The stochastic shortest path problem and its variations : foundations and applications to sport strategy optimization

Matthieu Guillot

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques et Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Matthieu GUILLOT

Thèse dirigée par **Gautier STAUFFER**, Professeur

préparée au sein du **Laboratoire Laboratoire des Sciences pour la Conception, l'Optimisation et la Production de Grenoble** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Le problème du plus court chemin stochastique et ses variantes : fondements et applications à l'optimisation de stratégie dans le sport

The stochastic shortest path problem and its variations: foundations and applications to sport strategy optimization

Thèse soutenue publiquement le **3 juillet 2020**,
devant le jury composé de :

Monsieur GAUTIER STAUFFER
PROFESSEUR DES UNIVERSITES, KEDGE BUSINESS SCHOOL - BORDEAUX, Directeur de thèse
Monsieur JORG RAMBAU
PROFESSEUR DES UNIVERSITES, UNIVERSITE DE BAYREUTH - ALLEMAGNE, Rapporteur
Monsieur LOUIS ESPERET
CHARGE DE RECHERCHE HDR, CNRS DELEGATION ALPES, Examinateur
Monsieur BRUNO SCHERRER
CHARGE DE RECHERCHE HDR, INRIA CENTRE NANCY GRAND EST, Examinateur
Monsieur FRANÇOIS CLAUTIAUX
PROFESSEUR DES UNIVERSITES, UNIVERSITE DE BORDEAUX, Rapporteur
Madame NADIA BRAUNER
PROFESSEUR DES UNIVERSITES, UNIVERSITE GRENOBLE ALPES, Présidente

Contents

Remerciements

Je voudrais tout d'abord remercier mon directeur de thèse, Gautier, sans qui rien n'aurait été possible, et à qui je dois beaucoup de ma formation scientifique actuelle.

J'aimerais également remercier tous les membres du jury qui ont accepté de lire et juger mon travail. Merci en particulier à Nadia, qui m'a beaucoup aidé dans cette période si particulière et qui m'a permis de soutenir dans les conditions que je souhaitais; et à Louis sans qui je n'aurais pas pu soutenir tellement il a été d'une aide immense à la fin de ma thèse.

Durant ces années de thèse, j'ai eu la chance de croiser beaucoup de monde au laboratoire. Merci à l'ancienne génération : Nico, Clément, Lucas (qui m'a tellement aidé), Lucile, Aurélie... Merci à Taume, Alex (meilleur co-bureau du monde), Dehia, et bien sûr Gricha qui m'a aidé tant dans le travail que dans les loisirs. Merci à Lucie qui a toujours été là dans les moments durs, et les bons aussi !

Et un immense merci à l'amour de ma vie, sans qui j'aurais craqué 1000 fois. On a affronté cette thèse à deux, puis à trois. On affrontera les autres épreuves de la vie ensemble, à trois ou à plus. Laura, voici notre manuscrit.

Introduction

Ces dernières années, le développement de 'super IAs' telles que AlphaGo [29] pour le jeu de Go et de AlphaStar [3] pour des jeux de stratégies en ligne comme Starcraft II a grandement bouleversé les communautés de ces disciplines. Les stratégies préexistantes suivaient des schémas (souvent appelés *metagames*) qui ont été largement dépassés par les stratégies mises en place par le programme informatique, explorant des techniques jusqu'ici inconnues ou crues obsolètes. Ceci a permis aux joueurs de considérer d'autres aspects du jeu. Au Go, les joueurs, après avoir analysé les coups originaux de l'IA, ont depuis adopté certains de ces coups dans des parties professionnelles. À Starcraft, l'IA a surtout brillé par la gestion minutieuse de ses unités, même avec un nombre d'action par minute limité. L'outil informatique peut donc avoir un impact fort sur l'évolution du sport ou de l'e-sport de manière générale et sur l'optimisation de stratégies en particulier. L'optimisation de stratégies dans le sport est un grand enjeu à la fois pour l'amélioration des performances des sportifs, mais également pour l'analyse prédictive de résultats. Dans le premier cas, il s'agit d'utiliser des données relatives aux sportifs, aux terrains et à de potentiels adversaires afin de déterminer la manière optimale de jouer en tenant compte de tous ces aspects. On peut ainsi déterminer quels aspects du jeu il serait intéressant d'améliorer en priorité afin d'avoir le plus grand impact positif sur les performances du ou des sportifs. Dans le deuxième cas, il s'agit de simuler le jeu des sportifs afin d'obtenir des éléments statistiques sur les résultats possibles (distribution de scores, probabilités de gain...).

Dans ce manuscrit, nous nous sommes concentrés sur le Golf, et ce pour deux principaux aspects. D'abord un golfeur se confronte au 'terrain' plus qu'aux autres golfeurs : les interactions directes entre les joueurs sont très limitées. Ainsi, la stratégie de chaque joueur peut se déterminer indépendamment des actions des autres joueurs (sous certaines hypothèses que nous détaillerons plus tard).

De plus, nous avons eu accès à une base de données, Shotlink [86], qui nous donne une grande quantité d'informations sur les joueurs professionnels. Ces données sont indispensables pour la création de profils de joueurs et leurs accès a été déterminant dans le choix de notre application.

Il est cependant possible d'adapter les modèles que nous avons développé dans d'autres disciplines sportives et ce même quand deux équipes sont impliquées. Les

travaux de Hoffmeister et Rambau [55] [56] utilisent le même type de modèles et permettent l'optimisation de stratégies des équipes de Beach Volley ou de Football.

Le Golf est un sport dans lequel des joueurs s'affrontent sur un *parcours* composé dix-huit *trous*. Chaque trou est composé de plusieurs éléments, dont une zone de départ, ou *tee*, une zone d'arrivée, ou *green*, où la pelouse est très rase avec en son sein un trou marqué par un drapeau (qu'on nommera dans la suite 'drapeau' pour éviter toute ambiguïté). Une zone d'herbe rase (mais moins que le green) appelée *fairway* relie le tee et le green. Entourant le fairway, on trouve une zone d'herbe plus haute, ou *rough*. On peut trouver également plusieurs obstacles : de l'eau, des zones de sables appelées *bunkers* et des arbres. Le golfeur doit acheminer la balle depuis le tee vers le drapeau en utilisant un *club*, parmi les quatorze clubs qu'il possède et dont il a fait la sélection. À chaque fois que le golfeur utilise un de ses clubs pour frapper la balle, son score (initialement de zéro sur le tee) est incrémenté de un en situation normale, ou de deux si il reçoit une pénalité. Une pénalité est par exemple octroyée à un joueur si la balle tombe dans l'eau ou en dehors des limites du terrain. Si la balle tombe dans l'eau, elle est alors replacée à la lisière de celle-ci ; et si elle sort des limites du terrain, elle est replacée à l'endroit où elle a été tirée. Le but du jeu est que la balle arrive au drapeau en un nombre minimum de coups.

D'un point de vue compétitif, il y a principalement deux types de compétitions qui diffèrent selon la façon de déterminer le vainqueur sur les dix-huit trous (la taille réglementaire des parcours de golf). Lors d'une compétition *Stroke Play*, chaque joueur additionne les scores qu'il a fait sur les dix-huit trous et le vainqueur est celui qui a obtenu le score minimum sur l'ensemble du parcours (ou sur plusieurs parcours : les tournois professionnels se jouent sur quatre "tours"). Lors d'une compétition *Match Play*, seulement deux joueurs s'affrontent. À la fin de chaque trou, les deux joueurs comparent leur score. Celui qui a obtenu le plus petit score marque un point. En cas d'égalité, chaque joueur marque un demi point. À la fin des dix-huit trous, le joueur ayant le plus grand nombre de points est déclaré vainqueur. Les enjeux pour le golfeur sont différents dans les deux modes : en Stroke Play, le score d'un joueur sur un trou a un impact direct sur le score final, alors qu'en Match Play, l'enjeu (sur chaque trou) est uniquement de faire un meilleur score que son adversaire.

On s'intéresse dans cette thèse à l'optimisation de stratégie au Golf. Sous certaines hypothèses que nous développerons ultérieurement, il est possible de modéliser le Stroke Play comme un problème de Plus Court Chemin Stochastique (PCCS) et le Match Play comme un jeu de plus court chemin stochastique à deux joueurs.

Stroke Play et Plus Court Chemin Stochastique

Le plus court chemin stochastique (PCCS) est un processus de Markov (MDP) particulier dans lequel un agent évolue dynamiquement sur un ensemble fini d'états. À chaque

période de temps, l'agent choisit une action parmi un ensemble d'actions disponibles qui le mènera aléatoirement dans un autre état suivant une distribution de probabilités connue. Chaque action induit un coût et le but de l'agent est d'atteindre à coup sûr un état puits particulier tout en minimisant le coût moyen des actions qui l'y mènent.

Formellement, une instance de plus court chemin stochastique est un tuple $I = (\mathcal{S}, \mathcal{A}, J, P, c)$, où $\mathcal{S} = \{0, 1, .., n\}$ est un ensemble fini de $n+1$ états, $\mathcal{A} = \{0, 1, .., m\}$ est un ensemble fini de $m+1$ actions, J est une matrice en 0/1 de m lignes et n colonnes. L'élément $J_{a,s} = 1$ si l'action $a \in \{1, .., m\}$ est disponible en l'état $s \in \{1, .., n\}$ et 0 sinon. On suppose sans perte de généralité qu'on peut partitionner $\mathcal{A}$ en $\mathcal{A} = \cup_{s \in \mathcal{S}} \mathcal{A}(s)$, où $\mathcal{A}(s)$ est l'ensemble des actions disponibles en $s \in \mathcal{S}$, ce qui signifie qu'une action n'est disponible qu'en un unique état [1]. P est une matrice sous-stochastique (la somme des éléments d'une colonne est au plus 1) de m lignes et n colonnes appelée matrice de transition. Un élément $P_{a,s} = p(s|a)$ est la probabilité d'atteindre l'état $s \in \{1, .., n\}$ sachant qu'on a effectué l'action $a \in \{1, .., m\}$. c est un vecteur de m éléments et $c(a)$ définit le coût de l'action $a \in \{1, .., m\}$. L'état 0 est un état puits particulier, dans lequel la seule action disponible est l'action 0 : $\mathcal{A}(0) = \{0\}$ et l'action 0 mène en l'état 0 avec probabilité de un.

Une *politique stationnaire* (on définira une notion de politique plus générale dans cette thèse, notamment dans le chapitre 1) est une fonction Π qui associe une distribution de probabilités sur les actions à chaque état. On peut représenter une politique stationnaire par une matrice stochastique (la somme des éléments d'une colonne est exactement égale à 1) de n lignes et m colonnes qui vérifie $\Pi(s, a) > 0 \Rightarrow J_{a,s} = 1$. Si Π est de plus une matrice en 0/1, la politique est dite déterministe. Une politique stationnaire est dite *propre* si $\mathbb{1}^T (P^T \Pi^T)^n . e_i < 1$ (e_i est un vecteur dont les n éléments valent 0, sauf le i^{eme} qui vaut 1). Cela signifie qu'à partir de n'importe quel état i, la probabilité d'atteindre l'état puits 0 après n périodes de temps est strictement positive. Si pour une instance de PCCS toutes les politiques sont propres, on dira qu'il s'agit d'un plus court chemin stochastique avec *terminaison inévitable*. En effet dans ce cas, n'importe quelle stratégie mène au puits avec une probabilité de un.

On peut constater que choisir une politique propre et stationnaire définit une chaîne de Markov absorbante (voir [81] pour plus de détails sur les chaînes de Markov), dont la matrice de transition est $Q^T = P^T \Pi^T$. En particulier, $(I - Q)$ est inversible et $(I - Q)^{-1} = \lim_{K \to +\infty} \sum_{k=0}^{K} Q^k$. On peut donc définir pour chaque $i \in \mathcal{S} \setminus \{0\}$,

$$J_\Pi = \lim_{K \to +\infty} \sum_{k=0}^{K} c^T \Pi^T (I - Q)^{-1} e_i$$

et $J^*(i) = \min\{J_\Pi(i) : \Pi \text{ propre et stationnaire}\}$. Bertsekas et Tsitsiklis [19] donne la définition suivante d'une politique optimale : une politique propre et stationnaire Π^* est dite optimale si pour tout état $i \in \mathcal{S} \setminus \{0\}$, $J_{\Pi^*} = J^*(i)$. Le problème qui consiste à trouver une telle politique s'appelle le problème du plus court chemin stochastique.

Le problème de PPCS apparaît naturellement lorsqu'on s'intéresse à l'optimisation de la stratégie de jeu d'un golfeur. En effet, lorsqu'un golfeur joue en Stroke Play,

[1] On peut faire une telle hypothèse, quitte à dupliquer des actions.

on peut raisonnablement supposer qu'il joue chaque trou indépendamment, sans tenir compte des autres joueurs et qu'il essaye de minimiser son score moyen (puisqu'il réitère le processus 18 fois). Cette hypothèse est discutable, surtout quand on arrive à la fin du parcours (un joueur aura peut-être tendance à prendre plus de risques si il est en retard par rapport aux autres joueurs sur les derniers trous). Cependant, si on considère des joueurs professionnels, ce que nous allons faire dans la suite du document, le niveau des golfeurs est assez uniforme pour qu'il ne soit pas rentable de prendre de tels risques (ou vraiment uniquement dans des situations extrêmes). Si on fait de telles hypothèses, on peut donc considérer qu'en Stroke Play, un joueur ne joue pas contre les autres joueurs, mais contre "le parcours". Optimiser la stratégie d'un golfeur en Stroke Play peut dans ce cadre se modéliser comme un plus court chemin stochastique. En effet, si on définit des états comme étant les endroits possibles où la balle peut se trouver, des actions comme les coups que peut effectuer le golfeur dont les coûts valent 1 (ou 2 si pénalité il y a) et une matrice de transition qui décrit où la balle peut atterrir pour chaque action du golfeur, en prenant en compte à la fois des éléments physiques (obstacles des trous, conditions climatiques, physique de la balle de golf...) et le niveau du joueur en lui même ; alors résoudre l'instance de plus court chemin stochastique ainsi définie donnerait la stratégie optimale que le joueur devrait adopter afin de minimiser son score moyen. De plus, si on suppose qu'un joueur professionnel joue sa stratégie optimale, nous pouvons, en simulant la stratégie optimale calculée, créer un jumeau numérique du joueur qui aurait les mêmes caractéristiques que le joueur. Ce jumeau peut être utilisé pour de la prévision de score, mais aussi pour détecter les points de jeu critiques à améliorer afin d'avoir le meilleur impact sur le score final.

