PLANIFICATION OPTIMALE DISCRÈTE ET CONTINUE : UN JOUEUR DE BILLARD AUTONOME OPTIMISÉ

par

Jean-François Landry

Thèse présentée au Département d'informatique
en vue de l'obtention du grade de philosophiæ doctor (Ph.D.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 19 décembre 2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-93236-0

Our file Notre référence
ISBN: 978-0-494-93236-0

Le 21 décembre 2012

le jury a accepté la thèse de Monsieur Jean-François Landry
dans sa version finale.

Membres du jury

Professeur Jean-Pierre Dussault
Codirecteur de recherche
Département d'informatique
Université de Sherbrooke

Professeur Philippe Mahey
Codirecteur de recherche
ISIMA, École Publique d'Ingénieurs en informatique
Université Blaise Pascal

Professeur Richard Egli
Président rapporteur
Département d'informatique
Université de Sherbrooke

Professeur Dominique Orban
Rapporteur
Département de mathématiques et génie industriel
École Polytechnique de Montréal

Professeur Jean-Claude Hennet
Rapporteur
Laboratoire des Sciences de l'Information et des Systèmes
CNRS

Professeur Alain Quilliot
Membre externe
ISIMA
École Publique d'Ingénieurs en Informatique

Sommaire

Le sujet de thèse de ce doctorat consiste en l'élaboration de méthodes pour la planification dans les domaines avec aspects continus, discrets et stochastiques.

Cette classe de problème, bien qu'assez générale, ne comporte pas pour l'instant de solution efficace et est souvent traitée de façon discrète plutôt que continue afin d'y appliquer les approches existantes. L'aspect stochastique apporte une difficulté supplémentaire à la recherche d'un plan optimal, et rend le problème d'autant plus intéressant. L'ensemble des approches et méthodes proposées dans cette thèse sont avant tout appliquées au jeu du billard, tout en gardant dans l'esprit qu'une généralisation permettrait son application à d'autres problèmes similaires.

En un premier lieu, une classification de ce type de problème par rapport aux recherches existantes sera effectuée, suivie d'une courte revue des approches actuelles possiblement applicables pour la recherche d'une solution acceptable. Un modèle général développé dans le contexte du jeu du billard sera présenté, ainsi que quelques indices sur la façon de le résoudre à l'aide de la programmation dynamique.

Deuxièmement, un modèle pour une approche à deux-couches sera proposé, utilisant un contrôleur robuste profitant de la finesse qui peut être exploitée des techniques d'optimisation non-linéaire.

Finalement, le modèle à deux-couches sera raffiné et quelques heuristiques de planifications seront proposée, afin de guider le contrôleur de façon à déterminer un plan efficace. On terminera à l'aide d'une synthèse des résultats et une discussion sur les perspectives futures.

Mots-clés: Optimisation, Contrôle Optimal, Modélisation, Planification, Intelligence Artificielle, Billard, Robotique, Programmation Dynamique

Remerciements

Je désire tout d'abord remercier mes deux directeurs de recherche, sans qui la création de cette thèse n'aurait pas été possible. Jean-Pierre Dussault, que je connais depuis maintenant plusieurs années, qui a su me conseiller et me guider tout au long de ces études et avec qui j'ai partagé de très bons moments ainsi qu'un nombre impressionnant d'heures à discuter du billard. Philippe Mahey, qui a su apporter un regard nouveaux à mes recherches et m'a généreusement accueilli à deux reprises en France, me permettant d'acquérir de nouvelles connaissances et de découvrir la magnifique région de l'Auvergne.

Mes bons amis Jaume, Polo et Basile, sans qui mes séjours à Clermont-Ferrand n'auraient pas été les mêmes, et avec qui j'ai partagé le plaisir des études doctorales. Alessandro et Nancy, sur qui j'ai toujours su compter et avec qui j'ai d'innombrables bons souvenirs.

Je voudrais également saluer Éric, Khalid, Bilel et tous les autres membres du laboratoire d'optimisation, qui ont toujours été d'une bonne compagnie et avec qui j'ai partagé bien des heures de discussions sur tous les sujets possibles. Stéphanie qui m'a accompagné et supporté pendant la dernière année de ces travaux.

Finalement un dernier remerciement à mes parents, qui n'ont jamais cessés de croire en moi et de m'encourager au fil de ces longues années d'études.

Il est aussi important de noter que ce travail n'aurait pas été possible sans l'aide financière apportée par l'organisme du CRSNG et l'organisme du FQRNT.

Abréviations et définitions

IA : Intelligence Artificielle

Cue : Bille de choc (bille blanche)

Cue stick : Queue de billard utilisée pour effectuer un coup

MDP : Markov Decision Process

NLOPT : Librairie d'optimisation (Non-Linear OPTimisation)

BOBYQA : Librairie d'optimisation (Bounds-constrained optimization without derivatives)

DIRECT-L : Librairie d'optimisation (DIviding RECTangles algorithm for global optimization)

TNC : Librairie d'optimisation (Truncated Newton with Constraints)

L-BFGS : Librairie d'optimisation (Low-storage Broyden–Fletcher–Goldfarb–Shanno)

CRS : Librairie d'optimisation (Controlled Random Search)

MLSL Librairie d'optimisation (Multi-Level Single-Linkage)

ISRES : Librairie d'optimisation (Improved Stochastic Ranking Evolution Strategy)

DP : Dynamic Programming

Table des matières

Liste des figures

Liste des tableaux

Chapitre 1

Introduction

Le sujet de thèse de ce doctorat consiste en l'élaboration de méthodes pour la planification dans les domaines avec aspects continus, discrets et stochastiques.

Étant donnée la complexité importante présente, il existe peu de techniques à ce jour pour effectuer de la planification sous de telles circonstances, et pourtant les problèmes avec aspects continus sont très communs dans la vie de tous les jours.

Le simple fait de conduire une voiture pour se rendre au travail le matin demande une gestion de plusieurs actions faisant partie du domaine continu. L'être humain est habitué à fonctionner dans ces situations à l'aide d'une dextérité acquise au fil des années. Par exemple il ne doit pas mesurer à chaque fois avec quel angle précis tourner le volant pour se garer; il va s'ajuster de façon approximative, et corriger si nécessaire. Beaucoup de connaissances sont acquises au préalable et c'est pourquoi il n'est pas toujours facile d'insérer cette information dans un programme afin de résoudre des problèmes de grande complexité.

La dextérité de l'être humain est particulièrement bien illustrée dans l'ensemble des jeux sportifs de haut niveau. On peut penser par exemple à un sport comme le tennis, qui requiert une précision très fine lors de l'exécution d'un coup. Le joueur doit maîtriser complètement l'aspect technique du jeu, afin de se concentrer sur l'aspect tactique et déjouer son adversaire. La complexité augmente d'autant plus quand on considère les jeux d'équipes, ou une collaboration doit être présente entre les joueurs

afin de suivre un plan de match et atteindre un but commun.

Deux aspects essentiels gouvernent le développement d'agents intelligent pour résoudre ces problèmes dans la vie de tous les jours. Une première couche traite plutôt de l'aspect mécanique et robotique du problème, c'est-à-dire la création d'un robot répondant à certaines spécifications pour effectuer une tâche précise dans un contexte particulier. Une deuxième couche traite plutôt de l'aspect calculatoire qui fera en général abstraction du robot, en analysant les données d'un problème comme par exemple un état initial donné et un état final recherché, pour y proposer une solution.

L'aspect robotique représente en lui seul une très grande difficulté, et sera parfois l'élément limitant pour le problème qu'on tente de résoudre. On doit gérer par exemple avec précision les actuateurs contrôlant les actions du robot, ou bien tenter de représenter un état donné le plus précisément possible à partir de données éparses récupérées de différents capteurs. Ce domaine, bien qu'intéressant, ne fera que l'objet de quelques mentions dans ce travail. Il n'est pas possible d'en faire totalement abstraction comme la précision des actions qu'il nous est possible d'exécuter en dépend directement, mais cet aspect sera traité de façon indirecte à l'aide de simulations physiques prenant en compte les imperfections et limites de la robotique.

Cela nous laisse donc avec l'aspect calculatoire du problème. La planification n'étant pas un domaine nouveau en intelligence artificielle et optimisation, il existe déjà plusieurs classifications de types de problèmes, et plusieurs approches pour les résoudre. On retrouve cependant des éléments clés permettant de distinguer les différentes classes.

En supposant qu'on tente de contrôler un agent quelconque afin d'atteindre un certain but fixé, il faut tout d'abord déterminer quelles sont les actions qui lui sont permises. Ces actions peuvent êtres discrètes ou continues, et peuvent varier en fonction de l'état présent. On peut également associer une dimension de coût à certaines actions, comme par exemple une ressource énergétique, et une durée potentielle de l'action. On peut supposer que l'exécution parallèle de plusieurs actions est possible, et que le résultat final d'une action n'est pas toujours garanti.

En général nous possédons également de l'information sur l'état spécifique du système au moment de prendre une action. Cette information peut être parfaite ou

partielle. Une action spécifique prise dans un état peut résulter aussi en un nombre fini ou infini de nouveaux états, qui auront également une certaines valeur en fonction du but ultime que l'agent tente d'atteindre.

On peut assez rapidement comprendre pourquoi il existe une infinité d'approches pour une infinité de problèmes particuliers. Malgré cela, il n'existe à ce jour que quelques travaux proposant une solution directe à la classe de problèmes abordée dans cette thèse. Celle de la planification en présence d'actions et d'états continus, soumis à une dimension stochastique.

Le jeu du billard, existant déjà depuis plusieurs siècles, comporte en lui-même tous ces aspects et plus. C'est pourquoi il sera au centre de cette thèse, et sera la plateforme sur laquelle des approches et modèles seront développés et mis au test.

Dans un premier temps, le chapitre 1 introduira la problématique de façon générale. On fera une introduction à la classe de problèmes étudiée suivie d'une courte synthèse des méthodes existantes et de l'état de l'art.

Le chapitre 2 introduira ensuite de façon plus formelle le problème sous sa forme continue, et suggèrera un premier modèle de programmation dynamique pour en arriver à le résoudre. Quelques tests démontrant le potentiel de l'approche seront proposés, et une courte étude de l'efficacité de différentes libriaries d'optimisation non-linéaires sera effectuée.

Le chapitre 3 proposera une approche globale pour la résolution du problème, en développant un modèle à deux-couches divisant contrôleur et planificateur afin de profiter au maximum de la connaissance de l'état du problème qu'on tente de résoudre. Un modèle pour un contrôleur robuste sera ensuite proposé, s'inspirant des méthodes d'optimisation robuste en programmation non-linéaire. On proposera aussi une formulation spécifique pour l'exécution de coups multi-objectifs. Un complément en francais sera également présenté à la fin du chapitre 2 traitant de divers aspects n'ayant pas paru dans la revue comme l'optimisation de la casse et l'analyse de la fonction objectif qui est optimisée.

Le chapitre 4 viendra proposer différentes heuristiques de planification, de façon à guider le contrôleur robuste de façon efficace et en limitant les calculs inutiles. On y présentera également quelques amélioration à la fonction d'évaluation d'un état de table, et on améliorera encore une fois le contrôleur robuste à l'aide d'une étude plus

poussée de la fonction à optimiser. Les dérivées de la fonction d'exécution d'un coup seront utilisées pour mieux guider la recherche d'une solution robuste. Un supplément en français est aussi présenté à la fin de ce chapitre avec quelques éléments n'ayant pas été soumis à la revue.

Finalement une courte synthèse du travail sera effectuée en conclusion avec quelques perspectives de travaux futurs.

1.1 Problématique

Nous allons présenter dans cette section les éléments essentiels de la problématique dont nous allons faire l'étude, c'est-à-dire les problèmes avec actions et états continus soumis à des perturbations stochastiques. Plus précisément, comme il réunit tous ces éléments et plus encore, le jeu du billard sera au centre des méthodes proposées et servira de plateforme pour tester et développer celles-ci. C'est pourquoi nous allons commencer par une description détaillée des éléments de ce jeu, suivi d'une brève introduction aux différentes variantes, afin de permettre au lecteur de se familiariser avec les éléments essentiels qui nous intéressent dans cette étude.

Le jeu du billard est une application très intéressante représentant un problème hautement non-linéaire, où la moindre variation d'un paramètre d'angle sur un coup peut provoquer des résultats très variés. Obtenir une solution optimale dans ce type de problème est très difficile car les solutions sont très dispersées dans l'espace des possibilités. La complexité de cette problématique est commune à plusieurs autres problèmes en planification ; on y retrouve la combinaison de deux aspects, l'un continu et l'autre discret. Dans le billard, cela est traduit par l'aspect technique, qui représente la difficulté à spécifier les paramètres pour effectuer un coup voulu, et l'aspect tactique, qui représente l'élaboration d'un bon plan de match.

Afin de traiter ce problème de façon optimale, il devient nécessaire de développer une méthode plus générale et mixte, travaillant sur ces deux aspects en même temps. C'est donc à l'aide de différentes méthodes comme la programmation dynamique, le parallélisme et les techniques d'optimisation robuste que je propose dans cette thèse de développer un algorithme plus général adapté à ce genre de problème de planification.

Nous allons tout d'abord commencer par introduire les différentes variantes de jeu de billard, afin de motiver l'intérêt de ce travail.

1.1.1 Variantes du billard

L'origine précise du billard n'est pas très claire. L'hypothèse la plus populaire propose que le jeu se serait tout d'abord développé à partir du Croquet, très populaire en Angleterre. Le jeu aurait évolué en utilisant la queue du bâton lorsque pris dans une impasse ou pour effectuer un coup délicat. Plus tard on aurait commencé à jouer sur des tables remplaçant l'utilisation d'un bâton par une queue longue et fine, avec la pointe recouverte de cuir, pour permettre un meilleur contrôle lors de l'impact avec la bille. Ainsi l'arrivée du billard moderne.

De façon générale, une partie de billard est jouée sur une table rectangulaire de dimensions 1:2, recouverte d'un feutre, permettant une certaine friction entre le tapis et les billes. Cette table possède 6 poches, situées aux quatre coins ainsi qu'au milieu des bandes centrales (sur le plus long côté). Une queue, habituellement d'une longueur approximative de 150cm, est utilisée pour exécuter le coup voulu. Un coup est exécuté en frappant la blanche, aussi appelée bille de choc, afin de percuter une seconde bille avec précision pour réussir à l'expédier dans la poche désirée. La dimension des billes, de la table, de la hauteur des bandes ainsi que l'ouverture des poches varie en fonction de la forme du jeu.

En général, l'ensemble des jeux de billard possèdent des règles assez communes. Il faut empocher un certain nombre de billes, possiblement dans un ordre donné et en spécifiant les combinaisons billes-poches, sans faire de fautes sinon le tour sera cédé à l'adversaire. Il est possible d'effectuer des coups défensifs afin de mettre l'adversaire en mauvaise position, mais si une partie s'éternise sur plusieurs coups et qu'il est jugé qu'aucun progrès suffisant n'est fait dans la partie, l'arbitre peut choisir de terminer la partie et d'en recommencer une nouvelle. Le début d'une partie est habituellement lancé à l'aide de la casse, où les billes sont ordonnées dans un triangle au bout de la table, et le joueur les dispersera en envoyant la bille blanche dans cette direction avec une grande vélocité.

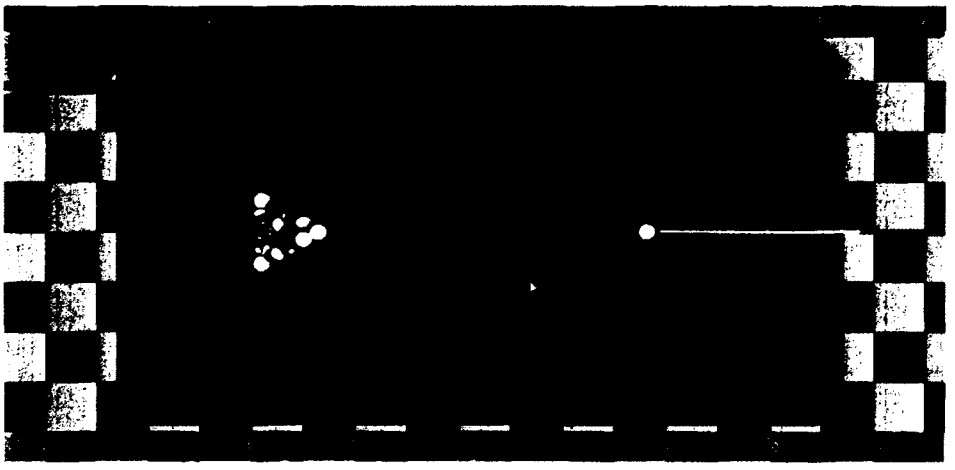

Figure 1.1: Table initiale pour le jeu de la 8.

1.1.1.1 Le jeu de la 8 (8-ball)

Règles: On retrouve 15 billes numérotées sur la table, les basses (1-7) les hautes (9-15) et la 8. Un joueur déterminé au hasard va effectuer la casse. S'il réussit à empocher une bille, il aura ensuite le choix de continuer en visant pour les hautes ou les basses. Il doit spécifier quelle bille il désire empocher dans quelle poche. Il continuera de jouer ainsi jusqu'à ce qu'il manque un coup ou qu'il effectue un coup défensif, instant auquel l'adversaire prendra la main. Le vainqueur sera le joueur ayant empoché toute ses billes, ainsi que la 8 (qui ne peut être empochée qu'à la fin). Si la 8 est empochée par accident, le joueur ayant commis la faute perdra la partie.

Aspects importants: Aucune importance dans l'ordre des billes à empocher sauf pour la 8. Possibilité de faire des coups défensifs. Il n'est pas requis de spécifier quelle bille sera empochée sur la casse.

1.1.1.2 Le jeu de la 9 (9-ball)

Règles: On retrouve 9 billes numérotées sur la table. Un joueur déterminé au hasard va effectuer la première casse et le gagnant effectuera la casse pour les prochaines parties. Le joueur ayant empoché une bille sur la casse peut continuer à jouer. Il doit absolument toucher la bille ayant la plus petite dénomination sur la table en premier mais ne doit pas nécessairement l'empocher, il peut empocher une

autre bille s'il le préfère. Le premier joueur ayant empoché la bille 9 gagnera la partie.

Aspects importants: Contraintes sur l'ordre des billes à empocher, impliquant une simplification de la planification des coups à effectuer, mais donnant une plus grande importance aux coups défensifs.

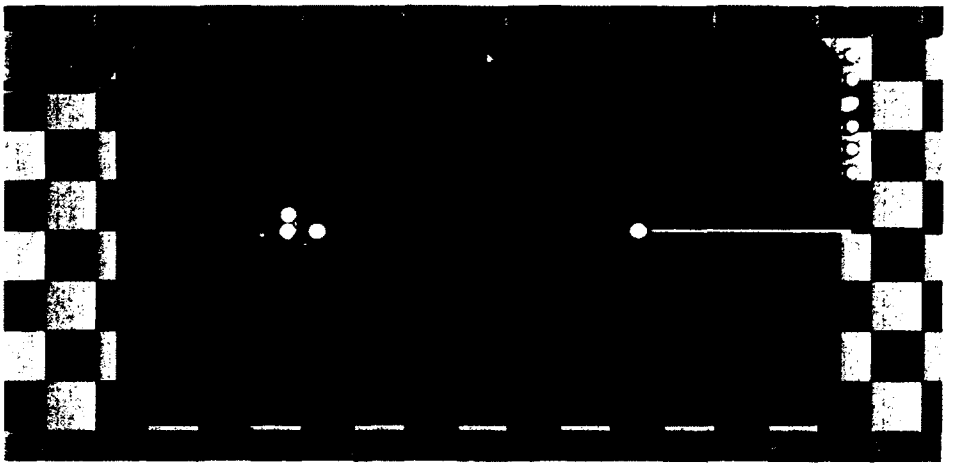

Figure 1.2: Table initiale pour le jeu de la 9.

1.1.1.3 Le Continu ("Straight")

Règles: Aussi appelé le 14.1 continu, ou simplement 14.1. On retrouve 15 billes numérotées sur la table dans ce jeu. Chaque coup réussi avec succès, défini par la réussite d'un joueur à empocher une bille dans une poche spécifiée au préalable, vaut un point. Le gagnant est le premier joueur arrivant à accumuler 100 points. Plusieurs parties doivent donc être jouées afin de déterminer un vainqueur. Il n'est pas nécessaire d'observer un ordre spécifique pour empocher les billes. Lorsque le joueur arrive à son dernier coup, il doit effectuer celui-ci tout en effectuant la casse et disperser les 14 autres billes pour son prochain coup, il est donc très important de bien se repositionner lors de son avant-dernier coup.

Aspects importants: Contrairement aux variantes décrites précédemment, la casse occupe une place beaucoup plus importante dans ce jeu, car même lorsqu'il effectue celle-ci le joueur doit spécifier la poche dans laquelle il empochera une certaine bille choisie, et il risque de perdre deux points s'il n'effectue pas une casse légale. Le

premier coup est donc habituellement un coup défensif. On retrouve souvent de très longues séquences de coups défensifs dans ce jeu.

1.1.1.4 Le Snooker

Règles: On y retrouve 15 billes rouges valant 1 point, 6 billes de couleurs différentes (jaune, verte, marron, bleue, rose et noire) valant de 2 à 7 points. Le joueur doit empocher une bille rouge suivie d'une autre couleur. Les billes de couleur sont replacées sur le jeu tant qu'il reste au moins une bille rouge sur le tapis. Les billes de couleur sont ensuite empochées en ordre croissant suivant leurs valeurs respectives, sans être replacées sur le tapis.

Aspects importants: Cette variante du jeu demande un très haut niveau technique, étant donné le diamètres plus petit des billes, la taille plus grande de la table ainsi que l'ouverture des poches plus petite. On y retrouve un échange important de coups défensifs lorsque la répartition des billes sur la table n'est pas très bonne.

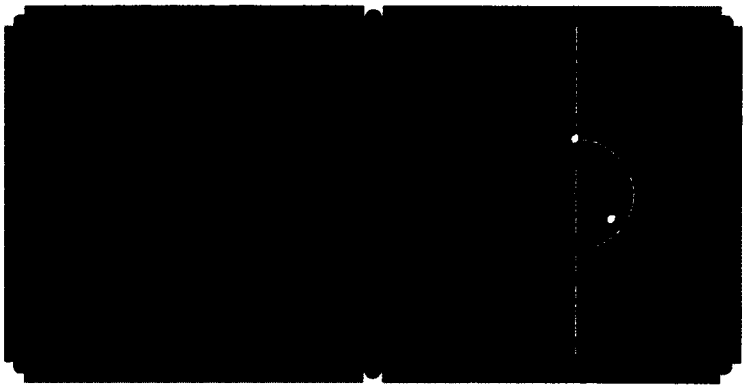

Figure 1.3: Table initiale pour le jeu du Snooker. On remarque que les billes sont plus petites et la table plus grande.

1.1.1.5 Le Carambole

Règles: Bien qu'il existe plusieurs variantes du Carambole, la plus populaire utilise 3 billes sur la table, aucune poche. Cette table possède de plus haute bandes que les autres variantes, étant donné des billes de plus grand diamètre. On utilise

un certain mécanisme pour réchauffer le tapis, afin de promouvoir l'effet de friction avec les billes. Un joueur doit frapper la blanche, frapper une des billes objet et ensuite frapper 3 bandes avant de venir frapper la 2^e bille objet, sans bien sûr entrer à nouveau en collision avec la première bille. Chaque coup réussi vaut un point, et le joueur continue jusqu'à ce qu'il perde son tour. Une partie est normalement sur 10 points ou plus.

Aspects importants: Ce type de jeu est totalement différent des autres variantes, du point de vue qu'il est très difficile de prévoir plus d'un coup à l'avance. Une très grande maîtrise de la technique du jeu est nécessaire, ainsi que de la géométrie de la table, afin d'arriver à faire un coup avec succès.

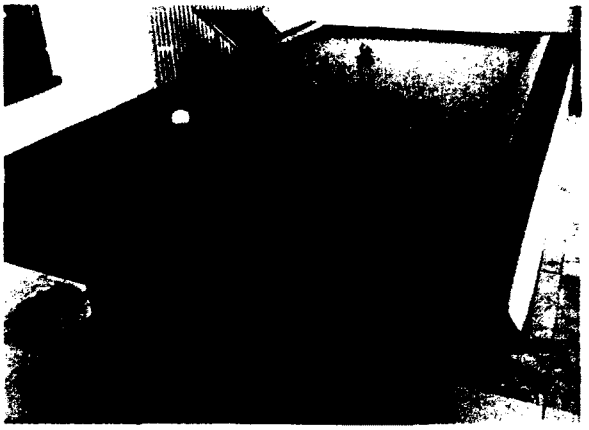

Figure 1.4: Table standard pour le jeu du Carambole. Aucune poche sur la table et des billes plus grosses en font un jeu très différent des autres variantes de billard.

1.1.1.6 Exemples

Voici en exemple un premier coup simple afin d'illustrer le choix auquel est confronté un joueur de billard lors de la planification d'un coup. On peut voir sur la figure 1.8 un coup très simple, où le joueur doit lui même décider de l'angle avec lequel il frappera la blanche. Il doit également contrôler la vitesse de la queue, et tenter de choisir un coup simple qu'il sera en mesure d'exécuter en fonction de son niveau d'expertise. L'ensemble de ces paramètres peut en fait être décrit à l'aide de cinq variables a, b, θ, ϕ, v. Les variables a et b représentant le déplacement horizontal

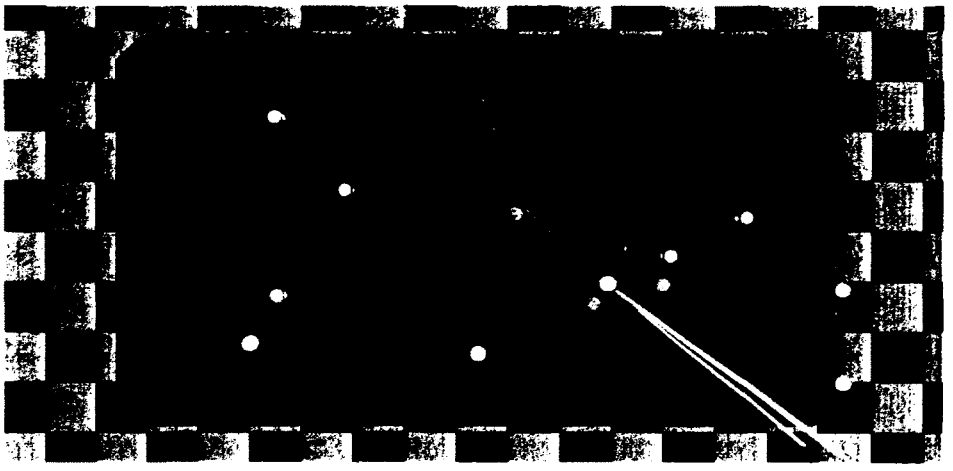

Figure 1.5: Trois coups avec une reposition différente.

et vertical par rapport au centre de la bille, θ représentant l'angle d'inclinaison la queue, ϕ pour indiquer l'orientation de la queue par rapport à l'axe x-y de la table, et finalement v pour spécifier la vitesse initiale donnée à la bille.

On peut voir dans la figure 1.6 un tracé de correspondant aux différents endroits où un joueur pourrait décider de frapper la blanche. Si on suppose que les autres paramètres sont fixés (vitesse, angle d'inclinaison de la queue, direction de la queue), on voit que déjà plusieurs possibilités s'offrent au joueur. Le résultat des différents coups correspondant à cette figure est illustré dans la figure 1.7. On peut voir que tous les coups ne seront pas effectués avec succès, et que le joueur devra ainsi choisir celui qui a la plus grande chance de réussite. Un autre exemple peut être observé dans la figure 1.5, où trois coups différents sont possibles. Il devra ainsi faire non seulement un choix de paramètres pour le coup à effectuer, mais aussi un choix sur la meilleure position à atteindre pour espérer réussir la plus grande séquence de coups possible.

1.1.2 Décomposition du problème

Pour en revenir au problème initial, l'objectif principal de ce travail n'est pas d'arriver à résoudre toute les variantes de billard, mais de définir un modèle pouvant

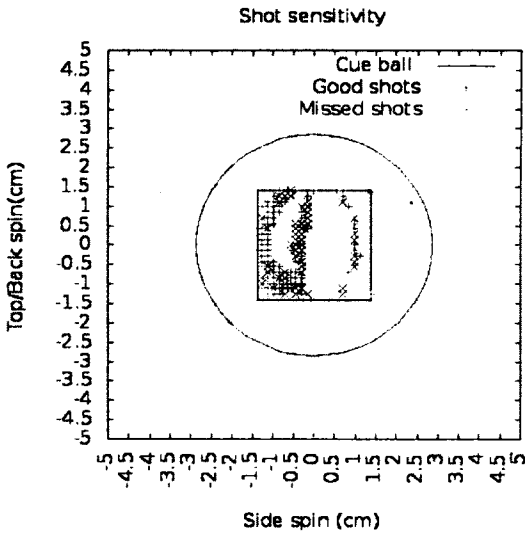

Figure 1.6: On peut observer sur ce graphique le tracé de la bille blanche, et le résultat d'un ensemble de coups, correspondant à l'état de table présenté dans l'illustration 1.7. Les croix en bleu représentent les coups où la bille visée n'est pas empochée.

représenter l'ensemble de ces problématiques, afin de pouvoir ensuite proposer une solution potentielle. La variante '8-ball' du billard sera donc au centre de cette étude comme c'est une des plus simples et connues qui combine quand même plusieurs des aspects les plus intéressants.

En se référant aux différentes variantes de billard décrites plus haut, il est possible de décomposer le problème de façon à isoler chacun des aspects les plus importants qui influent sur le résultat final d'une partie. On peut diviser en trois catégories l'ensemble des coups pour une partie: la casse, le coups offensif et le coup défensif.

1.1.2.1 Coups offensifs

Sélection du coup La sélection d'un coup consiste en l'analyse de tous les coups possibles sur la table, et de la décision par le joueur de l'exécution de ce coup pour tenter d'atteindre une certaine zone de reposition. On retrouve divers types de coups:

- direct: Un coup par lequel la blanche entrera en impact direct avec la bille visée, et cette dernière ira terminer sa trajectoire dans la poche.

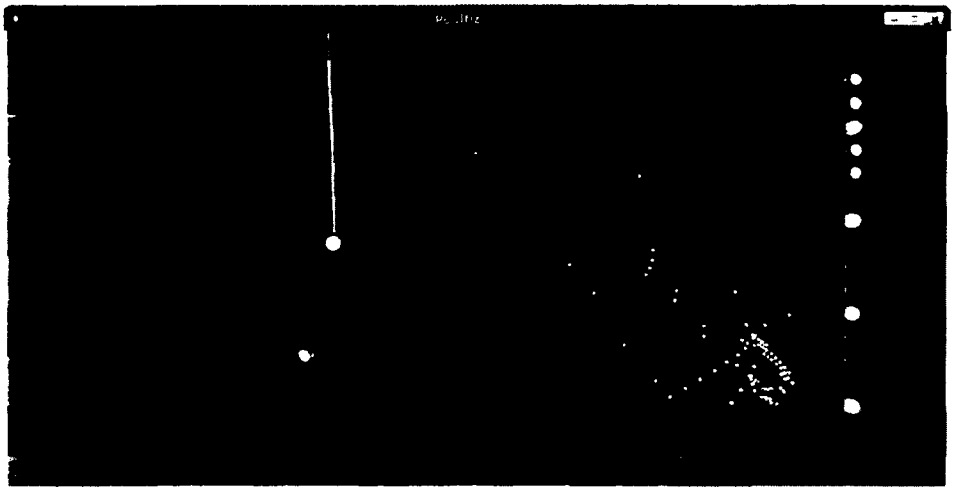

Figure 1.7: Il est possible d'observer sur cet état de table le résultat de divers coups effectués. Les points rouges représentent les endroits où la blanche s'immobilise après le coup, mais sans réussir à faire entrer la bille visée dans la poche au coin en bas à droite. Les points blancs représentent les coups réussis.

- indirect: Il existe deux types de coups indirects. Un coup par lequel la blanche va percuter la bille visée, et celle-ci va ensuite frapper une bande pour ensuite terminer sa trajectoire dans la poche. Un coup par lequel la blanche va tout d'abord percuter une bande pour ensuite frapper la bille visée, qui se dirigera ensuite dans la poche.
- combinaison: Un coup par lequel la blanche va percuter une bille, qui à son tour ira percuter une deuxième bille visée qui se dirigera dans la poche.

Zones de reposition En général lors d'une partie de billard, peu importe la variante, une certaine évaluation de la table doit être effectuée lors d'un coup. Cette évaluation consiste en une analyse des positions qu'il est possible d'atteindre pour la réalisation du prochain coup. Dans le cas le plus simple, le joueur possèdera une infinité de positions lui permettant plusieurs coups, dans le cas le plus complexe, ces zones seront assez petites et moins facilement atteignables.

Il est important de noter que ces zones de repositions sont en fait directement reliées aux coups suivants sur la table. La planification de la séquence de coups à exécuter en dépendra directement, en fonction bien sûr de la capacité du joueur à

Figure 1.8: Illustration du choix des paramètres pour un coup simple.

atteindre ces cibles. Un bon joueur doit donc posséder une bonne capacité d'analyse de la table, en fonction de son contrôle de la blanche.

Exécution d'un coup Un aspect essentiel consiste en l'exécution d'un coup voulu sur une table. Cet aspect est essentiel et le plus important du joueur de billard. Étant donné un certain état de table, un joueur doit avoir la capacité d'empocher une certaine bille dans une poche voulue, dans la mesure bien sûr où ce coup est possible. Cela revient à déterminer à quel endroit précis il va frapper la blanche, ainsi qu'avec quel angle et force. Une certaine pré-analyse est donc nécessaire de la part du joueur, afin de déterminer quels coups lui sont possibles, ainsi que la difficulté d'exécuter ceux-ci.

