situations where the distance of the object ball to the pocket differs significantly from the distance of the object ball to the cue ball. Again, the fit with $\sqrt{\cos(\alpha)}$ is still plausible, and the equivalence between (20–50) and (50–20), as well as (30–70) and (70–30) is observed.

Table 1. Optim vs 2d zone scan

| Player | Small noise | | Higher noise | |
|----------------|-------------|----------|--------------|----------|
| | Zones | Optim | Zones | Optim |
| Average vs Sum | 50 vs 50 | 50 vs 50 | 46 vs 54 | 57 vs 43 |
| Average vs Max | 50 vs 50 | 45 vs 55 | 50 vs 50 | 44 vs 56 |
| Max vs Sum | 60 vs 40 | 46 vs 54 | 50 vs 50 | 51 vs 49 |

Table 2. Zones Max vs Optim Max (small noise)

| Player | Wins |
|-----------|------|
| Zones Max | 56 |
| Optim Max | 44 |

7.3 PoolMaster Old vs. PoolMaster New

PoolMaster Old vs. PoolMaster New. The original POOLMASTER used a straightforward global table coefficient, and only a back-top spin. No ball speed was incorporated in the optimization model. Therefore, it might happen that sometimes the object ball was going in the right direction, but did not reach the pocket.

PoolMaster—improved uses the refined global table coefficient, any possible spin, bank and kick shots, and is able to break clusters. Under the higher noise model, though, the differences are quite a deception. The noise destroys the positioning computed by both players; since the differences lie in the refined positioning capabilities, the results are very close. This is not a surprise: a refined player relies on precision, destroyed by too much noise. And this is also why a realistic virtual pool tournament would probably benefit from a much better/advanced noise model instead of just approximate values.

In Tables 3 and 4, by small noise level, we mean Gaussian noise with mean zero and standard deviation $a : 0.08, b : 0.08, \theta : 0.003, \phi : 0.0185, v : 0.0085$ while higher level is $a : 0.8, b : 0.8, \theta : 0.03, \phi : 0.185, v : 0.085$.

Table 3. POOLMASTER vs. POOLMASTER NEW

| Player | Small noise | | Higher noise | |
|----------------|-------------|--------|--------------|--------|
| | Wins | Points | Wins | Points |
| POOLMASTER | 37 | 699 | 52 | 740 |
| POOLMASTER NEW | 63 | 800 | 48 | 767 |

Simple Player vs PoolMaster New. Here we compare POOLMASTER NEW to a simple greedy player, one which always chooses the easiest ball on the table, and does not deal with indirect or safety shots. The results are quite obvious, the simple player rarely manages to win a game since he never tries to correctly

Table 4. Simple Player vs. POOLMASTER NEW

| Player | Small noise | | Higher noise | |
|----------------|-------------|--------|--------------|--------|
| | Wins | Points | Wins | Points |
| Simple Player | 3 | 321 | 12 | 443 |
| POOLMASTER NEW | 97 | 970 | 88 | 880 |

reposition for the next shot. We can also see he won a few more games playing on a higher noise table, which probably indicates the other player also missed a few more shots due to noise.

8 Conclusion

In this paper we pursued the development of an optimized billiard player. Building on the strength provided by accurate position play, as described in [6], we studied choices of target and aimed at repositioning. We use new optimization ideas to set up a short list of good targets based on estimating shot difficulties, and on a global evaluation function.

We further refined the positioning model in several items. First, we took into account non vanishing final velocities, allowing to include mini breaks of clusters of balls as a position target; in this case, the cue ball must hit the cluster with some velocity in order actually to break it. Also, we considered kick, bank, and combination shots.

The resulting player is rather strong, but we did not take into account any look ahead strategy yet. Without noise in the execution of the optimized shot, the player never misses but sometimes gets stuck with nothing to play. With kick, bank, combinations shot added to the possibilities, it is very rare that the player finds itself without play, but it happens, most often when he fails to break a cluster.

Of course, a realistic simulation will incorporate some random noise, whatever small, which may dictate, in some circumstances, to opt for defensive play instead of taking the risk of the easiest shot, still too difficult. The use of a look ahead strategy will allow to avoid to recourse to a defensive shot most of the time. This is the next step toward an optimal player, namely the planning part of the game, which will probably be addressed by stochastic dynamic programming.

Acknowledgments

We wish to warmly thank the referees, whose comments were so helpful in improving the original version of this paper.

References

1. Alciatore, D.G.: The Illustrated Principles of Pool and Billiards. Sterling Publishing (2004)
2. Alian, M.E., Lucas, C., Shouraki, S.B.: Evolving Game Strategies for Pool Player Robot. In: 4th WSEAS Intl. Conf. on Sim., Mod. and Opt. (2004)
3. Alian, M.E., Shouraki, S.B.: A Fuzzy Pool Player Robot with Learning Ability. WSEAS Trans. on Electronic 1, 422–425 (2004)
4. Chua, S., Wong, E., Tan, A.W., Koo, V.: Decision Algorithm for Pool using Fuzzy System. In: iCAiET 2002: Intl. Conf. AI in Eng. & Tech. (2002)
5. Chua, S.C., Wong, E.K., Koo, V.C.: Performance Evaluation of Fuzzy-Based Decision System for Pool. Applied Soft Computing 7(1), 411–424 (2005)
6. Dussault, J.-P., Landry, J.-F.: Optimization of a Billiard Player – Position Play. In: van den Herik, H.J., Hsu, S.-C., Hsu, T.-s., Donkers, H.H.L.M. (eds.) CG 2005. LNCS, vol. 4250, pp. 263–272. Springer, Heidelberg (2006)
7. Greenspan, M.: Pool at the 10th Computer Olympiad (2005), <http://www.ece.queensu.ca/hpages/faculty/greenspan/papers/8ball.pdf>
8. GSL - GNU Scientific Library (2007), <http://www.gnu.org/software/gsl/>
9. Leckie, W., Greenspan, M.: An Event-Based Pool Physics Simulator. In: van den Herik, H.J., Hsu, S.-C., Hsu, T.-s., Donkers, H.H.L.M. (eds.) CG 2005. LNCS, vol. 4250, pp. 247–262. Springer, Heidelberg (2006)
10. Lin, Z.M., Yang, J.S., Yang, C.Y.: Grey Decision-Making for a Billiard Robot. In: IEEE Intl. Conf. Systems, Man and Cybernetics. IEEE Computer Society Press, Los Alamitos (2004)
11. Nash, S.: A Truncated-Newton Optimization Package (2006), <http://iris.gmu.edu/%7Esnash/nash/software/software.html>