Le plus court chemin stochastique est un problème intéressant à étudier en soit avec de nombreuses applications. Dans la robotique : dans [7], les auteurs décrivent comment manœuvrer un véhicule dans eaux agitées mais également dans la recherche opérationnelle en général [106], en finance de manière générale [8] et dans des modèles d'établissement de prix [76] en particulier ou encore en apprentissage automatique [99]. Il a été introduit pour la première fois par Eaton et Zadeh en 1962 [38], puis très largement étudié par Bertsekas et Tsitsiklis [19]. On connait à ce jour trois principales manières de résoudre exactement le problème (qui sont en fait des méthodes de résolution des processus de Markov) : Value Iteration, Policy Iteration et la programmation linéaire. Value Iteration est un algorithme itératif qui a pour principe d'approcher J^* grâce à la programmation dynamique [10, 58]. Policy Iteration est également un algorithme itératif qui nécessite une stratégie propre initiale et qui itère de stratégie propre en stratégie propre jusqu'à trouver une stratégie optimale [69, 33]. Le problème du plus court chemin stochastique peut également être formulé comme un programme linéaire et peut donc, comme la plupart des processus de Markov, être résolu en temps faiblement polynomial [69, 34, 33, 57, 52]. On pourra noter que Policy Iteration peut être interprété comme un algorithme du simplexe sur un tel programme linéaire.

L'existence d'algorithmes de résolution fortement polynomiaux pour résoudre le PCCS (polynomial en le nombre d'états et d'actions), et plus généralement les processus de Markov, est une question ouverte très importante dans le domaine. Dans le

cas 'avec escompte', c'est-à-dire le cas où il existe une probabilité non-nulle et constante d'atteindre l'état puits 0 en effectuant n'importe quelle action, Ye a été prouvé qu'il existait un algorithme de résolution fortement polynomial à facteur d'escompte fixé[109]. Il a prouvé par la suite que Policy Iteration (la version proposée par Howard [58]) avait une complexité fortement polynomiale [110]. Dans le cas 'sans escompte', cette question est toujours ouverte et (Howard) Policy Iteration a une complexité temporelle exponentielle [46]. Value Iteration a une complexité temporelle exponentielle et ce dans le cas discounted comme dans le cas undiscounted [42]. Même si l'on connait des algorithmes de résolution faiblement polynomiaux dans le cas undiscounted, la plupart des instances de MDP en général et de PCCS en particulier sont résolues grâce à Value Iteration et Policy Iteration quand elles sont de tailles moyennes. Pour de grandes instances, on préfèrera en général approcher les solutions optimales afin d'obtenir des solutions satisfaisantes en un temps raisonnable (en utilisant des méthodes de type programmation dynamique approchée par exemple [87])a.

Match Play et JPCCS

Un jeu de plus court chemin stochastique à deux joueurs est un jeu où deux joueurs, que nous appelons MIN et MAX, ont un but antagoniste. Un agent évolue dynamiquement dans un ensemble d'états partitionné en deux sous-ensembles : un ensemble d'états contrôlés par MIN et les autres contrôlés par MAX. En chaque état, le joueur qui le contrôle choisit une action disponible qui mènera l'agent dans un autre état selon une distribution de probabilités connue. Chaque action induit un coût et le but de MIN (resp. de MAX) est d'atteindre un état puits particulier tout en minimisant (resp. en maximisant) le coût moyen des actions qui l'y mène.

Formellement, une instance de jeu de plus court chemin stochastique (JPCCS) est un tuple $(\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$, où $(\mathcal{S} = \mathcal{S}_{MIN} \cup \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$ est une instance de plus court chemin stochastique et $\mathcal{S}_{MIN} \cap \mathcal{S}_{MAX} = \{0\}$. $\mathcal{S}_{MIN}$ est un ensemble d'états contrôlés par MIN et $\mathcal{S}_{MAX}$ un ensemble d'états contrôlés par MAX. Comme on suppose que chaque action ne peut être effectuée que dans un seul état, on peut également partitionner $\mathcal{A}$ en $\mathcal{A} = \mathcal{A}_{MIN} \cup \mathcal{A}_{MAX}$ avec $\mathcal{A}_i = \{a \in \mathcal{A} | \exists s \in \mathcal{S}_i, J(a,s) = 1\}$ pour $i \in \{MIN, MAX\}$. Une politique déterministe et stationnaire pour MIN (resp. pour MAX) est une fonction $\Pi_{MIN} : \mathcal{S}_{MIN} \to \mathcal{A}_{MIN}$ (resp. $\Pi_{MAX} : \mathcal{S}_{MAX} \to \mathcal{A}_{MAX}$), telle que $\Pi_{MIN}(s) = a \Rightarrow J(a,s) = 1$. Un couple de stratégie $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ définit une politique déterministe et stationnaire pour l'instance de PCCS $(\mathcal{S} = \mathcal{S}_{MIN} \cup \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$. On définit le coût du couple de stratégie $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ comme étant J_Π, le coût de la stratégie Π pour l'instance du PCCS. Soit Σ_{MIN} (resp. Σ_{MAX}) l'ensemble des stratégies déterministes et stationnaires pour MIN (resp. pour MAX).

Considérons maintenant les instances de JPCCS $(\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$ dont l'instance de PCCS associée $(\mathcal{S} = \mathcal{S}_{MIN} \cup \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$ est avec terminaison inévitable. Dans ce cas, on verra qu'il existe un *équilibre de Nash*, c'est-à-dire un couple de stratégies (Π_{MIN}, Π_{MAX}) tel que pour tout $i \in \mathcal{S} \setminus \{0\}$, pour tout $\Pi'_{MIN} \in \Sigma_{MIN}$ et tout

$\Pi'_{MAX} \in \Sigma_{MAX}$, $J_{(\Pi_{MIN},\Pi'_{MAX})} \geq J_{(\Pi_{MIN},\Pi_{MAX})} \geq J_{(\Pi'_{MIN},\Pi_{MAX})}$. Il en résulte du théorème minimax de Von Neumann que, pour tout $i \in \mathcal{S} \setminus \{0\}$:

$$\begin{aligned} J_{(\Pi_{MIN},\Pi_{MAX})}(i) &= \min_{\Pi'_{MIN}\in\Sigma_{MIN}} \max_{\Pi'_{MAX}\in\Sigma_{MAX}} J_{(\Pi'_{MIN},\Pi'_{MAX})}(i) \\ &= \max_{\Pi'_{MAX}\in\Sigma_{MAX}} \min_{\Pi'_{MIN}\in\Sigma_{MIN}} J_{(\Pi'_{MIN},\Pi'_{MAX})}(i) \end{aligned}$$

Un problème de jeux de plus court chemin stochastique consiste à trouver un tel équilibre de Nash.

Lors d'une compétition en Match Play, deux joueurs s'affrontent sur chaque trou du parcours afin de gagner le point lié à ce trou (ou de le partager en cas d'égalité). Le joueur qui commence est celui qui a gagné le trou précédent (et pour le premier trou il est tiré au hasard) et c'est toujours le joueur dont la balle est la plus loin du drapeau qui joue. Quand un joueur joue, les informations dont il dispose sont la position de sa balle, la position de la balle de son adversaire et la différence de score courante entre son adversaire et lui. On nomme MIN le joueur qui commence et MAX son adversaire. Pour un trou, on définit un ensemble d'états $\mathcal{S}$, dont les états ont une forme générique du type $s = (p_{MIN}, p_{MAX}, \delta)$ avec p_{MIN} la position de la balle de MIN sur le trou discrétisé, p_{MAX} la position de la balle de MAX et $\delta \in \mathbb{Z}$ la différence relative de score courante entre MIN et MAX : si l'on appelle $s_{MIN} \in \mathbb{N}$ le score de MIN et $s_{MAX} \in \mathbb{N}$ le score de MAX, on pose $\delta = s_{MIN} - s_{MAX}$. L'état initial est donc $s^{tee} = (p_{tee}, p_{tee}, 0)$ avec p_{tee} correspondant à la position du tee sur le trou. Même si la différence de score ne peut pas être bornée *a priori* (une différence de score quelconque engendrerait un ensemble d'états infini), on peut définir une différence de score $D \in \mathbb{N}$ qui n'a jamais été dépassée dans l'histoire du Golf (pour des joueurs professionnels, D ne dépasse généralement pas 2 ou 3 car en pratique, au delà d'une telle différence de score, le joueur en retard 'donne' le point à son adversaire). Dans la suite, on considèrera uniquement des joueurs professionnels, dont le niveau est assez proche pour que δ soit suffisamment facile à borner. On supposera donc que $\delta \in \{-D, -D+1, ..., D-1, D\}$. Un état $(p_{MIN}, p_{MAX}, \delta)$ est contrôlé par MIN si la distance de p_{MIN} au trou est supérieure à la distance de p_{MAX} au trou et est contrôlé par MAX sinon. Les actions disponibles en un état correspondent aux coups que peut jouer le joueur qui contrôle cet état. Quand MIN joue depuis $s = (p_{MIN}, p_{MAX}, \delta)$, la balle de MIN atterrit sur une autre position p'_{MIN} du trou et le nouvel état est $s' = (p'_{MIN}, p_{MAX}, \delta')$ avec $\delta' = \delta + 1$ ou $\delta' = \delta + 2$ si MIN s'est vu octroyé une pénalité (une pénalité est octroyée dans des cas bien particuliers, qui sont indiqués dans les règles officielles du golf [59]). Si $\delta' > D$, alors le nouvel état est l'état puits (définit plus bas), et le coût engendré par cette transition est de 1. De même, quand MAX joue depuis $s = (p_{MIN}, p_{MAX}, \delta)$, la balle atterrit en p'_{MAX} et le nouvel état après son coup est $s'' = (p_{MIN}, p'_{MAX}, \delta'')$ avec $\delta'' = \delta - 1$ ou $\delta - 2$ si MAX a obtenu une pénalité (de même, si $\delta'' < -D$, alors le nouvel état est l'état puits et le coût engendré est de -1). Ces transitions n'occasionnent aucun coût (à part quand l'état puits est atteint). On définit un état 'drapeau' comme un état du type $s_{flag} = (p_{flag}, p_{flag}, \delta_{flag})$ où p_{flag} correspond à la position du drapeau (il y a donc $2D + 1$ états drapeau). Le jeu s'arrête quand un état drapeau s_{flag} est atteint. MIN gagne le point si $\delta_{flag} < 0$, MAX le gagne si $\delta_{flag} > 0$ et MIN et MAX gagnent chacun $0,5$ point si $\delta_{flag} = 0$. Pour faire correspondre parfaitement le Match Play avec

une instance de JPCCS, on définit un état puits artificiel qui est atteint à partir des états drapeaux de manière sûre et avec un coût de 1 si $\delta_{flag} > 0$, de -1 si $\delta_{flag} < 0$ et de 0 si $\delta_{flag} = 0$ (ainsi que si la différence de score est inférieure strictement à $-D$ ou supérieure strictement à D, comme on l'a vu plus haut). Le coût d'un couple de politique $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ est l'espérance de gain dans $[-1, 1]$ du joueur MAX quand MIN suit la politique Π_{MIN} et MAX suit Π_{MAX} (c'est aussi l'opposé de l'espérance de gain de MIN). Le but de MIN (resp. MAX) est de minimiser (resp. maximiser) ce coût. La notion d'espérance a du sens dans la mesure où les joueurs jouent dix-huit trous. Évidemment, plus on approche de la fin plus l'hypothèse est contestable. On notera que les instances sont à terminaison inévitable car les deux joueurs peuvent terminer indépendamment. Il est donc possible de modéliser une compétition en Match Play comme un problème de JPCCS avec terminaison inévitable. Résoudre une telle instance à l'optimal donne ainsi un couple de stratégies $(\Pi^*_{MIN}, \Pi^*_{MAX})$ qui forme un équilibre de Nash.

Les jeux stochastiques ont été introduits pour la première fois par Shapley en 1953. Ce problème a de nombreuses applications, notamment en sécurité de réseaux [80], en économie [82][60], et en robotique [93] entre autres choses. On ne connait actuellement aucun algorithme de complexité temporelle polynomiale qui résout exactement les JPCCS. L'existence d'un tel algorithme est un problème non-résolu de longue date [30]. L'algorithme utilisé en pratique est Strategy Iteration. Cet algorithme itératif consiste en l'application alternative de Policy Iteration pour un joueur, en fixant la stratégie de l'adversaire jusqu'à obtenir un équilibre de Nash. Cet algorithme est de complexité temporelle exponentielle dans le pire des cas (qui résulte de la complexité exponentielle de Policy Iteration) [31].

Base de données Shotlink

Nous avons vu que le Stroke Play comme le Match Play peuvent se modéliser comme un plus court chemin stochastique ou un jeu de plus court chemin stochastique. Tandis que les états, les actions disponibles dans chaque état et les coûts de ces actions sont relativement naturels à définir, la matrice de transition, elle, est plus dure à caractériser. Elle décrit les probabilités d'atteindre les états quand on effectue une action. Elle renferme donc la complexité de la topologie du trou et surtout relate du niveau de précision des coups du joueur. Afin de construire cette matrice de transition, nous avons donc besoin de données relatives au joueur. Pour ce faire, nous avons utilisé une très grande base de données Américaine qui recense des millions d'informations relatives aux coups joués par des joueurs professionnels internationaux dans les compétitions Américaines, telles que les coordonnées de départ et d'arrivée de la balle, le type de terrain sur lequel le joueur a tiré, etc... [86].

Organisation du document

Dans le premier chapitre, nous étudions le problème du plus court chemin stochastique d'un point de vue théorique. Nous étendons le cadre d'étude de Bertsekas et Tsitsiklis [19] puis de Bertsekas et Yu [20] en adoptant un point de vue polyédral sur le problème. Nous prouvons que dans ce nouveau cadre, les algorithmes classiques de résolution (Value Iteration, Policy Iteration) convergent même en présence de cycles de transition (généralisation des cycles dans le cas stochastique) de coûts nuls. Nous introduisons également un nouvel algorithme de résolution. Ce chapitre a fait l'objet d'une publication dans le journal européen de recherche opérationnel (EJOR) [48].

Dans le deuxième chapitre, nous nous intéressons à la modélisation de l'optimisation des stratégies de Golf en plus court chemin stochastique, ainsi qu'à des méthodes de prédiction de score. Nous présentons notre modèle, ainsi que des résultats numériques et leur validation statistique. Les travaux correspondants ont été présentés dans des conférences internationales [71] [72], et un article de journal est en préparation.

Dans le troisième et dernier chapitre, nous nous intéressons aux jeux stochastiques et plus particulièrement aux formulations programmation linéaire de ces jeux. Nous présentons une formulation programmation linéaire en nombres entiers pour les JPCCS. Nous étudions également deux cas particuliers des JPCCS : les Jeux Stochastiques Simples et le problème du Plus Court Chemin Robuste. Les travaux de ce chapitre n'ont pas encore été présentés en conférence ou en journal, mais ont vocation à l'être.

Prérequis théoriques

Dans cette section, nous introduisons les prérequis théoriques nécessaires à la bonne compréhension de cette thèse. Nous y faisons référence tout au long de celle-ci. Nous introduisons également ici les notations que nous allons utiliser dans le document. La plupart des notations sont tirées de [49]. Nous invitons les lecteurs à s'y référer pour plus de détails et pour accéder aux preuves manquantes.