1.1.2.2 Coups défensifs

Un deuxième aspect présent dans toutes les variantes de billard consiste en l'exécution d'un coup défensif. En faisant un tel coup, un joueur décide de laisser volontairement le contrôle de la table à l'autre joueur. Cependant il doit quand même percuter une de ses propres billes en faisant un tel coup, afin de faire évoluer l'état de la table et d'éviter un échange infini de coups défensifs entre deux joueurs. Cet aspect ne représente pas le même niveau de difficulté et n'occupe pas la même importance dans

chacune des variantes. En effet, au Carambole par exemple, un joueur ne doit pas préciser qu'il exécutera un coup défensif ou offensif. Cependant comme il joue sur les mêmes billes que son adversaire, il devra porter particulièrement attention lorsqu'il effectuera des coups offensifs. En effet, s'il manque ceux-ci, il aura intérêt à laisser l'adversaire en mauvaise position. Pour le jeu de la 8 par exemple, les deux joueurs ont chacun un ensemble de billes sur lesquels ils jouent. Il est donc possible qu'un coup défensif soit parfois plus avantageux que d'effectuer un coup offensif très risqué. De cette façon le joueur peut espérer regagner le contrôle de la partie quand son adversaire manquera son coup et sera donc possiblement en meilleure position. Il est souvent possible d'observer de très grands échanges de coups défensifs dans les variantes comme le Snooker ou le Continu où les adversaires attendent qu'une erreur soit comise et que la table soit plus propice à l'exécution d'un coup réussi. Un joueur avec un peu moins de talent technique aura souvent avantage à jouer de façon plus défensive au billard, supposant bien sûr qu'il en soit de même pour son adversaire.

1.1.2.3 Choix de la casse

Le premier coup sur une table de billard, correspondant à la casse, occupe aussi une importance différente en fonction de la variante de billard. Les règles d'une casse légale varient en fonction du style de billard joué, mais pour le jeu du Snooker par exemple, le joueur doit positionner la blanche dans le demi-cercle sur la table (voir figure 1.3), et arriver à percuter au moins une des billes rouges dans le triangle à l'opposé de la table. Il pourra continuer à jouer s'il empoche une de ces billes. Pour le Continu, un joueur doit spécifier la bille qu'il désire empocher à chaque coup, incluant la casse. Comme il est impossible de prédire exactement la dispersion des billes, le premier coup au Continu est habituellement un coup défensif, où le joueur va tenter de disperser le moins possible les billes (en gardant le coup légal), afin de laisser le tour à son adversaire avec aucun coup possible. Au jeu de la 8, les chances d'empocher une bille sont assez grande et comme il n'est pas nécessaire au préalable de préciser la bille visée, on désire habituellement bien disperser les billes et tenter d'en empocher une sur la casse pour garder le tour et continuer à jouer.

1.2 État de l'art

Ces dernières années, la problématique du jeu de billard a su attirer de plus en plus l'attention de la communauté scientifique. D'un point de vue robotique, le défi représentant la création d'un robot ayant la précision experte d'un professionnel du billard n'est pas tâche facile. La détection précise de l'emplacement des billes sur la table, la calibration de la force donnée à un coup, ainsi que la précision de frappe de ce coup demandent tous une importante analyse. On peut voir dans [31], [45] et [21] [8] [56] que même si plusieurs chercheurs y travaillent, un robot possèdant toute les abilités d'un professionnel du billard n'existe pas encore.

Cependant, en faisant abstraction du robot et en supposant l'existence d'un simulateur représentant la réalité de façon idéale, il est possible de travailler sur l'aspect de planification des tâches de ce robot, afin de maximiser les chances de gagner une partie.

1.2.1 Tournoi "Computational Pool Tournament"

En 2005 fût lancée la première édition du "Computational Pool Tournament" à Taipei,Taiwan (voir [28]). Cette compétition, organisée par l'équipe de l'Université de Queens (Kingston), avait pour but de réunir en un seul endroit les personnes passionnées par le jeu du billard, et travaillant sur la création d'un joueur virtuel intelligent. Un simulateur physique, créé pour la compétition, donnait la possibilité pour tous les participants de réaliser un joueur utilisant la même plateforme, afin d'ensuite faire un mini-tournoi entre ces joueurs virtuels. Ce tournoi permettrait donc de démontrer l'efficacité des approches proposées, ainsi que de promouvoir l'intérêt pour ce jeu très complexe à résoudre.

Trois compétitions furent organisées, en 2005, 2006 et 2008. Lors de cette dernière, il fut déterminé que l'équipe de l'Université de Stanford organiserait la prochaine édition du tournoi, afin d'améliorer l'infrastructure utilisée pour le tournoi. Cette dernière édition du tournoi n'eut malheureusement pas lieu comme prévu en 2011, et il semblerait pour le moment qu'aucune autre organisation n'ait entrepris de continuer à s'en occuper. De façon globale, on peut classifier en trois catégories l'ensemble des méthodes utilisées lors de ces derniers tournois; les méthodes d'échantillonnage de

type Monte-Carlo, les méthodes heuristiques/hybrides, les méthode d'optimisation.

1.2.1.1 Simulateur physique

Le simulateur Poolfiz ([13]) utilisé lors des dernières compétitions permet de simuler en deux dimensions la physique d'une table de billard, en simulant l'interaction entre les billes sur la table suite à la perturbation du système (exécution d'un coup). La résolution analytique du système d'équations représentant le trajet des billes sur la table permet un calcul direct du moment des impacts afin de créer une simulation complète sans avoir recours à une méthode d'intégration pour calculer des estimations. Cet aspect est assez important car il permet entre autres de créer une simulation demandant beaucoup moins de temps de calcul.

Cependant il n'est pas facile de créer un simulateur répliquant la réalité de façon parfaite, et il est possible d'observer justement quelques-uns de ces défauts dans le simulateur de Poolfiz. Entre autres les collisions avec avec les bandes, qui sont assez complexes, sont pour l'instant très simplifiées et ne permettent pas d'effectuer certains coups. Cela change donc de façon importante la dynamique des coups en fonction des paramètres utilisés.

Une version modifiée de Poolfiz a été développée par l'équipe de l'Université de Stanford ([9]) dans le but d'augmenter l'efficacité des calculs. Fastfiz donne des résultats presque identiques mais avec un temps de calcul grandement réduit (4x plus rapide). Cependant il comporte toujours les même lacunes en ce qui concerne la fidélité de la simulation des impacts avec les bandes.

Le même format à été gardé pour le simulateur. L'impact de la queue de billard avec la bille est réduit à 5 paramètres représentant la force appliquée à la bille lors de la collision avec la queue (voir figure 3.1):

- a et b représentent l'endroit précis où la bille est frappée par rapport à son centre;
- θ l'élévation de la queue par rapport au plan horizontal de la table;
- ϕ l'orientation de la queue par rapport à la table;
- v, la vitesse initiale donnée à la bille.

Une fois ces paramètres donnés au simulateur, celui-ci retourne l'état de la table résultant.

Il existe aussi plusieurs autres simulateurs, professionnels et amateurs, créés souvent avec l'intention d'obtenir une plateforme de jeu réaliste. Cependant dans la plupart des cas il n'est pas possible de vérifier la fidélité physique de ces simulateurs, comme ces compagnies préfèrent ne pas divulger le code ou les approches utilisées. Dans les autres cas, les simulateurs utilisent souvent des approches de discrétisation pour déterminer la trajectoire des billes sur la table, ce qui demande encore plus de calculs est n'est pas nécessairement plus précis. Le simulateur Billiards ([51]), basé sur le moteur physique ODE, semble être plus fidèle à la réalité comme il dépend directement d'un simulateur physique général. Cependant il n'est pas sans désavantages. Entre autres le même coup (même paramètres) exécuté deux fois sur la même table ne donnera pas toujours le même résultat. Ceci est possiblement dû à une mauvaise utilisation du simulateur physique ODE par l'interface Techne du programme Billiards.

Un autre simulateur, [3] créé par Fray, J-M., utilise plutôt une interpolation de données acquises au préalable afin de simuler une table de billard. Des centaines de coups ont été effectués sur une table réelle, pour ensuite compiler ces données dans une base de donnée. Lorsqu'un coup est effectué sur le simulateur, une recherche est effectuée dans cette base de donnée afin d'en déduire l'état le plus proche et d'en interpoler l'état résultant. Cette approche fût élaborée après avoir échoué dans la réalisation d'un simulateur répliquant de façon parfaite une table réelle. Bien qu'empirique, le simulateur résultant n'en demeure pas moins convainquant pour le jeu du Carambole, mais nécessite pour chaque table différente une étude complète et une nouvelle base de donnée.

1.2.2 Méthodes d'échantillonnage

Parmi les approches existantes les plus intéressantes, celle proposée en 2006 par Mike Smith [58] demeure à ce jour la plus efficace en ce qui concerne le jeu de la 8. En effet, ce fût l'approche qui permit de gagner les deux premières éditions du tournoi. En discrétisant l'ensemble des actions possibles pour effectuer un coup sur une table, et en faisant un échantillonnage sur l'espace de ces actions, un arbre de recherche est ainsi établi permettant d'analyser l'efficacité d'une certaine séquence de coups (voir

figure 1.9). Une heuristique est utilisée pour évaluer les états de table et la valeur est ainsi propagée avec un certain facteur d'atténuation dans l'arbre de recherche.

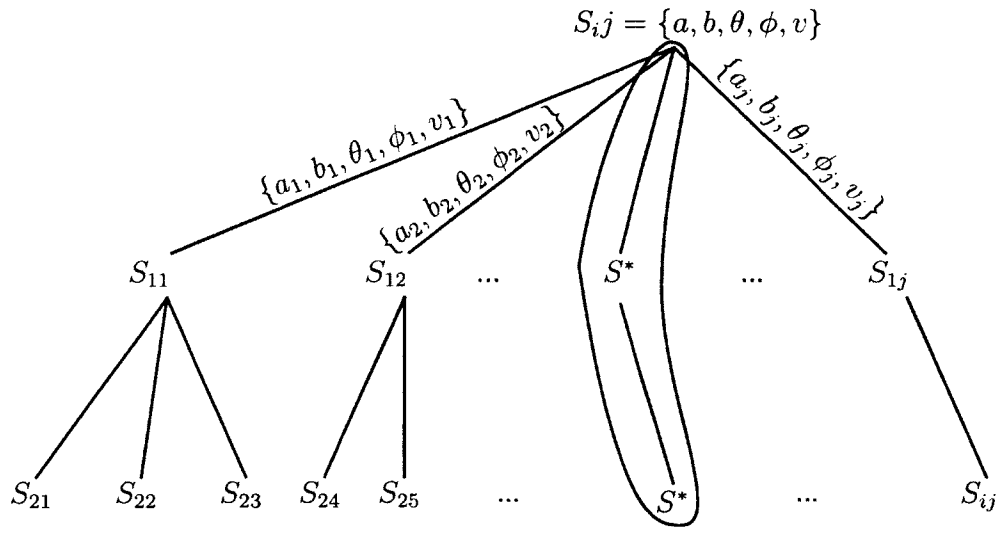

Figure 1.9: Approche par échantillonnage. L'espace de recherche est exploré en dicrétisant les paramètres a, b, θ, ϕ, v afin de déterminer les états atteignables. Le chemin optimal avec les états maximisant les chances de réussites est ensuite sélectionné (S^*)

Cependant, il est évident qu'une telle approche dépend beaucoup du temps disponible pour effectuer la recherche dans l'espace des actions. Un désavantage important est le manque d'analyse a priori de l'état de la table, ignorant ainsi une certaine connaissance qui pourrait être extraite afin de faciliter la recherche. Il est aussi facile d'imaginer que d'appliquer une telle approche dans une variante de jeu avec une plus grande complexité de planification pourrait devenir problématique, étant donné un espace de recherche encore plus grand.

Lors de la compétition en 2008, une équipe de l'université de Stanford raffina l'approche de Smith([58]), en mettant un important effort sur la maximisation du temps de calcul, en parallélisation de l'approche, ainsi qu'en créant un simulateur simplifié basé sur poolfiz, permettant d'effectuer plus de calculs dans le temps permis. Le résultat fût un joueur très performant gagnant facilement la dernière édition du tournoi.

1.2.3 Approches heuristiques

Différentes approches heuristiques ont aussi été développées pour trouver une solution au problème avec plus ou moins de succès. Un système, expert par exemple peut tenter d'identifier la situation (énumérer les coups intéressants) et interpoler les paramètres nécessaires pour effectuer un coup à l'aide de tables pré-calculées. Cependant il clair que l'efficacité d'une telle approche dépend directement de la capacité du programmeur à résoudre le problème.

1.2.4 Optimisation d'un coup

Cette approche, qui est à la base de ce travail, consiste en l'élaboration d'un modèle décrivant un coup sur une table de billard, afin de le minimiser en utilisant une méthode d'optimisation non-linéaire. On tente de trouver les paramètres optimaux pour effectuer un coup donné afin de se repositionner sur une cible précise sur la table, tout en réussissant le coup voulu.

Plus précisément, en supposant qu'un simulateur complet soit disponible ([43]) pouvant donner l'état résultant en fonction d'un coup effectué, il est possible de créer un problème de moindres carrés afin d'effectuer un coup et de se repositionner sur la table. Supposons que le vecteur x représente l'ensemble des paramètres décrivant un coup sur le simulateur. Il est possible de définir:

$$\min_x \frac{1}{2}\left(\|\mathbf{p}_c(x) - \mathbf{c}\|^2 + \rho\|\mathbf{p}_o(x) - \mathbf{p}\|^2\right)$$

où $\mathbf{p}_c$, $\mathbf{p}_o$ représentent les positions de la blanche et de la bille objet respectivement au temps final, $\mathbf{c}$ la cible de reposition pour la blanche, $\mathbf{p}$ la poche visée (voir fig 1.10) et ρ une pénalisation associé à la satisfaction de la contrainte.

À l'aide de ce modèle, il est donc possible de déterminer les paramètres nécessaires d'un coup pour empocher une bille visée et se repositionner où l'on désire.

1.2.5 Modèle général

Dans l'article [12], un modèle général du billard est proposé. Celui-ci est décrit comme faisant partie d'une classe de jeux nommée "Two-player Zero-sum Billiards

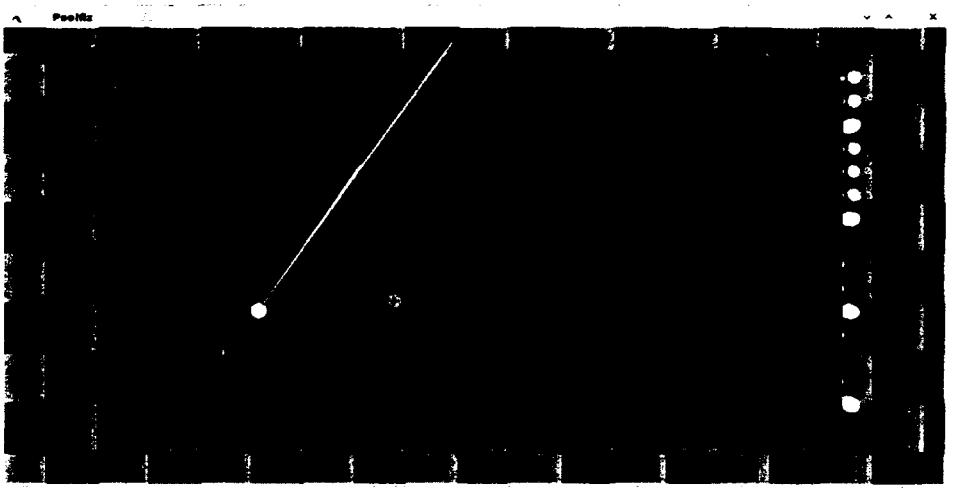

Figure 1.10: Optimisation pour l'exécution et le repositionnement d'un coup sur une cible visée

Game". Plus précisément, on décrit le jeu du billard par l'ensemble $(S, A, \lambda, p, s^0, C, r)$ où:

- $S \subset \mathbb{R}^n$ est un espace d'état compact de dimension n, représenté par un vecteur $s = (s_1, ..., s_n)$ de nombres réels.
- $A \subset \mathbb{R}^m$ est un espace d'action compact de dimension m pour les joueurs 1 et 2, où une action est représentée par un vecteur $a = (a_1, ..., a_m)$ de nombres réels. $a^t \in A$ est l'action choisie au temps t.
- $\lambda : S \mapsto \{1, 2\}$ est une fonction utilisée pour représenter quel joueur doit faire son coup pour un état donné de la partie. $\lambda(s^t)$ indique à qui est le tour de jouer dans l'état s^t.
- $p : S \times A \mapsto \Delta(S)$ est la fonction de transition, où $\Delta(S)$ est l'ensemble de toutes les distributions de probabilité pour S.
- s^0 correspond à l'état initial.
- $\lambda(s^0)$ représente le joueur qui commencera la partie.
- $C \subseteq S$ est l'ensemble des états terminants, un sous-ensemble fermé de l'espace des état.
- $r : C \mapsto \mathbb{R}$ est la fonction de récompense, où $r(s)$ spécifie le montant que le joueur 2 doit payer au joueur 1 si la partie termine dans un état où $s \in C$.

La partie commence avec l'état s^0, et le joueur $\lambda(s^0)$ spécifie une action. Le prochain état, s^1, est déterminé par $p(\cdot|s^0, a^0)$, et le jeu continue avec le joueur $\lambda(s^1)$ spécifiant le prochain coup. Le jeu continue ainsi jusqu'à ce que l'état $s^T \in C$ soit atteint, et le joueur 1 reçoit alors $r(s^T)$ du joueur 2.

On propose par la suite de définir $v(s)$ comme étant la valeur du jeu du billard, pour en arriver à l'équation minmax:

$$v'(s) = \begin{cases} \max_a \left[\int_S v(\cdot)dp(\cdot|s,a)\right] & si\ \lambda(s) = 1 \\ \min_a \left[\int_S v(\cdot)dp(\cdot|s,a)\right] & si\ \lambda(s) = 2. \end{cases}$$

Si on définit, pour $n \geq 1$, l'historique H^n, comme étant le produit de $S \times (A \times S) \times ... \times (A \times S)[S \times n(S \times A)'s]$, on peut définir l'élément $h^n = (s^0, a^0, s^1, ..., a^{n-1}, s^n)$ comme un historique. Il est donc possible de définir une stratégie σ comme étant $\sigma^n : H^n \mapsto A$. De cette façon, la stratégie $\sigma_1(s) = \text{argmax}_a \left[\int_S v^*(\cdot)dp(\cdot|s,a)\right]$ sera optimale et mènera donc vers la victoire avec le plus grand pourcentage de réussite.

Ce modèle, bien que décrivant fidèlement la problématique générale du billard, présente des inconvénients assez évidents rendant son utilisation impossible dans une application. En effet, on ne peut sans simplifications très importantes, explorer l'espace complet des états et actions. Même en discrétisant de façon importante le domaine, cela revient à explorer l'ensemble des coups possibles non seulement pour un joueur, mais aussi pour son adversaire. Un aspect important de ce travail sera donc de procéder a une simplification de ce modèle, de façon à trouver une solution optimale

Chapitre 2

Le billard d'un point de vue optimisation

Résumé

Depuis quelques années, il est possible d'observer un intérêt grandissant dans le domaine de la recherche pour les aspects robotique et calculatoire du jeu du billard. Le défi se montre très important et très intéressant. Nous allons nous concentrer dans ce travail sur l'aspect computationnel du problème, sans cependant oublier l'importance et la difficulté relevant de l'aspect robotique.

D'un point de vue calcul, on peut globalement diviser la tâche en trois aspects principaux : la simulation physique, l'exécution de coups, et la planification d'une séquence de coups. Ces aspects seront développés plus en détail à l'aide du domaine de l'optimisation. Le côté planification de la problématique relevant également du domaine de l'intelligence artificielle, et le lien entre les deux approches sera révélé progressivement dans le travail.

Nous allons présenter l'état de l'art dans ce tout nouveau domaine, ainsi que les obstacles demeurant présents avant qu'il soit possible de mettre au défi un professionnel du billard à l'aide d'un système robotique.

Commentaires

Cet article à été soumis à la revue "Wiley Encyclopedia of Operations Research and Management Science" le 12 juillet 2010, accepté sous révision d'un comité de lecture, et publié dans l'édition 2010 de la revue. Les trois auteurs sont listés en ordre alphabétique et non en ordre d'importance et se sont partagés la rédaction. La part de l'étudiant est très importante, ayant mis en place la plateforme de test utilisée pour appuyer les différents aspects discutés, et ayant travaillé avec les co-auteurs sur le développement du modèle proposé.

L'apport principal de cet article par rapport à la thèse concerne surtout le développement de l'approche pour la discrétisation du problème en une combinaison de billes poches cibles, et les tests sur les différentes librairies d'optimisation. En effet, on propose dans cet article un résumé général de l'état de l'art et on tente de situer la problématique par rapport aux recherches existantes. Pour cela, une courte description des différentes variantes du jeu du billard est fournie, afin d'en dégager les éléments essentiels. Ceci permet donc par la suite d'introduire la formulation générale d'un modèle de programmation dynamique, et d'y présenter quelques résultats préliminaires.

Ce travail sert également de preuve de concept pour motiver les chapitres suivants, qui iront plus en détail dans le développement d'un contrôleur robuste pour le jeu du billard, ainsi que d'un planificateur de haut niveau. L'ensemble des tests sur les librairies d'optimisation permettent de faire une courte analyse quant à l'efficacité d'utiliser une approche d'optimisation locale plutôt que globale.

L'article présenté est entièrement rédigé en anglais mais comprend aussi une section complémentaire en français à la fin du chapitre comportant des résultats n'ayant pas paru dans la revue. Il est important de noter une correction concernant la section 2.4.2.2 de l'article. Il semblerait que la raison pour laquelle il n'est pas possible de simuler deux fois le même coup avec le simulateur Billiards n'est pas directement relié à ODE mais possiblement dû à une mauvaise implémentation de la librairie Techne utilisée comme interface à ODE.

Computational pool: an OR—optimization point of view

Jean-Pierre Dussault, Jean-François Landry
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
Jean-Pierre.Dussault@usherbrooke.ca,
Jean-Francois.Landry2@usherbrooke.ca

Philippe Mahey
ISIMA - Campus des Cézeaux, BP 10125, 63173 Aubière CEDEX, France.
Philippe.Mahey@Isima.fr

Keywords: Pool, Billiards, Snooker, Game, Simulation, Planning, Strategy, Optimization, Artificial Intelligence

Abstract

Computational and robotic pool has received increasing interest in the past few years. The challenges are important. In this survey, we focus on the computational aspect, but nevertheless acknowledge that the robotic aspect is both very important and very difficult.

Computationally, one may roughly split the tasks into three main aspects: simulation of the physics, execution of the shots and planning of sequences of shots. We will discuss those aspects from an OR–optimization point of view. The planning aspect retains also the interest of the artificial intelligence community, and links between both approaches will be revealed.

We will present the state of the art in this young research area, as well as the obstacles that remain before a computer–robotic system actually challenges a human champion.

2.1 Introduction

The interest in computational pool is motivated by a variety of factors. The mechanics of pool has long served as a focus of interest for physicists, an interest which has extended naturally to the realm of computer simulation. Realistic and efficient simulators have led to the proliferation of computer pool games. These games include significant computer graphics components, often including human avatar competitors with unique personalities, as well as an element of artificial intelligence to simulate strategic play. There are numerous one-person games available, of various physical realism.

Though in the context of this chapter the primary objective is the creation of a computer pool player, we wish to extend our research beyond the scope of billiards. We hope that by researching the best way of making a perfect player we can develop new AI approaches, and possibly contribute to other problems of that nature.

There have been three instances of pool Olympiads where rival computational pool player competed. The competitions attracted participants from the artificial intelligence community, a pool player representing a challenge for the agent approach. Yet, only 8-ball tournaments were held, and the champions used a Monte Carlo search tree strategy [29, 30, 59, 10]. In this chapter, we develop a complementary approach based on OR techniques, simulation, dynamic programming, numerical optimization.

Although the problem we wish to solve is deterministic in nature, subject to the laws of physics, it is so easily influenced by many small factors that it can actually be seen as stochastic. This makes it very hard to create a perfect player, because even if he never misses, the outcome of the game is never pre-defined.

In this chapter we will explore the technical challenges of computational pool from an OR point of view. Section 2.2 describes the elements of the cue sports from a human perspective, including some discussion of the aspects of advanced human play. Section 2.3 introduces the several OR aspects to be detailed later on.

Section 2.4 presents the requirements to produce a simulation model. Section 2.5 addresses the optimization of a billiards player. This is work in progress.

Future trends are discussed in section 3.7.

2.2 The Cue Sports

There are many varieties of cue sport, and many different terminologies and rules that vary with geographical location. For example, the most popular North American pool game is 8-ball, in which there are seven "low" (solid) balls (1–7), seven "high" (striped) balls (9–15) and the 8-ball. The objective is to be the first to sink all balls in one group, and then the 8-ball. Alternately, snooker is likely the most popular cue sport in England, which is played on a larger table and consists of 15 red balls, and 6 of various colors. The balls are smaller than their North American counterparts, and the objective is to accumulate points by alternately sinking a red and then a colored ball, each having a certain value. In France, a game called carom is played, where only 3 balls are used on a pocket-less table, and the player must attempt to contact both object balls with the cue ball. In the three cushions ("trois bandes" in french) version of carom, the cue ball is required to rebound off of three rails before kissing the object ball.

2.2.1 Pool terminology

The basic elements common to all of these games are as follows. For a much more complete glossary of cue sport terms, see the Wikipedia page [2] from which we extract the terms relevant to our computational pool study.

- Cue: Also cue stick. A stick, usually around 55-60" in length with a tip made of a material such as leather on the end and sometimes with a joint in the middle, which is used to propel billiard balls.
- Cue Ball: Also cueball. The ball in nearly any cue sport, typically white in color, that a player strikes with a cue stick. Sometimes referred to as the "white ball", "whitey" or "the rock".
- Object Ball: Depending on context:
  1. Any ball that may be legally struck by the cue ball (i.e., any ball-on);
  2. Any ball other than the cue ball.
- Table: The flat playing surface. The table size varies, depending on the game played, but is always rectangular, being twice as long as is is wide. In most

games, the table has a pocket in each of the four corners, and one at the centers of each length.

1. Bed: The table is covered in a textured felt (or baize) material, usually of green color, to add a frictional damping effect to the shots.
2. Rail: A rubberized edge running along the inner boundary of the table, to accommodate rebounds following the collision of a ball (cushions in British English).
3. Table State: the position of all balls at rest on the table, ready for the next shot.

- Shot: The use of the cue to perform or attempt to perform a particular motion of balls on the table, such as to pocket (pot) an object ball, to achieve a successful carom (cannon), or to play a safety. Different classes of shot type include:
  - Direct Shot: A shot where the cue ball hits an object ball, which then reaches a pocket.
  - Kick Shot: A shot in which the cue ball is driven to one or more rails before reaching its intended target–usually an object ball. Often shortened to 'kick'.
  - Bank Shot: Shot in which an object ball is driven to one or more rails prior to being pocketed (or in some contexts, prior to reaching its intended target; not necessarily a pocket). Sometimes "bank" is conflated to refer to kick shots as well, and in the UK it is often called a double.
  - Combination Shot: Also combination, combo. Any shot in which the cue ball contacts an object ball, which in turn hits one or more additional object balls (which in turn may hit yet further object balls) to send the last-hit object ball to an intended place, usually a pocket. In the UK this is often referred to as a plant.
  - Safety shot:
    1. An intentional defensive shot, the most common goal of which is to leave the opponent either no plausible shot at all, or at least a difficult one.
    2. A shot that is called aloud as part of a game's rules; once invoked, a safety usually allows the player to pocket his or her own object ball without having to shoot again, for strategic purposes. In games such as

seven-ball, in which any shot that does not result in a pocketed ball is a foul under some rules, a called safety allows the player to miss without a foul resulting. A well-played safety may result in a snooker (no direct shot available).

- Spin: Rotational motion applied to a ball, especially to the cue ball by the tip of the cue, although if the cue ball is itself rotating it will impart (opposite) spin (in a lesser amount) to a contacted object ball. Types of spin include follow or top spin, bottom or back spin (also known as draw or screw), and left and right side spin, all with widely differing and vital effects. Collectively they are often referred to in American English as "English".

2.2.2 Specificity of game variants

To excel in any pool variant, one has to thoroughly understand the physics, accurately control the cue ball, and plan several shots in advance. However, each variant has its specificity, and the relative weight of those game aspects slightly differs from one variant to another. We examine here four among the most popular games and their specifics.

2.2.2.1 Games using numbered balls

On the American continent, most pool tables use numbered balls (1–15) in a variety of games. The "official" tables measure $4.5' \times 9'$, but smaller models ($4' \times 8'$ or even $3.5' \times 7'$) are quite common. Hereafter, we briefly discuss three popular game variants using such equipment.

2.2.2.1.1 Eight ball This is by far the most popular game played on the American continent. However, some of its popularity stems from its use in bars, on coin operated tables, and the rules of the game reflect this situation.

It is the variant where chance plays a major role, and this for three main reasons:

- no call is made on the break shot, so any ball pocketed by luck on the break is valid;
- on a foul play, pocketed balls remain in the pockets;

– both players do not play on the same set of balls.
Although not the primary variant in high level human competitions, this has been the variant in the first three computational pool tournaments.

Planning is limited to eight shots before the win. On the other hand, the presence of opponent's balls may complicate the plan.

2.2.2.1.2 Nine ball This variant is played with nine balls, and the specifics include the obligation to hit the lowest numbered ball on the table. Thus, planning is somewhat simpler here because of this restriction on which ball is first contacted by the cueball.

2.2.2.1.3 Straight pool This is played on similar tables as eight balls pool, but here, any ball has to be called including on the break shot, and balls sunk on a foul are pulled out the pocket and re-spotted on the table. Champions achieve runs of several hundreds consecutive successful shots. It is clear that this variant requires both clever planning, and extreme accuracy. High risk shots involving several rebounds on the rails or with other balls are rare, the players often preferring to resort to defensive shots than to risk loosing its turn leaving the opponent with an easy table.

2.2.2.2 Snooker

Snooker is the most popular variant in UK. It is played on a large ($6' \times 12'$) table using 15 red balls and six colored balls in addition to the cue ball. The large dimension of the table and the use of slightly smaller balls call for high accuracy. Here again difficult bank, kick or combination shots are often avoided by the frequent use of defensive shots. The name snooker itself refers to the situation where a player has no legal direct shot available.

2.2.2.3 Carambol

Carom is played on a pocket-less table using only three balls, slightly larger than numbered balls though. The game consists in having the cue ball hit the other two balls. In high level competitions, the three cushions variant is often prefered, in

which the cue ball has to rebound on three cushions before hitting the second of the other two balls. Obviously, this three cushions variant is extremely demanding in geometrical skills, and the sole difficulty of a single shot almost precludes any form of planning.

In opposition to most pocket billiards variants, the physics model of the collision with the rail is of primary importance here, as is the accurate control of the cue ball. On the other hand, the strategic planning aspect is significantly less important.

2.3 Overview of computational pool

In this section, we present the three basic building blocks used in a computational billiards player. As we discussed in section 2.2.2, different variants of the game require different balance and specificity in the building blocks.

2.3.1 Physics simulation

The simulator is required for two aspects in computational pool.

- The first one is to act as a common table and referee in order for the (computer) players to compete. In an ideal world, the simulated game would be indistinguishable from its counterpart as played on a real table. This involves meticulous modeling of the impacts between the cue, the balls, the rails and (whenever balls jump) the table. Moreover, the rolling motion of the balls on the cloth must be addressed faithfully.
- The second aspect is a tool on which the AI players rely in order to predict the outcome of a given shot. Here, speed–accuracy trade-offs are expected.

In the first three editions of the computer pool tournaments, the same simulator was used for both uses, some random noise being added on the shots played on the "common table". As the situation evolves, it is likely that referee simulators will get more and more complex and faithful, yielding high computational costs perhaps no more compatible with the requirements of computational players under limited resources. Ultimately, the games will be played on a real table by robots, and the simulators used by the players will be integral part of the player's strategy.

2.3.2 Control of the cue ball

When observing a champion at some variant of pocket billiards, two aspects are striking: he has mostly easy looking shots to play, but whenever he faces a difficult shot, he nevertheless sinks the ball. The first aspect is less spectacular, but is a consequence of his accurate repositioning after the impact on the object ball.

2.3.3 Strategic planning

Another aspect is that the champion takes enlightened decisions with respect to the order in which he will sink the balls. In computational pool, this aspect is addressed using some dynamic model of the game aspect.

Among the most evident strategic decisions is the choice to do a so called defensive shot. Instead of aiming to sink some object ball, the player strives to leave the cue ball in such a position that his opponent is left with a very difficult situation.

2.4 Modeling billiards physics

In this section we first summarize the mathematical models that have been developed over the years, going back to Coriolis in 1835. Next we discuss two computer implementations.

2.4.1 Mathematical models

All variants of billiards and pool games involve a cue, hard spheres, a bed which is a flat surface covered with a cloth, and rail cushions. Spheres slide and roll on the bed, impact on each other as well as on the rails, and the cue gives the initial impact to the cue ball. Thus, we need to model

- cue–cueball impacts;
- moving balls; mostly, sliding and rolling on the bed, but sometimes "jumping" off the bed for some tiny amount of time, and even sometimes jumping off the table;

- collisions between moving hard spheres, mostly elastic collisions between rigid bodies;
- collisions between hard spheres and cushions on the rails, partly inelastic collisions between a rigid body and a not so rigid one (the rail).

All of this is quite delicate to accurately simulate, but since the moving bodies are identical spheres, and the bed is planar, the physics equations are rather simple.

The interest in the mathematics of billiards goes back to at least Coriolis [23], and has attracted recent interest as well. In particular, David G. Alciatore, in addition to his book [5], maintains a web site [6] where he posts more mathematically oriented developments including fine tuning of physical models required to accurately predict experiments he conducts using high speed video cameras.