Programmation Linéaire

La programmation linéaire est un cas particulier de programmation mathématique. Un programme mathématique est une modélisation d'un problème d'optimisation de la forme :

$$\begin{array}{lll} \text{Minimiser ou Maximiser} & f(x) & \\ \text{Sachant que} & g_i(x) & \left\{ \begin{array}{c} \geq \\ \leq \\ = \end{array} \right\} \ 0 \quad \forall i \in \{1,..,m\} \\ & x \in X & \end{array} \qquad (ProgMath)$$

où $m \in \mathbb{N}$, $x \in \mathbb{R}^n$ ($n \in \mathbb{N}$) sont les variables, X est le domaine de définition de x, $f : \mathbb{R}^n \to \mathbb{R}$ est la fonction objectif et pour tout $i \in \{1, .., m\}, g_i : \mathbb{R}^n \to \mathbb{R}$ est une fonction de contrainte.

Dans le cas où f et $(g_i)_{i\in\{1,..,m\}}$ sont des fonctions linéaires et $X = \mathbb{R}^n$, on parle alors de programmation linéaire. Il s'agit donc de programmes mathématiques du type :

$$\begin{array}{lll} \text{Min/Max} & \sum_{j=1}^{n} c_j x_j & \\ \text{Sachant que} & \sum_{j=1}^{n} a_{ij} x_j - b_i \left\{ \begin{array}{l} \geq \\ \leq \\ = \end{array} \right\} 0 \quad \forall i \in \{1, .., m\} & (ProgLin) \\ & x \in \mathbb{R}^n & \end{array}$$

où pour tout $i \in \{1, .., m\}$ et tout $j \in \{1, .., n\}$, $c_j, a_{ij}, b_i \in \mathbb{R}$. De manière plus condensée, on utilisera une notation matricielle en posant $A = (a_{ij})_{1\leq i\leq m, 1\leq j\leq n}$, $c = (c_j)_{j\in\{1,..,n\}}$ et $b = (b_i)_{i\in\{1,..,m\}}$.

$$\begin{array}{lll} \text{Min/Max} & c^T x & \\ \text{Sachant que} & Ax - b \left\{ \begin{array}{l} \geq \\ \leq \\ = \end{array} \right\} 0 & (ProgLin) \\ & x \in \mathbb{R}^n & \end{array}$$

En notant que minimiser une fonction est équivalent à maximiser son opposé et en transformant les inégalités $A_i x \geq b_i$ en $-A_i x \leq -b_i$, où A_i est la i^{eme} ligne de A, on peut supposer sans perte de généralité que n'importe quel programme linéaire peut s'écrire sous la forme suivante (appelée forme *canonique*) :

$$\begin{array}{lllll} \text{Max} & c^T x & & & \\ \text{Sachant que} & Ax & \leq & b & (P_{Can}) \\ & x & \geq & 0 & \end{array}$$

Definition 1
Un polyèdre convexe est l'ensemble des solutions d'un système fini d'inégalités linéaires.

Definition 2
Soit $\bar{x} \in \mathbb{R}^n$ et (P) un programme linéaire sous forme canonique. On dit que $\bar{x}$ est une solution réalisable de (P) si et seulement si $A\bar{x} \leq b$ et $\bar{x} \geq 0$. On nomme $P = \{x \in \mathbb{R}^n | x \geq 0, Ax \leq b\}$ l'ensemble des solutions réalisables de (P). Par définition, P est un polyèdre convexe. On appellera valeur de $\bar{x}$ la valeur de la fonction objectif appliquée à x, i.e. $c^T\bar{x}$.

Definition 3
Soit $x^* \in \mathbb{R}^n$ et (P) un programme linéaire sous forme canonique. On dit que x^* est une solution optimale de (P) si et seulement si x^* est une solution réalisable et que pour tout $\bar{x} \in P$, $c^T x^* \geq c^T\bar{x}$. On nommera valeur optimale de (P) la valeur de x^*, i.e. $c^T x^*$.

Definition 4
Soit (P) un programme linéaire sous forme canonique.

- *(P) est dit irréalisable si et seulement si $P = \emptyset$*
- *(P) est dit non-borné si seulement si $\forall M \in \mathbb{R}$, $\exists \bar{x} \in P$, $c^T \bar{x} \geq M$*

Proposition 5
Soit (P) un programme linéaire. Soit (P) admet une solution optimale, soit il est irréalisable, soit il est non-borné.

Definition 6
Soit $\bar{x} \in \mathbb{R}^n$ et (P) un programme linéaire sous forme canonique. $\bar{x}$ est un sommet de P si $\bar{x} \in P$, et que pour tout $x, x' \in P$, $\bar{x} = \frac{x+x'}{2} \Rightarrow \bar{x} = x = x'$.

Definition 7
On dit qu'un programme linéaire est sous forme standard si il est de la forme

$$\begin{array}{lllll} Max & c^T x & & & \\ Sachant\ que & Ax & = & b & \\ & x & \geq & 0 & \end{array} \qquad (P_{Stand})$$

avec $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, et $A = (a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$ et $b \in \mathbb{R}^m$

Sans perte de généralité, on peut supposer que dans un problème sous forme standard, les colonnes de A sont linéairement indépendantes (si ce n'est pas le cas, il y a des contraintes redondantes ou l'ensemble des contraintes est vide). On supposera donc dans la suite que pour un programme linéaire sous forme standard, A est de rang plein, i.e. $rang(A) = m$.

On remarquera qu'à tout programme linéaire sous forme canonique, on peut associer un programme linéaire sous forme standard. Soit (P) un programme linéaire sous forme canonique:

$$\begin{array}{lllll} \text{Max} & c^T x & & & \\ \text{Sachant que} & Ax & \leq & b & \\ & x & \geq & 0 & \end{array} \qquad (P)$$

avec $c \in \mathbb{R}^{n'}$, $x \in \mathbb{R}^{n'}$, et $A = (a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n'}$ et $b \in \mathbb{R}^m$.

On pose $n = n' + m$ et on définit $\tilde{x} \in \mathbb{R}^n$, $\tilde{c} \in \mathbb{R}^n$, $\tilde{A} = (\tilde{a}_{ij})_{1 \leq i \leq n, 1 \leq j \leq m}$ et $\tilde{b} \in \mathbb{R}^m$ tels que:

- $\tilde{x} = (x_1, .., x_{n'}, , .., b_1 - A_1 x, .., b_m - A_m x)$
- $\tilde{c} = (c_1, .., c_{n'}, 0, .., 0)$
- $\tilde{A} = (A | I_m)$
- $\tilde{b} = b$

avec I_m la matrice identité de taille m et $(A|I_m) = \begin{pmatrix} a_{11} & ... & a_{1n'} & 1 & 0 & 0 & ... & 0 \\ a_{21} & ... & a_{2n'} & 0 & 1 & 0 & ... & 0 \\ . & & & & & & & . \\ . & & & & & & & . \\ . & & & & & & & . \\ . & & & & & & & . \\ a_{m1} & ... & a_{mn'} & 0 & 0 & ... & 0 & 1 \end{pmatrix}$

On considère le programme linéaire (sous forme standard) suivant :

$$\begin{array}{lll} \text{Max} & \tilde{c}^T\tilde{x} & \\ \text{Sachant que} & \tilde{A}\tilde{x} = \tilde{b} & \\ & \tilde{x} \geq 0 & \end{array} \qquad (\tilde{P})$$

On peut démontrer facilement que

- x est une solution réalisable de (P) si et seulement si $\tilde{x}$ est une solution réalisable de $(\tilde{P})$
- x est une solution optimale de (P) si et seulement si $\tilde{x}$ est une solution optimale de $(\tilde{P})$

On note que $\tilde{A}$ 'contient' la matrice identité I_m, donc $\tilde{A}$ est de rang plein. Dans toute la suite, on considèrera que les programmes linéaires en forme standard possèdent n variables et m contraintes.

Definition 8
Soit (P) un programme linéaire sous forme standard. On supposera sans perte de généralité que A est de rang plein (comme expliqué plus haut). On dit que $B \subseteq \{1,..,n\}$ est une base de (P) si et seulement si $|B| = m$ et $A_B = (a_{ij})_{1\leq i\leq m, j\in B}$ est inversible. Pour une base B, on appelle $N = \{1,..,n\} \setminus B$.

Les variables x_i avec $i \in B$ sont appelées les variables de base et les variables x_i avec $i \in N$ sont appelées variables hors-base.

Definition 9
Soit (P) un programme linéaire sous forme standard et B une base de (P). On définit $A_B = (a_{ij})_{1\leq i\leq m, j\in B}$, $A_N = (a_{ij})_{1\leq i\leq m, j\in N}$, $c_B = (c_j)_{j\in B}$, $c_N = (c_j)_{j\in N}$, $x_B = (x_j)_{j\in B}$ et $x_N = (x_j)_{j\in N}$.

Definition 10
Soit (P) un programme linéaire sous forme standard et B une base. (P) peut alors se réécrire :

$$\begin{array}{lll} \textit{Max} & c_B^T x_B + c_N^T x_N & \\ \textit{Sachant que} & x_B = A_B^{-1}b - A_B^{-1}A_N x_N & \\ & x_B, x_N \geq 0 & \end{array} \qquad (P^B_{Stand})$$

avec $x_B \in \mathbb{R}^m$ et $x_N \in \mathbb{R}^{n-m}$ (on peut toujours supposer que $n > m$, sinon l'ensemble des solutions réalisables est réduit à un point, car le rang de A est plein). L'unique vecteur

$\tilde{x} \in \mathbb{R}^n$ solution de (P^B_{Stand}) avec $\tilde{x}_N = 0$ est appelé solution de base B. De plus, si $\tilde{x}$ est réalisable, on parlera de solution de base réalisable et B sera appelée base réalisable.

Proposition 11
Soit (P) un programme linéaire sous forme standard et $x \in \mathbb{R}^n$. x est un sommet de P si et seulement si x est une solution de base de (P).

Méthodes de résolution et complexité

Il existe trois principaux types d'algorithmes permettant de résoudre de manière exacte les programmes linéaires : les méthodes *d'ellipsoïdes* [65], les méthodes de *points intérieurs* [62] et le *simplexe* que nous allons détailler car il est au cœur de certaines méthodes de résolution des plus courts chemins stochastiques et de variantes de ceux-ci.

Les méthodes de points intérieurs et d'ellipsoïdes sont des méthodes polynomiales [65, 62], mais ne sont pas fortement polynomiales (polynomial en le nombre de variables et de contraintes). Pour plus de détails sur ces méthodes, nous invitons les lecteurs à lire [94, 77, 108]. La résolution des programmes linéaires est donc un problème qui peut être résolu en temps (faiblement) polynomial. Cependant, un des algorithmes les plus utilisés en pratique n'est lui pas polynomial (comme nous allons le voir plus loin), il s'agit de l'algorithme du simplexe.

L'algorithme du simplexe a été créé par Dantzig en 1947 (et publié en 1951 [32]). L'idée générale de cet algorithme est de se déplacer de sommet en sommet du polyèdre des solutions réalisables afin de trouver une solution optimale s'il en existe. Cependant, il peut arriver que l'on reste sur le même sommet si plus de 3 contraintes concourent en un même point. On dit alors que (P) est dégénéré. Cela peut engendrer des difficultés au niveau de l'exécution de l'algorithme du simplexe et leur résolution est un pan important de la recherche en programmation linéaire [27, 107].

Definition 12
Soit (P) un programme linéaire sous forme standard et B une base de (P). Le vecteur de coûts réduits lié à la base B est définit par :

$$\bar{c}^B = c - c_B A_B^{-1} A$$

Pour une variable x_i, $i \in \{1, .., n\}$, le coût réduit de x_i est donc $(\bar{c}^B)_i$. On notera que pour toute variable de base x_i, on a $(\bar{c}^B)_i = 0$.

Proposition 13
Soit (P) un programme linéaire sous forme standard, B une base de (P) et $\tilde{x}_B$ la solution de base associée à B. Soit $x \in \mathbb{R}^n$ une solution réalisable de (P). On a :

$$c^T x = c_B^T \tilde{x}_B + (\bar{c}^B)^T x$$

Ainsi, si on dispose d'une base réalisable, comme on a $(\bar{c}^B)^T x = (\bar{c}^B)_B^T x_B + (\bar{c}^B)_N^T x_N = (\bar{c}^B)_N^T x_N$, un programme linéaire (P) peut se réécrire :

$$\begin{array}{lllll} \text{Max} & c_B^T \tilde{x}_B + (\bar{c}^B)_N^T x_N & & & \\ \text{Sachant que} & x_B & = & A_B^{-1} b - A_B^{-1} A_N x_N & (P^B_{Stand}) \\ & x_B, x_N & \geq & 0 & \end{array}$$

On sait par définition que $\tilde{x}$, la solution de base associée à B, est une solution réalisable, car B est une base réalisable. On considère cette solution comme la solution courante.

On constate ensuite que dans la fonction objectif, $c_B^T \tilde{x}_B$ est une constante. Il convient donc de maximiser $(\bar{c}^B)_N^T x_N$. On peut interpréter $(\bar{c}^B)_i$ comme le gain relatif pour la fonction objectif si on augmente la valeur de la variable x_i d'une unité.

Si pour tout $i \in N$, $(\bar{c}^B)_i \leq 0$, alors comme $x_N \geq 0$, $\tilde{x}$ est optimale car elle est réalisable et que $\tilde{x}_N = 0$. En effet, toute autre solution $x \in \mathbb{R}^n$ avec $x_i \neq 0$ pour un certain $i \in N$ aura une valeur de fonction objectif plus faible.

Sinon, si il existe $j \in N$ tel que $(\bar{c}^B)_j > 0$, alors on aura tendance à vouloir augmenter la valeur de x_j afin d'obtenir une valeur plus grande que celle de la solution courante (laissant les autres valeurs de $x_{j'}$ égales à 0 pour tout $j' \in N \setminus \{j\}$). Pour tout $i \in B$, on a :

$$x_i = (A_B^{-1} b)_i - (A_B^{-1} A_j x_j)_i$$

Sachant qu'on veut assurer la réalisabilité du problème, on veut s'assurer que $x_i \geq 0$ pour tout $i \in B$. Si on a $(A_B^{-1} A_j x_j) \leq 0$, alors le problème est non borné, car on pourra augmenter indéfiniment la valeur de x_j sans compromettre la réalisabilité de la solution. Sinon, on définit :

$$i^* \in argmin_{i \in B, (A_B^{-1} A_j x_j)_i > 0} \frac{(A_B^{-1} b)_i}{(A_B^{-1} A_j x_j)_i}$$

On nomme x_j la variable entrante et x_i^* la variable sortante. On définit une nouvelle base $B' = B \cup \{j\} \setminus \{i^*\}$ (on a donc $N' = N \cup \{i^*\} \setminus \{j\}$). D'après la définition du coût réduit, on a bien $c^T \tilde{x}_{B'} \geq c^T \tilde{x}_B$.

L'algorithme du simplexe est l'algorithme itératif qui consiste à répéter cette étape jusqu'à ce que tous les coûts réduits des variables hors-base soient négatifs :

Algorithm 1 L'algorithme du Simplexe

Input : A, b, c et une base réalisable B
  while $\exists j \in N$, $(\bar{c}^B)_j > 0$ **do**
    Choisir un tel j
    if $\forall i \in B, (A_B^{-1} A_j x_j)_i \leq 0$ **then**
      return Problème non-borné
    else
      Choisir $i^* \in argmin_{i \in B, (A_B^{-1} A_j x_j)_i > 0} \frac{(A_B^{-1} b)_i}{(A_B^{-1} A_j x_j)_i}$
      $B = B \cup \{j\} \setminus \{i^*\}$
    end if
  end while
  return $\tilde{x}_B$

Une itération du simplexe s'effectue en un temps de $O(m^2+n)$ et il existe un nombre fini de bases (maximum $\binom{m}{n+m}$). Si on s'assure de ne parcourir qu'une seule fois chaque base on peut atteindre une solution optimale (ou on établit que le problème n'est pas borné) en un temps d'au plus $O((m^2+n)\binom{m}{n+m})$. C'est le cas de la règle de Bland.

- Règle de Bland : on définit au préalable un ordre sur les variables et quand il y a plusieurs variables de coût réduit positif, la variable entrante choisie est celle de plus petit rang dans cet ordre. Cette règle permet de ne pas 'boucler' sur un sous-ensemble de bases [21], mais le nombre d'itération est exponentiel dans le pire cas [5].
- Règle de Dantzig : on définit la variable entrante comme celle qui a le plus grand coût réduit. Cette règle, bien que plus naturelle, peut ne pas faire terminer l'algorithme du simplexe (on cycle sur un sous-ensemble de bases) [66].

À l'heure actuelle, on ne connait aucune règle de pivot qui garantit une complexité temporelle polynomiale (toutes les règles de pivot connues nécessitent de parcourir un nombre exponentiel de solutions de base).

Dualité

Considérons un programme linéaire sous forme canonique (P) :

$$\begin{array}{lllll} \text{Max} & c^T x & & & \\ \text{Sachant que} & Ax & \leq & b & \\ & x & \geq & 0 & \end{array} \qquad (P)$$

Considérons que (P) est réalisable. Il est clair que pour toute solution réalisable $x_r \in \mathbb{R}^n$, $c^T x_r$ est une borne inférieure de la valeur optimale de (P). Il est légitime de

se poser la question de l'existence d'une borne supérieure sur la valeur d'un programme linéaire. C'est la dualité qui va nous donner les outils permettant de trouver une telle borne.

Supposons que pour tout $y \in (\mathbb{R}^+)^m$ on ait $A^T y \geq c$. Dans ce cas, pour tout $x \in \mathbb{R}^n$ solution réalisable de (P), on a :

$$b^T y \geq (Ax)^T y = x^T A^T y \geq x^T c = c^T x$$

Ainsi, si $A^T y \geq c$, alors $y^T b$ est une borne supérieure sur la valeur de n'importe quelle solution réalisable, donc également sur la valeur optimale de (P). On a ainsi potentiellement une infinité de bornes supérieures disponibles. On voudrait connaître la meilleure borne possible, c'est-à-dire la plus petite d'entre elles.

$$\begin{array}{lllll} \text{Min} & b^T y & & & \\ \text{Sachant que} & A^T y & \geq & c & \\ & y & \geq & 0 & \end{array} \qquad (D)$$

(D) est un programme linéaire appelé le programme linéaire dual de (P) ou plus simplement le *dual* de (P). On appellera (P) le primal de (D). Une solution réalisable de (P) est appelée une solution *primal-réalisable* et une solution réalisable de (D) est appelée une solution *dual-réalisable*. On peut noter que le dual de (D) est (P).

On cite trois importants théorèmes : le théorème de la dualité faible, de la dualité forte, et le théorème des écarts complémentaires (pour plus de détails voir par exemple [84]).

Théorème 14 Dualité faible
Soit (P) un programme linéaire sous forme canonique et (D) son dual. Soit $x \in \mathbb{R}^n$ une solution primal réalisable et $y \in \mathbb{R}^m$ une solution dual réalisable.

- *on a $c^T x \leq y^T b$*
- *si (P) est non borné, (D) est irréalisable*
- *si (D) est non borné, (P) est irréalisable*

Théorème 15 Dualité forte
Soit (P) un programme linéaire sous forme canonique et (D) son dual. (P) admet une solution optimale x^* si et seulement si (D) admet une solution optimale y^*. De plus, si x^* est une solution optimale pour (P) et y^* est une solution optimale pour (D), alors $c^T x^* = b^T y^*$.

Théorème 16 Écarts complémentaires
Soit (P) un programme linéaire sous forme canonique et (D) son dual. Soit $x^* \in \mathbb{R}^n$ et $y^* \in \mathbb{R}^m$. x^* et y^* sont des solutions optimales de (P) et (D), respectivement, si et seulement si :

i x^* est primal-réalisable et y^* est dual-réalisable

ii $(A^T y^* - c)^T x^* = 0$

iii $(Ax^* - b)^T y^* = 0$

Programmation Linéaire en Nombres Entiers

Soit (P) un programme linéaire sous forme canonique. On impose maintenant que les variables ne soient plus dans $\mathbb{R}$ mais dans $\mathbb{Z}$. Les méthodes de résolutions précédentes ne fonctionnent plus. Les méthodes de points intérieurs, les méthodes d'ellipsoïdes ainsi que l'algorithme du simplexe ne donnent pas des solutions entières dans le cas général. Cependant, on peut s'intéresser à l'enveloppe convexe des solutions entières $P_I = conv(\{x \in \mathbb{Z}^n | Ax \leq b\})$ (voir figure 1).

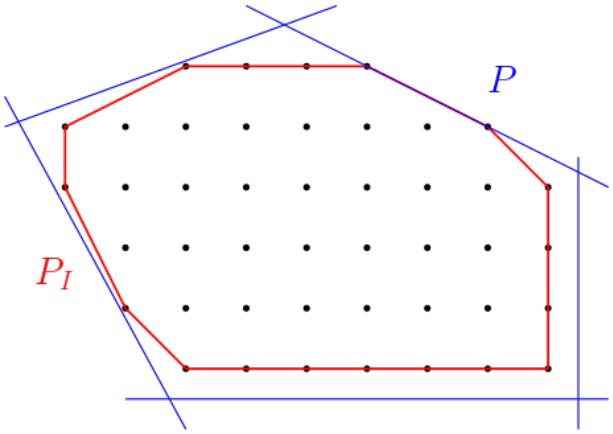


Figure 1 – Exemple d'un polyèdre et du polyèdre des solutions entières associé

Ce polyèdre est difficile à caractériser en cas général. Si malgré tout on arrive à bien caractériser ce polyèdre, des méthodes de type ellipsoïde peuvent donner des résultats intéressants, comme dans le cas du polyèdre des couplages dans un graphe [47]. D'autre part, il existe des cas particulier dans lequel on a $P = P_I$. C'est notamment le cas si la matrice A est totalement unimodulaire (TU).

Definition 17
Une matrice A est dite totalement unimodulaire si le déterminant de toute sous-matrice carrée de A vaut -1, 0 ou 1 (en particulier tous les coefficient de A sont dans $\{-1, 0, 1\}$).

Dans les formulation programmation linéaire des plus courts chemins, ou plus généralement pour les flots, la matrice définissant les contraintes est totalement unimodulaire [45]. Lorsque $P = P_I$, l'algorithme du simplexe donne une solution optimale entière, puisque les sommets de P ont des coordonnées entières.

Cependant dans le cas général, la programmation linéaire en nombre entier est un problème NP-complet [63]. Il existe cependant des algorithmes de résolution qui

s'exécutent en temps exponentiel dans le pire cas : on citera les algorithmes de type Cutting plane [64], Branch & Bound [84], et Branch & Cut [83].

Le Problème du Plus Court Chemin Déterministe

Definition 18
Un graphe orienté est un couple $G = (V, A)$ où $V = \{s_1, s_2, .., s_n\}$ est un ensemble fini de n sommets et $A \subseteq V \times V$ est un ensemble d'arcs. De plus, on dit que G est pondéré s'il existe une application $c : A \longmapsto \mathbb{R}$ appelée fonction de coûts.

Definition 19
Soit $G = (V, A)$ un graphe orienté. On définit pour tout $v \in V$, $\delta^+(v) = \{a \in A | \exists v' \in V, a = (v, v')\}$. On définit également pour tout $v \in V$ $N(v) = \{v' \in V | \exists a = (v, v') \in A\}$ l'ensemble des voisins de v.

Definition 20
Soit $G = (V, A)$ un graphe orienté, c une fonction de coûts et $s, t \in V$. Un chemin de s à t ou (s, t)-chemin est une suite alternée de sommets et d'arcs $p = (v_0 a_1 v_1 a_2 ... a_k, v_k)$ telle que $v_0 = s$, $v_k = t$ et pour tout $i \in \{1, .., k\}$, $a_i = (v_{i-1}, v_i)$. Le coût d'un chemin est la somme des coûts des arcs qui le composent : $c(p) = \sum_{i=0}^{n} c(a_i)$ (on notera par abus de notation $c(p)$ le coût d'un chemin). Un circuit est un chemin $p_c = (v'_0 a'_1 v'_1 a'_2 ... a'_k, v'_k)$ tel que $v'_0 = v'k$.

Soit $G = (V, A)$ un graphe orienté, c une fonction de coûts, ainsi que deux sommets particuliers $s, t \in V$ qu'on appellera respectivement l'origine et la destination. Le problème du plus court chemina (PCC) de s à t est le problème consistant à trouver un $(s-t)$-chemin de coût minimum. On présente en figure 2 un exemple d'instance de PCC, dans lequel les sommets sont représentés par des cercles comprenant leurs étiquettes en leur centre, un arc $a = (i, j)$ est représenté par une flèche allant de i à j et le coût d'un arc est indiqué à proximité de l'arc correspondant.

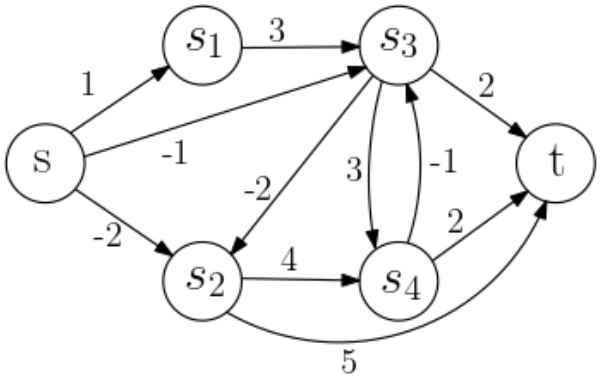


Figure 2 – Représentation graphique d'une instance de PCC

Il existe une solution au PCC si et seulement si (i) il existe un (s,t)-chemin dans G et (ii) il n'existe pas de circuit de coût strictement négatif (appelé aussi circuit absorbant).

Il existe différents algorithmes de résolution du problème du plus court chemin. Dans les cas où le graphe ne possède pas de circuit, l'algorithme de Bellman permet de trouver un plus court chemin d'un graphe en un temps polynomial en le nombre de sommets et d'arcs[11]. En effet, lorsque le graphe ne possède pas de circuit, il existe un ordre topologique sur les sommets du graphe.

Definition 21
Soit $G = (V, A)$ un graphe orienté. Un ordre topologique $\mathcal{O}$ sur V est un ordre $(v_1, v_2, ..., v_n)$ sur les sommets tel que pour tout $k, k' \in \{1, .., n\}$, $(v_k, v_{k'}) \in A \Rightarrow k < k'$.

Proposition 22
Soit $G = (V, A)$ un graphe orienté. Il existe un ordre topologique sur V si et seulement si G est sans circuit.

Dans le cas où tous les coûts des arcs sont positifs, l'algorithme de Dijkstra résout le problème en un temps polynomial également [35]. Dans le cas général, l'algorithme de Bellman-Ford trouve une solution optimale s'il en existe et trouve un circuit absorbant le cas échéant [45].

Dans le chapitre 1, nous détaillons une extension de l'algorithme de Dijkstra au cas du plus court chemin stochastique. Nous rappelons donc l'algorithme pour les PCC. Soit $G = (V, A)$ un graphe orienté et c une fonction de coût telle que pour tout $a \in A$, $c(a) \geq 0$. Soit $s \in V$, l'algorithme de Dijkstra calcule itérativement une s-arborescence de coût minimal.

Definition 23
Soit $G = (V, A)$ un graphe orienté et c une fonction de coût sur A. Soit $s \in V$, une s-arborescence est une union de chemins de s à v pour tout $v \in V$ sans circuit.

Une s-anti-arborescence est une union de $(v - s)$-chemins pour tout $v \in V$ sans circuit.

Le coût d'une s-arborescence ou d'une s-anti-arborescence est la somme des coûts des arcs qui la compose.

Au fur et à mesure des itérations, l'algorithme met à jour un ensemble Y de sommets (initialement vide), et des 'étiquettes' $y : V \longmapsto \mathbb{R}^+$ sur les sommets de G. Y représente l'ensemble des sommets pour lesquels la valeur de l'étiquette ne changera plus, et pour tout $v \in V$, $y(v)$ représente le coût du plus court (s, v)-chemin ne passant que par des sommets de Y. L'algorithme itère tant que $Y \neq V$. A chaque itération, l'algorithme va choisir un sommet z^* dont l'étiquette courante est minimale et qui n'est pas dans Y. Par minimalité des étiquettes et comme les coûts des arcs sont positifs ou nuls, on sait que l'étiquette de z^* représente le coût du plus court chemin de s à z^*. En effet, il ne serait pas 'rentable' de passer par des sommets qui ne sont pas dans Y. Ensuite, on place z^* dans Y et on met à jours les étiquettes de tous voisins v de z^* s'il est plus intéressant que le (s, v)-chemin passe par z^*. On détaille l'algorithme de Dijkstra.

Algorithm 2 Dijkstra Algorithm

$Y = \emptyset$
$y(s) = 0$, $y(v) = \infty \ \forall v \in V \setminus \{s\}$
while $Y \neq V$ **do**
  Choisir z^* tel que $y(z^*) = \min_{z \in V} y(z)$
  $Y \leftarrow Y \cup \{z^*\}$
  for all $v \in N(z^*)$ **do**
    if $y(v) > y(z^*) + c((z^*, v))$ **then**
      $y(v) = y(z^*) + c((z^*, v))$
    end if
  end for
end while

Soit $n = |V|$ et $m = |A|$, l'algorithme de Dijkstra est polynomial : il s'exécute en un temps de $O(n^2+m)$ L'algorithme peut être implémenté plus efficacement et ne nécessiter que $O(m + n\log(n))$ opérations [111].

Une manière de voir le problème du plus court chemin déterministe (d'ailleurs adoptée par Ford dans [45]) est de voir le problème comme un problème de flot particulier. L'idée est de voir un plus court chemin de s à t comme un flot réalisable de s à t de coût minimum. On peut donc formuler le problème grâce à un programme linéaire.