2.4.1.1 Rolling motion—2D simplifications

An important observation yields computationally tractable prediction for balls motion on the table. The trajectory of the balls is described by a piecewise quadratic parametric equation. This allows to simplify the collision detection to simultaneous quadratic equations, for which closed form formulæ are available. [41, 42, 54]. Moreover, extension to 3D simulation still may be described by piecewise parametric quadratic equations. Therefore, in principles, it is possible to implement a discrete event simulator without using numerical integration. Moreover, since the underlying parametric equations are simple piecewise quadratic, an improved implementation will provide sensitivities and derivatives, most useful for the optimization aspects to be discussed in section 2.5.3.1.

2.4.1.2 Ball–ball collisions

The balls are almost rigid bodies, and their collision is almost frictionless. The behavior of the object ball and cue ball after the collision are described by the so-called 30° and 90° rules. Actually, slight non-elasticity, small friction, speed, and spin all slightly modify the rules. In [6], one can find numerous technical notes providing fine points of billiards physics.

2.4.1.3 Ball–rail collisions

Those collisions are much more complicated. In most pocket billiards variants, though, kick or bank shots are not that frequent, so a simple approach is justified. In three cushions, as the name of the game suggests, those collisions are of prime importance. Larger balls used in carom variants and high energy ball–rail collisions imply the cushion undergo important deformations, no more faithfully computed using simple geometric models alone. For instance, in [3, 27], a one person carom game, shots were performed and captured with a camera, and in the actual game, interpolation is used to adjust the simulator to replicate the same result. This usage of fudge variables was necessary as the direct implementation of the physics of the collisions as described in [23] yielded different and inaccurate results when compared to real-life.

2.4.2 Existing physically documented simulators

While there are many one–person pool playing games, there are few efforts disclosing the actual equations or physical models underlying their simulator; we describe hereafter two softwares available under open source licenses.

2.4.2.1 PoolFiz – FastFiz

The first three instances of the pool Olympiads used the PoolFiz simulator [41, 42, 43], lately replaced by the re-implementation FastFiz [1]. This simulator implements the simplified 2D rolling equations. The actual movement of balls has a spatial component (how the balls move on the table) and a rotational component (how the balls spin). The spin may be represented as rotational velocities around the 3 main axes, $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$,$\mathbf{e}_3$ pointing upward perpendicular to the table's surface. The present implementation neglects any spin in the $\mathbf{e}_3$ axis. The simulator was made available through a common interface to the competitors, which used it to compute their shots. When used as the referee table, some random noise was added to the shot parameters.

In the simulator, the following parameters describe a shot:

- a and b representing the side-spin and top-spin with respect to the cue ball center;
- θ the elevation of the queue stick;
- ϕ the orientation of the queue stick on the table;
- v, the initial speed given to the cue ball.

2.4.2.2 Billiards

The billiards–techne project [51] adopts a different approach, namely it uses the ODE open source dynamics engine.

From [60], ODE is an open source, high performance library for simulating rigid body dynamics. It is fully featured, stable, mature and platform independent with an easy to use C/C++ API. It has advanced joint types and integrated collision detection with friction. ODE is useful for simulating vehicles, objects in virtual reality environments and virtual creatures. It is currently used in many computer games, 3D authoring tools and simulation tools.

As such, physical objects (cue, balls, table bed, table rails) are described using the ODE interface, and simulated as accurately as ODE allows. No advantage is taken from the simplified nature of perfect spheres rolling on a planar surface. While still in progress, the physics has 3D realism, and care has been taken to validate the results with high speed cameras observations to accurately model the cueball deflection when using English, an aspect absent from PoolFiz. The approach using ODE has some drawback, though: the simulator cannot produce noiseless results since it relies on a fairly sophisticated simulator. There are ways to reduce the random effects, but it is impossible, it seems, to completely eliminate randomness.

2.4.2.3 Simulators comparison

We illustrate in Figure 2.1 the differences in repositioning possibilities of the two simulators. Probably neither is realistic enough to predict reposition on a real table and further validation is certainly required. Important differences come from the fact that PoolFiz does not take side spin into account on ball–rail collisions, which explains the large area unattainable. Billiards, as compared to PoolFiz, results in less

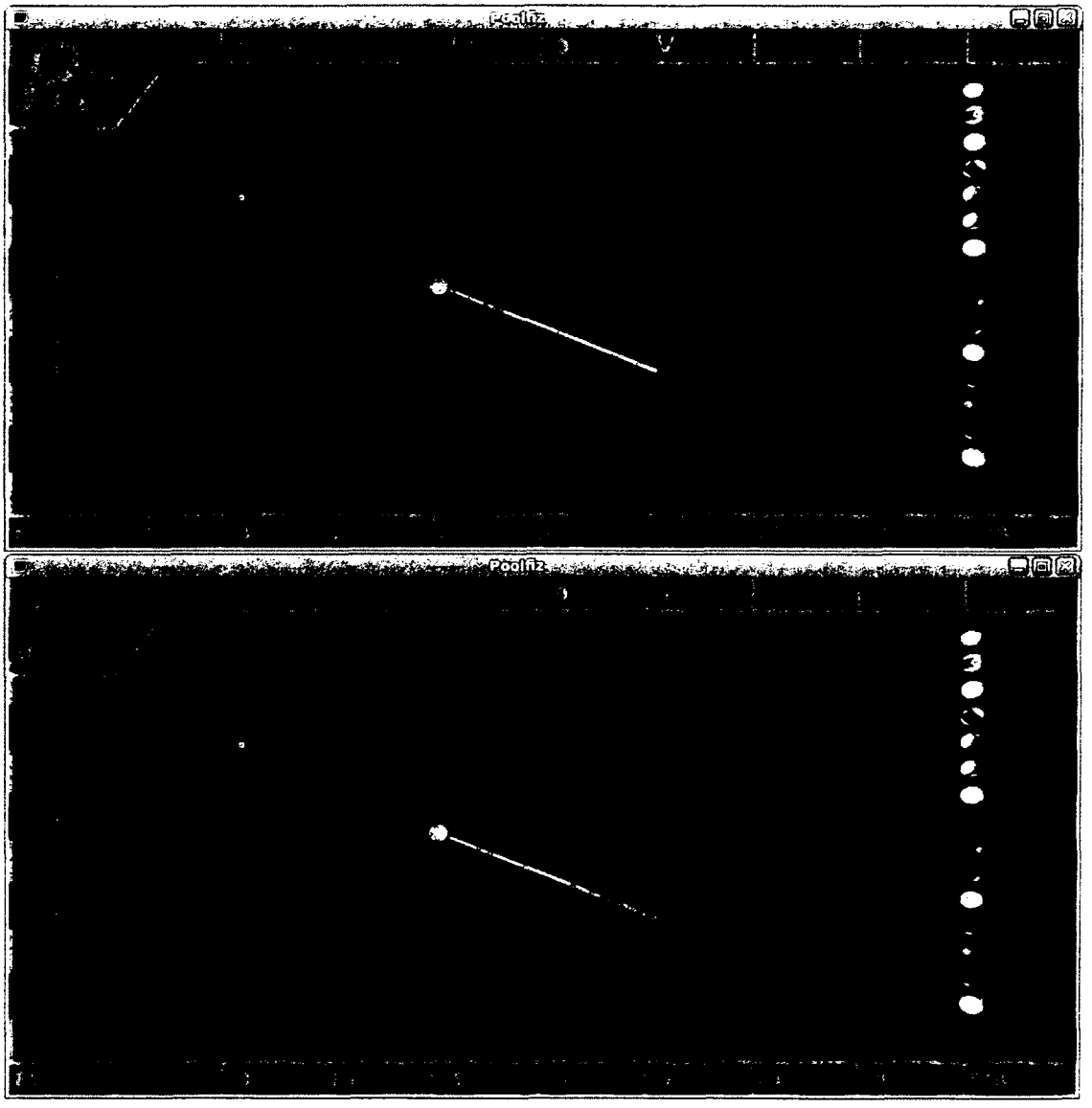

Figure 2.1: Reachable regions. For a simple shot in the upper right corner, we observe the reachable regions for Billiards (top) and PoolFiz (bottom).

influence of speed on the cue ball deflection after impact with the object ball.

2.5 Modeling billiards players

As discussed above, some kind of simulation of the physics is required for the computational player to be able to predict the outcome of a given shot. In this section, we assume such a simulation model is available in the form of a black box. The more accurate the simulator, the more accurate predictions and planning of the AI player. In the first Olympiads, all the players shared the same simulator, which also acted as table referee. When playing on a robot, or when using a highly accurate simulator, it is expected that different players will use different simulators for their planning.

Two aspects mainly require our attention. A sequence of shots has to be established, and thereafter, their execution has to be computed. By a shot, we mean a somewhat generalized shot including repositioning of the cueball. In the following subsections, we first address the planification of the sequence of shots as modeled by a dynamic programming (DP) formulation. We also discuss some solution strategies for this admittedly abstract DP formulation. Actual shots required by the dynamic programming planner are then computed using a local optimization model. We cannot avoid the temptation to report on our preliminary experience with respect to the local optimization.

Before going further, we must warn that the following modeling assumes the 8-ball game setting. As the model is rather abstract, we expect no fundamental problem in extending this to other pool variants, but of course several minute details will have to be adjusted for other forms of the game.

2.5.1 A dynamic programming formulation

Modeling billiard games is a difficult task and very few known frameworks fit correctly the specific characteristics of these games which are : continuous state and control spaces, actions taken at discrete unknown instants, a specific turn-taking game rule which allows a player to play several times as long as he keeps the hand, stochastic perturbations of moves. C. Archibald *et al.* [13] have proposed a stochastic game framework valid for finite or repetitive versions of the billiard games and they studied the existence of Nash equilibria associated with stationary Markov strategies.

We will first focus on the stochastic control problem mixing discrete and continuous aspects before introducing the game theoretic ingredients in the model.

Stochastic dynamic programming has been introduced to model dynamic discrete processes where random events occur after decisions are applied at each stage of the process. The state space X specifies the position of each ball on the table. It is continuous here but may be discretized to locate balls on the table. The control space $\mathcal{U}$ corresponds to the value of the cue parameters which are adjusted at the beginning of the shot. Each transition is the result of a nonlinear constrained optimization problem that represents the dynamic trajectory of the balls on the table under random perturbations. As seen below, one tries here to minimize the distance of the cue ball to a specified location while maximizing the probability of sinking the object ball. The overall objective function is to maximize the reward of the player (which is positive when the object ball is sunk in the pocket). The way to model that situation is to add a Boolean variable y_t at each stage which is one when the object ball is sunk. Observe that, when $y_t = 0$, the hand is lost and the opponent will inherit the current table. Thus, in difficult situations, a defensive step will try to minimize the opponent's reward. Note also that, as long as expectations are used in the stochastic case, a value function can be estimated by backward calculations, starting at an ideal state to pocket the last ball.

Let $\pi = (u_0, \ldots, u_{T-1})$ be a feasible policy (series of actions for the states $(s_1, \ldots, s_T)$), $P[s' \mid s, u]$ the probability of going from state s to state s' while taking action u, $R(s)$ the reward for reaching state s. The stochastic program can be stated in the following way (using here a finite length game) :

$$\begin{aligned} \text{Maximize} \quad & \gamma P[y_T = 1 \mid (u_1, \ldots, u_{T-1}; \omega_1, \ldots, \omega_{T-1})] + \textstyle\sum_t E[R_t(s_t, y_t, \omega_t)] \\ & (s_{t+1}, y_{t+1}) = \text{argmin}_{s(\tau) \in X, u_t \in \mathcal{U}} \; E[F(s_t, u_t, s(\tau), \bar{s}_t, \omega_t)] \\ & s_t \in X, y_t \in \{0, 1\} \end{aligned} \tag{2.1}$$

where $\omega_t \in \Omega_t$ is a random variable with normal distribution and γ is an harmonization factor. The transition step associated with the internal minimization sub-problem will be detailed below. It relies on a target position for the cue ball corresponding to state

$\bar{s}_t$. The trajectory is represented by $s(\tau)$ where $\tau \in [0, \tau_f]$ is the continuous time of the motion of the balls, $s_t = s(0), s_{t+1} = s(\tau_f)$.

Let now $V^\pi(s)$ be the optimal expectation of the final reward when applying the sub-policy $(u_t, \ldots, u_{T-1})$ from state s_t through the final state.

The equation:

$$V^\pi(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^\pi(s')$$

will be at its optimum if it satisfies the condition:

$$V^*(s) = R(s) + \max_{\pi(s)} \gamma \sum_{s'} P(s'|s, \pi(s)) V^*(s')$$

Thus, the sub-policy $\pi^* = \{u_t^*, u_{t+1}^* \ldots, u_{T-1}^*\}$ will be optimal for this problem.

For billiards, as stated in [13], we are faced with a continuous-state and continuous-action domain, which introduce further complications.

We find ourselves evaluating an integral instead of a sum:

$$V(s) = R(s) + \max_{\pi} \gamma \int_{s'} P(s'|s, \pi(s)) V(s') ds'$$

The optimal strategy of discrete dynamic programs is commonly solved by performing *backward computations* from the final state back to the current state. We will discuss that strategy below, after saying a few words about the underlying game theoretic issues.

2.5.2 Game issues

Let us consider first the simpler situation of a finite game (such as 8-ball). It is not easy to analyze the general strategy which interprets a defensive shot as the one which will *leave a difficult table to the adversary, thus expecting to take back the hand later to finish the game.* This means indeed that the final profit associated with $P[y_T = 1]$ is still valid. Nevertheless, we should here pull from a possible table at stage t to an unknown table at $t + q$ where q is the number of estimated shots left to the opponent. We refer to Archibald *et al* [11] for an interesting A.I. point of view of

the strategic issues.

2.5.3 Solution strategies for the DP model

The general problem of finding $V^{\pi}(s)$ for a deterministic problem may be quite complex by itself, and often becomes intractable as more variables come into play. It is even worst for a problem with a continuous state and action space. However, in our case, we can benefit from the knowledge we have of the game we are solving and extract important information to narrow our search to a limited number of action-state space combinations.

Assumption 1 : $y_t = 1, t = 1, \ldots, T$

Assumption 1 corresponds to the so-called *win-off-the-break* winning sequence. The optimal value is hence the expected sum of remaining rewards (for each shot) to get through the game. The way to cope with the fact that the success of the game is now deterministic is to add a *technical reward* to an easier shot, or, equivalently, to a better position for the cue ball on a given table.

Assumption 2 : The cue and object balls are the only balls expected to move.

With assumption 2, we get a direct estimate of the future positions of all future object balls. The player strategy corresponds to the Optimistic Planner described in [11].

Observation : Later, we will naturally relax Assumption 2 to authorize collisions with other balls (and with the rails). Then, a new question will arise : should we consider only side collisions due to the trajectory of the cue and object balls, or should we include different shots aiming at repositioning the cue ball so that a collision happens with another ball (or a cluster of dangerously close balls) ? That latter situation is now under study. The skill model to compute the optimal trajectory of the balls will here be much more intricate as it will involve elastic collisions and singularities (see [48]).

Back to the stochastic dynamic program, assuming Assumptions 1 and 2, we will now proceed backward to estimate the optimal value of a given table. We may start with the last table which can be modeled as the current table where we want to sink, not the current object ball, but the the one which should be sunk at the last shot.

To compute all $V^{\pi}(s_{T-1})$, we must compute the pocket angle for each remaining balls and then perform an inverse step to position the cue ball in all feasible states at $t = T - 1$. The optimization black box will be very useful as it already works with an estimate $\bar{s}_t$ of the desired repositioning state.

Observation : The curse of dimensionality will probably make these computations cumbersome and time-consuming if T is large, but we can begin testing increasingly complex situations setting $T = 2, 3, \ldots$ which can be easily adapted from the general case.

2.5.3.1 Optimization approach

Here we describe briefly the minimization sub-problem associated with the transition step at stage t as defined in model (2.1).

Suppose we are using the black box simulator described earlier, with the action variable u_t being the finite-dimensional vector of the five parameters a, b, θ, ϕ and v. The trajectories $s(\tau)$ of the balls moving on the table at a given shot depend thus on the initial state $s(0) = s_t$ and on u_t, the latter being perturbed by the random variable ω_t.

To compute an optimal action u_t, we minimize a least-square function associated with the desired reposition $\bar{s}_t$ of the cue ball on the table and the desired sinking of the object ball. The objective function can thus be modeled in the following way :

$$\begin{aligned} F(s(\tau), u_t, \omega_t) = \quad & \|s(\tau_f) - \bar{s}_t\|^2 \\ \text{subject to} \quad & s(\tau) = \psi(s_t, u_t, \omega_t) \\ y = 1 \quad & \text{if } s_{obj}(\tau) = \bar{p} \text{ for } \tau \in [0, \tau_f] \end{aligned}$$

where ψ is the dynamic transition function associated with the simulator, s_{obj} is the component of s associated with the object ball, and $\bar{p}$ is the component of $\bar{s}_t$ associated with the pocket. The solution of the minimization sub-problem will then give $s_{t+1} = s(\tau_f)$ with the corresponding value of y updating y_{t+1}. In the stochastic case, the function is replaced by an expectation value computed from the known distribution of the noise on the action variables.

The detailed formulation of the minimization problem and its resolution by an

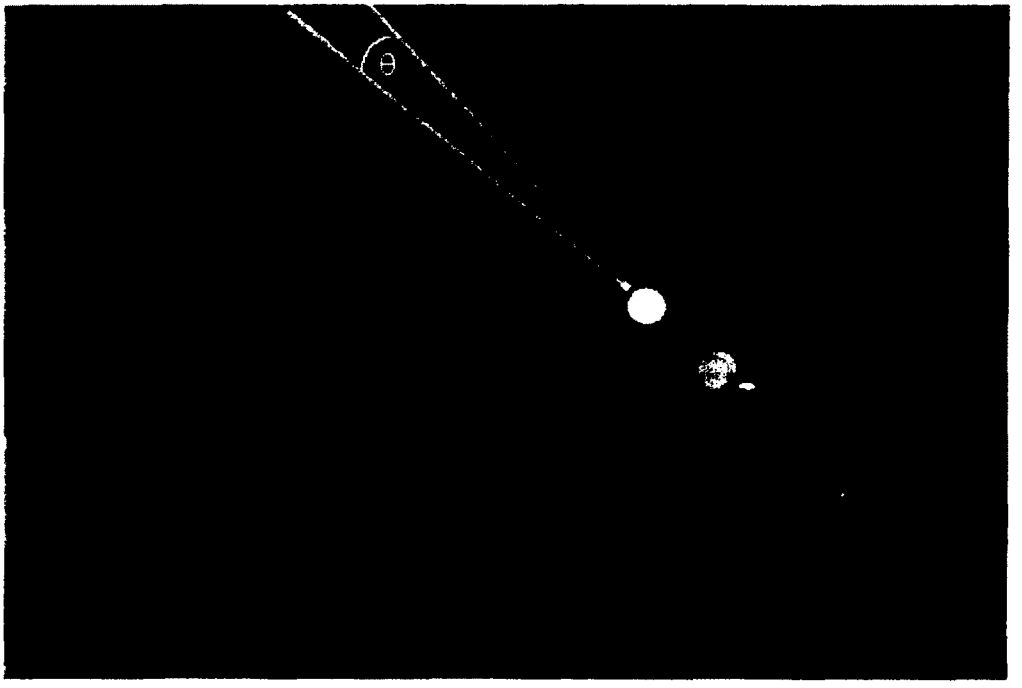

Figure 2.2: Shot difficulty. The distance of the cue ball to the object ball, of the object ball to the pocket, the cut angle and the angle with the pocket all influence the difficulty of a shot, proportional to the admissible error with respect to the "perfect" shot that will sink the object ball right in the center of the pocket.

adaptation of Powell's derivative-free Quasi-Newton method (see [ ]) can be found in [ ].

2.5.3.2 Value of Billiards

In a general matter, the difficulty of executing a shot on a billiard table is defined by two aspects: i) the distance between the cue-ball, object ball and pocket, and ii) the angle at which the cue-ball will hit the object ball. Other factors, such as the closeness of the cue-ball to the rails or other balls partially blocking the path of the shot, also come into play. However, we can still get a fair estimate of a shot difficulty by using the first two aspects described. A real and precise difficulty coefficient is almost impossible to compute since it will not only rely on the percentage of success of the computed shot, but also on the impact of this shot for the cue-ball repositioning.

As shown in Figure 2.5.3.2, both the angle and distance factor result in a calculable angle θ for the shot difficulty. A shot aimed at the center of this angle should normally result in the best success percentage for this shot.

Once we have computed this coefficient, we can use it to evaluate any given table state, simply by taking the maximum value of all the available shots for a given cue ball position. If we now divide our billiard table into a 50x100 grid, we can take the value of $\max_{i=0}^{n} \theta_i$ at each of these points and come up with the best repositioning targets for the next shots by removing each time the ball we would be aiming for (since we assume it will be pocketed).

However, this first and seemingly logical evaluation of a billiard table state is not necessarily applicable to all variants of the game, and does not for the moment take into account defensive shots. It does however provide an estimation of table state values for offensive play.

2.5.3.3 Results

To show the effectiveness of the optimization model discussed in this paper, we ran some tests on randomly generated table states and computed possible shots using local and global optimization libraries. We show here only two of the best performers we tested for the sake of brevity. BOBYQA was selected since it was the fastest of the approaches tested and DIRECT-L was the most successful with the best solutions found. Both approaches were implemented using the NLopt library ([36]). BOBYQA, a local optimization method by Powell [53] was tested with five different starting points, distributed over the possible shot parameters. DIRECT-L, a global optimization method by [37], was tested with only one starting point.

A total of 1000 random table states were generated, and each possible shots on these tables were explored, thus coming close to a total of 20,000 various shots, which were computed using both libraries. Each of these shots represents a combination of an object-ball, an aimed pocket, a cue-ball and a repositioning target. As can be seen below, types of shots included direct, kick, bank and combine shots.

The local optimization method, even though launched five times with various starting points, gets a slightly lower success percentage for bank and combine shots.

DIRECT-L	Success %	Avg. time	Avg. time succ. shots	Avg. Dist.
Direct shots (4310)	97.21%	0.37 sec.	0.38 sec.	0.15m
Kick shots (6950)	95.59%	0.78 sec.	0.78 sec.	0.19m
Bank shots (7520)	72.0%	0.85 sec.	0.99 sec.	0.59m
Combination shots (1150)	85.13%	2.02 sec.	2.09 sec.	0.4m

BOBYQA	Success %	Avg. time	Avg. time succ. shots	Avg. Dist.
Direct shots (4310)	96.10%	0.08 sec.	0.07 sec.	0.34m
Kick shots (6950)	88.21%	0.13 sec.	0.12 sec.	0.43m
Bank shots (7520)	35.45%	0.12 sec.	0.16 sec.	0.68m
Combination shots (1150)	46.00%	0.31 sec.	0.34 sec.	0.62m

Table 2.1: Results of 19930 different shots computed with the local optimization method BOBYQA and global optimization method DIRECT-L. Average times(seconds) are displayed for all shots, as well as for successful shots. The average distance (meters) is the distance from the repositioning target.

The DIRECT-L, however, being a more global approach, achieves better results but at a higher cost in calls to the simulator. What these results actually show us is that we can obtain satisfying results to compute single shots with very few calculations. The trade-off between the solution's quality and the computation time needs to be carefully studied to achieve the best compromise, but this will also vary with respect to the variant of game played. It should also be noted that in the case a solution is not found for a given shot, it is usually because it is not possible because of other balls or rails restricting the queue motion. Thus a global method may be able to find a solution by doing a spectacular shot, but one that is not very reliable when played on a stochastic system. Further testing is needed to adapt the search criteria to the type of game played, and the level of noise present in the simulator.

2.6 Perspective

We described a new application field of computational billiard and proposed a method of approaching the problem from an OR perspective. The field is really emerging, and much is to be expected in some near future. However, the ultimate

goal to have a computer robot challenge human champions appears quite remote. We did not discuss the robotic aspect of this challenge, but it should be clear that good progress has been made on the computational aspect, and that much remains to be done in this, once again, new field.

From an OR specific point of view, we have described a stochastic dynamic programming framework to model an optimal strategy of a generic billiard player in the presence of noise. The key contribution is an improved estimation of the table value at some stage that takes in consideration the expectation of the difficulty to complete a sequence of winning shots and not only the current one. The present modeling is coherent with the conclusions of Archibald *et al* [11] who state that 'strategic skill' is decisive whenever 'the execution skill' is imperfect.

On the other hand, we have presented an improved player simulator based on an accurate optimization tool to drive the cue ball trajectory towards the desired spot with the 'best' table value. Numerical simulations on repeated launching of 8-ball games were performed with relative success, including indirect shots like bank and combine shots, confirming the robustness of our player.

Current work aims at improving the player skill to cope with more realistic situations including the precise modeling of elastic collisions with banks and the breaking of clusters of balls.

Finally, the introduction of defensive strategies to take in consideration the game theoretical aspects will be the direction of further studies. It is indeed evident that the development of future competitions between virtual players controlling robots acting on a real table will progressively increase the level of noise effects, thus turning more crucial the improvement of the 'strategic skill' of the players.

2.7 Acknowledgments

We are grateful to François Gaumond and Benoit Hamelin for their careful reading of the manuscript and useful comments and François Gaumond's work on the Billiards–PoolFiz interface. We also wish to thank the referees for their useful constructive suggestions.

2.8 Compléments

La rédaction de cet article s'effectua en parallèle avec la préparation d'un joueur pour participer à la 5ième édition du "Computational Pool Tournament", organisée par l'Université de Stanford. Malheureusement la compétition fût annulée le jour même où elle devait avoir lieu pour cause de bris techniques et n'eût jamais lieu. Cependant, en préparation à cette compétition, un aspect intéressant et assez important avait aussi été étudié. Celui de la génération d'une casse optimale.

2.8.1 Casse optimale

Lors d'une partie de billard, aucun élément ne sera aussi important que la casse pour déterminer la suite de la partie. En effet, sur une table réelle, les professionnels attachent une importance particulière à ce premier coup. En fonction du type de jeu, la casse sera défensive ou offensive, et la dispersion des billes permettra ou non de terminer la table avec facilité.

Cet aspect du billard est néanmoins l'un des plus complexes à prédire. En effet, plusieurs facteurs vont influencer la dispersion des billes; la position initiale dans le triangle, l'usure du tapis, l'humidité dans la salle, l'élasticité des billes lors des collisions. Cependant, même avec tous ces facteurs, si les billes sont bien positionnées dans le triangle au départ et sont bien serrées les unes contre les autres, on peut prédire certains comportements typiques. Pour ne pas empocher la blanche par exemple, on sait qu'il faut s'assurer que sa trajectoire après le premier impact soit distant avec les poches aux coins. On sait aussi par exemple que pour une casse défensive au jeu du continu, on peut viser la dernière bille au coin du triangle de facon à ce qu'au moins deux billes se déplacent (casse légale, voir figures 2.3 et 2.4), mais reviennent à leurs point de départ pour laisser l'adversaire en mauvaise position.

Dans le cadre des dernières "Computational Pool Olympiads", le simulateur utilisé étant Poolfiz, il a justement été possible pour une équipe d'en tirer profit. Le simulateur Poolfiz simule une casse aléatoire en déplaçant les billes d'une distance minimale aléatoire apres les avoir mises dans le triangle. Ainsi lors de chaque casse les collisions seront variées et le résultat différent. Cependant en simulant des millions de casses, il est possible d'en extraire un certain comportement, et de déterminer certaines casses

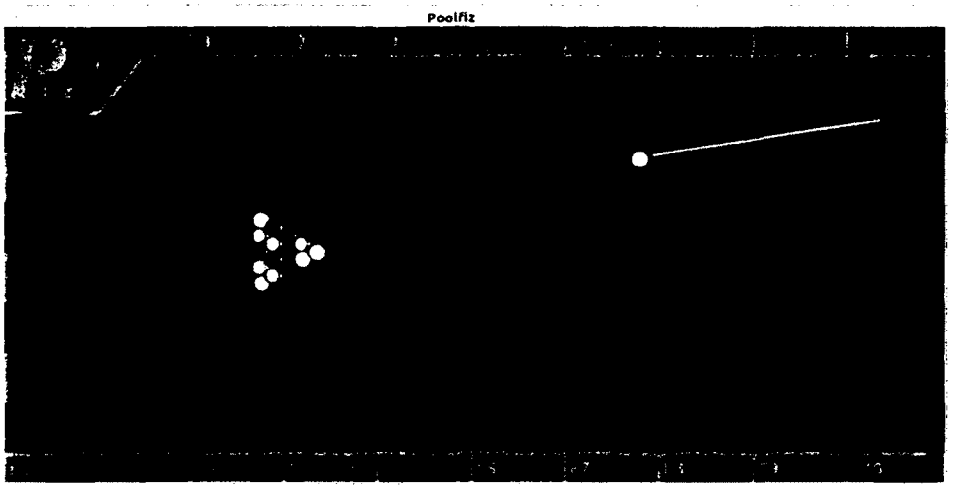

Figure 2.3: Il est possible d'observer dans cette figure les paramètres choisis pour une casse optimale. On peut voir que la bille du coin en haut à droite sera percutée et ira frapper la bande pour y revenir. L'impact sera également transféré à l'autre bille du coin, qui sera également envoyée contre la bande pour revenir à sa position initiale

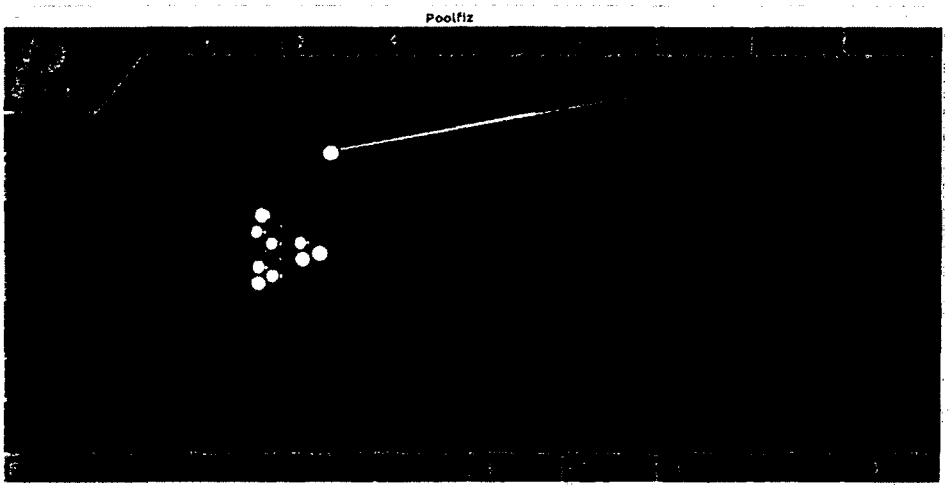

Figure 2.4: Résultat de la casse idéale, où au moins deux billes ont touché les bandes, mais aucun coup n'est possible pour l'adversaire. Ceci est une casse défensive typique pour le jeu du Straight.

plus efficaces que d'autres.

C'est en préparation du 5e 'Computational Pool Olympiads' que nous avons donc procédé à une analyse plus poussée du comportement relié à ce coup.

2.8.1.1 Métriques d'évaluation de la casse

Pour le jeu de la 8, il n'est généralement pas nécessaire d'effectuer une casse défensive comme il est assez facile d'empocher au moins une bille lors de la casse et de continuer à jouer. Nous allons donc concentrer nos efforts dans la génération de casses offensives. Afin de déterminer la casse optimale, il faut avant tout déterminer un critère d'évaluation précis pour ce coup. Au jeu de la 8, on peut facilement distinguer une très mauvaise casse d'une bonne casse. Par exemple une bonne casse nous permettra de garder le contrôle de la partie, et d'être en position favorable pour un coup. Une mauvaise casse signifie qu'on aura perdu le tour en empochant la blanche, ou bien en n'empochant aucune autre bille. Cependant un autre facteur important, et un peu plus difficile à évaluer entre aussi en compte comme la dispersion des billes ou la position favorable de celles-ci.

Deux mesures ont été utilisées afin de classifier la valeur d'une casse réussie. La première est une mesure de la moyenne des distances entre chaque billes sur la table. Cette distance nous permet d'éviter les casses où les billes seront ensuite très peu dispersées. Un désavantage d'utiliser la moyenne est qu'évidemment, il est possible que deux ou trois billes soient très proches, et compliquent donc l'état de la table. Pour éviter ce type de casse, nous utilisons comme deuxième métrique la distance des deux billes les plus proches sur la table.

2.8.1.2 Génération de la casse

Avec les métriques décrites plus haut, il a été possible de générer un sous-ensemble de casses intéressantes. Pour y arriver, il a fallu procéder à une discrétisation des positions possibles où mettre la blanche pour effectuer le coup. L'espace correspondant à la position en Y (largeur de la table) a été divisé en 10 parties égale d'où les coups ont été testés. Pour chaque position, un sous ensemble des paramètres en a,b,θ,ϕ,v a été choisi en les discrétisant de la façon suivante:

- a: 3 échantillons
- b: 3 échantillons
- θ: 3 échantillons
- ϕ: 20 échantillons
- v: 1 échantillons

L'espace du paramètre ϕ a été discrétisé de façon plus importante car il correspond à la direction à laquelle la blanche sera envoyée. Cette direction a été ajustée afin qu'on vise d'un extrème à l'autre du triangle. Chacun de ces coups a été simulé 500 fois avec un bruit gaussien ajouté aux paramètres. Les résultats ayant un taux de succès de 80% et plus ont été conservés. C'est-à-dire les coups empochant au moins une bille lors de la casse et n'empochant pas la bille de choc ou la bille numéro 8. Parmi ces coups générés, les 10 meilleurs ont été sélectionnés en utilisant les deux métriques décrites précédement, et ont été utilisés pour lancer des mini-tournois de 100 parties. Finalement, la casse donnant le meilleur taux de succès a été conservée. Celle-ci correspond aux paramètres $a = 3.3$, $b = 3.3$, $\theta = 10.0$, $\phi = 288.561$, et $v = 4.3$, avec la position de la blanche derrière la ligne à $(x, y) = (0.21748, 1.677)$.