On définit un vecteur $x \in \mathbb{R}^m$ sur les arcs, ainsi que la matrice d'adjacence $(A^G_{ij})_{i,j \in \{1..n\} \times \{1,..,m\}}$ de G. Cette matrice possède n lignes et m colonnes et pour chaque arc $j = (i, i')$, $A^G_{ij} = 1$, $A^G_{i'j} = -1$ et $A^G_{i''j} = 0$ pour tout autre $i'' \neq i$, $i'' \neq i'$. On définit le programme linéaire suivant :

$$
\begin{array}{lll}
\text{Min} & c^T x & \\
\text{Sachant que} & A^G_i x = \begin{cases} 1 \text{ si i = s} \\ -1 \text{ si i = t} \\ 0 \text{ sinon} \end{cases} & \text{pour tout } i \in \{1, .., n\} \\
& x \geq 0 &
\end{array}
\qquad (P^G_{PCC})
$$

Le dual de (P^G_{PCC}) est (D^G_{PCC}) :

$$
\begin{array}{lllll}
\text{Max} & y_s & & & \\
\text{Sachant que} & y_j - y_i & \leq & c_a & \text{pour tout } a = (i, j) \in A \\
& y_t & = & 0 & \\
& y & \in & \mathbb{R}^n &
\end{array}
\qquad (D^G_{PCC})
$$

On peut prouver que dans ce cas particulier, on peut trouver une solution optimale entière si elle existe [45], ce qui définit un plus court chemin de s à t. En effet, la matrice A^G est totalement unimodulaire.

Chapter 1

The Stochastic Shortest Path Problem: A polyhedral combinatorics perspective

The following chapter has been taken from an article that has been already published in the European Journal of Operational Research (EJOR) in 2018 [48]. Consequently, this chapter can be read independently from the other chapters. However, it contains all the formal definition of Stochastic Shortest Path problem needed in the rest of this manuscript.

1.1 Introduction

The Stochastic Shortest Path problem (SSP) is a Markov Decision Process (MDP) that generalizes the classic deterministic shortest path problem. We want to control an agent, who evolves dynamically in a system composed of different *states*, so as to converge to a predefined *target*. The agent is controlled by taking *actions* in each time period[1]: actions are associated with costs and transitions in the system are governed by probability distributions that depend exclusively on the previous action taken and are thus independent of the past. We focus on finite state/action spaces: the goal is to choose an action for each state, i.e., a *deterministic and stationary policy*, so as to minimize the total expected cost incurred by the agent before reaching the (absorbing) target state, when starting from a given initial state.

More formally, a stochastic shortest path instance is defined by a tuple $(\mathcal{S}, \mathcal{A}, J, P, c)$ where $\mathcal{S} = \{0, 1, \ldots, n\}$ is a finite set of *states*, $\mathcal{A} = \{0, 1, \ldots, m\}$ is a finite set of

[1] We focus here on discrete time (infinite) horizon problems.

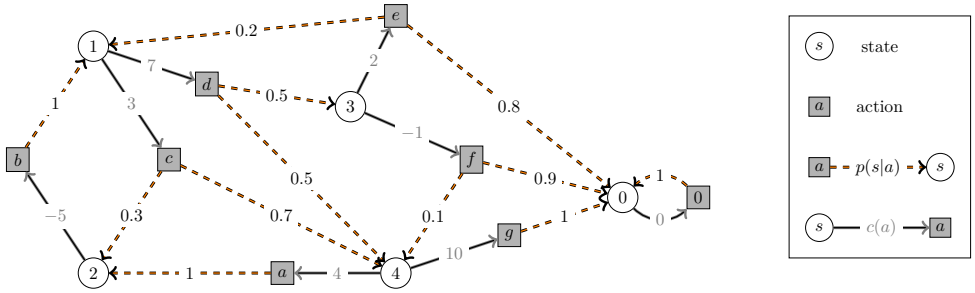


Figure 1.1 – A graphical representation of a SSP (with target state 0): circles are states, squares are actions, dashed arrows indicate state transitions (probabilities) for a given action, and black edges represent actions available in a given state with corresponding cost.

actions, J is a 0/1 matrix with m rows and n columns and general term $J(a,s)$, for all $a \in \{1,...,m\}$ and $s \in \{1,...,n\}$, with $J(a,s) = 1$ if and only if action a is available in state s, P is a *row substochastic matrix*[2] with m rows and n columns and general term $P(a,s) := p(s|a)$ (probability of ending in s when taking action a), for all $a \in \{1,...,m\}$, $s \in \{1,...,n\}$, and a cost vector $c \in \mathbb{R}^m$. The state 0 is called the *target* state and the action 0 is the unique action available in that state. Action 0 leads to state 0 with probability 1. When confusion may arise, we denote state 0 by $0_{\mathcal{S}}$ and action 0 by $0_{\mathcal{A}}$. A *row substochastic* matrix is a matrix with nonnegative entries so that every row adds up to at most 1. We denote by $\mathcal{M}_{\leq}(l,k)$ the set of all $l \times k$ row substochastic matrices and by $\mathcal{M}_{=}(l,k)$ the set of all *row stochastic* matrices (*i.e.* for which every row adds up to exactly 1). In the following, we denote by $\mathcal{A}(s)$ the set of actions available from $s \in \{1,...,n\}$ and we assume without loss of generality[3] that for all $a \in \mathcal{A}$, there exists a unique s such that $a \in \mathcal{A}(s)$. We denote by $\mathcal{A}^{-1}(s)$ the set of actions that lead to s i.e. $\mathcal{A}^{-1}(s) := \{a : P(a,s) > 0\}$.

We can associate a directed bipartite graph $G = (\mathcal{S}, \mathcal{A}, E)$ with $(\mathcal{S}, \mathcal{A}, J, P)$ by defining $E := \{(s,a) : s \in \mathcal{S} \setminus \{0\}, a \in \mathcal{A} \setminus \{0\}$ with $J(a,s) = 1\} \cup \{(a,s) : s \in \mathcal{S} \setminus \{0\}, a \in \mathcal{A}^{-1}(s)\} \cup \{(0_{\mathcal{S}}, 0_{\mathcal{A}}), (0_{\mathcal{A}}, 0_{\mathcal{S}})\}$. G is called the *support graph*. We sometimes call the vertices/nodes of G in $\mathcal{S}$ the *state nodes* and the vertices/nodes of G in $\mathcal{A}$ the *action nodes*. A *$\mathcal{S}$-walk* in G is a sequence of vertices $(s_0, a_0, s_1, a_1, ..., s_k)$ for some $k \in \mathbb{N}$ with $s_i \in \mathcal{S}$ for all $0 \leq i \leq k$, $a_i \in \mathcal{A}$ for all $0 \leq i \leq k-1$, $(s_i, a_i) \in E$ for all $0 \leq i \leq k-1$, and $(a_{i-1}, s_i) \in E$ for all $1 \leq i \leq k$. k is called the *length* of the walk. s_k is called the *head* of the walk. We denote by W_k the set of all possible $\mathcal{S}$-walk of length k and $W := \cup_{k \in \mathbb{N}} W_k$. A *policy* Π is a function $\Pi : (k, w_k) \in \mathbb{N} \times W \mapsto \Pi_{k,w_k} \in \mathcal{M}_{=}(n,m)$ satisfying $w_k \in W_k$, and $\Pi_{k,w_k}(s,a) > 0 \implies J(s,a) = 1$ for all $s \in \{1,...,n\}$ and $a \in \{1,...,m\}$. We say that a policy is *deterministic* if Π_{k,w_k} is a 0/1 matrix for all k and w_k, it is *randomized* otherwise. If there exist k and $w_k, w'_k \in W_k$ that share a same head and such that

[2]Observe that it is usually not a stochastic matrix as state 0 and action 0 are left out.
[3]If not we simply duplicate the actions.

$\Pi_{k,w_k} \neq \Pi_{k,w'_k}$, we say that the policy is *history-dependent* (otherwise it is usually said to be memoryless or *Markovian*). If Π is a constant function, we say that the policy is *stationary*. A policy Π induces a probability distribution over the (countable) set of all possible $\mathcal{S}$-walks. When Π is stationary, we often abuse notation and identify Π with a $n \times m$ matrix.

We let $y_k^\Pi \in \mathbb{R}_+^n$ be the substochastic[4] vector representing the state of the system in period k when following policy Π (from an initial distribution y_0^Π). That is $y_k^\Pi(s)$ is the probability of being in state s, for all $s = 1, ..., n$ at time k following policy Π. Similarly, we denote by $x_k^\Pi \in \mathbb{R}_+^m$ the substochastic[5] vector representing the probability to perform action a, for all $a = 1, ..., m$, at time k following policy Π. Given a stationary policy Π and an initial distribution y_0^Π at time 0, by the law of total probability (and because each action is available in exactly one state), we have $x_k^\Pi = \Pi^T \cdot y_k^\Pi$ for all $k \geq 0$. Similarly, we have: $y_k^\Pi = P^T x_{k-1}^\Pi = P^T \cdot \Pi^T \cdot y_{k-1}^\Pi$ for all $k \geq 1$. Hence the state of the system at time $k \geq 0$ follows $y_k^\Pi = (P^T \cdot \Pi^T)^k \cdot y_0^\Pi$. The value $c^T x_k^\Pi$ represents the expected cost paid at time k following policy Π. One can define for each $s \in \mathcal{S} \setminus \{0\}$, $J_\Pi(s) := \limsup_{K \to +\infty} \sum_{k=0}^K c^T x_k^\Pi$ with $y_0^\Pi := e_s$, and $J^*(s) := \min\{J_\Pi(s) : \Pi \text{ deterministic and stationary policy}\}$[6] ($e_s$ is the characteristic vector of $\{s\}$ i.e. the 0/1 vector with $e_s(s') = 1$ iff $s' = s$). Bertsekas and Tsitsiklis [19] introduced the notion of *proper* stationary policies: a stationary policy Π is said to be *proper* if $\mathbf{1}^T (P^T \cdot \Pi^T)^n \cdot e_s < 1$ for all $s = 1, ..., n$, that is, after n periods of time, the probability of reaching the target state is positive, from any initial state s. We say that such policies are *BT-proper* (*BT-improper* otherwise) as we will introduce a slight generalization later. Bertsekas and Tsistiklis [19] defined a stationary policy Π^* to be *optimal*[7] if $J^*(s) = J_{\Pi^*}(s)$ for all $s \in \mathcal{S} \setminus \{0\}$. They introduced the *Stochastic Shortest Path Problem* as the problem of finding such an optimal stationary policy.

1.1.1 Literature review

The stochastic shortest path problem is a special case of Markov Decision Process and it is also known as total reward undiscounted MDP [15, 16, 88]. It arises naturally in robot motion planning, from maneuvering a vehicle over unfamiliar terrain, steering a flexible needle through human tissue or guiding a swimming micro-robot through turbulent water for instance [2]. It has also many applications in operations research, artificial intelligence and economics: from inventory control, reinforcement learning to asset pricing (see for instance [106, 76, 8, 99]). SSP forms an important class of MDPs as it contains finite horizon MDPs, discounted MDPs (a euro tomorrow is worth less than a euro today) and average cost problems (through the so-called vanishing discounted factor approach) as special cases. It thus encapsulates most of the work on finite state/action MDPs. The stochastic shortest path problem was introduced first by Eaton and Zadeh

[4] It is in general not a purely stochastic vector as state 0 is left out.
[5] It is in general not a purely stochastic vector as action 0 is left out.
[6] lim sup is used here as the limit need not be defined in general.
[7] Note that it is not clear, a priori, whether such a policy exists.

in 1962 [38] in the context of pursuit-evasion games and it was later studied thoroughly by Bertsekas and Tsitsiklis [19].

MDPs were first introduced in the 50's by Bellman [10] and Shapley [95] and they have a long, rich and successful history (see for instance [15, 16, 88]). For most MDPs, it is known that there exists an optimal deterministic and stationary policy [88]. Building upon this fact, there are essentially three ways of solving such problems exactly (and some variants): *value iteration* (VI), *policy iteration* (PI) and linear programming (LP). Value iteration and policy iteration are the original 50+ years old methods [10, 58]. The idea behind VI is to approximate the infinite horizon problem with a longer and longer finite one. The solution to the k-period approximation is built inductively from the optimal solution to the $(k-1)$-period problem using standard dynamic programing. The convergence of the method relies mainly on the theory of contraction mappings and Banach fixed-point theorem [15] for most MDPs. PI is an alternative method that starts from a feasible deterministic and stationary policy and iteratively improves the action in each state so as to converge to an optimal solution. It can be interpreted as a simplex algorithm where multiple pivots are performed in each step [69, 33]. As such it builds implicitly upon the geometry of the problem to find optimal solutions. Building explicitly upon this polyhedra, most MDPs can also be formulated as linear programs and as such they can thus be solved in (weakly) polynomial time [69, 34, 33, 57, 52].

In the context of the SSP, some hypothesis are required for standard methods and proof techniques to apply. Bertsekas and Tsitsiklis [19] proved that VI, PI and LP still work when two assumptions hold, namely, when (i) there exists a BT-proper policy and (ii) any BT-improper policy Π have at least one state s for which $J_\Pi(s) = +\infty$. In particular they show that one can restrict to deterministic policies. Their assumptions naturally discriminate between BT-proper and BT-improper policies. Exploiting further the discrepency between these policies, Bertsekas and Yu [20] showed that one can relax asumptions (i) and (ii) when the goal is to find an optimal *BT-proper* stationary policy. They could show that applying the standard VI and PI methods onto a perturbated problem where c is modified to $c + \delta \cdot \mathbf{1}$ with $\delta > 0$ and letting δ tends to zero over the iterations, yields an optimal BT-proper solution if (j) there exists a BT-proper policy and (jj) J^* is real-valued. Moreover they could also show that the problem can still be formulated (and thus solved) using linear programming, which settles the (weak) polynomiality of this extension. Some authors from the AI community proposed alternative extensions of the standard SSP introduced by Bertsekas and Tsitsiklis. It is easy to see that the most general one, titled Stochastic and Safety Shortest Path problem [101], is a special case of Bertsekas and Yu's framework (it is a bi-objective problem that can be easily modeled in this framework using artificial actions of prohibited cost).

The question of whether SSP, in its original form or the later generalization by Bertsekas and Yu, can be solved in strongly polynomial time[8] is a major open problem for MDPs (see for instance [110]). It was proven in a series of breakthrough papers that it is the case for *fixed* discount rate (basically the same problem as before but where the transition matrix P is such that there is a fixed non-zero probability of ending up in 0

[8] a polynomial in the number of states and the number of actions

after taking any action). The result was first proved using interior point methods [109] and then the same author showed that the original policy iteration method proposed by Howard was actually strongly polynomial too [110] (the analysis was later improved [50]). The problem is still open for the undiscounted case but Policy Iteration is known to be exponential in that setting [46]. In contrast, value iteration was proved to be exponential even for the discounted case [42]. Because SSPs can be formulated as linear programs, the question relates very much to the existence of strongly polynomial time algorithms for linear programming, a very long-lasting open problem that was listed as one of the 18 mathematical problems of the 21st century by Smale in 1998 [97]. A possible line of attack is to study simplex-type of algorithms but existence of such algorithms is also a long standing open problem and relates to the Hirsch conjecture on the diameter of polyhedra. These questions are central in optimization, discrete geometry and computational complexity. Despite the fact that SSP exhibits strong additional properties over general LPs, these questions are still currently out of reach in this setting, too.

In practice, value iteration and policy iteration are the methods of choice when solving medium size MDPs. For large scale problems (i.e. most practical applications), approximate solutions are needed to provide satisfying solutions in a reasonable amount of time [87]. The field is known as Approximate Dynamic Programming and is a very active area of research. Most approximation methods are based on approximate versions of exact algorithms and developping new exact approaches is thus of great practical interest.

In this chapter, we propose an extension of the frameworks of Bertsekas and Tsitsiklis [19] and Bertsekas and Yu [20]. We prove in Section 1.3 that, in this setting, there is an optimal deterministic and stationary policy. Then we show in Section 1.4 that the standard Value Iteration and Policy Iteration methods converge, and we give an alternative approach that generalizes Dijkstra's algorithm when the costs are nonnegative.

Remark

We became aware recently [89] that some related results were published in 1990 by Bendali and Quilliot [13]. Similarly to Bertsekas and Tsitsiklis [19], they extended the shortest path problem to stochastic environments. Their approach was different though: they studied the natural extensions of directed graphs to the stochastic setting (they named the corresponding extensions stochastic networks) and they studied the extensions of arborescences and cycles and their role in an alternative notion of stochastic shortest path. They could prove results similar to Bertsekas and Tsitsiklis [19] (namely that linear programming could be used to solve the problem and they also described a method similar to Value Iteration). They also proposed an extension of Dijkstra's algorithm. These results were totally unknown to the MDP community until now, probably due to the fact that they were published in French and in a different community (Bendali and Quilliot were apparently equally unaware of the work of Bertsekas and Tsitsiklis [19]). While there are non empty intersections with our work too (in particular for Dijkstra's algorithm), our results are more general: our framework (stricly) encap-

sulates both the frameworks of Bertsekas and al. and of Bendali and Quilliot. Besides, our results were proved independently with different techniques.

1.1.2 Notations and definitions

For a graph G, we denote by $V(G)$ the vertex/node set of G and by $E(G)$ its edge set. Given a directed graph $G(V, E)$, and a set $S \subset V$, we denote by $\delta^+(S)$ the set of arcs (u, v) with $u \in S$ and $v \notin S$, and by $N^+(S)$ the set of vertices $v \in V \setminus S$ such that $(u, v) \in E$ for some $u \in S$. For convenience when S is a singleton, we denote $\delta^+(\{u\})$ by $\delta^+(u)$ and $N^+(\{u\})$ by $N^+(u)$. Then we define inductively $N_k^+(u) := N^+(N_{k-1}^+(u)) \setminus N_{k-1}^+(u) \cup \ldots \cup N_0^+(u))$ for $k \geq 1$ integer with $N_0^+(u) = \{u\}$. We denote by $R^+(u)$ the set of vertices *reachable* from u i.e. $R^+(u) = \bigcup_{k \geq 0} N_k^+(u)$. We can define $\delta^-(u)$, $N^-(u)$, $N_k^-(u)$ and $R^-(u)$ analogously. Clearly $v \in R^-(u)$ if and only if $u \in R^+(v)$. $R^-(u)$ are the vertices that can reach u. When confusion may arise, we denote $R^+(u)$ by $R_G^+(u)$ (and similarly for the other notations). We say that a graph $G(V, E)$ is *strongly connected* if for all $u, v \in V(G)$, we have $u \in R^+(v)$ and $v \in R^+(u)$. We denote by $\mathbb{1}_A$ the indicator function associated with a set A i.e. $\mathbb{1}_A$ is a 0/1 function with $\mathbb{1}_A(a) = 1$ if and only if $a \in A$. For a vector $x \in \mathbb{R}^d$ and $I \subseteq \{1, ..., d\}$ we denote by $x[I]$ the restriction of x to the indices in I and $x(I) := \sum_{i \in I} x(i)$. For $s \in \{1, ..., n\}$, we denote by e_s the 0/1 vector of $\mathbb{R}^n$ with $e_s(i) = 1$ if and only if $i = s$.

1.1.3 Main contributions and roadmap

In this chapter, we revisit the Stochastic Shortest Path problem, a well-known problem in Markov Decision Processes. We shed some new light on this well-established problem, both structurally and algorithmically. Our approach is to mimic the polyhedral analysis of the deterministic shortest path problem.

On the structural side, we study the polyhedra associated with the natural linear relaxation of the problem. We show that extreme points of the polyhedra are associated with deterministic and stationary policies by generalizing the flow decomposition property (a fundamental result in network flow theory). This allows to: (i) formally prove that we can restrict to such policies for this problem, a fact that was somewhat taken for granted in earlier works ; (ii) relax the conditions under which the problem is well-defined and (weakly) polynomial: this is the case now when there is a way to reach the target from any initial state, and there is no 'transition cycle' of negative cost (the extension of negative cost cycles to the stochastic setting) ; (iii) simplify the analysis of the problem.

On the algorithmic side, building upon our polyhedral findings: (i) we prove that the two standard methods for MDPs, i.e. Value Iteration and Policy Iteration, converge in our more general setting ; (ii) we also give a new iterative algorithm based on the standard primal-dual algorithm for linear programming. When the costs are nonnegative, this algorithm can be seen as a generalization of Dijkstra's algorithm to the stochastic setting.

We believe that our result closes some important algorithmic and structural gap between the deterministic problem and the stochastic extension. All in all, our new approach allows to generalize, unify and simplify most results on the SSP for finite state/action spaces and we believe that we set the appropriate and natural framework to study the problem in this case.

Besides, our approach has several strengths with respect to the literature: (i) Approaching the problem from a polyhedral combinatorics perspective is new. Polyhedral combinatorics has been a powerful tool in harmonizing and simplifying many fundamental results in combinatorial optimization. We believe that this new perspective on the problem might help address important remaining open questions such as the existence of a strongly polynomial time algorithm ; (ii) Our framework properly encapsulates the deterministic shortest path problem, in contrast with prior works ; (iii) Our proofs are elementary for people familiar with network flow theory and it should thus provide a new entry point to the problem for people in combinatorial optimization not familiar with Markov Decision Processes. This should help grasping further interest from this community ; (iv) Our generalization allows to capture many important subproblems that were not fitting in the previous frameworks, such as the so-called MAXPROB problem, where the goal is to reach a target with maximum probability: this is a core problem in optimal control, artificial intelligence and game theory.

We now try to give an overview of the different propositions, lemmas and theorems that follow. This should help the reader familiar with the deterministic shortest path problem and its relation with network flow theory to navigate smoothly through the next sections.

In Section 1.2, we introduce our new framework for the stochastic shortest path problem. We show in particular (Lemma 1.2) that our framework properly encapsulates the one proposed by Bertsekas and Tsitsiklis [19]. In the deterministic setting the standard assumptions for the shortest (s,t)-path problem are that (i) there exists a path from any node to t and (ii) there is no negative cost cycle. Assumption 24 generalizes these two assumptions to the stochastic setting in the most natural way and we prove that the corresponding assumptions are easily checked (Lemma 1.3 , Lemma 1.4 and Lemma 1.5). We also introduce the natural linear programming relaxation of the problem, the so-called network flux relaxation (see (P_s)). This is again the natural generalization of the network flow formulation for the deterministic version.

In Section 1.3, we prove that the network flux relaxation is actually a formulation by showing that the extreme points are associated with *proper*[9], deterministic and stationary policies (Corollary 28). This result builds upon an extension of a fundamental theorem for network flows: *the flow decomposition theorem*. The idea in the deterministic case is to decompose a flow in paths and cycles. In the stochastic setting this translates into a decomposition in terms of proper, deterministic and stationary policies and *transition cycles*. The proof of the corresponding theorem (Theorem 1.7) builds upon different basic properties of *flux vectors* (that is, solutions to the network flux relaxation), namely, that if a flux 'goes through' a node, then this node has to be connected to the sink (Proposition

[9]The new definition of proper is given in the beginning of the next section.

25), and the fact that one can easily associate a flux vector to a proper, deterministic and stationary policy (Proposition 26). These properties are trivial when instantiated in the determininistic setting but a bit more technical in the stochastic case. As in the deterministic case, the idea behind the flow decomposition theorem is to first identify "paths" (Lemma 1.8) and then decompose "cycles" (Lemma 1.9). Lemma 1.6 shows that each step in the decomposition can be performed efficiently. Finally Lemma 1.10 generalizes Bellman optimality conditions to the stochastic setting (that is, if the optimal policy *can* visit a certain node, the policy should be optimal from this node too).

Bellman optimality conditions are then exploited in Section 1.4 to derive several iterative algorithms. We first prove that Value Iteration (the generalization of Bellman-Ford algorithm), converges in value to the optimal solution (Theorem 1.11) and we show that we can in fact extract a series of proper, deterministic and stationary policies that converge to the optimal policy (Theorem 1.12). We then prove that we can exploit the linear programming formulation to derive simplex-type of algorithms: any standard (single pivot) simplex method will find an optimal policy in a finite number of steps as the linear program is non degenerate (Theorem 1.13), and Howard's Policy Iteration method (a multi-pivot simplex algorithm) also converges in a finite number of steps (Theorem 1.14). The latter result exploits the fact that the objective function is nonincreasing in each iteration (Proposition 29). Finally Theorem 1.15 exploits the standard primal-dual algorithm for linear programming to provide a natural extension of Dijkstra's algorithm.

1.2 Our new framework

We start with a simple observation:

Lemma 1.1

Let Q be a matrix with $\lim_{k\to+\infty} Q^k = 0$. Then $I - Q$ is invertible, $\sum_{k\geq 0} Q^k$ is well defined and $\sum_{k\geq 0} Q^k = (I - Q)^{-1}$.

Lemma 1.2

For BT-proper stationary policies, $\lim_{K\to+\infty} \sum_{k=0}^{K} x_k^{\Pi}$ is finite for any initial state distribution y_0^{Π}.

Proof. $x_k^{\Pi} = \Pi^T \cdot P^T \cdot x_{k-1}^{\Pi}$ for all $k \geq 1$ and $x_0^{\Pi} = \Pi^T y_0^{\Pi}$. Therefore $x_k^{\Pi} = (\Pi^T \cdot P^T)^k \cdot \Pi^T y_0^{\Pi}$, where y_0^{Π} is the original state distribution. It follows that $\sum_{k=0}^{K} x_k^{\Pi} = \sum_{k=0}^{K}((\Pi^T \cdot P^T)^k \cdot \Pi^T y_0) = (\sum_{k=0}^{K}(\Pi^T \cdot P^T)^k) \cdot \Pi^T y_0$ and because of the standard Lemma 1.1, it implies that $I - \Pi^T \cdot P^T$ is invertible and that $\lim_{K\to+\infty} \sum_{k=0}^{K} x_k^{\Pi} = (I - \Pi^T \cdot P^T)^{-1} \cdot \Pi^T y_0$. ($\lim_{k\to+\infty}(\Pi^T \cdot P^T)^k = 0$ by definition of BT-properness since $\mathbf{1}^T(P^T \cdot \Pi^T)^n \cdot e_i < 1$ for all $i = 1, ..., n$).

□

We now extend the notion of proper policies introduced by Bertsekas and Tsitsiklis using this alternative (relaxed) property and *from now on we will only use this new definition*.

Given a state $s \in \{1, ..., n\}$, a policy Π is said to be *s-proper* if $\sum_{k\geq 0} x_k^\Pi$ is finite, when $y_0^\Pi := e_s$. Observe that $\sum_{k\geq 0} y_k^\Pi$ is also finite for s-proper policies (as $y_k^\Pi = P^T x_{k-1}^\Pi$). In particular $\lim_{k\to+\infty} y_k^\Pi = 0$ and thus the policy leads to the target state 0 with probability 1 from state s. A s-proper policy is thus a policy that converges to the target with probability one and whose expected number of visit in each action is finite. The expected cost of such a policy is thus the well-defined value $c^T \sum_{k\geq 0} x_k^\Pi$. The *s-stochastic-shortest-path problem* (s-SSP for short) is the problem of finding a s-proper policy Π of minimal cost $c^T \sum_{k\geq 0} x_k^\Pi$. We say that a policy is *proper* if it is s-proper for all s and it is called *improper* otherwise. The *stochastic shortest path problem* (SSP) is the problem of finding a proper policy Π of minimal cost $c^T \sum_{k\geq 0} x_k^\Pi$ where $y_0^\Pi := \frac{1}{n}\mathbf{1}$. It is easily seen that the stochastic shortest path problem, as defined here, is also a special case of the s-SSP as one can add an artificial state with only one action that leads to all states in $\{1, ..., n\}$ with probability $\frac{1}{n}$. In the following two sections, unless otherwise stated, we restrict to the s-SSP. In this context, we often abuse notation and we simply call proper a s-proper policy.

Since for any policy Π (possibly history-dependent and randomized), Π_{k,w_k} are stochastic matrices, we have at any period $k \geq 0$, $\sum_{a\in\mathcal{A}(s)} x_k^\Pi(a) = y_k^\Pi(s)$ (remember that each action is available in exactly one state). We also have $y_{k+1}^\Pi(s) = \sum_{a\in\mathcal{A}} p(s|a) x_k^\Pi(a)$ for all $s \in \{1, ..., n\}$. In matrix form this is equivalent to $y_k^\Pi = J^T x_k^\Pi$ and $y_{k+1}^\Pi = P^T x_k^\Pi$. This implies $J^T x_{k+1}^\Pi = P^T x_k^\Pi$ for all $k \geq 0$. We also have $J^T x_0^\Pi = e_s$. Now $x^\Pi := \sum_{k=0}^\infty x_k^\Pi$ is well-defined for proper policies. Summing up the previous relations over all periods $k \geq 0$ we get $(J-P)^T x^\Pi = e_s$. Hence the following linear program is a relaxation of the s-SSP problem[10].

$$\begin{array}{ll} \min & c^T x \\ (J-P)^T x & = e_s \\ x & \geq 0 \end{array} \qquad (P_s)$$

Observe that for a deterministic problem (i.e. when P is a 0/1 matrix), $(J-P)^T$ is the node-arc incidence matrix of a graph (up to a row as it does not containt the row associated with the sink node) and the corresponding LP is the standard network flow relaxation of the deterministic shortest path problem (again up to a row as we remove the (redundant) flow conservation constraint for the sink node). The vector x is sometimes called a network *flux* as it generalizes the notion of network flow.

[10]We would like to stress on the fact that the LP relaxation we consider here is almost (except for the right hand side) the standard LP formulation of the problem of finding an optimal deterministic and stationary policy and it was already known for quite some time for many special cases of SSP (see [20] for instance). However while usually, the LP formulation comes as a corollary of other results, here we reverse the approach and introduce this formulation as a natural relaxation of the problem and we derive the standard results as (reasonably) simple corollaries. This is what allows to simplify, generalize and unify many results from the litterature. This is a simple yet major contribution of this chapter. The notation and terminology is taken from [49].

We call a solution x to $(J-P)^T x = 0, x \geq 0$ a *transition cycle* and the cost of such a transition cycle x is $c^T x$. Negative cost transition cycles are the natural extension of negative cost cycles for deterministic problems. One can check the existence of such objects by solving a linear program.

Lemma 1.3

One can check in (weakly) polynomial time whether a stochastic shortest path instance admits a negative cost transition cycle through linear programming.