2.8.2 Tests avec différentes librairies d'optimisation

Plusieurs tests ont été effectués afin de déterminer la librairie d'optimisation la plus adéquate pour le modèle que nous tentons de résoudre. Les deux approches avec le plus grand potentiel ont été sélectionnées pour être présentées dans l'article au début de ce chapitre, mais nous pouvons observer ici l'ensemble des ces autres résultats. Nous avons utilisé les différences finies pour les méthodes nécessiant les dérivées. Le coût de calcul d'une dérivée était donc de 5 appels au simulateur.

Comme l'intérêt de ce travail n'est pas de faire une étude approfondie des différentes méthodes d'optimisation, seulement une courte description de chaque approche sera donnée ci-dessous. Pour plus de détails, voir en référence [36] et [61].

- CRS: Approche d'optimisation globale sans dérivées, appartenant aussi à la classe d'algorithme évolutionnaire (Evolutionary Algorithm).
- ISRES: Approche d'optimisation globale sans dérivées, appartenant aussi à la classe d'algorithme évolutionnaire.

CRS	Success %	Average time	Avg time successful	Avg Fx
Direct shots(4290)	99.0909%	1.78099	1.77783	1176.49
Kick shots(6920)	98.4393%	4.30604	4.30756	1616.67
Bank shots(7470)	90.7497%	4.7451	4.94957	2196.93
Combine shots(1150)	93.4783%	9.47304	9.6229	1894.01

Table 2.2: Shot optimization results: CRS library

ISRES	Success %	Average time	Avg time successful	Avg Fx
Direct shots(4290)	98.8112%	4.66017	4.68351	1321.37
Kick shots(6920)	94.6821%	5.12939	5.34957	5006.46
Bank shots(7470)	72.3025%	2.80251	3.5336	14477.3
Combine shots(1150)	89.5652%	6.20391	6.32448	4960.32

Table 2.3: Shot optimization results: ISRES library

- L-BFGS: Approche d'optimisation de type Quasi-Newton nécessitant les dérivées (calculées à l'aide des différences finies).
- MLSL: Algorithme d'optimisation globale avec contraintes, effectuant plusieurs optimisations locales en partant de différents points de départ, et utilisant certaines heuristiques de "clustering" afin d'éviter les recherches menant au mêmes optima locaux déjà trouvés.
- TNC: Approche d'optimisation itérative utilisant une approximation de la solution des équations de Newton pour chaque itération, nécessitant aussi les dérivées.

Il est possible de voir que les techniques d'optimisation globales donnent des résultats assez bon en général, avec un taux de succès proche de 100% pour CRS et IRES, mais avec un temps de calcul également très élevé. Si on regarde L-BFGS, on observe

L-BFGS	Success %	Average time	Avg time successful	Avg Fx
Direct shots(4290)	97.2727%	0.320858	0.313046	1438.33
Kick shots(6920)	77.5145%	0.77026	0.659085	7524.51
Bank shots(7470)	43.0120%	0.467258	0.569153	45537.1
Combine shots(1150)	52.7826%	1.52541	1.7403	16771.9

Table 2.4: Shot optimization results: L-BFGS library

MLSL	Success %	Average time	Avg time successful	Avg Fx
Direct shots(4290)	94.8252%	1.11108	1.05578	1475.48
Kick shots(6920)	84.0607%	1.43231	1.38133	3118.41
Bank shots(7470)	42.3159%	0.774303	0.753445	28490.8
Combine shots(1150)	43.3913%	1.10149	0.887475	11824.3

Table 2.5: Shot optimization results: MLSL library

TNC	Success %	Average time	Avg time successful	Avg Fx
Direct shots(4290)	97.4825%	0.534545	0.527489	1028.74
Kick shots(6920)	82.4133%	0.850118	0.755925	10232.4
Bank shots(7470)	42.0482%	0.843525	0.712038	34940.7
Combine shots(1150)	36.5217%	1.34901	1.2066	37263

Table 2.6: Shot optimization results: TNC library

aussi un très bon taux de succès pour les coups directs, avec très peu de temps de calcul, mais les coups indirects et les combinaisons souffrent cependant beaucoup. La librairie MLSL semble aussi donner des résultats acceptables, en prenant un temps moyennement élevé, mais donne de piètres résultats pour les coups par la bande et les combinaisons. Finalement la librairie TNC est également proche des performances de L-BFGS en temps de calcul et qualité des résultats, mais performe moins bien pour les combinaisons.

Il très difficile de tirer des conclusions définitives de ces résultats, mais il est quand même évident que les méthodes d'optimisation globales donnent de meilleurs résultats pour ce qui est de l'exécution des coups indirects. Cela pourrait donc suggérer qu'il serait bénéfique d'utiliser une librairie particulière en fonction du type de coup à optimiser, et du temps de calcul disponible. Cependant il est important de mentionner que pour l'élaboration de ces tests, les points de départ donnés aux librairies d'optimisation pour minimiser la fonction n'étaient pas nécessairement réalisables, ce qui expliquerait en parti pourquoi les méthodes globales ont presque toujours trouvé une solution. Il a aussi été possible d'observer visuellement sur plusieurs des ces coups calculés que les paramètres choisis étaient souvent assez extrèmes (proche des limites), ce qui porte à croire que l'ajout de bruit affecterait un peu plus ces coups.

Chapitre 3

Un contrôleur robuste pour une approche à deux-niveaux dans le cadre du jeu du billard

Résumé

Le problème d'effectuer de la planification dans un domaine continu, et ce en présence de bruit, représente un important défi au niveau de la modélisation et de la complexité de calcul. Le jeu du billard offre une problématique très intéressante, tant au domaine de l'optimisation que celui de l'intelligence artificielle. Nous proposons dans ce travail un contrôleur spécialisé pour le jeu du billard, s'inspirant de l'optimisation robuste en combinaison avec des ajustements précis prenant avantage de la connaissance du domaine. Une formulation multi-objectif d'un contrôleur robuste sera présentée afin d'établir les outils nécessaires à l'exécution de n'importe quel coup voulu sur une table, dans le cadre d'une approche à deux-niveaux pour le jeu du billard. Quelques résultats seront ensuite présentés afin de démontrer le potentiel de l'approche, suivi d'une courte discussion sur la suite des travaux.

Commentaires

Cet article à été soumis à la revue 'Elsevier - Entertainment Computing - Games and AI' le 19 décembre 2011, accepté sous la révision d'un comité de lecture, et publié en août 2012.

La part de l'étudiant dans cet article est très grande ayant rédigé l'ensemble du texte et généré les tests démontrés. L'apport par rapport à la thèse est important. En effet, c'est dans cet article que sera présenté l'approche à deux-couches proposant une nouvelle méthode pour traiter la classe de problèmes présentée. La plupart des méthodes proposées jusqu'à présent dans la littérature pour la résolution de problèmes avec les particularités retrouvées dans le jeu du billard ne profitent pas du fait qu'on puisse extraire une grande connaissance du domaine avec un peu d'analyse. Il est aussi important de noter que le problème est généralement traité de façon discrète plutôt que continue, et que bien de l'information utile est ignorée.

Pour combler ce manque et pour tenter d'innover dans le domaine, une approche à deux couches composée d'un contrôleur robuste et un planificateur est proposée. Les éléments clés de cet article, qui contribueront à ce chapitre seront surtout centrés autour du contrôleur robuste dans le modèle. C'est ainsi que tous les outils nécessaires à la construction d'une approche complète sera proposée, auxquels le planificateur pourra se fier. Quelques pistes dans la fin de l'article laissent aussi un aperçu de la matière qui sera couverte dans le prochain chapitre.

L'article présenté est entièrement rédigé en anglais mais comporte aussi une section complémentaire en français à la fin du chapitre comportant des résultats n'ayant pas été soumis pour la parution dans la revue.

A robust controller for a two-layered approach applied to the game of billiards

Jean-Pierre Dussault, Jean-François Landry
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
Jean-Pierre.Dussault@usherbrooke.ca,
Jean-Francois.Landry2@usherbrooke.ca

Philippe Mahey
ISIMA - Campus des Cézeaux, BP 10125, 63173 Aubière CEDEX, France.
Philippe.Mahey@Isima.fr

Keywords: Pool, Billiards, Snooker, Game, Simulation, Planning, Strategy, Optimization, Artificial Intelligence

Abstract

Planning issues in a continuous domain in the presence of noise lead to important modeling and computational difficulties. The game of billiards has offered many interesting challenges to both communities of AI and Optimization. We propose here a refined controller for billiards based on robust optimization combined with specific adjustments to take advantage of the domain knowledge. A multi-objective formulation of a robust controller will be presented to provide the tools needed to execute any desired shot on the table, as part of a two-layered approach for the game of billiards. Some results will be then shown, followed by a short discussion on future work.

3.1 Introduction

A lot of research is currently done in the AI and optimization community on game-related problems, often in the form of robotics (Robocup challenge [24]) or computational games competitions (ICGA [35] and AAAI [4] conferences). Some of these games hide very complex problems which humans are often able to decompose very quickly but demand an important analysis if to be modeled and solved by a computer. It is our belief that some of the most interesting challenges can be found in the mixed continuous-discrete stochastic domain problems, and in one example in particular, billiards.

In everyday world, many of the variables affecting our decisions are continuous, like the speed at which we drive a car, or the time spent to reach a destination. Others are discrete, like the fact that we need to stop at one or two places before our final destination. We usually benefit from a lot of knowledge which was previously acquired to help us plan, and it is not an easy task to design efficient planning algorithms, as it is to gather this knowledge and organize it in an intelligent manner. As such, planning in a continuous domain with noisy parameters, which is the case of billiards, usually requires a great deal of analysis and domain knowledge. Adding a notion of continuity to the possible actions, making them infinite, and considering the resulting state space to be infinite as well, greatly complicates things. If we picture a planning in such a domain in which one wishes to plan a sequence of actions to reach a specific goal, our search will quickly become intractable.

If approaching these types problems from a human's point of view, a simple way to see things would be to divide them in two layers: planner and controller. Indeed, when trying to devise a plan for a series of tasks to perform, it is not necessary to plan every little detail with our planner. A simple general assessment of the best targets to aim for to minimize the chances of mistakes might be enough to provide us a general plan to proceed. We can then consult our controller that will compute all the necessary details (strike force, angle...) to provide us with precise information at base-level and let us know of the real value of such a plan, and if indeed viable. Of course, not every domain can be easily approached, and simply building an accurate and fast-enough simulator is another problem of its own.

3.1. INTRODUCTION

We will focus in this work on one particular well-known game and hobby, billiards. A lot of work has already been published on this topic, usually on the of the aspects of robotics ([50], [31], [44], [21], [56]), physics ([43], [14], [64], [63], [47], [55]) or player AI ([59], [20], [10], [38]). The latter, notably an automated decision-making process for this game, is the one which we will further discuss.

Billiards is a very interesting and challenging game, characterized by the presence of a theoretically infinite set of possible shots, each of which leading to a different outcome, thus creating a vast continuous search space. One must also account for the opponent's shots, by either creating an environment in which he will be able to keep the turn, or leave the opponent to play a very risky shot.

The physical aspects of the game must be simulated to a high level of accuracy if hoping to one day apply the solutions found to possibly challenge real-world professional players. Creating such a simulation also comes with its share of problems. Although many game simulators currently exist on the market, it is unclear as to how faithful these are to real world physics. Aspects like ball-ball collisions or ball-rail collisions are often simplified to create something visually pleasing, but not always physically accurate.

For the moment, we take for granted the availability of a physical simulator, which will be described briefly, while still keeping in mind that the approach used should be customizable to suit specific changes in the simulator.

It is our goal to explore this category of problems, and propose a solution using optimization methods, to search the state space in an intelligent manner by using knowledge deduced from the problem. The novel contribution we propose consists in the following aspects: a redesign of a previous optimization model to account for multi-objective shots to complete the array of tools needed by the controller, a robust formulation of this model to account for the stochastic aspect of the game, and finally a proposition of a general two-layered planning approach where we single-out the controller component with evidence of its good performance.

In the first part of this paper, we will do a recall on the general problematic surrounding the game of billiards to single out the research aspects on which we wish to expand. We will present some refinements to a previously defined optimization model ([38]) for the game of billiards. We will next take a closer look at the specifics

of the game, to determine a list of tools needed by the controller. We will then proceed to the modeling of a robust form of the problem we are trying to solve to account for the stochastic nature of the game, to finally conclude with some results and future work.

3.2 The game of billiards

Of the many existing billiards variants, we choose here to take a closer look at the game of 8-ball. This specific variant has been the focus of previous work in [38] [12] [31] [43] [58]. It has also been used as a testing platform for various agents in three past computational tournaments. Although the challenges present in the various types of billiards games vary, the aspects we discuss here do not.

The game of 8-ball is one of the most commonly played variants in North-America. It consists of 15 numbered balls, divided in a set of high (1-7) and low (9-15) balls as well as the 8th ball. To win at 8-ball, a player must successfully pocket all of his balls, high or low, and pocket the 8^{th} ball last. If the 8^{th} ball is pocketed before by accident, the player loses the game. Each shot must be specified (ball, pocket), and failure to complete the shot as called will result in a loss of turn. The first shot (break) is decided in a random fashion, and if a player pockets a ball on the break, he may then choose for which balls he will play. If no balls are pocketed on the break, the turn and choice is given to the opponent. A detailed description of the specific rules can be found on the site of the Billiard Congress of America ([15]).

It is possible to view this game as a combination of two fundamental aspects. The first and possibly most important of these aspects is related to the technical parameters needed to sink a ball in a pocket. Indeed, once a ball and pocket have been selected, a player must find out what are the optimal parameters to successfully complete this shot. If a player doesn't have the ability to execute such a shot, he will fail to advance in the game. The second aspect of the game is one that applies to better players, and that also makes the game such an interesting challenge; the ability to plan a sequence of shots. When a player executes his first shot, if he's fairly confident of his success, he will gain a great advantage carefully repositioning for his second shot. He must however analyse and decide which position to reach, and which

order would represent the optimal sequence of balls to pocket. Such an analysis will depend on various parameters, namely the technical level of the player, his weaknesses and strengths in executing particular shots. In various games, other parameters like points and faults will come into play, of which we will make abstraction in this paper since these will need to be addressed at the planning stage of the process.

3.2.1 Physics simulation

The poolfiz simulator ([43]) used in previous competitions allows for a two dimensional physics simulation of a pool table, by simulating the various ball interactions on the table following shot execution. The analytical solution of a quartic equation describing the ball trajectories on a table allows for a direct and precise computation of impact times, as well as a very fast simulation since no approximation methods are used.

The format used to specify shots on the simulator corresponds to the impact force transmitted to the cue ball by the cue stick, and is translated to the following 5 parameters(Figure 3.1):

- a and b represent the horizontal and vertical offset from the center of the cue ball;
- θ the elevation of the cue stick relative to the horizontal plan of the table;
- ϕ the orientation of the cue stick (in degrees) relative to the table;
- v, the initial velocity given to the cue ball.

By feeding these parameters to the simulator, the resulting table state is given, and an analysis of the shot's success or failure, as well as its value, may be computed. Since the simulator is deterministic, the same results will always be returned given the same input parameters are used.

3.2.2 Modeling the game of billiards

A defining and very important aspect of billiards is the principle of turn-taking (see [12]) in the game. It corresponds to the fact that in billiards, one can play all the way until the end of the game if he doesn't miss a shot, but he's also got the possibility of giving the turn to his opponent by the execution of a safety shot, if he

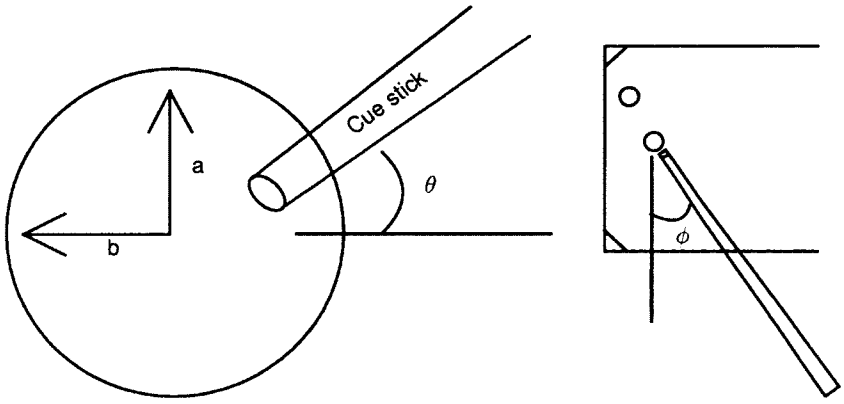

Figure 3.1: Side view on left figure to illustrate shot parameters. Top view on the right figure to illustrate cue angle parameter (ϕ).

thinks this global strategy will be beneficial in his overall game-plan.

In reality, defensive play is one of the most interesting and difficult aspects of the game of billiards. It demands a very precise analysis of the game state to weight the value of doing an offensive shot versus a defensive one. If a shot is too risky, it may be more valuable to play defensively, but if no good defensive shot is playable, one might as well take his chances and continue offensively. This kind of situation is easier to see in the games of Snooker and Straight, where it is often possible to have long exchanges of defensive shots, until one opponent sees an opportunity for a great shot, or one of the players misses his defensive shot.

The general model proposed in [12] defines the winning strategy as a perfect Markov equilibrium between two players. This implies the existence of a value function v on the state space such that

$$v(s) = \max_u [\int_S v(s')p(s'|s,u)]$$

where $s \in S$ is the state of the table and p is the probability to reach state s' when applying action u in the state s. Observe that, on his side, player 2 will minimize the same value function, but the model only considers offensive strategies defining sequences of successful shots until the player loses his turn. A defensive strategy should be an additional decision to leave the table in the worst possible state for the

game. Proceeding with a discretization approach in such a situation might prove problematic since the breadth of the search space will quickly explode. It is our impression that instead of exploring the state space, a lot of time could be gained by rather exploiting the game knowledge to our advantage, and thus eliminating useless calls to the simulator.

Evidently, exploiting the knowledge of the game also has its downsides. One must be very careful not to fall into the trap of encoding each and every possible situation as a rule-based expert system, since it may become very hard to decide with confidence if a shot really is better than another one. However some basic pool knowledge can easily be applied when analysing a table to quickly discard useless shots, as discussed in [38].

3.3 A two-layered approach

If we have to look at any professional tournament game, we can observe that a player will never execute a shot to which he doesn't know the exact outcome. Unless trying to break a cluster, each and every shot will be carefully planned to maximise the chance of a good reposition, and even when going for a cluster, it will usually be a controlled shot aimed to separate one or two balls from the pack and continue playing. It is very seldom that other balls than the cue and target balls will move on the table since the more collisions we have the harder it becomes to accurately predict the resulting trajectories, especially when noise is present in the shot. There is no reason why it should be different for a computer simulation. Thus if we are able to carefully determine the best positions to reach on the table for the next shots, we should be able to easily find a shot sequence maximizing our chances of winning.

For our model, this adds up to the following points:

- Discretization of the domain using repositioning targets.
- Partial ordering of the targets using shot difficulty coefficient.
- Listing of best shot sequences by planner.
- Consultation of the controller to discard bad shots and compute success percentages.
- Reorganization of shot sequences in order of success percentages.

This decision process is illustrated in the simplified Figure 3.2. It is important here to note that the shots which are generated by the planner correspond to a ball-pocket-target combination, rather than specific shot parameters. The shot parameters themselves will be found by the optimization procedure.

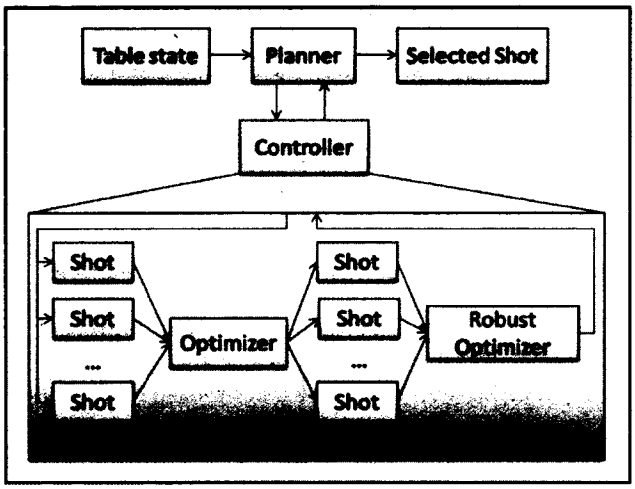

Figure 3.2: Illustration of a two-layered approach, where the planner first elaborates a plan by considering the best targets on the table and generating various shots to optimize, to then consult the controller, which in return will provide the list of achievable solutions.

We hope to differ from the planning approaches used in [58] and [10] by using a technique that first generates a plan based on interesting positions on the table, to then explore the best of these plans in a robust manner. The planner will then adjust the initial plan with the updated information and decide of the optimal step to be taken. The contribution we make in this paper mainly relies on the improvement of the controller aspect of the game, since the planning aspect will have to be adapted on the type of game being played, as well as on the stochastic aspect introduced by noise addition to the shot execution. We wish however to stress the importance of the robust controller in relation to the planner in the presence of noise-induced parameters. In a deterministic game, the use of a planner might be rendered useless (discussed in section 3.5) since a strong controller will most likely always find a shot, but with a stochastic game, the solution found will not always be robust and thus not always optimal. We still provide a few leads in the last section as to the direction

we will take in future work to plan in an efficient way.

To illustrate the necessity of a planner in some situations, and to motivate the existence of a two-layered approach, we provide in figures 3.3 and 3.4 an example of two possible shots leading to two completely different positions on the table. For these two shots, gaussian noise was added to the shot parameters using the standard deviations of the past Computational Pool Olympiads (ϕ=0.125 deg, θ=0.1 deg, v=0.075 m/s, a=0.5mm, b= 0.5mm). Although in most cases it would seem that a very good controller on its own can compensate for the lack of planning in the game of 8-ball, in this specific case, if the shot with the best position is chosen (in relation to pocket distance and angle), it results in a much harder next shot. Both shots are robust to noise (succeed in pocketing the first and second 100% of the time) but over an average of 500 games starting from this state, the shot 1 in Figure 3.3 led to victory in only 56% of the cases, while if the second best shot was selected (shot 2 in Figure 3.4), its different position on the table led to a 96% success rate. The reason for this is simple, if shot 2 is selected, it is easier to then follow with shot 1 and stay in good position for the 8th ball since it is in the same half of the table. If shot 1 is selected first however, a long shot has to be made to reach the next ball, thus failing more often. This is an isolated case not often encountered in 8-ball but one that serves as a reminder that a strong controller cannot always account for the lack of planning.

The following sections address the controller aspect of the game which will execute the shots requested by the planner and return the resulting robust shot with a success percentage.

3.4 Controller

The fundamental aspect of our approach consists in actually developing each and every tool a good player would have at his disposal. By this we intend to create a player with the ability to find out exactly what parameters to use for a shot, if such a shot is possible. Thus a part of this work consists in a pre-analysis of a table state which will inform us on the targets to reach, and of a formulation of the problem as a least-square problem to solve by minimisation techniques.

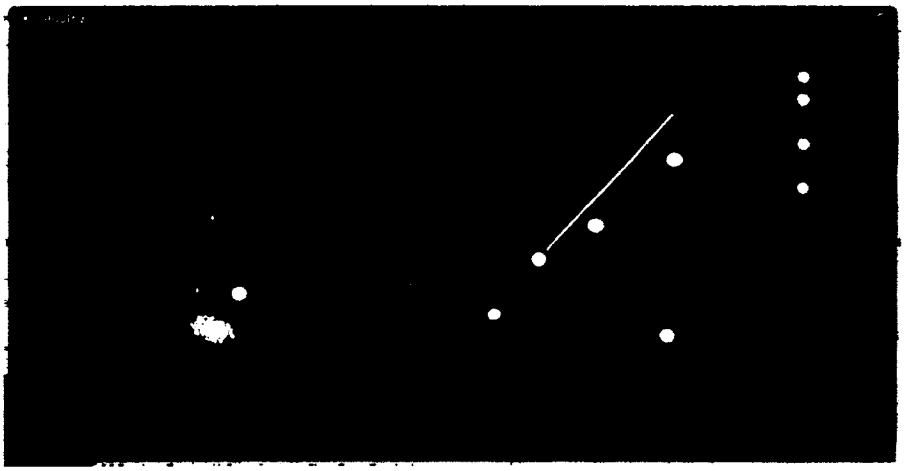

Figure 3.3: Shot 1. Robust shot leading to best possible position on the table in average. Success percentage to terminate the game: 54% (when testing with 500 states resulting from noise-induced parameters). White dots represent the various cue-ball position reached for noise-induced shot.

3.4.1 Multi-objective shots

In this section we will revise a previous optimization model to account for the problems of breaking clusters, and doing multi-objective shots, which can be quite useful in some variants of billiards games.

3.4.1.1 Direct shots

We first explored the aspect of position play in [ ] but reiterate here the basic principles, from which we will derive other useful techniques for our player. To compute a direct shot, we start by performing a quick analysis which allows to determine at which point the cue ball should hit the target ball for it to reach the pocket. This is only an estimate, based on the 90-degree rule (see [ ]), but allows us to find a suitable starting point for our optimization procedure. We can also eliminate shots for which the angle isn't achievable.

We can then proceed to define an objective function, which corresponds to the position of the cue-ball at its final resting time following a shot. We add a constraint to this function, representing the fact that we want to sink the target ball in a

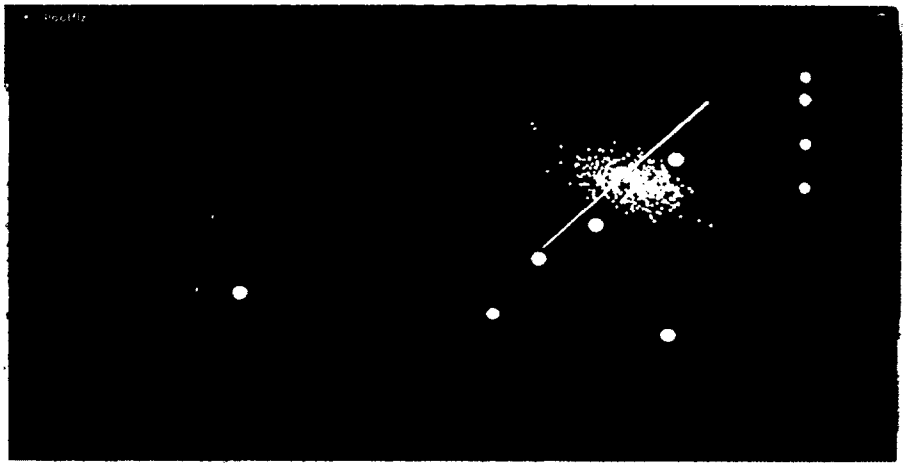

Figure 3.4: Shot 2. Robust shot with a slightly lower value than in shot 1. Success percentage to terminate the game: 96% (when testing with 500 states resulting from noise-induced parameters). White dots represent the various cue-ball position reached for noise-induced shot.

given pocket. However since this constraint will likely not be solved when we start optimizing our function, we model it as a penalty added to our objective function. This allows us to find a suitable solution to pocket the target ball first, and then proceed to find the best possible parameters to get us as close as possible to the target position. Assuming vector x contains the values related a given shot on the simulator, we can formulate the following objective function to minimize:

$$\min_x \frac{1}{2}\left(\|\mathbf{p}_c(x) - \mathbf{c}\|^2 + \rho\|\mathbf{p}_o(x) - \mathbf{p}\|^2\right)$$

where $\mathbf{p}_c$, $\mathbf{p}_o$ represent the outcome of a call to the simulator, used as a black-box for the moment, with the vector of shot parameters x (corresponding to $[a,b,\theta,\phi,v]$ in section 3.2.1). More precisely, $\mathbf{p}_c$, $\mathbf{p}_o$ are the positions of the cue and target ball at final resting time, $\mathbf{c}$ the target position we wish to reach with the cue ball, $\mathbf{p}$ the pocket at which we aim and ρ a penalty value to insure our target ball is pocketed (shown in Figure 3.5).

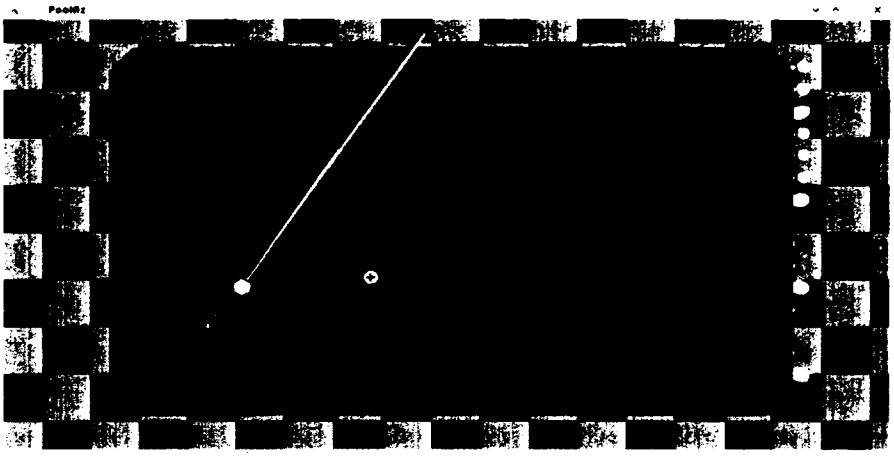

Figure 3.5: Shot optimisation for repositioning on a specific target for the next shot.

3.4.1.2 Breaking clusters

We have already discussed the aspect of breaking clusters in ([ ]) using pre-computed tables to facilitate function optimization. The approach used made it possible to break clusters but not in every situations, and the success of the optimization was highly dependent on the starting points. As such, we proceed to model our objective function in a different and more efficient way. We first re-use our initial model in (3.4.1.1), but instead of aiming for a repositioning target on the table, we switch this target to aim for a cluster target. We divide our objective function in two parts. The first part will be to satisfy the constraint of pocketing the target ball and to try to reposition at the center of the cluster we are trying to move. Obviously, trying to replace the cue ball into a spot where another ball is laying is quite difficult, and impossible in most cases. However, we can attempt to aim for that position, and as soon as we get to a collision with our target, switch to the second part of our objective function, which includes a minimum velocity for this collision, thus dispersing balls and hopefully obtaining a better table state.

We get:

$$\min_x \frac{1}{2} \begin{cases} (\|\mathbf{p}_c(x) - \mathbf{c}\|^2 + \rho\|\mathbf{p}_o(x) - \mathbf{p}\|^2) & \textit{if } i{=}1 \\ (\text{obj2}(x) + \rho\|\mathbf{p}_o(x) - \mathbf{p}\|^2) & \textit{otherwise} \end{cases}$$

with $\text{obj2}(x) = \|\mathbf{v}_c(x) - \mathbf{v}_{\min}\|^2$ as our second objective, where $\mathbf{v}_{\min}$ represents a minimum velocity, and $\mathbf{v}_c(x)$ the velocity at time of impact. i represents a boolean value, which is set to true if there is an impact with the second target. The minimization of our second objective $\text{obj2}(x)$ will ensure a minimum speed is given to the cue ball at impact time, dispersing our cluster. We do not specify here a $p_c(x)$ since we don't mind if the point of impact changes, as long as the impact occurs at a specified minimum speed.

Another problem, which arises when optimizing a function with many clustered balls in the table state is the time needed to do the simulations. For example, using poolfiz, a break shot demands 300 times more time to simulate versus a single impact shot. Our optimization methods also suffer when too many events occur after a shot since the function becomes very chaotic and the slightest variation in the parameters will result in a totally different table state. As such, it is not worth trying to optimize such a function considering the computing time constraints, and to work around this issue, we can proceed to a very simple trick. If two or more balls are together in a cluster, we can aim for the center of this cluster, and remove all other balls on the table. Once a solution is found, we can put the balls back on the table and test our shot. Of course this approach doesn't guarantee that the solution found will always be usable, but it is our experience that in most cases it easily breaks the cluster and demands little computing time.

3.4.1.3 Double shots

Having just defined a way to execute complex shots to break clusters on the table, we can use this secondary objective, and modify it to instead pocket a second target ball if desired. Indeed, by simply redefining $\text{obj2}(x)$ to $\text{obj2}(x) = \|\mathbf{p}_{o2}(x) - \mathbf{c_2}\|^2$ in our previous model, we will try to send our second ball (defined by position $\mathbf{p}_{o2}(x)$) to the desired pocket $\mathbf{c_2}$, as can be seen in Figure 3.6.

It is possible to see that we removed our cue ball repositioning constraint, since it most likely will be very hard or even impossible to control the cue ball final position while having constraints on two previous hits. Of course, only a limited number of impacts may occur before our ball loses all of its velocity, but results show that it is easily possible to find the parameters of a two-ball shot if one exists. This particular

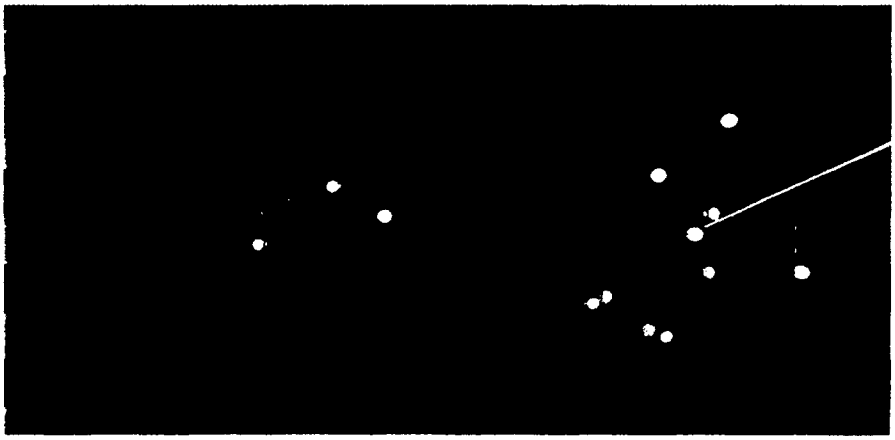

Figure 3.6: Double shot execution, where the first target ball is pocketed to the center pocket and the second target ball, to the corner pocket.

kind of shot is of little interest in the game of 8-ball, since it usually proves too risky and unnecessary. However, in the game of 9-ball for example, one player is always constrained on hitting a specific ball first, and this type of shot is exactly the kind that might allow the player to win the game early.

It may also be important to note that the current version of the poolfiz simulator doesn't impart the correct velocity after ball-rail impacts. This greatly reduces the ability to reach various targets on the table, and also prevent fancier shots otherwise possible on a real table. It will be interesting to test our optimization model on a more faithful simulator to see the improvement in constraint satisfaction, and hopefully new interesting shots.

3.4.2 A robust model

The initial model developed was targeted at providing our AI player with the tools necessary to reposition anywhere he liked on the table. This feature remains quite powerful and very desirable in that it is a very efficient way of finding shots with good repositioning value. The strength of this approach is also illustrated by the fact that a player is able to finish virtually any random table state he is given.

This model, however, tends to lose most of its advantage in presence of noise-induced parameters. As is discussed further in section 3.5, the success rate fell far

down when noise was added to the shot parameters. Two main hypothesis can actually be made as to the explanation of these results; either the wrong shot is selected amongst those generated, such that the shot chosen will have a completely different outcome with the slightest amount of noise, either the shots generated are riskier to achieve better repositioning, resulting in a higher sensitivity to noise.

If the wrong shot is selected, it is simply a matter of adjusting the evaluation function that is used to compare the available shots. This can be solved by launching many more mini-tournaments to determine statistically the best weights to be given to each computed shot. This calibration will need to be done one way or another.

If instead the problem lies within the shot generation model, an adjustment will be needed to our objective function. In the current model, the goal of aiming for pre-defined target for the next shot works very well without noise. However it is possible that the strength of the player (i.e. his ability to reposition almost anywhere on the table) is also his greatest weakness. In hope of reaching a given target, it is possible that the shots selected will be very fancy, that is, will have strong spin effect. Actually in the current model, it is not possible to measure this sensitivity unless we do sampling after a solution has been found to the optimization model. This is not very helpful except in the case of shot selection, since it will only help to evaluate the shots which were computed, but will not provide us with more robust ones. This problem is illustrated in Figures 3.7 and 3.8, where the white dots illustrate the type of shots we want to favour. We can roughly estimate a shot difficulty by measuring the distance and angle between the balls and pocket and this is useful in selecting good positions on the table, but shot difficulty also depends on the parameters used to execute the shot. As such, we need a way to find robust shot parameters, to sink the object ball, and also retain a decent position for our next shot.

3.4.2.1 Robust counterpart

Robust optimization, as detailed in [17] and [16], can be defined as a methodology to treat uncertain problems usually formulated as:

$$\min_u \{f_0(u,\zeta) : f_i(u,\zeta) \in K_i, i = 1,...,I\} \tag{3.1}$$

Figure 3.7: Illustration of possible repositioning targets, and sensitivity to noise when trying to pocket the ball in the bottom corner pocket using various shot parameters. White dots represent positions which are attainable while still pocketing the target ball when noise-induced parameters are used (90 to 100% of the time). Red dots represent positions which are reachable, but don't guarantee we will successfully pocket the target ball (success rate is between 70 and 90%).

where u is the vector of decision variables, f_0 the objective function, and $f_1, ..., f_i$, the structural elements of the problem (or constraints), and ζ the data of a particular problem instance. In our problem, this would correspond to the objective function of maximizing the value of the position, f_0, under the constraints of pocketing the balls $f_1, ..., f_i$ as described in model (4.1), with the vector ζ representing the shot results with noise-induced parameters.

Many methods can be used to solve robust optimization problems, unfortunately most of them are not directly applicable to our case since for the moment we use the simulator as a black-box and can't take advantage of any function-specific knowledge. However, we can still use some basic principles to find a good solution if one exists.

The first aspect which needs to be addressed is the value of a specific repositioning target. When choosing shot parameters, it is possible it will become necessary to trade-off the repositioning value to a certain extend. It is, however, also possible it will not be affected at all. This is why it would be better to aim for a shot remaining in a good zone rather than at a specified position. A good model should include two fundamental scalable variables; a shot success probability α_1, and a repositioning

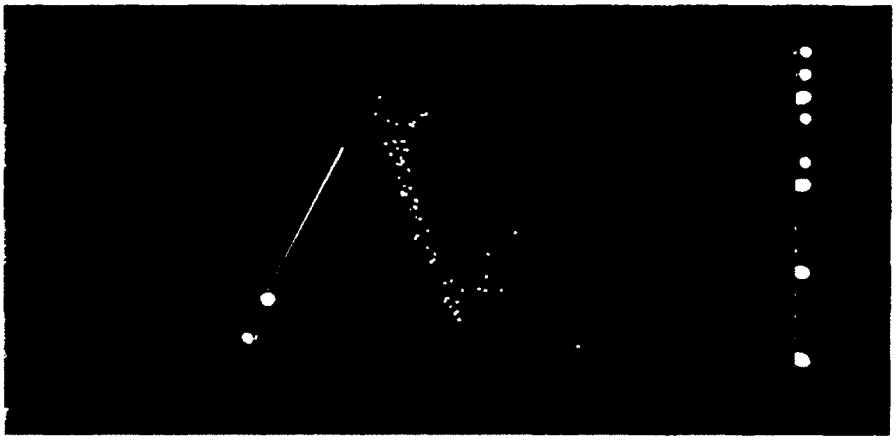

Figure 3.8: Harder shot where we can see the sure shots have considerably diminished. The distribution of the reachable zone is most likely affected by bottom center pocket which would result in a missed shot when noise modifies the cue ball trajectory and directs it in the pocket.

success probability α_0.

Again, taking our initial model, using balls' final resting positions as $b_i(u)$, with the vector u representing our initial shot parameters and $b_{\text{cue}}(u)$ to represent our cue ball. Let's use p as our pocket position for pockets 1 to 6.

The noise-less model, or as we will call it here a nominal model, can be defined as a function minimizing the distance between the cue-ball's final position and a given target t on the table, under the constraint of pocketing the target ball.

$$\begin{aligned} &\min f(u) = ||b_{\text{cue}}(u) - t|| \\ &\text{s. t.} \\ &\quad ||b_i(u) - p|| = 0 \end{aligned} \tag{3.2}$$

As we just mentioned, this model doesn't hold very well with noise-induced parameters. To counter this, it is possible to redefine it instead as:

$$\begin{aligned} &\min f(u) = \text{Posval}(u) \\ &\text{s. t.} \\ &\quad ||b_i(u) - p|| = 0 \end{aligned} \tag{3.3}$$

3.4. CONTROLLER

The objective here will be to minimize the value of a reposition Posval(u) (with a smaller value representing a better position), while still satisfying the same constraint of pocketing a ball. The fundamental difference with this parameter is that we no longer try to reach a precise position on the table, but rather maximize the value of a position in a zone for the next shot. The problem with this approach is the discontinuity of the function. Indeed, many positions will have a constant value of 0, and will create stationary points from which it will be hard to climb out. However it is possible to use a two-step approach, by first minimizing the distance to a target using model (4.2), to then switch to (3.3) when we have reached the desired zone. The goal of this reformulation of the initial problem is to introduce a model for which we will be able to formulate a robust counterpart. Indeed, if we wished to robustify model (4.2), we would stray from our initial objective which is to reposition in a decent zone for our next shot. By using model (3.3), we are able to relax our objective function so that it becomes easier to minimize under a robust constraint.

Assuming our perturbed shot parameters(for a given noise) vector ζ. If we consider U as the set of the normal range of the parameters under which our shot still remains successful $U = \{\zeta : b_i(u, \zeta) = p\}$,

We can formulate the comprehensive robust counterpart of (3.3) as:

$$\begin{aligned} &\min_{u} \sigma \\ &\text{s. t. } \text{Posval}(u, \zeta) \leq \sigma + \alpha_0 \text{dist}(\zeta, U) \\ &\qquad ||b_n(u, \zeta) - p|| \leq \alpha_1 \text{dist}(\zeta, U) \end{aligned} \tag{3.4}$$

$\forall \zeta$. Thus, our minimized value σ will represent a robust solution, with constraints and objective weighted by the adjustable α values, and the noise effect on the satisfaction of our constraint. Of course, to solve this model, we will have to proceed to a discretization of our normal range values U, but it should provide us with a good estimate of a shot's chances of success. It should be noted here that we used a comprehensive robust counterpart to describe our problem, since it is possible to imagine a solution where the constraints (pocketing the ball) will not be satisfied for every ζ in U. This would result in a possible imbalance where a shot with guaranteed success but a guaranteed useless position, would be chosen over a shot with 95% success rate

and very good reposition.

3.4.2.2 Penalized robust model

As was needed for the earlier described position-play model in section 3.4.1.1, we will now proceed to a slight reformulation of the robust model (4.5) to consider the constraints as penalties to our objective function. Solving our problem in this manner allows us to optimize a function that will not only maximise the value of the position reached, but also minimize the distance between the target ball and the pocket so that when searching for an optimal solution, we remain in the region closest to satisfying the constraint of pocketing this ball.

$$\min_{u} f(u) = \max(\text{Posval}(u, \zeta) - \text{Val}_{\min}, 0) + \rho||b_n(u, \zeta) - p||_{\infty} \ \forall \zeta \qquad (3.5)$$

By finding the parameters of u which minimize this function, we should be able to reach a solution that is not only robust, but also leaves us a very good zone for the next shot. We use the infinity norm on our constraint as to penalize it in a linear fashion to facilitate its optimization, and try to reach a position with a minimum value $\text{Val}_{\min}$ for the zone.

3.5 Results

We present in this section the result of various tests, mainly to illustrate the potential of the proposed approach, but also to show the progress which has been made in regards to alternative methods used to solve the game of billiards. As has been mentioned before, it is very hard and complex to design a physically-accurate simulator for the game of billiards. We are currently working on such a simulator [1], in hope of having it carefully and precisely calibrated to a real billiards table. In the meantime, however, two options are available to us; the Poolfiz ([43]) and the Fastfiz ([32]) simulators. Poolfiz was used as the basis for the first three computational pool tournament. Subsequently, Fastfiz, a re-engineered version of poolfiz, was developed

1. Julien Ploquin, forthcoming Master's thesis, Université de Sherbrooke

by the Stanford Computational Billiards Group ([32]) to be used in future competitions. Fastfiz has the advantage of being must faster than poolfiz, and as such, allows for more computations in the same time, and for such reason we made the necessary conversions in PoolMaster to use it. A deep and well-described analysis of the winning player of the last computational pool olympiads ([10]) allows us to get an idea of the level at which it currently stands. Since we are in preparation for the next tournament, we will do some comparisons here with the another player (Pickpocket) that will not be participating this year and which was made available to us by its author[2], and was also used for comparisons in [10]. It is important to keep in mind that these comparisons are simply to show the validity of the robust controller model proposed here. The true benefit of such a model may be more apparent in other forms of billiards (9-ball, snooker, straight), but as we show here, even the robust controller on its own performs relatively well for the game of 8-ball.

For these tests, Pickpocket is used exactly as provided by the author, using the default settings, where a search breadth of 15 shots is done at the first search level. The only thing which was adjusted was the noise level used for the shot evaluations, which was adjusted to the correct values depending on the noise used for the various tests.

PoolMaster was used with the optimization library NLOPT([36]), using the method BOBYQA ([53]) to minimize the objective functions. Bound-constraints were set on the shot parameters to limit shot velocity and stay within the bounds of a physically possible shot. A sample of 15 shots was done to evaluate the value of $f(u)$ in model (4.5).

3.5.1 Random tables

To determine the efficiency of the robust approach, we generated 500 tables with balls randomly positioned, and constrained the player to play on stripes on each of these table states. In such a way, we hop to minimize the variations which may occur from a table state to another after a break shot, to better compare the two players. A maximum time of 10 minutes was allowed players to compute their shots

2. We wish to thank Mike Smith for graciously providing us with his player, Pickpocket, to perform various tests.

on the same computer, to conform with the previous rules used for computational pool tournaments. Players were all given the same starting table states and played until they missed a shot, if they finished all their shots, this counted as a success, otherwise as a failure. We justify the sample size of 500 table states by pointing at the comparison graphs. These show a cumulated average of the successes for cleaning the table at different noise levels. It can be seen that the average usually stabilised at a sample of around 300. We were also interested to see how the two players would fare at different noise levels. Such a study has already been done in depth in [11] to determine optimal noise level for a competition. We are not interested in reproducing these tests here, but we nonetheless performed each test at levels of 0, 0.5 and 1 of the parameters used in previous tournaments (standard deviations of ϕ=0.125 deg, θ=0.1 deg, v=0.075 m/s, a=0.5mm, b= 0.5mm) to see if the improvement of the player would be similar independently of noise levels.

To first gauge the effectiveness at which the nominal model could clean off the random table states with the added capacity to break clusters, we ran some preliminary tests with no noise added to the shot parameters. The result was a success rate of 100% for 500 games (not shown in graphs for obvious reasons), which means the player is now well-rounded enough to deal with almost any given situations on the table. By watching some of the games of these tests, we were able to see that some extraordinary, and very unlikely shots were performed in many games. These kinds of shots are rarely observed on real life games and for a good reason, they are highly susceptible to the sightless change in the shot parameters. The same success rate was observed for the robust version of PoolMaster. Pickpocket reached an average of around 90% for these tests, as can be seen in Figure 3.9. This means that even without noise, it is not guaranteed that any approach will always find a solution in any given state. Of course if the cue ball is isolated in a corner by adversary balls, it is possible that no offensive shots are possible. Thus it is more than likely that the robust and non-robust versions of PoolMaster performed some interesting gymnastics to find a solution in every table state.

We next proceeded to tests the players with various levels of noise. First at a level of 0.5x (in Figure 3.10), we saw the success percentage fall all the way down to 32% with the non-robust approach. This gives us a great deal of information concerning

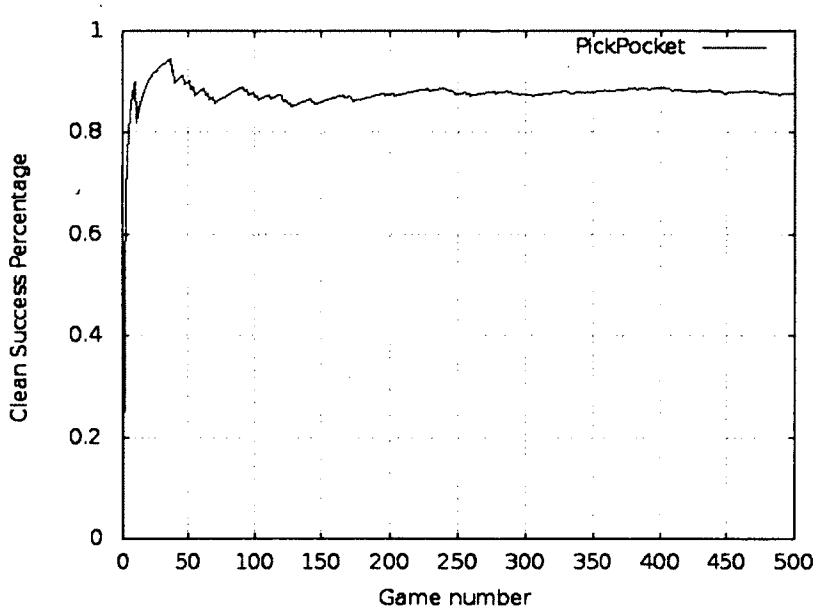

Figure 3.9: Average rate to clean off the table without missing a shot when no noise is applied on the shot parameters. Score for PoolMaster is not displayed since it is at 100% for the whole set.

the problem at hand. It means that shot planning will probably not be as important as shot evaluation no matter which method is used. This agrees with the findings in [10] which states that a better solution is not necessarily found if a deeper search is done rather than a more thorough search at the first level using the same time constraints. The robust approach, however, yielded a much better average of 82% in this case, while Pickpocket reached an average of close to 66%. Although it is too early to conclude on the efficiency of the robust method, it does show some worthy potential in low noise situations.

The final tests for the random table states were performed at a noise level of 1x. These results are shown in Figure 3.11. The non-robust approach, predictably, performed quite poorly with an average of 14%. This illustrates the fragility of the shots generated when noise is not part of the evaluation. The robust approach of PoolMaster stabilized at a rate of 64% to clean off the game, while Pickpocket remained at around 50%. A non-negligible amelioration of 50% was achieved by using

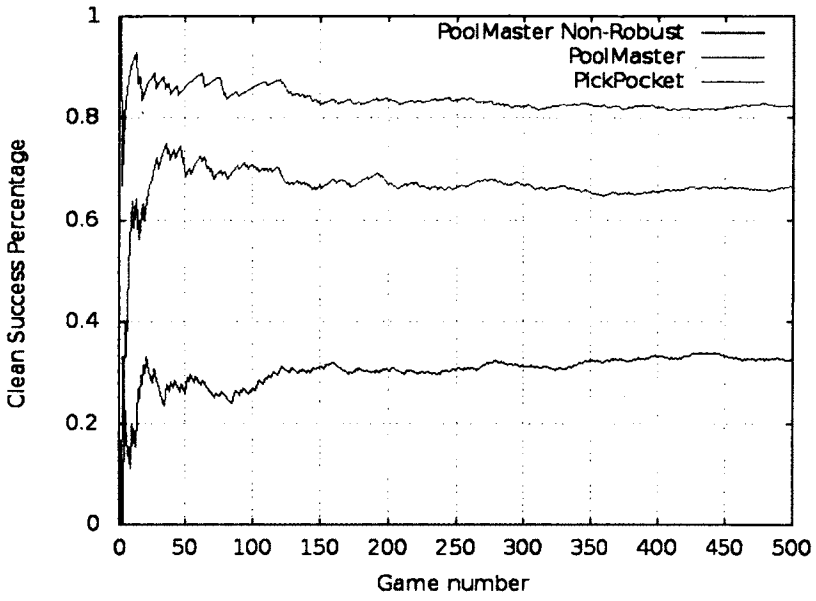

Figure 3.10: Average rate to clean off the table without missing a shot when a noise factor of 0.5 is used.

the robust model for PoolMaster. This shows great potential as any improvement to the function for position evaluation will likely provide better positioning, which at the moment doesn't take into account rail or ball proximity, thus limiting the range of possible shots importantly. The difference in success rate with Pickpocket also seems to be consistent with the 0.5x noise level, thus confirming the value of the robust approach.

If we compare these results to those obtained by the defending champion Cuecard [11], the average success rate is not as high. Cuecard was able to clean-off-the-break with success close to 70% of the time at a noise level of 1x. It is important to remember however that the set-up for tests used here is not exactly the same. In [11], each player used the same break, which was we assume was the same as in [10] (the result of extensive off-line search). In this case we completely removed the break from the equation by only using randomly generated tables. It is important to remember that since players are always constrained on aiming for stripe balls, it may be possible that in given situations, the only possible shot is highly risky, thus making a perfect score

very hard to reach unless using noiseless shots.

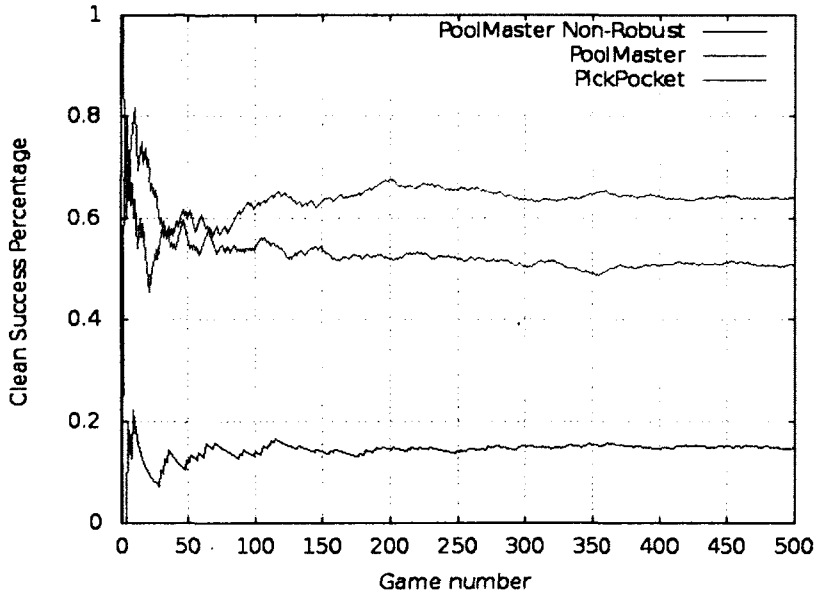

Figure 3.11: Average rate to clean off the table without missing a shot when a noise factor of 1x is used.

3.5.2 Full games

We provide here some full games between the robust version of PoolMaster and Pickpocket, in order to give a general idea of the state of the player. Since many factors come into play in a tournament like the break shots, safety shots and positions selected for ball-in-hand, these have to be taken for what they are; a general view of how the player would behave in a tournament when all its features are put to the test.

As can be seen in Figure 3.12, PoolMaster stabilizes at averages close to 75% over Pickpocket with noise levels of 0 and 0.5x. We could have expected PoolMaster to perform better when no noise was added to the shot parameters, however since both players were using their own break shots, it may have played an important part in keeping the turn after the break. Since the break shots are always different even when

no noise is added to the shot parameters, it is not guaranteed that it will succeed every time.

If we now look at the results for a noise level of 1x, the difference between the two players is not as significant, with PoolMaster finishing at an average of 64%. Once again if we refer to [10] where tests were performed against the same player, the margin we have here is not as large (74% reported for Cuecard when using a single-CPU version). However since many other factors come into play when playing full games, such as ball-in-hand behind line (see [15] for detailed rules), safeties and initial choice of breakshot, these will have to be fully tested individually.

Overall, we can infer from these tests that PoolMaster in its current state will have a tendency to perform much better at lower noise levels, also confirming some of the conclusions in [12] that a player doing no planning would be at its advantage in lower-noise situations.

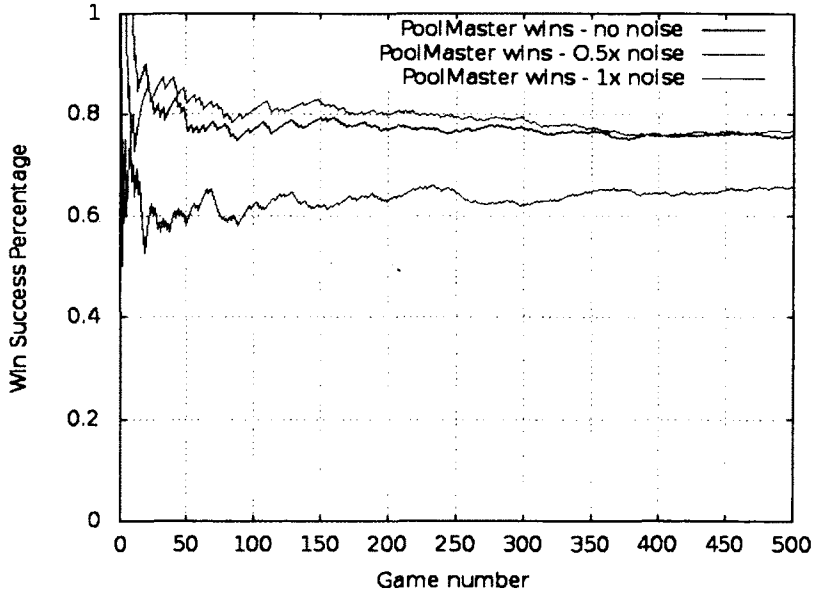

Figure 3.12: Cumulated average success rate for PoolMaster against Pickpocket, using noise levels of 0, 0.5x and 1x

3.6 Future work

The aspects discussed in this paper were mainly focused at the controller aspect of our planner. The reason for this choice is that at the moment, 8-ball was the best variant of billiards to test strategies with the upcoming 4th Computational Pool Olympiad. It was also our feeling that even though planning could be helpful in a very small number of cases in 8-ball like shown in figures 3.3 and 3.4, given the short time-limits, the advantage was offset by the time required to perform the additional computations to look-ahead while it could be better used to find a better robust shot. It is very likely however that in other variants, planning will be more challenging and require greater analysis, such that the importance of a good planner may be as great as the need for a good controller. For this reason, and to hint onto the direction of future work to complete the two-layered approach, we still provide a base model for the planner.

3.6.1 Planner

Since we possess a controller on which we can rely to provide us with feasible shots, and a given success rate, we can define a new way of discretizing our search space. If a defines an action on a table state s, and our decision variables defined by:

- b: target ball, from 1 to n,
- p: target pocket, from 1 to 6,
- c: repositioning target,

then it is possible to define the transition function as $s' = f(s, b, p, c)$.

The model defined in 2.3 remains valid with actions u now redefined for each combination of the decisions (b, p, c) for all possible shots. Thus, we find:

$$v'(s) = \begin{cases} \max_a \left[\int_S v(\cdot) dp(\cdot|s,a)\right] & \text{if } \lambda(s) = 1 \\ \min_a \left[\int_S v(\cdot) dp(\cdot|s,a)\right] & \text{if } \lambda(s) = 2 \end{cases}$$

with actions a now redefined. The set of actions A is now partitioned in a subset of $\{b, p, c\}$ combinations for all possible shots. Thus, we find ourselves with a shot sequence represented by:

$$\left\{\begin{array}{c} b_1 \\ p_1 \\ c_1 =? \end{array}\right\} \rightarrow \left\{\begin{array}{c} b_2 \\ p_2 \\ c_2 =? \end{array}\right\} \rightarrow \ldots \rightarrow \left\{\begin{array}{c} b_n \\ p_n \\ c_n =? \end{array}\right\}$$

where we know the target balls and target pockets, but wish to determine the repositioning targets to reach our objectives on a n shot horizon.

3.6.1.1 Shot sequence planning

To solve model (3.2.3), it is first necessary to proceed to a few simplifications. Indeed, it is clearly impossible to do a complete search of the min-max for the game of billiards. If we take the game of chess for example, the complete sequence of shots is directly dependent on our opponent. It means the structure of the game forces the turns to be alternated between both players, and as such it is mandatory to follow a strategy in which we will minimize the possible value of the shot chosen by the opponent. With billiards, however, this kind of situation remains a special case; a player that makes no mistakes will have the possibility to finish the game without his opponent even doing a single shot. Thus there exists strictly two situations in which the adversary will gain his turn:

- A defensive shot.
- A missed shot.

Since the global aspect of the model is already about minimizing the chances of missing a shot, we still need to find a way to assess defensive shots.

A defensive shot, optimally, would normally correspond to a computation by which a player would decide that it is more advantageous to let his opponent play in a bad position, hoping to regain control of the game afterwards, rather than play offensively. Since billiards is a continuous game in nature, the infinity of shots available to the opponent is already hard enough to predict without also trying to predict the case where the opponent would play more than one shot without losing his turn. It is why using this model, we will restrict our search to successful defensive shot to a horizon 1. This means that a defensive shot leading to a situation where the opponent has any possible shot at all will be defined as a failed defensive shot.

Thus when planning our shot sequence we'll have to deal with the fact that we try to maximise our chances to keep playing until the end, with the special case

of possibly playing defensively with a horizon of 1 shot. Assuming that we know the order in which we will execute the shots on the table, then the problem can be resumed as a simple problem of finding the repositioning targets.

If we define the following variables:

- $b(u)$: target ball position after fixing the shot parameters u
- p: Pocket for the given target ball.

If we consider u_n as the shot parameters of our n^{th} shot, we can minimize function f as:

$$\begin{aligned} \min f(u_0, u_1, ..., u_n) = \text{Posval}(c_0, c_1, ...c_n) \ \forall t : 0...n \\ \text{s.t. } \|b_t(u_t) - p_t\| = 0, t = 1, ..., n \end{aligned} \tag{3.6}$$

where the function f would correspond to the position value heuristic described in [ ], which associates a value with a position relative to the available shots at this position(depending directly on the position reached c_n).

By using this model, we wish to search for the repositioning targets $c_1, c_2, ...c_n$ to reach after each shot on the table, as can be seen on Figure 3.13.

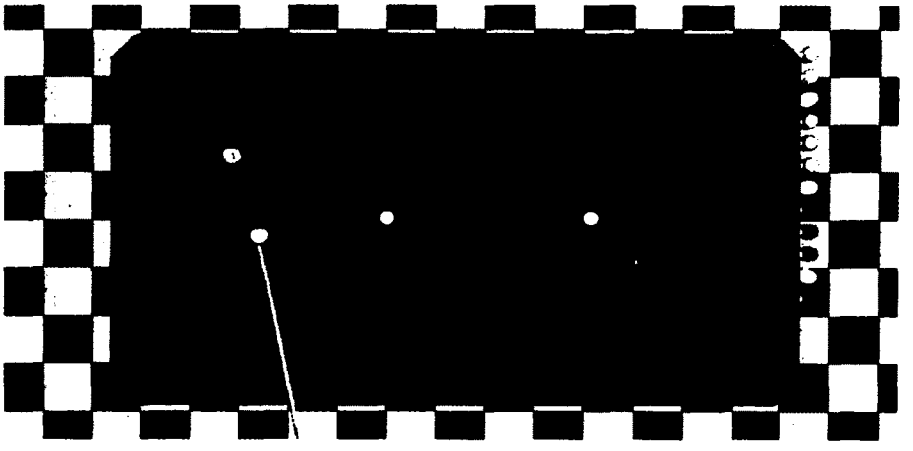

Figure 3.13: Shows sequence determined by the repositioning targets c_1, c_2, c_3.

Thus, by finding the solution of (4.1), we will find a list of targets to reach to maximise our chances of shot success. However, we should not forget that this is a very simplified version of the problem. We ignore for the moment the dynamical aspect of billiards, which makes it very hard to define targets $c_1, c_2, ...c_n$ without having an impact on the rest of the system. When optimizing a shot, it is possible that the

cue ball enters in contact with more than one ball, thus changing the global table state and changing the value of the solution we were hoping to find. It illustrates the convenience the robust version of the problem where we aim for a given zone instead of a specific target.

3.7 Conclusion

The aspects which are described in this paper represent the tools we developed to obtain a fully functional player, with all of the abilities observable in a professional player. We have shown the potential of a using multi-objective model to achieve any desired shots on the table, so that the controller used can easily be applied to any forms of billiards games. We have also shown the effectiveness of using a robust model to only generate shots that will succeed under noise-induced parameters. Our hope is that by carefully using these tools with proper planning, it may be possible to create a decision-making AI able to challenge world-class professional players. A lot of work remains to be done on the analysis of safety shots versus offensive shots, but these are also very much game-dependent, and will need to be adjusted accordingly. The planning aspect for the sequence of shots to be chosen is also likely to be a lot more important when other variants than 8-ball will be played. As our test results showed for 8-ball, a success rate of 100% when no noise is added to the parameters means that it is easily possible to find a solution which leads to victory. Thus it is only a matter of making these solutions robust to noise to create a stronger player. Some situations like the one shown in section 3.3 will still require planning in the game of 8-ball, but their occurrence seems to be very rare. It also motivates us to explore other games variants in future work like Straight pool, where a player has to call each of his shots, even on the break, making the game planning and multi-objective shots much more important. It will also be interesting to test the approaches described here in a game like Carambole, where less planning is required but better shot exploration is needed, thus possibly taking advantage of the technical skills of the player.

3.8 Compléments

3.8.1 Valeur d'une position sur la table

Dans l'article décrit dans cette section, il est précisé que pour la formulation robuste du problème, il n'est plus nécessaire de prendre en compte la distance avec une position voulue sur la table, mais plutôt la valeur de la position de façon générale. Ce changement permet entre autre de relaxer la fonction que nous tentons de minimiser, de façon à trouver une solution robuste non plus pour la distance avec la cible de reposition visée, mais pour la valeur finale de la position atteinte. Cependant quelques détails ont été omis quant à la description de cette fonction dénomée $Posval()$. La valeur cette fonction comme utilisée dans la section 3.4.2.1 correspond à un estimé de la valeur du coup pour une position donnée de la blanche et une combinaison bille-poche pour le prochain coup comme présenté dans l'article [38].

Plus clairement, pour les coordonnées (en x, y) de la position de la blanche sur la table, la valeur d'une combinaison bille-poche avec une poche au coin sera défini par: $Posval(x, y) = \kappa_c$, où

$$\kappa_c = \frac{\cos\alpha}{v_{op}v_{co}}. \tag{3.7}$$

Pour les coups aux centres, $Posval(x, y) = \kappa_s$ où:

$$\kappa_s = \frac{\cos\alpha \ \cos\gamma}{v_{op}v_{co}}. \tag{3.8}$$

v_{op} représente la distance entre la bille visée et la poche, v_{op} la distance entre la blanche et la bille visée, et α l'angle de coupe pour réussir le coup au centre de la poche.

Le paramètre γ est introduit pour l'équation des coups aux poches du centre pour prendre en compte le fait qu'un coup en ligne directe avec la poche est plus facile qu'un coup en angle. γ va représenter l'angle entre la bille objet et l'axe $\mathbf{x}$ de la poche, et sera défini: $\cos\gamma = \hat{\mathbf{v}}_{op}.\hat{\mathbf{x}}$ avec $\hat{\mathbf{x}} = \mathbf{x} = [1, 0, 0]$ et $\hat{\mathbf{v}}_{op}$ le vecteur entre la position de la bille objet et celle de la poche. La description détaillée de cette fonction ainsi que la démonstration de sa correspondance à la valeur réelle de la difficulté d'un coup est disponible dans [38].