We will prove in the sequel that the extreme points of $P_s := \{x \geq 0 : (J-P)^T x = e_s\}$ 'correspond' to proper deterministic and stationary policies. Hence, when the relaxation (P_s) has a finite optimum (i.e. when there is no transition cycle of negative cost and when a proper policy exists), this will allow to prove that, the s-SSP admits an optimal proper policy which is deterministic and stationary. This answers, for this problem, one fundamental question in MDP theory "Under what conditions is it optimal to restrict to deterministic and stationary policies ?" [88].

We can assume without loss of generality that there exists a path between all state node s' and 0 in the support graph G. Indeed, if there is a state node s' with no path to 0 in G, then no s-proper policy will pass through s' at any point in time (because then the probability of reaching the target state, starting from s', is zero, contradicting $\lim_{k\to+\infty} y_k^{\Pi} = 0$) ; we could thus remove s' and the actions leading to s' and iterate. Under this assumption, there is always a s-proper policy. Indeed the randomized and stationary policy Π that chooses an action uniformly at random among $\mathcal{A}(s')$, in each state $s' = 1, ..., n$, will work: in this case, for each state s', there is in fact a non zero probability of choosing one of the paths from s' to 0 after at most n periods of time.

Lemma 1.4

Consider a s-SSP instance where there exists a path between all state node s' and 0 in the support graph G. Then the policy that consists, for each state $s' \in \mathcal{S} \setminus \{0\}$, in choosing uniformly at random an action in $\mathcal{A}(s')$ is a proper stationary policy.

The discussion above also gives a simple algorithm for testing the existence of a proper policy for any instance of the SSP.

Lemma 1.5

One can check in time $O(|\mathcal{S}| \cdot (|\mathcal{S}| + |\mathcal{A}| + |E|))$ whether a s-SSP instance with support graph $G = (\mathcal{S}, \mathcal{A}, E)$ admits a proper policy or not.

We are now ready to introduce the new assumptions that we will use to study the stochastic shortest path problem. They are the very natural extensions of the standard assumptions for the deterministic shortest path problem.

Assumption 24
We consider s-SSP/SSP instances where:

- *there exists a path between all state node s' and 0 in the support graph G, and*
- *there is no negative cost transition cycle.*

As already observed, these assumptions can be checked in (weakly) polynomial time. Moreover, these assumptions implies that (P_s) has a finite optimum (from standard LP arguments). Also Bertsekas and Yu's framework is a special case of our setting as in the presence of negative cost transition cycles, $J^*(s')$ is not real-valued for some state s' [11]. The main extension, with respect to Bertsekas and Yu, is that we allow for non-stationary proper policies in the first place.

1.3 Existence of an optimal, deterministic and stationary policy

In this section, we will prove essential properties about $P_s := \{x \geq 0 : (J - P)^T x = e_s\}$. This will allow to prove that, under Assumption 24, we can restrict to optimal proper, deterministic and stationary policies.

We start with a few definitions. Let $G = (\mathcal{S}, \mathcal{A}, E)$ be the support graph of our s-SSP instance and let $x \in \mathbb{R}^m$. We define G_x to be the subgraph of G induced by the vertices in $\mathcal{A}_x \cup N_G^+(\mathcal{A}_x) \cup N_G^-(\mathcal{A}_x)$ where $\mathcal{A}_x := \{a \in \{1, ..., m\} \text{ with } x(a) > 0\}$. G_x is called the *support graph of x in G*. Again we call *state nodes* the vertices/nodes of G_x that are in $\mathcal{S}$ and *action nodes* the vertices/nodes of G_x that are in $\mathcal{A}$. We denote by E_x the set of edges of G_x. A transition cycle x is *simple* if for all state nodes s' in $V(G_x)$, there exists exactly one edge of $N_G^+(s')$ in E_x, i.e. $|N_{G_x}^+(s')| = 1$, and G_x is strongly connected.

The main theorem of this section is an extension of the *flow decomposition theorem*, which is a fundamental result in network flow theory (see [1]). It asserts that any network flux is a convex combination of network flux 'associated with' proper policies plus a conic combination of simple transition cycles (see Theorem 1.7). Before we can prove this theorem , we need a couple of useful propositions. The proof of the first proposition builds upon simple flow conservation arguments.

[11] One can prove using Lemma 1.9 and basic geometry that when there exists a negative cost transition cycle, there exists also a *simple* transition cycle of negative cost (see the definition in the second paragraph of Section 1.3): all state nodes s' on this cycle will have $J_\Pi(s') = -\infty$, where Π is the deterministic and stationary policy that consists in choosing the unique action $a \in \mathcal{A}(s')$ with $x(a) > 0$ for each state node s' on the cycle and any action for the state nodes that are not on the cycle.

Proposition 25
Let $x \in \mathbb{R}^m$ be a feasible solution of (P_s). There exists a path between all states reachable from s in G_x and $0_{\mathcal{S}}$. In other word, for all $s' \in R^+_{G_x}(s)$, we have $s' \in R^-_{G_x}(0_{\mathcal{S}})$.

Proof. Let us define $\bar{x} \in \mathbb{R}^{|E_x|}$ as follows: $\bar{x}((s'', a)) := x(a)$ for all $a \in A_x$ and s'' the (unique) state with $a \in \mathcal{A}(s'')$, and $\bar{x}((a, s'')) := P(a, s'') \cdot x(a)$ for all $a \in A_x$, and $s'' \in S$ such that $P(a, s'') > 0$. Observe that $\bar{x}$ is only defined on E_x and that $\bar{x} > 0$. Because x is a feasible solution to (P_s), $\bar{x}$ satisfies $\bar{x}(\delta^+_{G_x}(v)) - \bar{x}(\delta^-_{G_x}(v)) = \mathbb{1}_{\{s\}}(v) - \mathbb{1}_{\{0_{\mathcal{S}}\}}(v)$ for all $v \in V(G_x)$ and $\bar{x} \geq 0$. It is thus a unit $(s, 0_{\mathcal{S}})$-flow in $V(G_x)$. Now let us assume that there exists $s' \in R^+_{G_x}(s)$ with $s' \notin R^-_{G_x}(0)$. Summing up all flow constraints over $v \in R^+_{G_x}(s')$, we get $\bar{x}(\delta^+_{G_x}(R^+_{G_x}(s'))) - \bar{x}(\delta^-_{G_x}(R^+_{G_x}(s'))) = \mathbb{1}_{R^+_{G_x}(i)}(s)$. We have $\bar{x}(\delta^+_{G_x}(R^+_{G_x}(s'))) = 0$ by definition of $R^+_{G_x}(s')$. But then $\bar{x}(\delta^-_{G_x}(R^+_{G_x}(s'))) + \mathbb{1}_{R^+_{G_x}(s')}(s) = 0$. Since $\bar{x}(\delta^-_{G_x}(R^+_{G_x}(s'))) \geq 0$, this implies $s \notin R^+_{G_x}(s')$ and $\bar{x}(\delta^-_{G_x}(R^+_{G_x}(s'))) = 0$. Now because $s \notin R^+_{G_x}(s')$ and $s \in R^-_{G_x}(s')$ (by hypothesis), there is at least one arc of E_x in $\delta^-_{G_x}(R^+_{G_x}(s'))$ but this implies $\bar{x}(\delta^-_{G_x}(R^+_{G_x}(s'))) > 0$ as $\bar{x} > 0$, a contradiction. □

Given a proper, deterministic and stationary policy Π, we denote by G_Π the subgraph of G induced by the state vertices in $\mathcal{S}$ and the actions vertices in Π. Now let G^s_Π be the subgraph of G_Π induced by the vertices in $R^+_{G_\Pi}(s)$. G^s_Π is called the *support graph* of Π (it is easily seen that it corresponds to the subgraph induced by the states and actions that we might visit under policy Π when starting from s). Because Π is proper, 0_S is reachable from each state s' in G^s_Π. Let us denote by $\mathcal{S}'$ the state vertices in G^s_Π and $\Pi(\mathcal{S}')$ the actions associated with $\mathcal{S}'$ in Π. We also denote by $P_{\mathcal{S}'}$ the restriction of P to the columns in $\mathcal{S}'$ and the rows in $\Pi(\mathcal{S}')$ (since Π is determinisitic, $P_{\mathcal{S}'}$ is a $|\mathcal{S}'| \times |\mathcal{S}'|$ matrix). $P^T_{\mathcal{S}'}$ can be interpreted as the transition matrix associated with $\mathcal{S}'$ when following policy Π (we do not leave $\mathcal{S}'$): $P_{\mathcal{S}'}(\Pi(s'), s'')$ gives the probability of ending in state s'' (in one iteration) when starting in state s' and using $\Pi(s')$. Hence, if we assume that the rows of $P_{\mathcal{S}'}$ are ordered according to S', then $P^T_{\mathcal{S}'} e'_i$ defines the state of the system after one iteration of policy Π if we start in state $i \in \mathcal{S}'$ (e'_i is the restriction of e_i to to the indices in $\mathcal{S}'$). Now as already observed, $0_{\mathcal{S}}$ is reachable from any node in $\mathcal{S}'$ and it thus follows that $(P^T_{\mathcal{S}'})^k e'_i$, the state of the system after k steps, tends to zero as k tends to infinity (remember that $0_{\mathcal{S}}$ is left out). Because this is true for any $i \in \mathcal{S}'$, we have $\lim_{k \to +\infty} (P^T_{\mathcal{S}'})^k = 0$ and thus $(I_{\mathcal{S}'} - P_{\mathcal{S}'})$ is invertible by Lemma 1.1. Now observe that $(I_{\mathcal{S}'} - P_{\mathcal{S}'})^T x^\Pi[\Pi(S')] = e'_s$ for $x^\Pi := \sum_{k=0}^{+\infty} x^\Pi_k$, with $y^\Pi_0 := e_s$. Indeed $x^\Pi(a) = 0$ for all $a \notin \Pi(S')$ and thus $(I_{\mathcal{S}'} - P_{\mathcal{S}'})^T x^\Pi[\Pi(S')] = e'_s$ corresponds to the constraints of (P_s) associated with the rows in $\mathcal{S}'$. We thus have the following result.

Proposition 26
Given a proper, deterministic and stationary policy Π, the flux vector x^Π associated with Π and defined by $x^\Pi := \sum_{k=0}^{+\infty} x^\Pi_k$, with $y^\Pi_0 := e_s$ satisfies $x^\Pi[\Pi(S')] = (I_{\mathcal{S}'} - P_{\mathcal{S}'})^{-T} e'_s$ and $x^\Pi(a) = 0$ for all $a \notin \Pi(\mathcal{S}')$, with $\mathcal{S}'$, $\Pi(\mathcal{S}')$, $I_{\mathcal{S}'}$, $P_{\mathcal{S}'}$ and e'_s defined as above.

The following proposition is easy to prove using similar flow arguments as in the proof of Proposition 25.

Proposition 27
Let Π be a proper, deterministic and stationary policy. We have $G^s_\Pi = G_{x^\Pi}$. Moreover if $x \in P_s$ and $\Pi(S) \subseteq \mathcal{A}_x$, then G^s_Π is a subgraph of G_x.

Before proving Theorem 1.7, we need a final lemma.

Lemma 1.6
Let $G = (\mathcal{S}, \mathcal{A}, E)$ be the support graph of a s-SSP instance and assume that there is a path from every state vertex s' to $0_\mathcal{S}$ in G. Then in time $O(|\mathcal{S}| + |\mathcal{A}| + |E|)$, one can find a proper, deterministic and stationary policy Π.

Proof. We know that, $0_\mathcal{S} \in R^+(s')$ for all s', is enough to ensure that there is a proper policy by Lemma 1.4. Now if there exists a state vertex s' in G with $|\mathcal{A}(s')| > 1$, we can delete from G an action in $\mathcal{A}(s')$ that does not remove $0_\mathcal{S}$ from $R^+(s')$. Such an action exists as it is enough to keep an action $a \in A(s')$ with minimum distance to 0 (in terms of arc) to ensure that $0_\mathcal{S}$ is still in $R^+(s')$ after deletion (by minimality of the distance to 0, such an action has a directed path to $0_\mathcal{S}$ that does not go through s'). If $|\mathcal{A}(s')| = 1$ for all s' then the only possible policy is proper (from Lemma 1.4), deterministic and stationary. We can implement such a procedure in time $O(|\mathcal{S}| + |\mathcal{A}| + |E|)$ by computing $N_k^-(0)$ for all $k \leq |\mathcal{S}| + |\mathcal{A}|$ and a $0_\mathcal{S}$-anti-arborescence A using a breadth first search algorithm: we then keep only the actions in A. □

We are now ready to prove the main theorem of this section.

Theorem 1.7
Let $x \in \mathbb{R}^m$ be a feasible solution of (P_s). In strongly polynomial time, one can find $k, k' \in \mathbb{N}$ with $1 \leq k$, $k + k' \leq m$, $x_1, ..., x_k \in \mathbb{R}^m$, $x'_1, ..., x'_{k'} \in \mathbb{R}^m$, $\lambda_1, ..., \lambda_k \in [0, 1]$, and $\lambda'_1, ..., \lambda'_{k'} \geq 0$ such that $x_1, ..., x_k$ are feasible solutions of (P_s), $x'_1, ..., x'_{k'}$ are simple transition cycles, $\sum_{j=1}^k \lambda_j = 1$ and $x = \sum_{j=1}^k \lambda_j x_j + \sum_{j'=1}^{k'} \lambda'_{j'} x'_{j'}$. Moreover, the vectors x_j are network flux corresponding to proper, deterministic and stationary policies, i.e. for all $j \in 1, \ldots, k$, there exists a proper, deterministic and stationary policy Π_j such that $x_j = x^{\Pi_j}$.

Proof. We will start with a slightly simpler version.

Lemma 1.8
Let $x \in \mathbb{R}^m$ be a feasible solution of (P_s). In strongly polynomial time, one can find $k \in \mathbb{N}$, $x_1, ..., x_k, x_c \in \mathbb{R}^m$,and $\lambda_1, ..., \lambda_k \in [0, 1]$ such that $1 \leq k \leq m - |\mathcal{A}_{x_c}|$, $x_1, ..., x_k$ are feasible solutions of (P_s), x_c is a transition cycle, $\sum_{j=1}^k \lambda_j = 1$ and $x = \sum_{j=1}^k \lambda_j x_j + x_c$. Moreover, the vectors x_j are network

flux corresponding to proper, deterministic and stationary policies, i.e. for all $j \in 1, \ldots, k$, there exists a proper, deterministic and stationary policy Π_j such that $x_j = x^{\Pi_j}$.