3.8.2 Repositionnement double

Nous avons exploré dans les chapitres précédents la modélisation d'une fonction objectif permettant un repositionnement très précis sur la table. Cet outil permet non seulement d'imiter les capacités techniques d'un joueur professionnel, mais également d'effectuer des coups assez spectaculaires pour se sortir des impasses. Cependant, même si ce type d'approche est très efficace dans le cas déterministe, les résultats sont en revanche assez décevants lorsqu'on perturbe les paramètres du coup en y ajoutant un bruit gaussien. En effet, lorsque nous décomposons les éléments essentiels d'un coup, deux aspects critiques sont identifiables. Le premier correspond à la capacité d'empocher la bille visée avec la plus grande précision. C'est-à-dire, de trouver la vitesse et l'angle de frappe permettant de réussir le coup à chaque fois même sous l'influence du bruit. Le deuxième aspect correspond aussi à la robustesse du coup mais cette fois-ci après l'impact avec la bille visée, afin d'atteindre une position favorable pour le prochain coup, et pour assurer que celui-ci sera réalisable sans trop de difficulté. Il faut donc s'assurer que la zone que nous tentons d'atteindre permettra facilement d'accéder à un ou plusieurs autres coups, afin de ne jamais atteindre une impasse. Bien sûr, ce deuxième aspect est en grande partie dépendant d'une planification qui pourra déterminer la séquence optimale de coups à faire.

Dans cette optique et en continuant avec le modèle précédent du chapitre 3, il serait possible de créer un modèle effectuant non seulement une optimisation sur les paramètres du premier coup, mais également du deuxième coup, afin de ne plus reposer l'optimisation sur la valeur de la reposition pour le prochain coup, mais plutôt sur la valeur de la position pour deux coups à l'avance, en ayant comme contrainte de réussir les deux premiers coups. De cette façon il est facile de voir qu'une planification sur plusieurs coups serait possible, tout en maximisant les chances de réussite.

Pour y arriver, on reprends le modèle précédent:

$$\begin{aligned} \min_{u} f(u) = max(\text{Posval}(u,\zeta) - \text{Val}_{\min}, 0) + \\ \rho||b_n(u,\zeta) - p||_\infty \ \forall\zeta \end{aligned} \tag{3.9}$$

où *Posval* représente la valeur de la position finale de la bille en fonction des paramètres du coup initial correspondant au vecteur u, et $\text{Val}_{\min}$ représente une

valeur minimale acceptable pour la position. La deuxième expression, $||b_n(u,\zeta) - p||$ représente la distance entre la bille visée b_n et la poche visée p. Le tout, pour l'ensemble des paramètres ζ pouvant être générés lors de l'ajout de bruit gaussien.

Comme nous ne possédons pas directement l'ensemble continu ζ lors de nos calculs, il nous est nécessaire d'estimer celui-ci en faisant une discrétisation des paramètres possibles et en échantillonnant plusieurs fois le résultat de chacun de ces paramètres.

En pratique, les règles du jeu de la 8 nous forcent à rajouter deux paramètres à cette expression afin d'éviter de perdre le tour. En effet, pour qu'un coup soit considéré comme légal et pour ne pas perdre le tour, la bille de choc doit tout d'abord entrer en impact avec une des billes appartenant au joueur (si celles-ci sont déjà attribuées). Une fonction représentant la distance entre la bille frappée lors du premier impacte et la bille visée permet de prendre en compte ce cas particulier.

Le deuxième paramètre devant être ajouté sert à empêcher d'empocher la bille de choc après l'exécution du coup. Celui-ci représentera la vitesse finale avec laquelle la bille de choc s'est empochée, afin qu'elle soit minimisée pour devenir nulle, représentant donc une position sur la table et non pas dans la poche.

Pour rajouter un deuxième coup à notre évaluation, nous avons seulement besoin d'ajouter les variables nécessaires au vecteur u pour effectuer le deuxième coup.

Dans le cas le plus simple, ceux-ci sont définis par: $u = [a, b, \theta, v]$ avec le 5^e paramètre (ϕ) précisé avec un simple calcul géométrique suivant la règle des 90 degrés (voir [5]). Nous allons donc maintenant optimiser le vecteur $u = [a, b, \theta, v, a_2, b_2, \theta_2, v_2]$, où a_2, b_2, θ_2, v_2 seront les paramètres du deuxième coup.

La nouvelle fonction correspond maintenant à :

$$\begin{aligned} \min_u f(u) = \rho_1 ||b_n(u,\zeta) - p_1||_\infty + \rho_2 ||b_n(u,\zeta) - p_2||_\infty + \\ max(\text{Posval}(u,\zeta) - \text{Val}_{\min}, 0)\ \forall \zeta \end{aligned} \tag{3.10}$$

où $\rho_2 ||b_n(u,\zeta) - p_2||_\infty$ représente la distance entre la bille visée pour le deuxième coup et la poche. Cependant comme cette fonction est maintenant beaucoup plus complexe à optimiser et comporte deux partie distinctes (et donc une discontinuité), une recherche sera tout d'abord effectuée à l'aide du modèle inital en 3.9 afin de trouver un bon point de départ réalisable.

3.8.2.1 Estimation de ζ satisfiant les contraintes

Pour le moment, comme il n'est pas possible d'obtenir directement les valeurs de l'ensemble ζ, nous procédons à un échantillonnage pour chaque évaluation de la fonction $f(x)$. Le seul problème associé à cette approche est bien sûr que la qualité de l'évaluation dépendra directement du nombre d'échantillons utilisé. Cependant il est envisageable qu'une analyse géométrique plus poussée avant de faire un coup pourrait peut-être finir par remplacer cette méthode. Il faut également noter qu'un autre inconvénient au niveau de l'implémentation de cette méthode est d'associer une valeur lorsqu'un coup n'est pas valide. C'est à dire qu'il est possible que lors de l'évaluation de la fonction objectif pour un coup donné, les paramètres utilisés pour le coup représentent un coup physiquement impossible. La plupart de ces cas sont évités en spécifiant des contraintes sur les paramètres du coup comme par exemple une vitesse maximale de 4.5m/s ou un interval allant de -14 à 14mm pour l'élévation de la queue par rapport au centre de la bille. Cependant aucune contrainte ne modélise directement l'intéraction physique avec la table, donc si la queue entre en collision avec celle-ci la valeur du coup est élevée à un coefficient minimum spécifié au préalable.

3.8.2.2 Échecs des tests

Les premiers tests ont permis de vérifier qu'il y avait en fait très peu de potentiel pour ce type de modèle. En effet, en générant quelques tables aléatoires et en lançant plusieurs fois l'optimisation, il est possible de remarquer que les résultats diffèrent de façon très importante à chaque fois. Il n'y a donc pratiquement aucune stabilité/robustesse par rapport à la solution trouvée. Cela peut s'expliquer par le fait que lors de l'optimisation, même si le point de départ est en fait une solution initiale permettant d'empocher la première bille, les paramètres de ce coup vont certainement varier, en même temps que les paramètres du deuxième coup. Il est facile d'imaginer en plus qu'en modélisant le système de façon robuste, à chaque fois que le premier coup est simulé, l'état résultant pour le prochain coup sera différent. Comme nous ne calculons pas un point de départ réalisable à chaque fois pour le deuxième coup aussi, celui-ci aura beaucoup moins de chance d'être réussi. L'angle de la queue, ϕ, est également fixé pour les deux coups en utilsant la règle des 90 degrés. Cela fonctionne

bien lorsqu'on tente de calculer un coup, mais lorsqu'on optimise sur deux coups, le deuxième coup aura un angle phi qui changera constamment, et donc la trajectoire de la bille changera grandement même si on optimise pas les paramètres de ce deuxième coup.

3.8.2.3 Optimisation à deux-phases

Ceci nous force à considérer d'autres alternatives pour la planification de plus d'un coup à l'avance. Comme mentionné auparavant, un des problèmes est dû au fait que les paramètres du deuxième coup dépendent directement du premier coup, et ceux-ci sont soumis à l'évaluation robuste (avec l'ensemble ζ). Pour réellement optimiser deux coups, il faudrait en fait que les paramètres du premier coup soient fixés lorsqu'on optimise le deuxième coup, ce qui correspondrait en fait à utiliser une fonction objectif composée de deux parties. Ce type de problème est en fait déjà profondément étudié dans la littérature sous forme de problème à deux-phases (ou stages) avec recours (voir [34]). On peut plus facilement illustrer cette approche à l'aide d'un arbre décrivant les différents scénarios pouvant être atteints. Chaque scénario (une infinité dans le cas du billard) représente l'exécution des paramètres du contrôle (le coup) avec le bruit ajouté. L'arbre des scénarios représentera donc l'ensemble de tous les scénarios atteignables. La profondeur de l'arbre représentera l'exécution de chaque coup séparé.

On peut donc représenter le problème sous la forme suivante (tiré de [39]) :

$$\begin{aligned} \min_u f(u) + E[h(u,\zeta)] \\ \text{où } f(u) = \rho||b_n(u,\zeta) - p||_\infty \\ \text{et } h(u,\zeta) = max(\text{Posval}(u,\zeta) - \text{Val}_{\min}, 0) + \rho_2||b_n(u,\zeta) - p||_\infty \forall \zeta \end{aligned} \tag{3.11}$$

On considère ici le terme $h(u,\zeta)$ comme étant dépendant de l'exécution du premier coup représenté par $f(u)$ avec les paramètres u. Ce terme corresponds donc à l'exécution du deuxième coup, une fois les paramètres du premier coup appliqués. On peut aussi remarquer que dans cette nouvelle formulation, on ne tente plus de minimiser la valeur de la fonction *Posval* pour le premier coup. Comme on tente de

planifier deux coups à l'avance, il n"est plus nécessaire de trouver la meilleure position possible pour se repositionner, car la position trouvée sera automatiquement celle qui maximise la chance d'un deuxième coup intéressant.

3.8.2.4 Variantes

Il est possible d'approcher le modèle décrit par l'équation 3.11 de diverses façons. En effet, même la recherche d'une solution robuste en utilisant le modèle initial 3.9 ne garanti pas toujours une solution. Lors d'un coup difficile par exemple, il est très possible que le taux de succès soit assez bas, rendant donc la planification inutile. Une solution possible est de commencer par la recherche d'une solution robuste au modèle initial, pour ensuite partir de cette solution pour faire une recherche avec un modèle à deux étapes si celle-ci est assez bonne ou si notre $f(x) = 0$.

Un autre aspect sur lequel il est possible de diverger est de simplifier la recherche du deuxième coup en n'optimisant pas de façon robuste. En effet, comme il est déjà assez difficile d'obtenir une solution robuste pour un coup, le faire pour deux coup devient assez ambitieux, et n'aboutira probablement pas dans la plupart des cas. Il est aussi important de noter qu'en optimisant de façon robuste pour deux coups, on augmente exponentiellement le temps de calcul en fonction du nombre d'échantillons utilisé pour les évaluations de $f(x)$.

Finalement, un problème se présente aussi quant à l'évaluation de $h(u, \zeta)$. Comme pour le modèle initial, $h(u, \zeta)$ est en fait une valeur qui représente un min-max où nous tentons de minimiser le pire cas possible pour chaque évaluation de la fonction objectif. Cependant, comme nous prenons la moyenne des valeurs de $f(x)$, il est possible de se retrouver avec une solution privilégiant une reposition très bonne pour un seul coup tandis que les autres coups sont ratés. Une solution possible serait de garder un compteur des coups évalués comme étant des succès (par exemple un coup qui empoche la bille avec une certitude de 95%), pour ensuite minimiser cette valeur plutôt que la moyenne.

Il existe aussi plusieurs autres facteurs qui peuvent influencer le résultat final d'une partie en utilisant le modèle proposé. Parmi ceux-ci, les plus signifiants sont la taille de l'échantillon pour chaque évaluation de $f(x)$, le seuil minimum pour Val_{min}, et finalement le poids accordé à ρ et ρ_2.

3.8.3 Tests

Des tests ont été effectués en utilisant l'optimisation à deux-phases décrite ci-haut. Comme les résultats n'étaient pas concluants, ce modèle n'a pas fait l'objet d'une analyse plus poussée. Nous avons quand même choisi d'en parler car il est possible que de meilleurs résultats seraient obtenus en appliquant la solution à différentes formes de jeux.

Afin de déterminer la stabilité des solutions trouvées en utilisant les méthodes robustes et non-robustes pour la planification à deux étapes, nous avons fait quelques tests sur un état de table particulier où le coup calculé lors d'une partie a été manqué. Le résultat est assez surprenant, car il semblerait que même en optimisant le même coup à 5 reprises, la solution trouvée varie à chaque fois. Bien sûr, si il est très possible qu'il existe plus d'une solution pour un état de table donné, mais dans le cas illustré plus bas, la valeur des solutions varie de façon assez importante, de même que la valeur de la fonction objectif $f(x)$.

Dans la figure 3.14, on peut voir que la solution trouvée varie à chaque fois. Il est possible qu'il existe plus d'une solution, mais dans ce cas les solutions trouvées sont loin d'avoir la même valeur, donc il y a un problème au niveau de la stabilité. Dans le cas où on optimise seulement de façon robuste sans planification, la solution semble plus stable, ce qui porte à croire que le problème est peut-être dans la façon de résoudre le modèle à deux-phases.

Cela a quand même apporté une motivation pour faire quelques tests dans une autre situation (figure 3.16), qui démontre encore plus l'instabilité, même avec un échantillonnage de 50 coups pour le bruit ajouté aux paramètres.

3.8.4 Analyse de la fonction objectif

Afin d'illustrer de façon concrète le type de fonction que nous tentons de minimiser, il est possible de fixer quelques paramètres et d'en faire varier seulement un ou deux. En discrétisant ce paramètre choisi, il est ensuite possible d'en tracer une fonction et donc de visualiser ce que nous tentons de minimiser. C'est à partir de l'analyse de ces graphes qu'il a été possible de déterminer un nouveau modèle d'optimisation robuste. En effet, il semblerait que l'optimisation robuste utilisant le modèle décrit

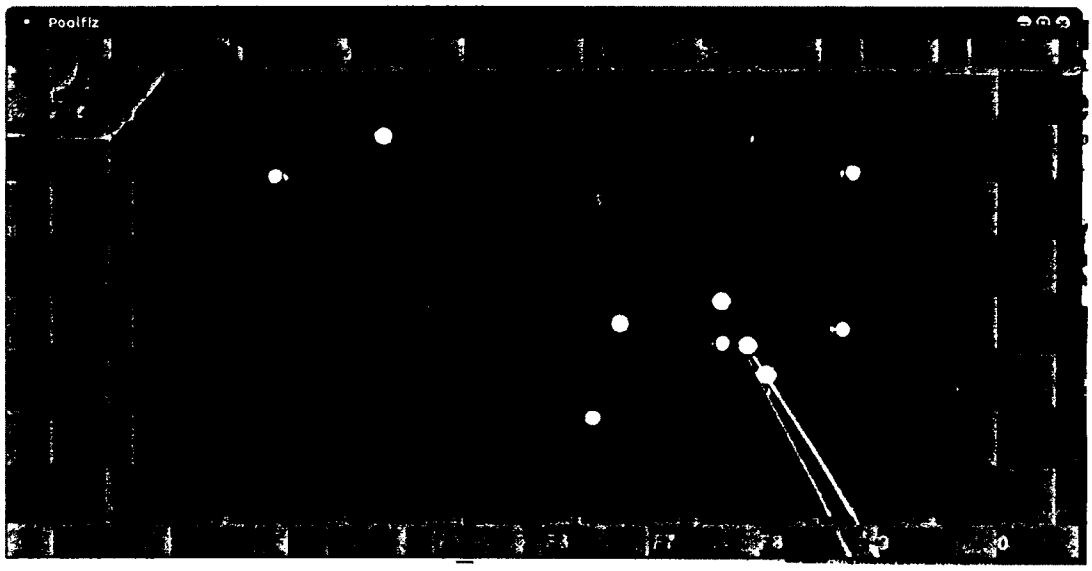

Figure 3.14: Coup optimisé à deux-phases, avec le deuxieme coup optimisé sans bruit. La valeur du deuxieme coup est donc la moyenne des évaluations du f(x) pour ce coup. Chaque évaluation de f(x) fait appel à 50 échantillons de bruit différents.

dans le chapitre 3 soit assez chaotique. On peut voir un exemple dans la figure 3.17. Pour générer cette figure, nous avons tout d'abord fixé l'ensemble des paramètres d'un coup et seulement fait varier la vitesse, que nous avons discrétisé entre 0.3m/s et 4.5m/s. La fonction tracée correspond à la fonction d'évaluation décrite dans le modèle robuste.

À première vue dans la figure 3.17 il est difficile d'en tirer une conclusion. Il semblerait que pour tous les coups ayant une vitesse supérieure à 1.2m/s la valeur de la fonction tombe à l'infini. Cependant en regardant la table dans la figure 3.18 on peut plus facilement expliquer ce comportement. En effet il semblerait qu'à plus grande vitesse, la bille blanche termine sa trajectoire dans la poche en haut au coin. Cela est donc peu intéressant pour nous. Cependant si nous regardons de plus proche le graphe en agrandissant plusieurs fois dans la figure 3.19, on voit maintenant beaucoup mieux les variations dans la fonction à optimiser.

Cela n'est pas très encourageant, car on voit une très grande variation d'une vitesse à l'autre. C'est pourquoi nous avons décidé d'améliorer le contrôleur robuste dans le chapitre 4. En utilisant les dérivées pour étudier la sensibilité de la reposition par

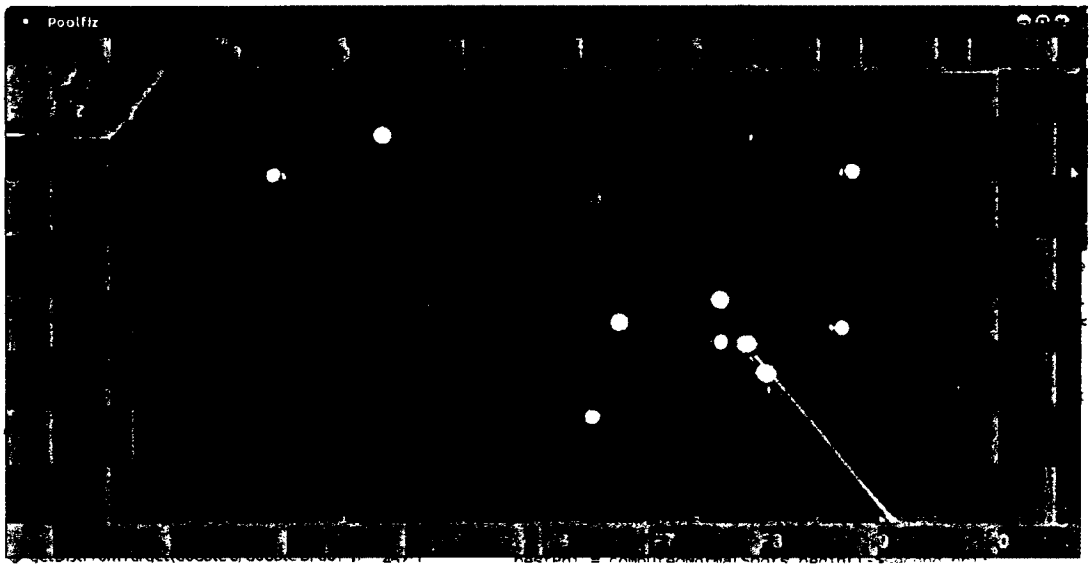

Figure 3.15: Coup optimisé de façon robuste sans planification.

rapport aux paramètres, on arrive à minimiser une fonction un peu moins chaotique que lorsqu'on échantillonne à chaque évaluation de f(x). Une autre idée correspond à pénaliser la valeur de la fonction f(x) par rapport à sa distance avec les paramètres les plus proches donnant une valeur qui tends vers l'infini. En utilisant une telle approche, lors de l'optimisation, on favorisera un coup qui demeure dans une zone avec une distance acceptable d'un coup raté.

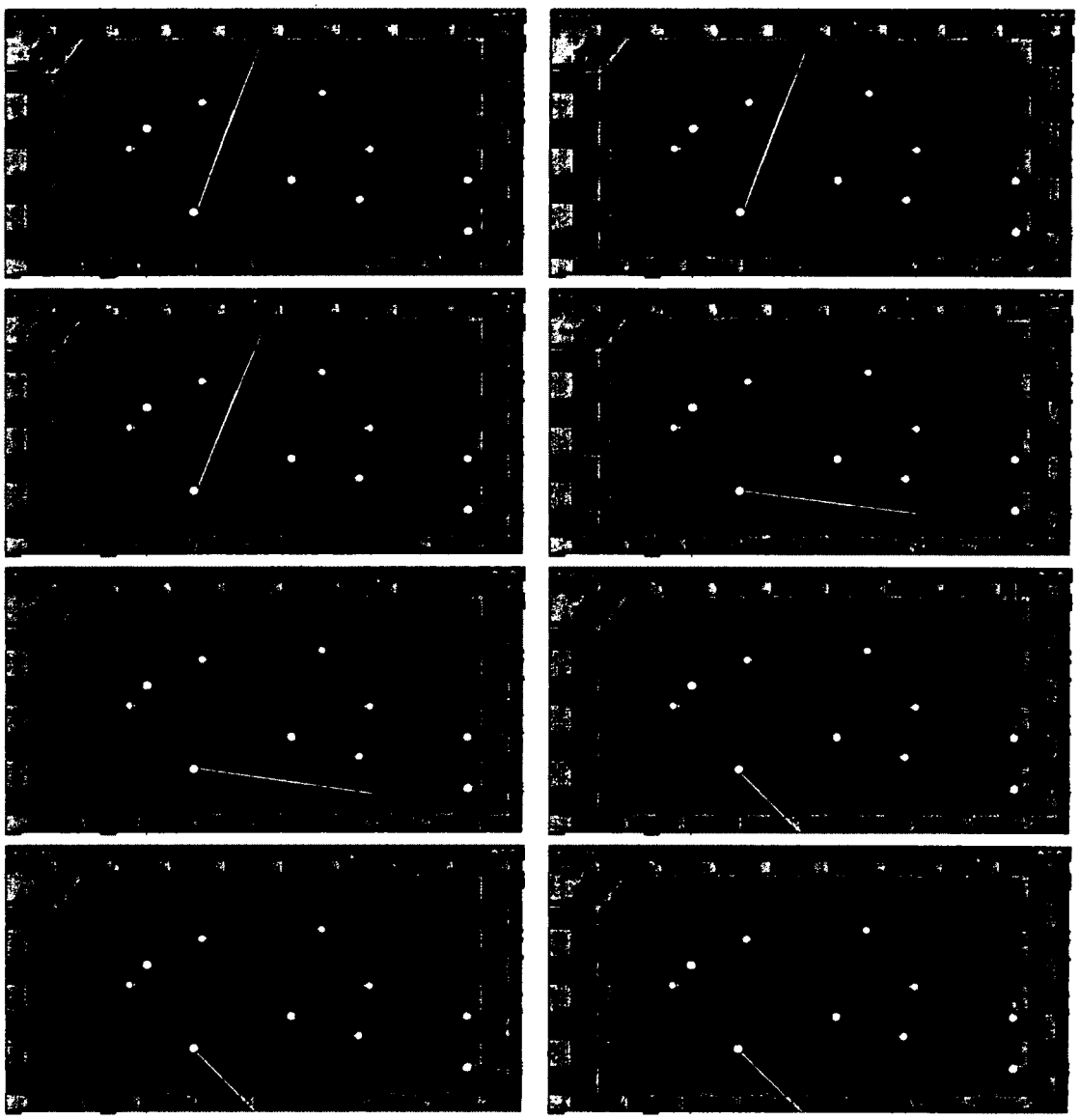

Figure 3.16: Huit exemples de coups différents résultant de huit appels consécutifs à l'optimiseur. On peut observer une très grande variabilité dans la valeur de la solution trouvée. Ce n'est qu'en effectuant un échantillon de 500 évaluations pour chaque $f(x)$ que la même solution stable fût trouvée à chaque fois (donc la solution existe).

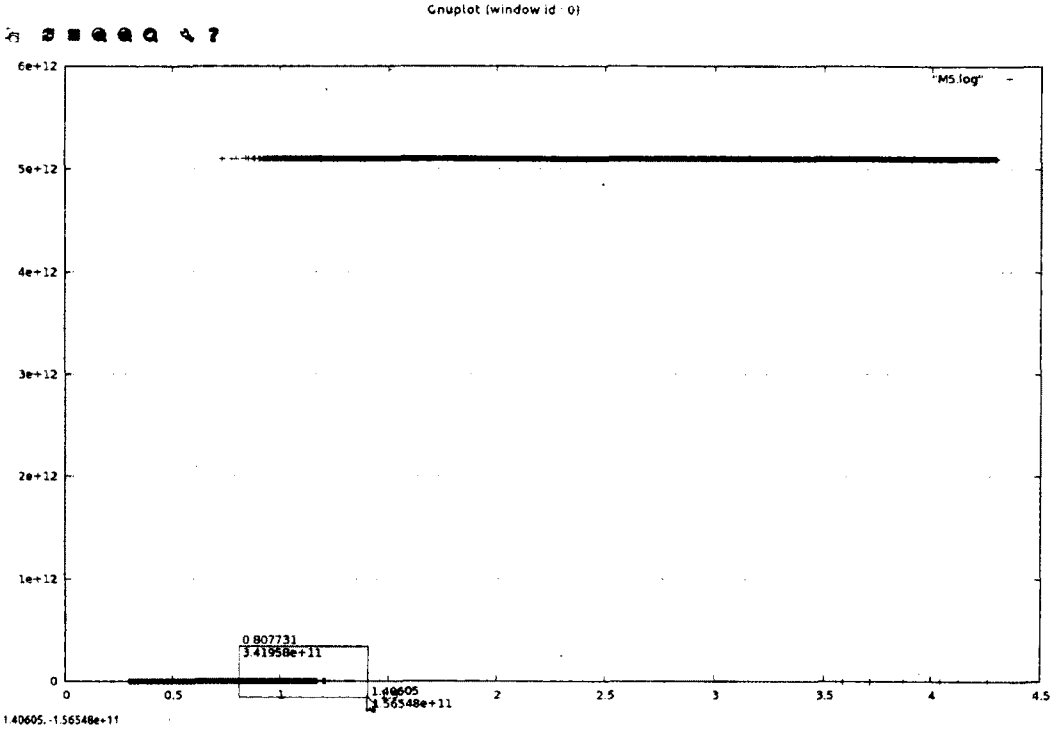

Figure 3.17: On peut observer un graphe de la vitesse (X) en fonction de la valeur de l'évaluation de la fonction (Y). La vitesse est variée entre 0.3m/s et 4.5m/s.

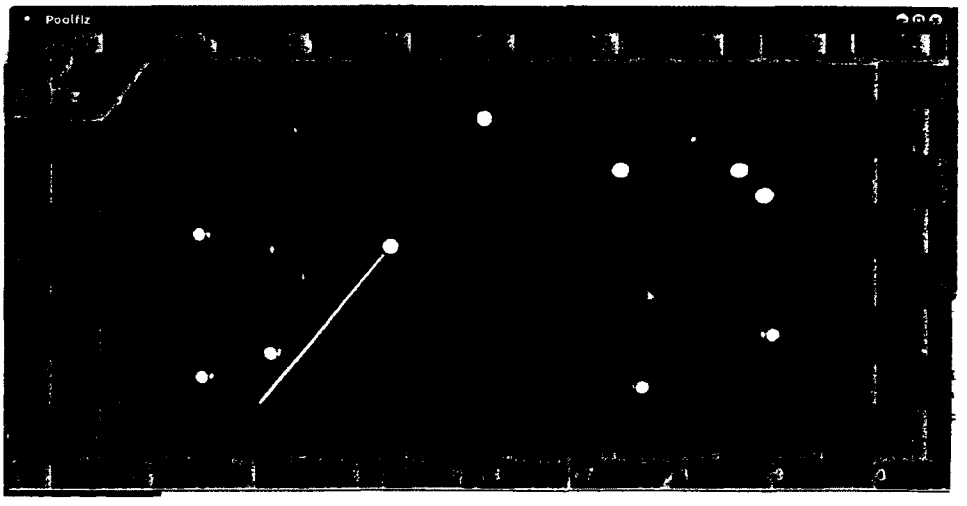

Figure 3.18: Le coup utilisé pour tracer le graphe de la figure 3.17

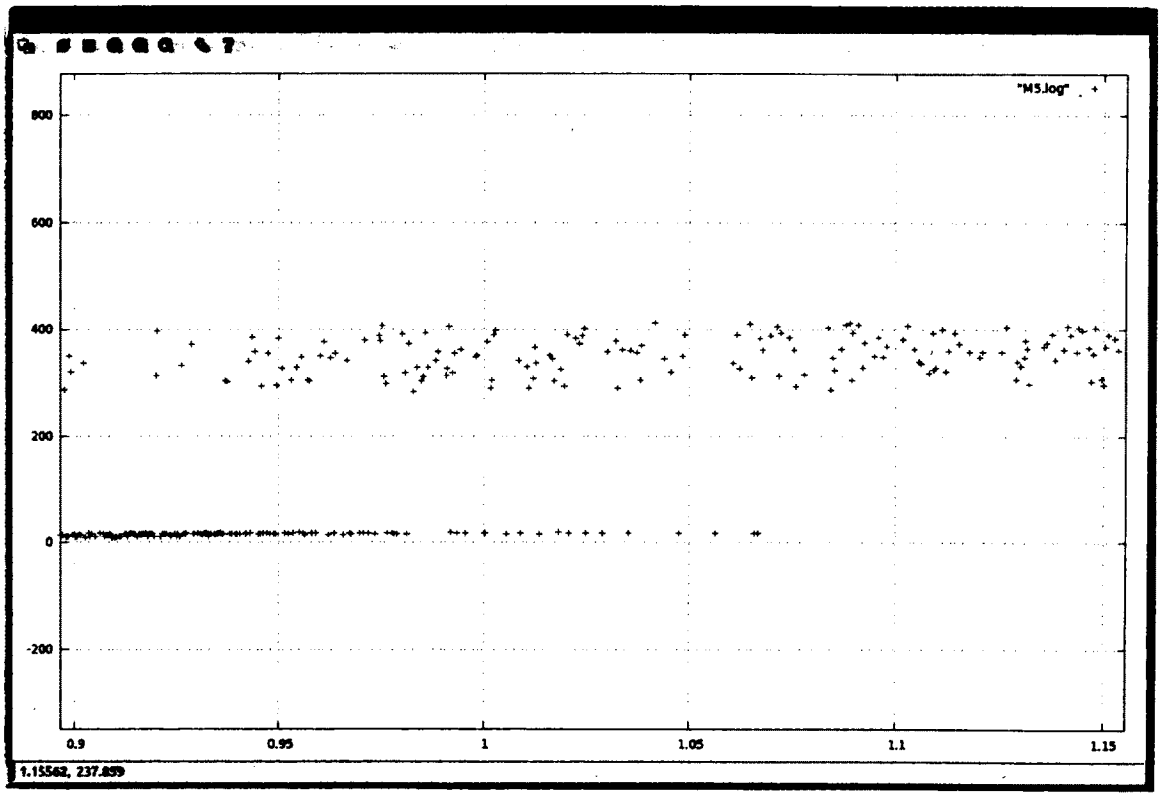

Figure 3.19: Vue rapprochée de la figure 3.17.

Chapitre 4

Un planificateur heuristique pour une approche à deux-niveaux appliqué du jeu du billard

Résumé

Plusieurs difficultés peuvent surgir lorsqu'il est question de générer des plans optimaux dans un domaine avec états et actions continus. De telles difficultés sont encore plus importantes lorsque le domaine est soumis à des simulations stochastiques. Le jeu du billard regroupe tous ces défis et bien plus, et a fait l'objet de plusieurs études dans les dernières années.

Ce que nous proposons dans ce travail est le développement d'une approche de planification dans un système à deux-couches, où un contrôleur optimal très précis sera guidé par un planificateur de haut-niveau afin de générer les plans optimaux d'un point de vue global étant donné quelconque état initial. Nous allons tout d'abord introduire le modèle général pour cette classe particulière de problèmes, et ensuite proposer quelques heuristiques afin de générer des plans et de résoudre le modèle en un temps raisonnable. Quelques améliorations au modèle du contrôleur optimal robuste pour le billard seront aussi présentées dans le but d'améliorer l'optimisation des coups au premier niveau.

Commentaires

Cet article à été soumis à la revue "IEEE Transactions on Computational Intelligence and AI in Games" le 24 septembre 2012, et est en attente pour évaluation par un comité de révision.

Pour ce dernier article, la part de l'étudiant consiste aussi en une contribution majeure concernant la rédaction, le développement des méthodes proposées ainsi que les résultats présentés.

L'apport essentiel de cet article par rapport à la thèse est de finalement venir conclure les développements de l'approche à deux couches présentés auparavant. Ce dernier chapitre est essentiel étant donné qu'il vient finaliser et conclure les éléments qui avaient été mentionnés dans le chapitres 3 mais n'avaient pas été approfondis.

Plus précisément, un apport important est fait au niveau de l'aspect planification de la problématique. Plusieurs heuristiques sont proposées afin d'arriver à démontrer le concept de planification en deux-niveau, pour pouvoir ainsi faire une contribution importante au domaine. On y présente aussi des améliorations non-négligeables au contrôleur robuste, illustrant encore une fois l'ensemble des gains possible avec un peu plus d'analyse par rapport au domaine étudié.

Gardant le même format que l'article précédent, celui-ci permet aussi d'effectuer une comparaison directe et de constater d'importantes améliorations.

A heuristic-based planner for a two-layered approach applied to the game of billiards

Jean-Pierre Dussault, Jean-François Landry
Département d'informatique, Université de Sherbrooke,
Sherbrooke, Québec, Canada J1K 2R1
Jean-Pierre.Dussault@usherbrooke.ca,
Jean-Francois.Landry2@usherbrooke.ca

Philippe Mahey
ISIMA - Campus des Cézeaux, BP 10125, 63173 Aubière CEDEX, France.
Philippe.Mahey@Isima.fr

Keywords: Pool, Billiards, Snooker, Game, Simulation, Planning, Strategy, Optimization, Artificial Intelligence

Abstract

Several difficulties can be encountered when trying to generate optimal plans in a domain with continuous action and state space. Such difficulties are even greater when the domain is subject to stochastic simulations. The game of billiards combines all of these challenges and more, and has been the focus of many studies in the past few years. What we propose in this paper is the development of a planner for a two-layered approach, where a precise optimal controller will be guided by a high-level planner to generate the best plan possible given any random initial state. We will first introduce the general model for this particular class of problems, and then propose several heuristics to generate plans to solve the model in a timely manner. Several improvements to the optimal robust controller for billiards will also be presented in order to improve single shot optimizations.

4.1 Introduction

The game of billiards is a game of skill, tactics and ingenuity. It been played all over the world for countless years and is even officially recognized as an Olympic sport by the International Olympic Committee. It has also been the focus of many recent projects in the research community. The engineering challenge to create a robot precise enough to compete and even win against the best professional players is a very strong motivation for the field of robotics. Many projects have been developed from the mid 1990s to today, (see [31], [45], [21], [19], [8], [56] [7], [50]) with approaches ranging from a robotic arm set-up on a gantry over a table to a standalone robot moving freely around. Another interesting aspect which was investigated was the usage of augmented reality tools to help guide and teach new players how to play the game or better select shots on the table. Efforts have been made in [40] and [33] with interesting results. Some work has even been done in video analysis in [25] to extract information from tournaments played on television to get game summaries.

Another aspect, the focus of this paper, concerns the intelligence required to plan and select the best shot sequences on the table, given a complete physical simulator of the game. A lot of work has also been done in this regard. The usage of fuzzy logic has been documented in [22] as a way to evaluate shot difficulty on a given table state. Bayesian network models have also been proposed in [52] in the same context. Monte-Carlo sampling approaches were used in [58] to not only evaluate shot difficulty but generate possible plans. Finally, least-square optimization approaches were described in [38] as an efficient approach for position-play.

When playing the game of billiards as amateurs, we are usually confronted by one key problem: cue-ball control. Although we know which shot we are aiming for, executing it is a completely different matter. It is a matter of skill, technical precision, and mechanical physics. We can usually estimate the trajectory of the cue-ball given an average cue-strike but it takes years of practice to perfectly master the shot execution, and to be able to gauge which shot we can successfully complete. Thus for an amateur, the best shot is almost always the easiest one. This makes the game very interesting since as we progress, we are able to execute harder shots leaving us with a new array of possible shots which opens up a new level to the game

and adds in the complexity of planning. This is also the area where we see the pro's come in. Having better cue-control and position play they can envision more than one shot ahead, to be able to select the shot which may be harder to execute but yields a better chance of finishing the table.

In [39], we previously explored the necessary elements to build a robust controller capable of achieving a high-level of success for completing randomly-generated table states. What we propose in this paper is to complete the two-layered approach which was suggested, by expanding on the description of high-level planner that will establish a preferred shot sequence to maximize chances of finishing the game with success. As a complement to the proposed planner, we will also show some noticeable improvements to the robust controller. Finally we will discuss the results of several tests aimed at demonstrating the benefits of the various elements presented in this paper.

4.2 A two-layered approach

The challenge of solving the game of billiards is not an easy one. The mix of continuous and discrete aspects, in a world subject to stochastic simulations brings forward many complex elements to the model, and demands a very careful analysis. We proposed a novel approach in [39] under the form of a two-layered model, using a planner and controller to establish the optimal shot sequence. What we wish to outline here are the elements necessary for the completion of this approach, with a special focus on the planning aspect which will guide and interact with a robust controller.

As can be seen in figure 4.1, our two-layered model is essentially composed of a controller and planner which will interact together so that the planner may adjust its overall game plan depending on the output from the low-level controller. An analogy can be made with a car travelling from one city to another where a planner will give it the exact streets to follow, and the controller will adjust the acceleration and rotate the wheels depending on the traffic and road conditions.

The planner, detailed in the following section, receives a given table state and perform a preliminary analysis to determine which shots are possible from the cue

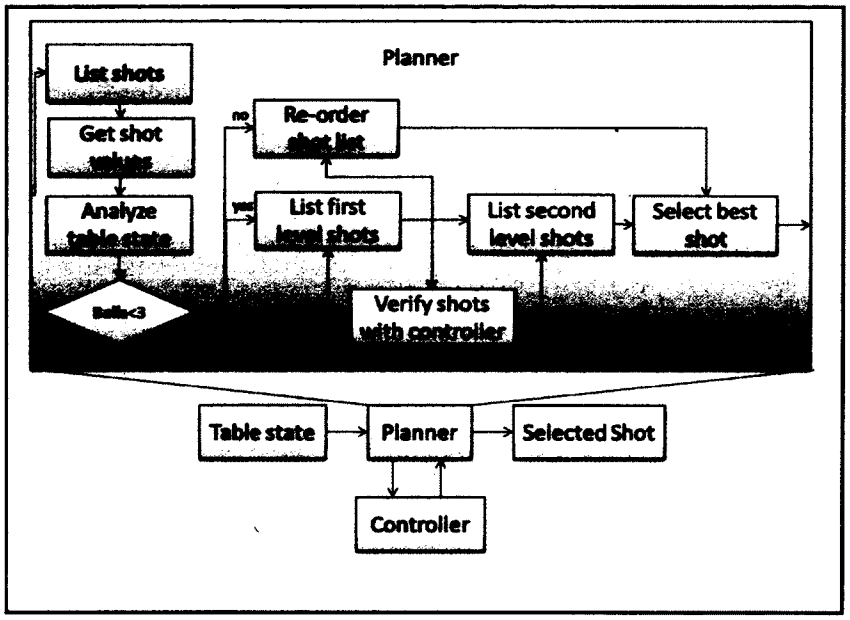

Figure 4.1: Illustration of the two-layered approach, with details on the planning to be used.

ball's position. This shot list will be composed of direct, indirect and combine shots, and is basically a list of pocket-ball combinations, with the queue angle to execute the shot as a starting point for the optimization. The shot list will then be analysed and a value representing the shot difficulty will then be associated to each.

Next we will proceed to the planning of the sequence of shots to execute, depending on the number of balls remaining on the table. If we only have 3 balls remaining, we will use a backward computation approach to compute the shots, otherwise we will use one of the heuristics described below to specify a preferred shot-order. The planner will then access the controller and optimize the selected shots. Based on the results, further evaluations will be done if necessary, otherwise the best shot will be selected and executed.

4.3 High-level planner

As the difficulty of planning in a continuous domain is very high, several heuristics often have to be used to help narrow the search space. In the case of our solver, since we are playing the game of billiards, we can extract several key components from the given table state to help decide which shot sequence will lead to a better outcome. We describe several heuristics in this section which help the planner establish a good shot sequence.

Given a cue ball position on the table, we can first elaborate the list of possible shots from that point in order of difficulty. A typical list would include a combination of the following shots:

- Straight shots: The cue-ball first collides with the target-ball and it goes directly in the pocket.
- Combination shots: The cue-ball first collides with a given ball, which sends it colliding on the target-ball and sends it in the pocket.
- Kick shots: The cue-ball first hits the rail, and then collides with the target-ball to send it in the pocket.
- Bank shots: The cue-ball first hits the target-ball and has it rebound on the rail to then end in the pocket.
- Cluster shots: The cue-ball sinks the target ball directly or indirectly, and then collides with a given cluster to disperse or better reposition the balls.

Many of these shots, however, are not necessarily viable candidates as they can be quite complex to execute (i.e. far shots) and have a minimal success rate. For this reason, only the first few (best ones) are considered.

The model we propose to use here is the same one we previously described in [39]. Instead of partitioning the search space by discretizing our continuous shot parameters, we will instead analyse the table state. Our discretization will be defined as a sequence of shots and repositionning targets.

If we define $b(u)$ at the target ball position after fixing the shot parameters u, p as the pocket for the given target ball, and u_n as the shot parameters of our n^{th} shot, we can minimize function f as:

$$\begin{aligned} \min f(u_0, u_1, ..., u_n) = \text{Posval}(c_0, c_1, ...c_n)\ \forall t : 0...n \\ \text{s.t. } \|b_t(u_t) - p_t\| = 0, t = 1, ..., n \end{aligned} \tag{4.1}$$

where the function f would correspond to the position value heuristic described in [38], which associates a value with a position relative to the available shots at this position(depending directly on the position reached c_n). By using this model, we wish to search for the repositioning targets $c_1, c_2, ...c_n$ to reach after each shot on the table. The problem we encounter however is that the positions we fix are not always reachable, and it is not always possible to determine this in advance. This is why we introduce in our nominal controller a penalty value to reposition for a specific position and maximise position value.

The goal of this is two-fold. If we were to only optimize our shot to reach a given position, we are not guaranteed such a shot always exist. Thus by adding a penalty to maximize the position value, we can at least find a solution that will get us closer. However if we were to only optimize the position value, the problem arises when our shot is in a dead-zone (i.e. a zone where we can't compute a next shot). Thus by combining these two aspects, we will guide our optimization model to reach the best position for our next shot if one exists.

We'll now describe some of the heuristics used to select the best sequence of balls to pocket.

4.3.1 Partially ordered search

The first heuristic we propose to determine the optimal shot sequence is a simple shortest-path approach. Indeed, if we step back and look at a given table state, the most successful sequence of shot will usually be the one where the cue-ball travels the least distance. The less distance it has to cover, the less chances it has to deviate from its predicted path or to collide with an obstacle when noise is added. It is also an obvious behaviour observed when watching professional players. They will rarely cross over the whole table to do a shot in the distance if a closer shot is possible. It is also usually easier to control the cue-ball position when less speed is given to the cue ball.

If we have a look at the same example that was used in [ ] to demonstrate the importance of planning in some specific situations, we can see that by using the shortest-path search we just described to determine the ball sequence, the correct ball would have been selected and the table state would be finished with a much higher success percentage.

If the first shot is selected as can be seen in figure 4.2, the total distance covering the shortest path between the balls corresponds to 3.77 meters, while if the shot 2 is selected 4.3, only 2.63 meters separate the balls, and less distance will be covered by our cue ball.

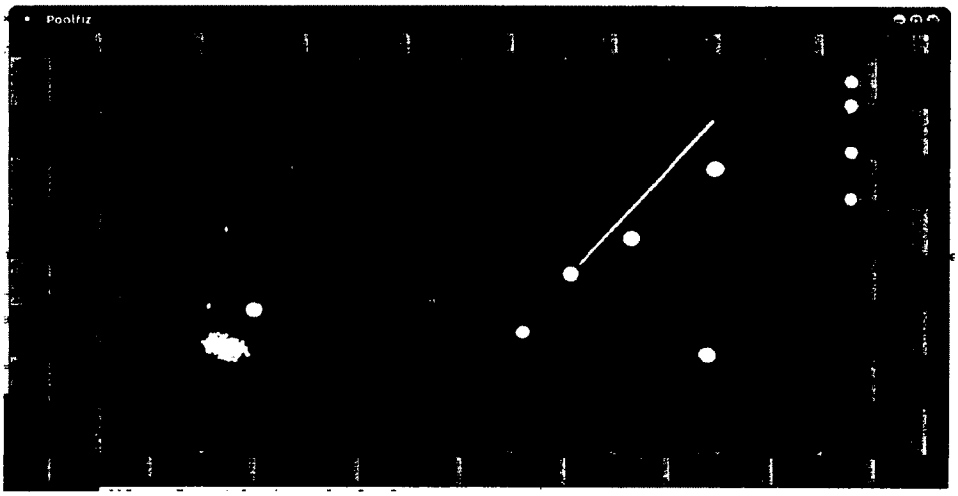

Figure 4.2: In this figure, if we choose to reposition for the ball in the bottom left of the table, the shortest path separating all of the remaining balls will be of 3.77 meters. The success-rate to finish the table was of 54% in this case, based on 500 trials.

4.3.2 Ball-order clustering

Going through many of the lost games using strategy described in 4.3.1, one of the aspects which often came out is that the imposed shot-order can be a constraint in some situations. For example if we look at figure 4.4, we can see that if we choose to pocket the ball in the bottom left corner, as long as we choose to reposition for one of the three closest balls, the overall distance covered by the cull ball will most likely

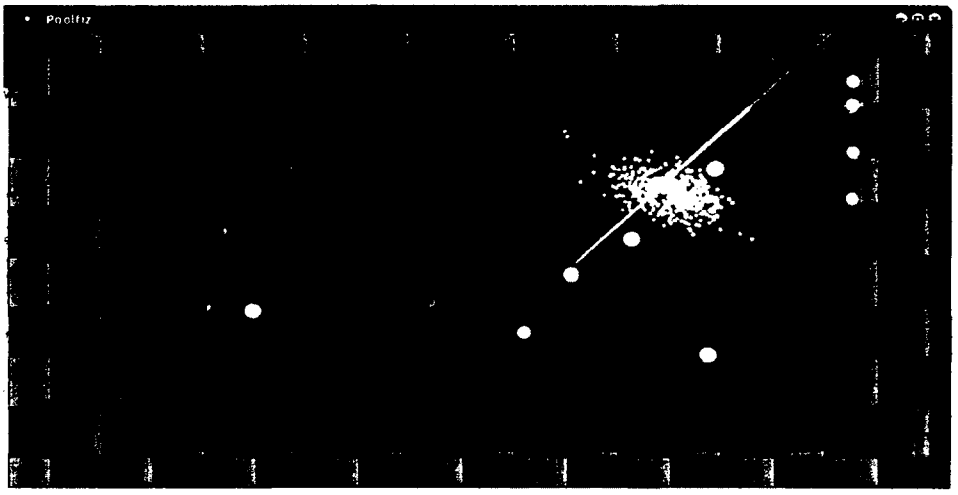

Figure 4.3: In this figure, if we choose to reposition for the ball in the top right of the table, the shortest path separating all of the remaining balls will be of 2.63 meters. The success-rate to finish the table was of 96% in this case, based on 500 trials.

be very similar. In some other situations however, if we put too much importance on respecting the selected order of shots, we may end-up selecting one which will is slightly riskier without having gained any significant reward for the risk taken. Thus by evaluating the table state and clustering balls prior to evaluating the shortest path between them, we can insure that a riskier shot is only selected when necessary, and that we look at the table as a global state where we want to pocket all the balls in a given zone before proceeding to the next one.

Determining the clusters to which each ball belongs is another problem in itself. However in this case since we only want a good approximate and are not too concerned with precision, we decided to use a simple k-means clustering approach [ ]. We start with 2 clusters, which are initialized as the position of two of our balls. We then iteratively attach each ball to its closest cluster, and recompute the centroid of each of these clusters. We continue until we have converged and our balls are set and no longer switch between clusters.

We finally check what is the longest distance between a ball and its cluster. If this distance is too big (a distance of 30cm was used), we increment our cluster size (K) and regenerate the clusters. This is necessary as the k-means approach requires

we specify the number of clusters beforehand. By recomputing our clusters when needed, we are guaranteed to have a decent separation between each of our clusters, and since we only have a maximum of 8 balls to regroup in clusters, the computation cost is negligible. The result of this process is illustrated in figure 4.4.

The choice of not using a fixed cluster size in our evaluation can be explained by the fact that if two balls are slightly far away from each other and still clustered together because of this constraint, then when selecting the preferred shot order they will have the same priority. However we wish to give a higher priority to the closest one, given that a solution is generally found faster by exploring these shots first. We prevent this by ensuring that our clusters are really representative of small clusters of balls which are close in distance.

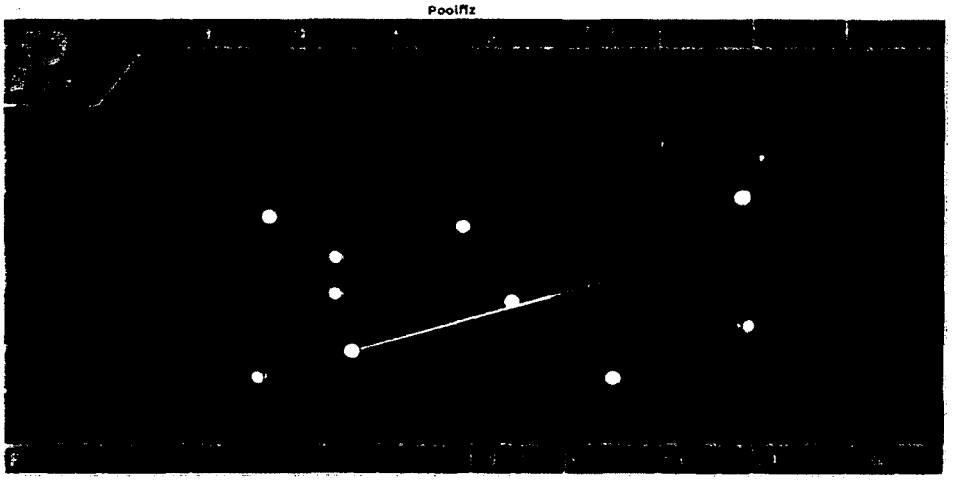

Figure 4.4: In this figure, we are able to see five different zones which were selected as a result of a k-means clustering of the balls on the table.

4.3.3 Dynamic programming

Our final addition to the planner is the use of dynamic programming to do a backwards search on the remaining ball-pocket combinations on the table. We described the general idea in section 4.2 but also mentioned how it was intractable to directly implement. However, if we implement a slight variation of it when only a few balls remain on the table, we may actually find good results.

The reason for not using using dynamic programming unless there are only a few shots left on the table is simple; as we advance in the game, less balls on the table means we have less chances of disturbing the table state when executing our shots. If an optimal shot sequence exists to finish the game, we are more likely to find it this way and execute it properly.

The first step is to establish the desired positions to execute our nth shot. For example if we desire to pocket balls 1, 2 and 3, we will start by removing our balls 1 and 2 from the table, generate the best position to pocket ball 3 in the easiest pocket, and then compute a shot from this position. If such a shot is successful, we'll return its value and propagate it through our search tree, such that we will then compute the parameters to pocket ball 2 in a pocket, to reposition for ball 3.

Since we have discretized our state space into states we wish to reach, we are no longer guaranteed optimality. However we can still get a good approximation this way and find out if a specific shot sequence is better than another. Our controller being semi-robust, we will also be informed with a good estimate of the difficulty of a shot sequence compared to another, thus will be able to make the right decision.

One important factor to take into account about using this approach is that it will most likely be a very powerful tool for other variants of the game of billiards. The game of Straight for example requires a player to be very careful when selecting his last shot since he will need to position the cue-ball in such a way as to be able to rebound and re-break the balls which are put back in a triangle to continue the game. By planning in advance with DP we will be able to foresee such a situation and select the best shot to continue the game.

4.3.4 Table state evaluation

When establishing the list of shots to execute on the table, it is also important to list the best targets for the cue-ball to reach for the next turn. Defining a list of desirable targets relies heavily on a good evaluation of shot difficulty. The planner will thus greatly benefit from any improvement in this area. Previously, the approach used in [38] was only based on the cut-angle and distance between the cue-ball, target-ball and pocket. Based on the 90° rule (see [5]), one can have a good estimate of the

best impact position for the cue-ball, and thus subsequently have a good estimate of the shot's difficulty. The methods described in [57] and [10] also seemingly use a similar approach, however using precomputed coefficients in a tabulated database for a given cue-ball, target-ball position and pocket.

The main issue with these estimations of the shot difficulty is that quite often, the direct path between a target-ball and pocket will be clear of obstacles, but a slight deviation of this path which may still pocket the target-ball may be partially blocked by another obstacle ball.

Another issue also arises when we evaluate the value of a corner shot. If the target-ball is sufficiently close to the rail, its chances of success will be better than if it is aligned directly at 45° angle with the pocket, as it can use the rail to reach its final point.

The geometry corresponding to these aspects has been studied in parallel to the writing of this paper and the details can be found in [18]. We will still describe here a few of the key elements which we use for our evaluation. We can see in figure 4.5 that a direct shot, with the cue-ball and pocket-ball directly aligned with the pocket is actually harder than a shot in which the target-ball is close to the rail (figure 4.6). The shot closer to the rail will have the possibility of a higher deviation from its path while still ending up in the same pocket, while the direct shot might rebound off the side of the pocket.

Thus to correctly compute the value of a corner shot, in the case where no ball is partially blocking the path, we first determine to which zone it belongs. Using some basic geometry, we can use the following developments.

If the ball is in the first zone, we can use the following formula to compute our target ball - pocket zone: $\Delta\theta_A = \arccos\left(\frac{a^2+b^2-c^2}{2ab}\right)$ with a defined as the vector between the target ball and one side of the pocket, b as the vector between the target ball and the other side of the pocket, and finally c as the vector between the two sides of the pocket.

If the ball is in the second zone, we use the same formula but use a mirror table to get the position of our ghost ball on the pocket and compute b (see figure 4.6).

We then use basic geometry to determine any ball is completely or partially blocking its path. To do so requires to do two verifications. The first is to check whether

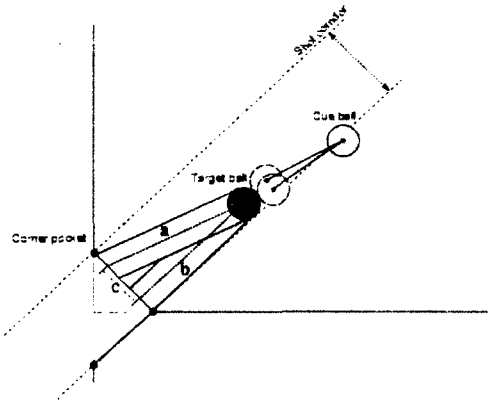

Figure 4.5: Corner shot illustrating a straight-in shot as well as the corresponding corridor. The ball can only be pocketed directly, if a collision with the rail occurs, it will deviate and rebound off the desired trajectory.

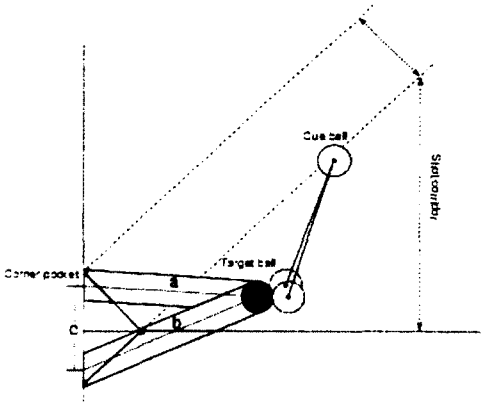

Figure 4.6: Cornershot illustrating a shot with a greater angle, with a shot corridor including possible rebounds on the rail.

or not the ball is in the region delimited by the angle of the zone for which the ball is pocketable. The second is to check whether or not the ball intersects with one of the sides of the triangle representing the angle. If a ball is partially or completely obstructing the path, we will adjust the angle by computing the tangent between our target-ball and obstacle ball. An example of partially blocked zone is shown in figure 4.7.

Finally, two last conditions also need to be checked when computing the value of the shot. The first can be seen in figure 4.8, and corresponds to a case where the cut angle is too small, and we have to actually adjust the angle to correctly represent the zone. The second is if the cue-ball is too close to the target ball. In such a case it may not see the complete angle corresponding to the target-ball – pocket trajectory.

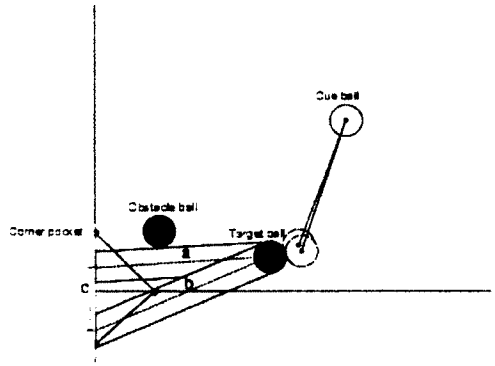

Figure 4.7: Same corner shot, but this time blocked by an obstacle ball. The zone is then rectified using the tangent point of the line between the obstacle ball and cue ball.

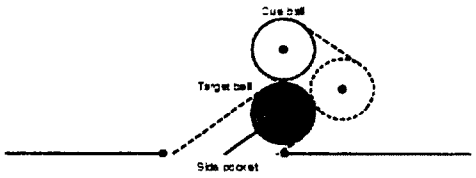

Figure 4.8: A side shot where the cue ball is positioned very close to the target ball. We can see that a correction will need to be made on the angle corresponding to the shot difficulty since the cue-ball cannot directly reach the position which would normally be desirable for greatest chance of success.

4.4 Improvements to the robust controller

The controller presented in [·], while very effective in low-noise situations, was not as impressive when higher noise-levels were used. Obviously, one of the possible solutions is to simply use a higher sample-size when evaluating $f(x)$ at a given point. However the cost of using a higher number of simulator calls can quickly escalate while only providing a marginally better solution. Since the problem we are presented with is not actually stochastic but rather deterministic, with a stochastic aspect simulated with the addition of noise to the shot parameters, a smarter search or modeling of the function should be possible in such a way as to provide solutions in a better neighborhood. We'll provide a few adjustments to our previous controller in this sections, in order to get better solutions while using less calls to the simulator.

As a reminder, the noise-less model which was previously used to compute the parameters necessary to pocket a target-ball in a specific pocket while aiming to reposition the cue-ball at a specific position corresponds to:

$$\begin{aligned} &\min f(u) = ||b_{\text{cue}}(u) - t|| \\ &\text{s. t.} \\ &\quad ||b_i(u) - p|| = 0 \end{aligned} \tag{4.2}$$

where the distance between the cue-ball's final position and a given target t on the table will be minimized, under the constraint of pocketing the target ball. To help find a solution satisfying the constraint of pocketing the target-ball, it can be reformulated as a penalty minimizing its distance with the pocket.

4.4.1 Directional vector

The success of a shot in a stochastic environment, which we will define here as sinking a target ball into a desired pocket, is closely related to one important element; the parameters used to execute the shot. These parameters must be selected very carefully, in a manner which will not only pocket the target ball, but send it with a velocity that will guarantee its success even under noise perturbations. For an easy shot, however, this doesn't always mean we have to hit the center of the pocket dead center. As a matter of fact, in certain cases where the ball is almost already in the pocket, the direction almost doesn't matter as long as we gently push it in the pocket with a minimal velocity. For other shots, harder ones, the success rate will however be directly related to this direction.

Thus to improve our shot generation technique, one approach we propose here is to computed the center of the zone in which the target ball should be sent. We then trace the directional vector from the target ball to the center of this zone. Finally after executing a shot, we check the cue ball and target ball collision to get the post-impact velocity of the target ball. We then penalize our objective function with the distance between this vector and the directional vector for the center of the pocket. We add this parameter to our objective function with a penalty based on the shot difficulty coefficient. We choose to penalize based on the shot difficulty coefficient

since as we said before, for an easy shot it is not as important to hit the pocket dead-center, and this way we may have more room to achieve a better position for the next shot. Supposing our target-ball's velocity after impact is represented by $\tilde{v}_{bi}$, and the vector representing the center of the zone $\tilde{z}_p$, we can define variable $\bar{d} = ||v_{bi} - z_p||$ as the norm of the difference between our two vectors, and $\Delta\theta_A$ our shot difficulty.

The function will now be reformulated as:

$$\begin{aligned} \min_u f(u) = \max(\text{Posval}(u) - \text{Val}_{\min}, 0) + \rho_0||b_i(u) - p||_\infty \\ +\rho_1||b_{\text{cue}}(u) - t|| \ + \rho_2(\Delta\theta_A * \bar{d}) \end{aligned} \tag{4.3}$$

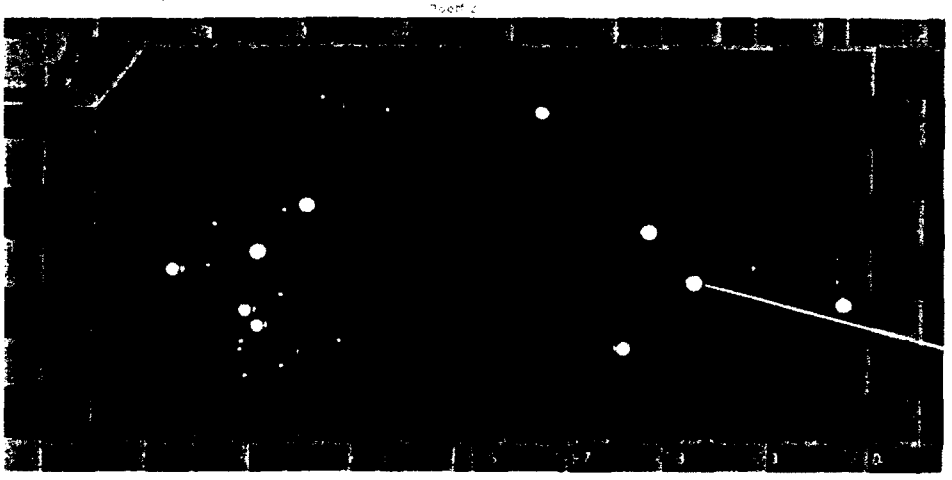

Figure 4.9: A side shot in which we see a slight offset from the pocket center (black dotted line), allowing for a better reposition while still guaranteeing a high shot success rate.

4.4.2 Robust positioning using derivatives

As we just discussed, by using a directional vector to specify the direction in which to send the object ball, we are almost guaranteed that if the penalized parameter is satisfied, our shot will be robust. However one key element we have ignored until now is the repositionning ability of our controller. When we specify a target to repositioning for the next ball, we can either analyse the table and find the best possible target, or we can try to find the center of the best possible zone for a given

shot. Those two are not always the same since it possible that in some cases the best possible target is in the smallest zone, thus hardly reachable. In other cases, the biggest zone might put us in a position that is rather mediocre, leaving us in a hard spot for the next shot. The usage of zones to generate repositionning targets was first and foremost used to aim for specific positions which would be robust under most circumstances. However we observed that in many cases, this was not sufficient to generate good positions and that better shots were often missed because of it.

One of the key aspects making a shot's repositionning value robust or not is the rate of dispersion from the noiseless shot's position in relation to the variation of the shot parameters. Or to simply put, how sensitive will the reposition be to the slight variation in the shot parameters. An example is shown in figure 4.10 where the computed shot is highly sensitive to the smallest variation in shot parameters, while a second example in figure 4.11 shows a more robust shot. This aspect can actually be quantified with a very common tool; derivatives. By computing the derivatives of the function used to simulate a shot in relation to the position reached by the ball, we can obtain a very good estimate of its robustness, and we can then add this parameter to the objective function we are trying to minimize to generate safer shots.

The objective function will now have an added parameter $\rho_3 * \eta$ accounting for the derivative multiplied by a penalty value:

$$\begin{aligned} \min_u f(u) = \max(\text{Posval}(u) - \text{Val}_{\min}, 0) + \rho_0 ||b_i(u) - p||_\infty \\ +\rho_1 ||b_{\text{cue}}(u) - t|| \ + \rho_2 (\Delta\theta_A * \bar{d}) + \rho_3 * \eta \end{aligned} \tag{4.4}$$

This added parameter, which allows for our computed shots to be partially robust concerning in reposition, is for the moment computed with finite differentiation. We hope in the future to be able to exploit automatic differentiation tools in order to get the derivatives directly from the simulator, however as was studied in [49] it is not yet clear if the gains will be noticeable.

4.4.3 Final shot evaluation

The modifications we made to the robust controller here all correspond to a nominal version of the problem. Although the shots computed this way should be fairly

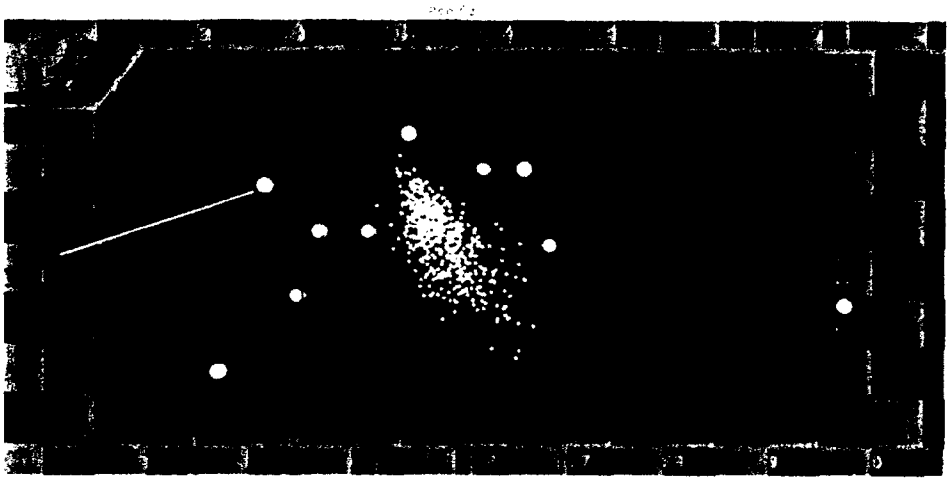

Figure 4.10: In this shot to the upper side pocket, we are able to see a number of possible positions at which the cue ball will end after being simulated 500 times with random gaussian noise added to the parameters.

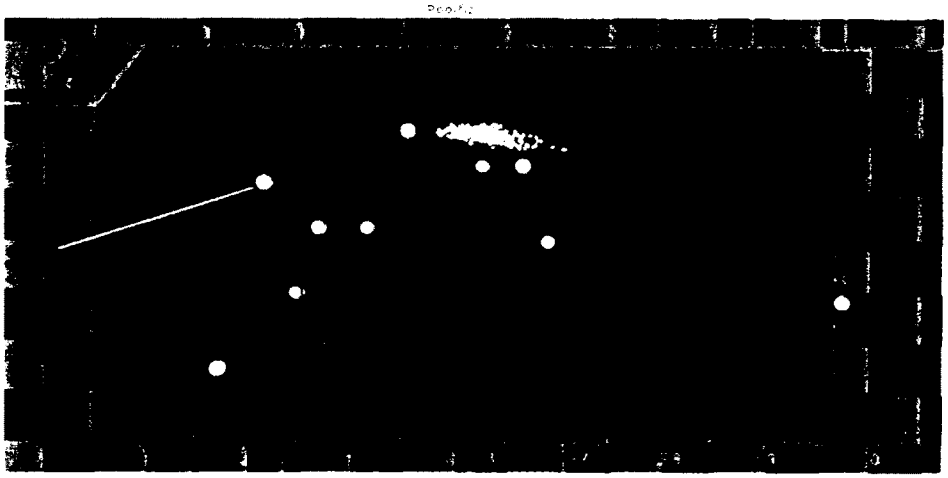

Figure 4.11: In this shot to the upper side pocket, we are able to see that the resulting positions of the cue ball after the shot are not as dispersed as widely as in the last figure [cite figure]. Thus the norm of the derivative of the position relative to the shot parameters is not as big.

robust, they still might behave slightly different than predicted when noise is added. The main reason for this is that several discontinuities can still happen when we execute the shot with noise-induced parameters, such as pocketing the 8-ball by accident, or having the cue-ball pocketed. Although these aspects are part of the nominal model as constraints, they are not for the moment treated in a robust fashion. What we can do however is use the same robust model as in [39] to evaluate the value of a shot after its nominal counterpart is computed.

Since calls to the simulator are the most expensive in terms of CPU resources, for every evaluation done by our optimizer to minimize our model, the value of the result is stored given the specific parameters used. Thus as we go along and try to compute shots for various repositionning targets, we gather and keep all shot information in a vector. It is also important to note that the optimization library used for our computations is actually a local optimizer ([53]), which will have a tendency to make more evaluations as it progressively gets closer to the solution. Thus by recording those calls to the simulator, we can gather important information about the robustness of the shot.

When we want to get an idea of the robustness of a solution which was provided by the nominal model, we will simply go through the vector of previous shots, and use the values of all the shots in a given neighbourhood corresponding to the noise level used for the game. If not enough calls were made in the neighbourhood of our solution we can then generate some more to get a better estimate of the shot's value.

If we recall the model presented in [39], a robust shot value corresponds to minimizing:

$$\begin{aligned} &\min_{u} \sigma \\ &\text{s. t. } \text{Posval}(u,\zeta) \leq \sigma + \alpha_0 \text{dist}(\zeta, U) \\ &\qquad ||b_n(u,\zeta) - p|| \leq \alpha_1 \text{dist}(\zeta, U) \end{aligned} \tag{4.5}$$

where we have our perturbed shot u parameters(for a given noise) vector ζ. If we consider U as the set of the normal range of the parameters under which our shot still remains successful $U = \{\zeta : b_i(u,\zeta) = p\}$, $\forall\zeta$. Thus, our minimized value σ will represent a robust solution, with constraints and objective weighted by the adjustable α values, and the noise effect on the satisfaction of our constraint.

To get our shot value, all we need now is to get the value of σ given our shot parameters u.

4.5 Results

To illustrate the potential of the presented approach and show its improvement over past models, the results of various tests will be shown and discussed in this section.

The game on which we performing tests at this time is one where the aspects of the planner and controller are often very closely interleaved. Some improvements done at the controller-level may generate better shots but leave us in positions from which it will be difficult to finish the game. Thus it is sometimes very difficult to gauge the gain of a particular modification to the model in the overall problem. The same was noticed from [10] when they tried to dissect the key elements of their winning player.

In our case, we have nonetheless been able to discern some stable and very nice improvements to the results by using the methods described in this paper. The test parameters we used were the same as in [39]. We feel that by generating random tables rather than playing a full-game with breakshot we are able to better illustrate the planning ability of the player. In reality, professional players actually spend a lot of time perfecting their breakshots. Depending on the style of game played, it can have a crucial importance to break defensively or offensively and sometimes requires a great deal of skill. The study of the perfect breakshot is however very dependent on the simulator, and possible advantages can be exploited from analysing it to always leave the player in a favourable position to clean the table (mentioned in [10] as an important element). For the sake of curiosity we will nonetheless provide a few clean-off-the-break results, with a breakshot which was selected after doing some testing on the Fastfiz simulator.

Overall, five hundred table states were generated, with balls randomly positioned. The same 500 tables used in this article were used in [39], thus allowing for a direct comparison to be done. Success is defined as the player sinking all the balls on the table successfully without missing a single shot. Averages are cumulated with a value of 1 for a success and 0 for a failure. Random noise is added to each of the computed

shots parameters. This noise was added in the previous computational pool olympiads (see [28]) to simulate the imperfections which would normally be present on a real pool table, and to simulate different skill levels of real players. We choose to simulate with a noise level of 1x and 0.5x of the parameters used in the last competition (Standard deviations for each parameters: ϕ=0.125 deg, θ=0.1 deg, v=0.075 m/s, a=0.5mm, b= 0.5mm). The same maximal computing time of 10 minutes per table state was allowed. For the standard noise level, the average computing time (Intel Core Duo 2.53ghz) was of approximatively 280 seconds per shot, while for $1/2x$ noise level, only about 150 seconds per game were required.

In figures 4.14 and 4.12, we can observe that one of the major improvements to the success rate was brought by the new shot evaluation function, which is now more precise than before, and thus allows for better repositionning on the table. The gain is more noticeable in higher noise-level tables, which suggests that positioning in these states is probably more of an important factor since a better player (less noise) will still manage to complete his shots even in tougher positions. We can also observe a clear gain of 8% (in figure 4.14) when planning using the cluster-based shortest path heuristic. This demonstrates that planning can actually be done at a higher level, without the necessity to compute every shot in a tree-like search. It is hard to comment on the efficiency of the cluster-based heuristic vs shortest path, since the difference between the two approaches is only of 2%, which could be to the statistical significance of the sample.

Finally, if we look at figures 4.13 and 4.15, which shows the best results obtained using all the improvements described in this paper, compared to the results which were originally obtained in [39], we see a gain of over 20% success rate for the 1x noise level and of 9% for the 1/2x noise level. Obviously the gain were not as important in lower noise-level table states, since the margin for improvement is smaller, but we nonetheless get very close to a perfect success rate.

If we now have a look at figures 4.16 and 4.17, we can see the outcome of 500 games, where a player would break, and then try to finish all the balls on the table. As in [11] the player was allowed to re-break until successful, and allowed a maximum of 6 minutes to compute all of his shots, with a binary indicator of 1 for victory and 0 for failure. An average success rate of around 80% can be observed for games played

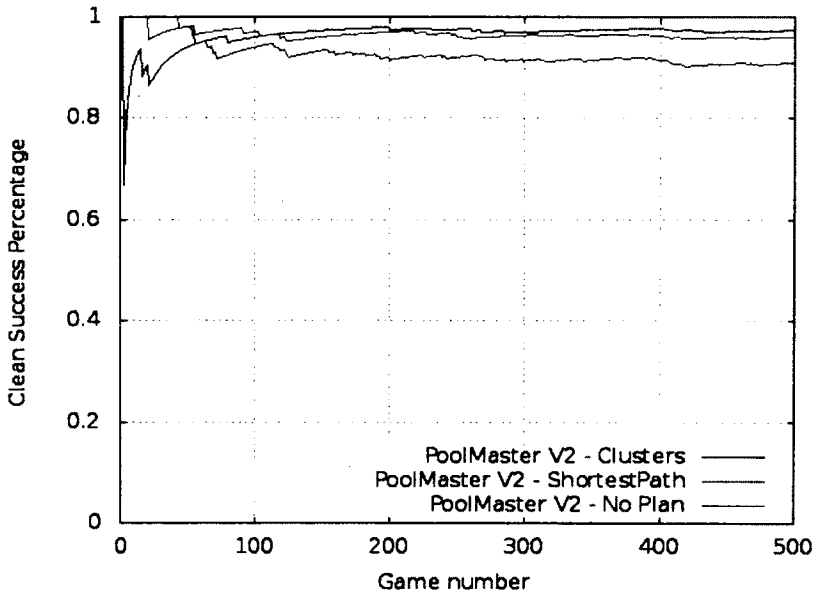

Figure 4.12: Cumulated averages for the results of 500 games played on randomly dispersed table states with $\frac{1}{2}$x noise level. An average of 91% success rate is achieved when no planning is used, 96% for shortest-path and 97.4% when using the cluster heuristic.

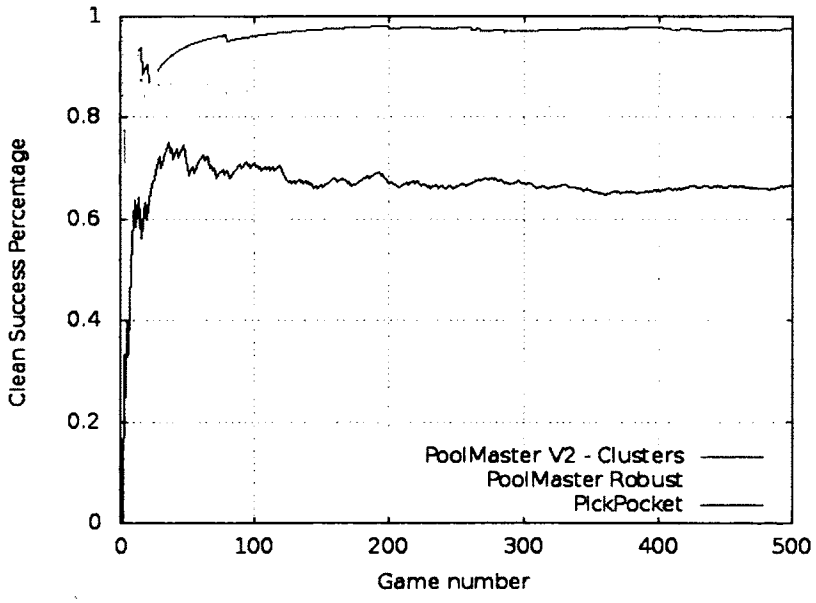

Figure 4.13: Cumulated averages for the results of 500 games played on randomly dispersed table states with $\frac{1}{2}$x noise level. These results are the results which were first presented in [39]. A noticeable improvement was made over PickPocket ([58]) and the old model, which had a success percentage of 84%

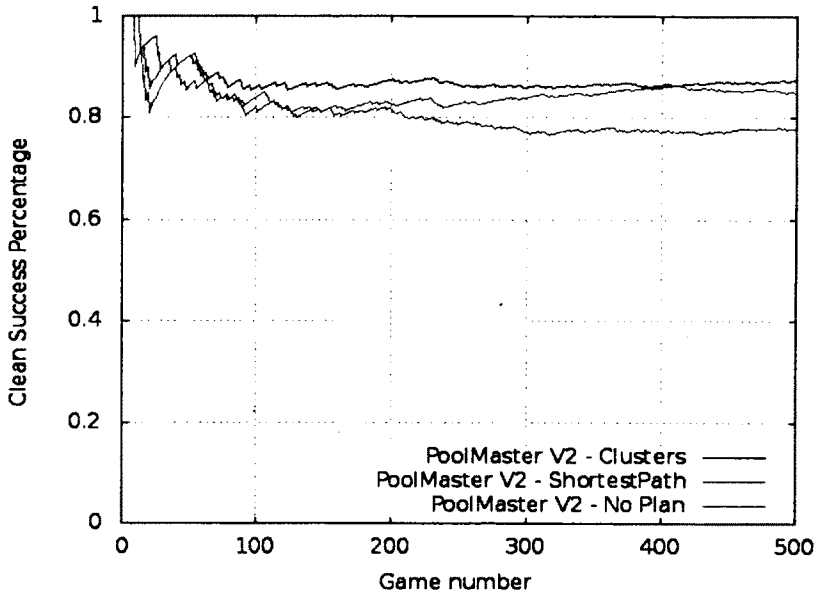

Figure 4.14: Cumulated averages for the results of 500 games played on randomly dispersed table states with 1x noise level. We can see the results when using the new improvements to the controller (77%), with the shortest-path heuristic (85%), and finally the cluster-based heuristic (87.4%).

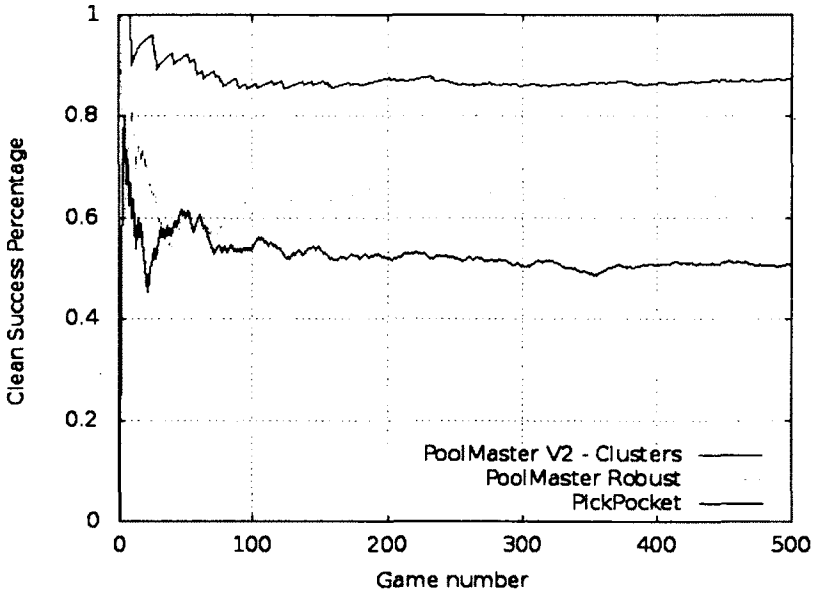

Figure 4.15: Cumulated averages for the results of 500 games played on randomly dispersed table states with 1x noise level. These results illustrate a gain of over 20% with a success rate of 87.4% while the old version was at 64%.

on a 1x noise level, while a higher 93% was attained for $\frac{1}{2}$x noise level. The last published results in [11] seemed to be hovering around $60 - 65\%$ for 1x noise level, and $80 - 85\%$ for a $\frac{1}{2}$x.

As a disclaimer, it should be noted that it is very hard to do a direct comparison between different players/approaches without actually confronting them in a tournament. Since the tournaments for the Computational Pool Olympiads are not being held anymore we have to find other means for benchmarking. As such, these tests should not be taken as a definite proof of the value of a method over another, but rather as an indication of the potential of the described approach. Many other factors have influence on the outcome when playing full-games.

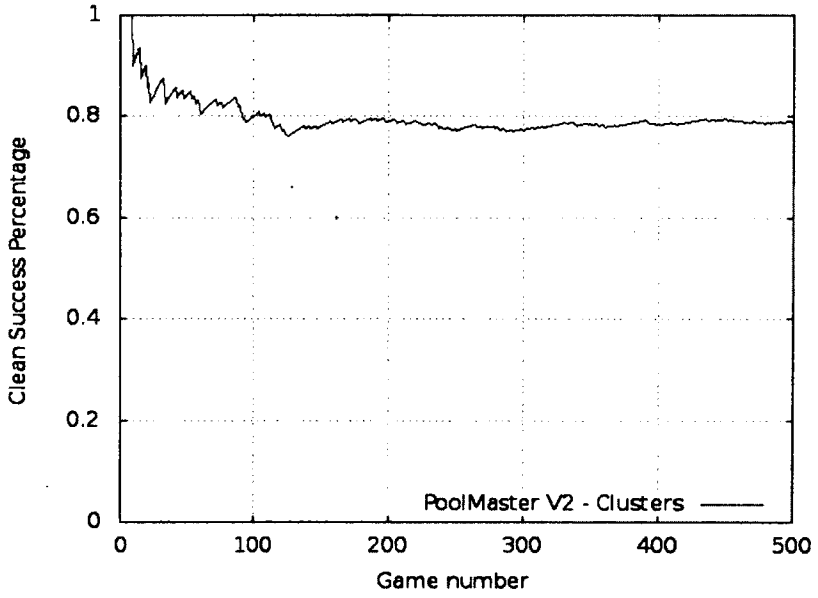

Figure 4.16: 500 games with breakshot, at a noise level of 1x

4.6 Perspective

As we can see, noticeable improvements have been made over the version of the model using the controller alone. The usage of a two-layered system provided in incremental but non-negligible advantage over a single-layered approach. We are

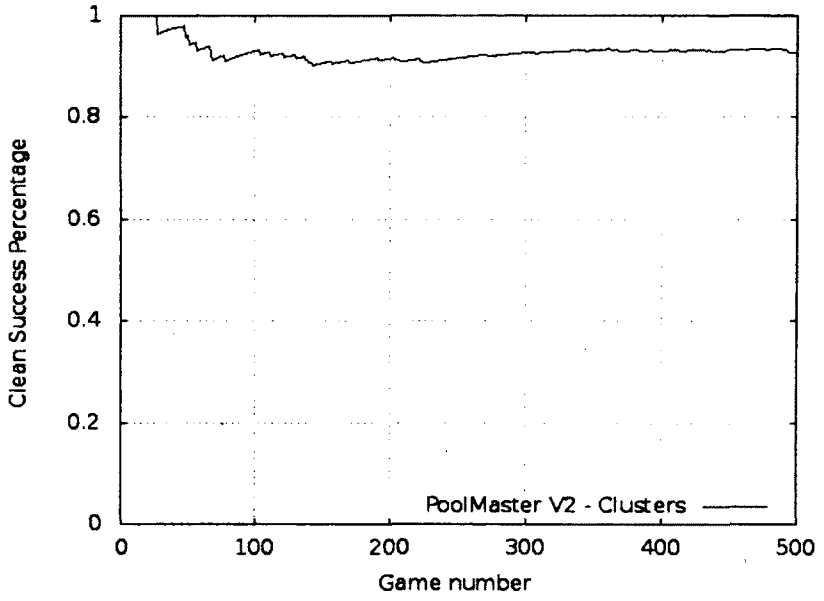

Figure 4.17: 500 games with breakshot, at a noise level of $\frac{1}{2}$x

also able to draw some interesting conclusions from our tests. Although a robust controller by itself accounts for most of the successful games, it still needs to rely on a planner of some sort, even in the simplest form, to get scores closer to perfection. The planner described in this work acts as a general manager of the table states, by identifying a preferred shot-order, and decides whether this plan should be followed or not, depending on the feedback from the low-level controller. Obviously the planer itself is domain-dependent, and in the case of billiards, game-dependent. However we have shown that with a very good low-level controller, such an approach can be very efficient to solve a complex problem dealing with continuous states and actions under uncertainty.

Future work will most likely be focused on variations of the planner to improve its planning abilities, possibly by using learning algorithms to identify trouble-some states. Such a tool might also prove helpful to predict success rate given specific table state, and thus provide an efficient way to plan safety shots ahead of time.

Conclusion

De nos jours on tente de résoudre de plus en plus de problèmes complexes étant donné une puissance de calcul toujours grandissante. Cependant, malgré cela, certains problèmes demeurent trop difficiles pour être résolus directement même lorsqu'une solution théorique existe. Les travaux présentés dans cette thèse apportent un nouveau regard sur la planification dans un domaine précis où peu de solutions efficaces existent pour l'instant. Le défi d'effectuer une planification quelconque lorsqu'on possède une infinité de choix à notre disposition est très grand, mais comme il a été démontré, il est possible d'y arriver en faisant quelques choix judicieux. En utilisant des connaissances tirées du domaine de la recherche opérationnelle et de l'intelligence artificielle, et en modélisant le problème de façon efficace, il a été possible de proposer une approche nouvelle et très performante pour la résolution de problèmes avec actions et états continus, soumis à des perturbations stochastiques.

Une contribution très importante pour le problème se trouve au niveau du contrôleur, qui bien sûr est adapté précisément pour le jeu du billard mais pourrait facilement être adapté pour d'autres applications. La modélisation du problème de façon continue permet de profiter de l'expertise approfondie des dernières années dans le domaine de l'optimisation non-linéaire. Pour le cas déterministe, en traitant le problème comme une simple fonction à minimiser, on peut facilement et rapidement arriver à trouver une solution optimale. Comme on l'a démontré dans ce travail, sans la présence du bruit, le contrôleur arrive à toujours trouver une solution en un temps remarquable. Cela démontre l'intérêt pour l'utilisation de l'optimisation comme outil pour gérer l'aspect technique présent dans ce type de problème. Bien sûr, les choses se compliquent lorsqu'on tente de prendre en compte la perturbation de la fonction à minimiser par un bruit gaussien. On doit alors faire appel à une analyse plus poussée de

la fonction à optimiser afin de profiter de l'information disponible comme par exemple les dérivées, ou la proximité avec la dernière évaluation de la fonction donnant une solution ne satisfaisant pas les contraintes. De cette façon, des solutions robustes sont générées sans avoir recours à un échantillonnage aléatoire lors de l'exploration du domaine.

Une autre contribution importante se situe au niveau de l'approche globale proposée, c'est-à-dire l'approche à deux-couches. Bien des modèles sont suggérés dans la littérature pour arriver à traiter l'explosion combinatoire relative aux problèmes avec aspects stochastiques, cependant dans le contexte du problème traité, les solutions existantes manquent de raffinement quant à l'avantage qui peut être tiré de traiter spécifiquement les aspects techniques et tactiques dans la génération de plans. Nous avons démontré qu'en approchant le problème d'un point de vue global, il est possible d'en tirer certaines informations nous aidant beaucoup pour la planification. Il a ainsi été possible de développer un planificateur assez efficace avec l'aide quelques simples heuristiques.

Les perspectives pour la suite des travaux sont multiples. En un premier temps, il sera très intéressant d'explorer l'utilisation de techniques d'apprentissage pour arriver à déterminer à l'avance les états particuliers où le planificateur possède très peu de chances de succès, de façon à prévoir plus facilement les coups défensifs. Il serait aussi intéressant d'utiliser ces techniques pour arriver à mieux déterminer les états atteignables afin de minimiser les appels inutiles au contrôleur.

En un deuxième temps, il sera intéressant de modifier légèrement le planificateur afin de mettre en place un joueur pouvant participer aux diverses formes de jeu du billard. Peu de modifications seraient nécessaires pour le jeu de la 9 ou le continu par exemple, où le contrôleur proposé dans ces travaux devrait être particulièrement efficace.

En un troisième temps, il est possible d'envisager la possibilité de tirer encore plus avantage du simulateur physique pour l'évaluation de la robustesse des coups. En effet bien que les dérivées nous donnent beaucoup d'informations concernant la robustesse de la reposition, il est possible qu'en effectuant une analyse plus poussée de la géométrie d'un coup, on puisse déterminer à l'avance la sensibilité de chaque paramètre pour un coup donné. Cela permettrait donc d'optimiser un coup en ayant

spécifié une certaine sensibilité comme contrainte à respecter.

Finalement l'application de diverses méthodes d'analyse et de simulation pour arriver à mieux spécifier les différentes constantes utilisées dans le programme serait une amélioration importante. En effet il existe plusieurs paramètres, comme par exemple la valeur du poids des diverses pénalités utilisées lors de l'optimisation, qui ont été choisis de façon subjective. Certaines de ces variables ont un impact très important sur la valeur de la solution générée lors de l'optimisation. Le développement d'une plateforme de test pouvant elle-même déterminer ces paramètres en fonction de la problématique ou de l'importance de l'aspect stochastique pourrait se montrer très utile.

Enfin, l'application de la méthode proposée dans un domaine différent que celui du billard serait certainement intéressant à explorer. Cela permettrait d'explorer l'efficacité et la capacité d'un tel modèle à être adapté à diverses situations.

Bibliographie

[1] « FastFiz ».
http ://www.stanford.edu/group/billiards/FastFiz-0.1.tar.gz.

[2] « Glossary of cue sports terms », 2010.
WWW Wikipedia.

[3] « Classic Billard », 2012.
Canal + multimédia, http ://www.gamekult.com/jeux/canal-classic-billard-J8610.html.

[4] AAAI.
« Association for the Advancement of Artificial Intelligence ».
`http://www.aaai.org`.

[5] David ALCIATORE.
The Illustrated Principles of Pool and Billiards.
Sterling, 2004.

[6] David G. ALCIATORE.
« Pool and Billiards Physics Resources ».
http ://billiards.colostate.edu/physics.

[7] M.E. ALIAN et S.B. SHOURAKI.
« A Fuzzy Pool Player Robot with Learning Ability ».
Dans *WSEAS Trans. on Electronic*, volume 1, pages pp422–425, 2004.

[8] Mohammad Ebne ALIAN, Saeed Bagheri SHOURAKI, M.T. Manzuri SHALMANI, Pooya KARIMIAN et Payam SABZMEYDANI.
« Robotshark : a gantry pool player robot ».
Dans *ISR 2004 : 35th Intl. Sym. Rob.*, March 2004.

[9] Christopher ARCHIBALD, Alon ALTMAN, Michael GREENSPAN et Yoav SHOHAM.
« Computational Pool : A New Challenge for Game Theory Pragmatics ».
AI Magazine, 31(4):33–41, 2010.

[10] Christopher ARCHIBALD, Alon ALTMAN et Yoav SHOHAM.
« Analysis of a winning computational billiards player ».
Dans *IJCAI'09 : Proceedings of the 21st international joint conference on Artifical intelligence*, pages 1377–1382. Morgan Kaufmann Publishers Inc., 2009.

[11] Christopher ARCHIBALD, Alon ALTMAN et Yoav SHOHAM.
« Success, strategy and skill : an experimental study ».
Dans *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems : volume 1 - Volume 1*, AAMAS '10, pages 1089–1096. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[12] Christopher ARCHIBALD et Yoav SHOHAM.
« Modeling billiards games ».
Dans Carles SIERRA, Cristiano CASTELFRANCHI, Keith S. DECKER et Jaime Simão SICHMAN, éditeurs, *AAMAS (1)*, pages 193–199. IFAAMAS, 2009.

[13] Christopher ARCHIBALD et Yoav SHOHAM..
« Modelling billiards games ».
Dans Sierra DECKER, Sichman et CASTELFRANCHI, éditeurs, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagents Systems (AAMAS 2009)*, 2009.

[14] Jane BAYES et William SCOTT.
« Billiard Ball Collision Experiment ».
Dans *Am. Jour. Physics vol. 3 no. 31*, pages 197–200, March 1963.

[15] BCA.
« Billiard Congress of America », May 2008.
`http://www.bca-pool.com`.

[16] Aharon BEN-TAL, Stephen BOYD et Arkadi NEMIROVSKI.
« Extending Scope of Robust Optimization : Comprehensive Robust Counterparts of Uncertain Problems ».
Mathematical Programming, 107:63–89, 2006.

10.1007/s10107-005-0679-z.

[17] Aharon BEN-TAL et Arkadi NEMIROVSKI.
« Robust optimization – methodology and applications ».
Mathematical Programming, 92:453–480, 2002.
10.1007/s101070100286.

[18] Nicolas BUREAU.
« Sensibilité des coups au billard ».
Dans *CaMUS*, pages 9–26. Université de Sherbrooke, 2012.
http ://camus.math.usherbrooke.ca/revue/volume2.html.

[19] B.R. CHENG, J.T. LI et J.S. YANG.
« Design of the Neural-Fuzzy Compensator for a Billiard Robot ».
Dans *IEEE Intl. Conf. Networking, Sensing & Control*, pages 909–913, March 2004.

[20] S. C. CHUA, E. K. WONG et V. C. KOO.
« Performance evaluation of fuzzy-based decision system for pool. ».
Appl. Soft Comput., 7(1):411–424, 2007.

[21] S.C. CHUA, E.K. WONG et V.C. KOO.
« Pool Balls Identification and Calibration for a Pool Robot ».
Dans *ROVISP 2003 : Proc. Intl. Conf. Robotics, Vision, Information and Signal Processing*, pages 312–315, January 2003.

[22] S.C. CHUA, E.K. WONG, Alan W.C. TAN et V.C. KOO.
« Decision Algorithm for Pool Using Fuzzy System ».
Dans *iCAiET 2002 : Intl. Conf. AI in Eng. & Tech.*, pages 370–375, June 2002.

[23] Gaspard-Gustave CORIOLIS.
Théorie mathématique des effets du jeu de billard.
J. Gabay, 1835.

[24] Javier Ruiz del SOLAR, Eric CHOWN et Paul-Gerhard PLÖGER, éditeurs.
« RoboCup 2010 : Robot Soccer World Cup XIV [papers from the 14th annual RoboCup International Symposium, Singapore, June 25, 2010] », volume 6556 de *Lecture Notes in Computer Science*. Springer, 2011.

[35] ICGA.
« International Computer and Games Association ».
http://ilk.uvt.nl/icga/.

[36] Steven G. JOHNSON.
« The NLopt nonlinear-optimization package », 2010.
http ://ab-initio.mit.edu/nlopt.

[37] D. R. JONES, C. D. PERTTUNEN et B. E. STUCKMAN.
« Lipschitzian optimization without the Lipschitz constant ».
Journal of Optimization Theory and Applications, 1993.

[38] Jean-François LANDRY et Jean-Pierre DUSSAULT.
« AI Optimization of a Billiard Player ».
Journal of Intelligent & Robotic Systems, 50:399–417, 2007.
10.1007/s10846-007-9172-7.

[39] Jean-François LANDRY, Jean-Pierre DUSSAULT et Philippe MAHEY.
« A robust controller for a two-layered approach applied to the game of billiards ».
Entertainment Computing, (0):–, 2012.

[40] L.B. LARSEN, M.D. JENSEN et W.K. VODZI.
« Multi Modal User Interaction in an Automatic Pool Trainer ».
Dans *ICMI 2002 : 4^{th} IEEE Intl. Conf. Multimodal Interfaces*, pages 361–366, Oct. 2002.

[41] Will LECKIE et Michael GREENSPAN.
« Pool Physics Simulation by Event Prediction 1 : Motion Transitions ».
Intl. Comp. Gaming Ass. Journal, 28(4):214–222, Dec. 2005.

[42] Will LECKIE et Michael GREENSPAN.
« Pool Physics Simulation by Event Prediction 2 : Collisions ».
Intl. Comp. Gaming Ass. Journal, 29(1):24–31, Mar. 2006.

[43] Will LECKIE et Michael A. GREENSPAN.
« An Event-Based Pool Physics Simulator ».
Dans van den Herik et al. [62], pages 247–262.

[44] Z.M. LIN, J.S. YANG et C.Y. YANG.
« Grey Decision-Making for a Billiard Robot ».

Dans *IEEE Intl. Conf. Systems, Man and Cybernetics*, pages 5350–5355, October 2004.

[45] Fei LONG, Johan HERLAND, Marie-Christine TESSIER, Darrly NAULLS, Andrew ROTH, Gerhard ROTH et Michael GREENSPAN.
« Robotic Pool : An Experiment in Automatic Potting ».
Dans *IROS 2004 : IEEE/RSJ Intl. Conf. Intelligent Robotic Systems*, pages 361–366, Oct. 2004.

[46] J. B. MACQUEEN.
« Some Methods for Classification and Analysis of MultiVariate Observations ».
Dans L. M. Le CAM et J. NEYMAN, éditeurs, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[47] Wayland C. MARLOW.
The Physics of Pocket Billiards.
Marlow Advanced Systems Technologies, 1995.

[48] B. M. MILLER et J. BENTSMAN..
« Optimal control problems in hybrid systems with active singularities ».
Nonlinear Analysis, 65:999–1017, 2006.

[49] Jorge J. MORÉ et Stefan M. WILD.
« Do You Trust Derivatives or Differences ? ».
Preprint ANL/MCS-P2067-0312, Argonne Mathematics and Computer Science Division, 2012.

[50] Thomas NIERHOFF, Omiros KOURAKOS et Sandra HIRCHE.
« Playing pool with a dual-armed robot ».
Dans *ICRA*, pages 3445–3446. IEEE, 2011.

[51] Dimitris PAPAVASILIOU.
« Billiards ».
http ://www.nongnu.org/billiards.

[52] David F. PERCY.
« Stochastic Snooker ».
Journal of the Royal Statistical Society. Series D (The Statistician), 43(4):pp.

585–594, 1994.

[53] Michael James David POWELL.
« The BOBYQA algorithm for bound constrained optimization without derivatives ».
Rapport Technique, Department of Applied Mathematics and Theoretical Physics, Cambridge England, 2009.

[54] David SÉNÉCHAL.
« Mouvement d'une boule de billard entre les collisions ».
Unpublished manuscript, 1999.

[55] Ron SHEPARD.
Amateur Physics for the Amateur Pool Player.
self published, 1997.

[56] Sang William SHU.
« *Automating Skills Using a Robot Snooker Player* ».
Thèse de doctorat, Bristol University, 1994.

[57] Michael SMITH.
« PickPocket : An Artificial Intelligence For Computer Billiards ».
Mémoire de maîtrise, University of Alberta, 2006.

[58] Michael SMITH.
« Running the table : an AI for computer billiards ».
Dans *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 994–999. AAAI Press, 2006.

[59] Michael SMITH.
« PickPocket : A computer billiards shark ».
Artif. Intell., 171(16-17):1069–1091, 2007.

[60] Russel SMITH.
« ODE ».
http ://www.ode.org/.

[61] TNC.
« TNC : A Truncated-Newton optimization package », 2005.
http ://iris.gmu.edu/%7Esnash/nash/software/software.html.

[62] H. Jaap van den HERIK, Shun chin HSU, Tsan sheng HSU et H. H. L. M. DONKERS, éditeurs.
« Advances in Computer Games, 11th International Conference, ACG 2005, Taipei, Taiwan, September 6-9, 2005. Revised Papers », volume 4250 de *Lecture Notes in Computer Science*. Springer, 2006.

[63] J. WALKER.
« The Physics of the Draw, the Follow, and the Massé (in Billiards and Pool) ».
Dans *Scientific American*, July 1983.

[64] R. Evan WALLACE et Michael SCHROEDER.
« Analysis of Billiard Ball Collisions in Two Dimensions ».
Dans *Am. Jour. Physics vol. 56 no. 9*, pages 815–819, September 1988.