Proof. We prove first that such a decomposition exists for any $x \in P_s$. Let x be a smallest counter-example (in terms of $|\mathcal{A}_x|$). Because x is a feasible solution of (P_s), we know by Proposition 25 that there exists a path between all states reachable from s in G_x and $0_{\mathcal{S}}$. Now from Lemma 1.6, we know that there exists a proper, deterministic and stationary policy Π to which we can associate and compute a flux x^{Π} using Proposition 26. Let $1 \geq \lambda \geq 0$ be the maximum value such that $x' := x - \lambda x^{\Pi} \geq 0$. By Proposition 27 we have that $G_{x^{\Pi}}$ is a subgraph of G_x and thus $\lambda > 0$ (as $x > 0$ on $\mathcal{A}_x$). If $\lambda = 1$, x' is a solution to $(J - P)^T x = 0, x \geq 0$ and $x := x^{\Pi} + x'$ provides a decomposition for x, a contradiction (note that $|\mathcal{A}_{x'}| < |\mathcal{A}_x|$ as $\mathcal{A}_{x'}$ must miss at least one action of Π leading to $0_{\mathcal{S}}$ with non zero probability). If $\lambda < 1$, by maximality of λ, there is an arc $a \in \mathcal{A}_x$ such that $x(a) > 0$ and $x'(a) = 0$. Hence $\mathcal{A}_{x'} \subset \mathcal{A}_x$ and $\frac{1}{1-\lambda} x'$ is a solution to (P_s) with $|\mathcal{A}_{x'}| < |\mathcal{A}_x|$. By minimality of the counter-example, we can assume that there exists a decomposition for $\frac{1}{1-\lambda} x'$. Now we can get a decomposition for x from the decomposition for $\frac{1}{1-\lambda} x'$ by scaling the multipliers by 1-λ and using x^{Π} with multiplier λ, this is a contradiction. Clearly, we can make the proof algorithmic and because $A_{x'} \subset A_x$ at each iteration, the algorithm will terminate with a set of k solutions $x_1, ..., x_k$ to (P_s) and a vector x_c satisfying the theorem in at most $|\mathcal{A}_x| - |\mathcal{A}_{x_c}| \leq m - |\mathcal{A}_{x_c}|$ steps. □

The following lemma builds upon similar ideas.

Lemma 1.9

Let $x' \neq 0 \in \mathbb{R}^m$ be a transition cycle. In strongly polynomial time, one can find $k' \in \mathbb{N}$, $x_1, ..., x_{k'}, \in \mathbb{R}^m$,and $\lambda'_1, ..., \lambda'_{k'} \geq 0$ such that $1 \leq k' \leq |\mathcal{A}_{x'}|$, $x'_1, ..., x'_{k'}$ are simple transition cycles and $x' = \sum_{j=1}^{k} \lambda'_{j'} x'_{j'}$.

Proof. We prove first that such a decomposition exists for any transition cycle $x' \neq 0$. Let x' be a smallest counter-example (in terms of $|\mathcal{A}_{x'}|$). We focus on the support graph $G_{x'}$. By minimality of the counter-example, we can assume that $G_{x'}$ is connected. Now $G_{x'}$ has to be strongly connected otherwise it would contradict flow conservation constraints (using similar argument as in Proposition 25). Observe also that $0_{\mathcal{S}}$ is not in $V(G_x)$. Let us consider any action a in $\mathcal{A}_{x'}$ and let us call e the edge between a and the unique node s with $a \in \mathcal{A}(s)$. We can consider the graph G_a obtained by taking the subgraph of $G_{x'} \setminus e$ induced by the vertices that are reachable from a (in $G_{x'} \setminus e$), by 'splitting' action a. Let s be the unique state where a is available. We add two artificial node states s_0, t_0 and an artificial action a_0 that leads to t_0 with probability 1, such that a is removed from the set of actions available in s and a becomes the only action available in s_0. Let G'_a be the corresponding graph. We can consider an instance of s_0-SSP with target state t_0 in G'_a. Now x' can easily be converted into a feasible network flux $\bar{x}$ for the corresponding problem by simply setting $\bar{x}(a') = \frac{x'(a')}{x'(a)}$ for all $a' \neq a_0$ and $\bar{x}(a_0) = 1$.

We can then apply Lemma 1.8 to $\bar{x}$ to generate $x_1, ..., x_k$, $\lambda_1, ..., \lambda_k$ and x_c obeying the corresponding lemma. Now we can convert $x_1, ..., x_k$ into simple transition cycles of our original instance by setting, for all $i = 1, ..., k$ and for all $a' \neq a_0$, $x'_i(a') = x_i(a')$. It follows that $x' = x'(a) \cdot (\sum_{i=1}^{k} \lambda_i x'_i + x_c)$. Remember that $k \geq 1$, so x'_1 exists. Now for $\mu = \min_{a'}\{\frac{x'(a')}{x'_1(a')}\}$, $x'' = x' - \mu x'_1$ is still a transition cycle, but $|\mathcal{A}_{x''}| < |\mathcal{A}_{x'}|$ by the choice of μ, so by minimality of the counter-example, x'' can be decomposed into simple transition cycles and so $x' = x'' + \mu x'_1$ too, a contradiction. We can again make the proof algorithmic and so $k' \leq |A_{x'}|$. □

Theorem 1.7 is a direct corollary of Lemma 1.8 and Lemma 1.9: we apply Lemma 1.9 to the transition cycle x_c returned by Lemma 1.8 .

□

We can now exploit Theorem 1.7 to prove that under our assumptions, we can restrict to deterministic and stationary policies.

Corollary 28
Under Assumption 24, the s-SSP admits an optimal proper, deterministic and stationary policy.

Proof. We know from linear programming that when a LP has a finite optimum, we can find an optimal solution in an extreme point. For (P_s), existence of of finite optimum is guaranteed by Assumption 24 : the first conditions implies the existence of a solution by Lemma 1.4 and the second conditions ensure that the value is bounded. But an extreme point x of P_s cannot be expressed as a convex combination of other points of P_s by definition. As such, using Theorem 1.7, x must be equal to x^{Π} for some proper, deterministic and stationary policy Π. Now $c^T x^{\Pi}$ is precisely the cost of policy Π. Hence we have a feasible solution to our original problem which is optimal for the linear relaxation (P_s). It is thus optimal for the original problem. □

We can deduce from what preceeds a result which is standard for the deterministic shortest path problem: *Bellman optimality conditions.*

Lemma 1.10

Let Π be an optimal proper, deterministic and stationary solution to the s-SSP (under Assumption 24). Let G^s_{Π} be the support graph of Π. For all state vertex s' in G^s_{Π}, Π is optimal for s'-SSP.

Proof. Observe first that s'-SSP satisfies Assumption 24. Now suppose Π is not optimal for s'-SSP. We know from Corollary 28 that s'-SSP admits an optimal proper, deterministic and stationary policy $\Pi_{s'}$. Now the (history-dependent and non stationary) policy Π' that consists in applying policy Π to problem s-SSP, up to when state s' is reached (if it ever is) and then applying policy $\Pi_{s'}$ is a proper policy. The value of this policy

is (strictly) better than the value of Π as there exist realizations where s' is reached, a contradiction. □

1.4 Algorithms

We focused, up to now and without loss of generality, on the s-SSP problem. Bellman optimality conditions (i.e. Lemma 1.10) also tells us that, under Assumption 24, we can actually restrict attention to the SSP problem as well without loss of generality. Indeed we already observed that SSP can be converted to a s-SSP problem by simply adding an artificial state s and a unique action available from s that lead to all states $i = 1, ..., n$ with probability $\frac{1}{n}$. Now there is a one-to-one correspondence between the policies of SSP and the policies of the auxiliary s-SSP problem and hence any proper, deterministic and stationary solution Π to SSP is optimal if and only if it is optimal for the auxiliary problem. But by Lemma 1.10, an optimal policy Π^* for SSP is optimal for s'-SSP for all $s' = 1, ..., n$ (as all s' are in G_{Π^*}). It is easy to see that all theorems from the previous section extend naturally to the SSP setting. Of course, some definitions and results have to be slightly adapted: for instance, the flux vector x^Π associated with a proper deterministic and stationary policy is now $x^\Pi := \sum_{k=0}^{+\infty} x_k^\Pi$ with $y_0^\Pi := \frac{1}{n}\mathbf{1}$ and it satisfies $x^\Pi = (I - P_\Pi)^{-T}\frac{1}{n}\mathbf{1}$ (see Proposition 26 for the previous relation), where P_Π is the $n \times n$ matrix obtained from P by keeping only the rows corresponding to actions in Π. For algorithmic reasons, it is more convenient to deal with the SSP problem as there is no problem of degeneracy: the feasible basic solution x^Π (it is indeed now the basic solution associated with the basis $(I - P_\Pi)^T$) has positive values on the actions in Π. In this section, we will therefore focus on the SSP problem. The corresponding linear programming formulation is (in principle, the right hand side should be $\frac{1}{n}\mathbf{1}$ but we simply rescaled it):

$$\begin{array}{lll} \min & c^T x & \\ (J - P)^T x & = \mathbf{1} & (P) \\ x & \geq 0 & \end{array}$$

One possible way of solving the previous model is to use any polynomial time algorithm for linear programming. This would lead to weakly polynomial time algorithms for SSP. As pointed out in the introduction, there are two standard alternatives for solving a MDP: Value Iteration and Policy Iteration. We prove in the next two sections the convergence of these methods under Assumption 24. Then we give another new iterative method based on the standard primal-dual approach to linear programming: this can be considered as a natural generalization of Dijkstra's algorithm.

1.4.1 Value Iteration

We denote by $\mathcal{P}$ the set of all proper policies for SSP. For all $s = 1, ..., n$, we define $V^*(s)$ to be the optimal value of (P_s) (again under Assumption 24), i.e. $V^*(s) := \min_{\Pi \in \mathcal{P}} c^T x^\Pi$

with $y_0^\Pi = e_s$. This is refered to as the *value* of state s. We have in particular $V^*(s) = \min_{\Pi\in\mathcal{P}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi$ by definition of x_k^Π. In the following, we show that we can switch the min and lim operators with some care. We need first to introduce an auxiliary SSP instance obtained from $(\mathcal{S}, \mathcal{A}, J, P, c)$ by adding an action of cost $M(s)$ for each state $s = 1, ..., n$ that lead to state 0 with probability one, with $M(s)$ "big enough". We call aux-SSP this auxiliary problem (we slightly abuse notation and we still denote by c the corresponding cost function). Observe that in aux-SSP, there are proper policies that terminate in at most k time periods for all $k \geq 1$, from any starting state. Indeed one can always chose an auxiliary action in period $k-1$. Let us denote by $\mathcal{P}^k$ the proper policies in aux-SSP that terminate in at most k steps and by $\mathcal{P}_{aux}$ the proper policies for aux-SSP. Observe that $V_K(s) := \min_{\Pi\in\mathcal{P}^K} \sum_{k=0}^K c^T x_k^\Pi$ is well-defined for each $K \geq 1$. In fact we can prove by induction that it follows the dynamic programming formula: $V_k(s) = \min\{V_{k-1}(s), \min_{a\in\mathcal{A}(s)} c(a) + \sum_{s'} p(s'|a) V_{k-1}(s')\}$ for all $k \geq 2$ and $V_1(s) = M(s)$ for all $s = 1, ..., n$ (an optimal, deterministic non-stationary policy Π_K^* can be recovered easily too): $V_k(s)$ is indeed the optimal value starting from s among policies in $\mathcal{P}^k$. The following result can be seen as an extension of Bellman-Ford algorithm for the deterministic shortest path problem.

Theorem 1.11

For all $s = 1, ..., n$, if $M(s) \geq V^*(s)$, then we have $V^*(s) = \lim_{K\to+\infty} V_K(s)$.

Proof. We will prove that $\min_{\Pi\in\mathcal{P}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi = \lim_{K\to+\infty} \min_{\Pi\in\mathcal{P}^K} \sum_{k=0}^K c^T x_k^\Pi$ with $y_0^\Pi := e_s$, for all $s = 1, ..., n$, by proving both inequalities.

$\boxed{\leq}$ Let Π_K^* be an optimal solution to $\min_{\Pi\in\mathcal{P}^K} \sum_{k=0}^K c^T x_k^\Pi$ computed by dynamic programming (as described above). Π_K^* is a proper policy for aux-SSP for all K. By feasibility of Π_K^*, we thus have $V_K(s) = c^T \sum_{k=0}^K x_k^{\Pi_K^*} \geq \min_{\Pi\in\mathcal{P}_{aux}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi$ (observe that this minimum is well defined since we are still satisfying Assumptions 24 in aux-SSP). By construction $\{V_K(s), K \geq 1\}$ is nonincreasing, hence because it is bounded from below, it converges and $\lim_{K\to+\infty} V_K(s)$ is well-defined. Taking the limit we get $\lim_{K\to+\infty} c^T \sum_{k=0}^K x_k^{\Pi_K^*} \geq \min_{\Pi\in\mathcal{P}_{aux}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi$. But

$$\min_{\Pi\in\mathcal{P}_{aux}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi \geq \min_{\Pi\in\mathcal{P}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi$$

if $M(s)$ is chosen so that auxiliary actions can be assumed not to be used in an optimal policy Π^* for $\mathcal{P}_{aux}$. This is the case for $M(s) \geq V^*(s)$ as we could consider the (non stationary) policy that applies policy Π^* up to the first time we want to use an artificial action and then apply an optimal policy Π^{**} for $\mathcal{P}$: the corresponding policy has a value no greater than the former.

$\boxed{\geq}$ Let Π^* be an optimal proper deterministic and stationary solution to

$$\min_{\Pi\in\mathcal{P}} \lim_{K\to+\infty} \sum_{k=0}^K c^T x_k^\Pi$$

(Π^* exists in our setting by Corollary 28). Let us denote by $\bar{\Pi}$ the policy of $\mathcal{P}_{aux}$ that chooses the auxiliary action for each state. Consider the policy Π_K of $\mathcal{P}^K$ obtained from using Π^* in periods $0, ..., K-1$ and policy $\bar{\Pi}$ in period K. By feasibility of Π_K, we have $c^T x_K^{\Pi_K} + \sum_{k=0}^{K-1} c^T x_k^{\Pi_K} \geq \min_{\Pi \in \mathcal{P}^K} \sum_{k=0}^{K} c^T x_k^{\Pi}$. Now taking the limit as K tends to infinity, we have the result since $\lim_{K \to +\infty} x_K^{\Pi_K} = \lim_{K \to +\infty} x_K^{\Pi^*} = 0$ as Π_K differs from Π^* only in period K, and Π^* is s-proper.

□

Notice that it is easy to find initial values for $M(s)$ satisfying the previous theorem. Indeed one can use $V^{\Pi}(s)$, the values for state s when using policy Π for any s-proper policy Π. We can actually easily find a proper, deterministic and stationary policy for SSP (i.e. for all s simultaneously) by extending Lemma 1.6 to SSP.

The algorithm that consists in computing V_k iteratively is called *Value Iteration*. Value Iteration was already known to converge for SSP in the presence of transition cycles of cost zero, when initialized appropriately, see Bertsekas and Yu [20].

We now explain how to recover an optimal proper, deterministic and stationary policy, if we were lucky enough to get the optimal vector V^* after some iterations (if we build a feasible policy Π_k at each iteration, it may happen that we discover that Π is optimal by computing V^{Π} and observing that it satisfies the Bellman equations). Let us consider the dual linear program (D) of (P) :

$$\begin{array}{ll} \max & y\mathbf{1} \\ y(J-P)^T & \leq c \end{array} \qquad (D)$$

By definition of $V^*(s)$ and by Lemma 1.10, we know that V^* satisfies $V^*(s) = \min_{a \in \mathcal{A}(s)} c(a) + \sum_{s'} p(s'|a) V^*(s')$ for all $s = 1, ..., n$. Also extending Corollary 28 to SSP, we know that there exists an optimal proper deterministic and stationary policy Π^* with $V^*(s) = c(\Pi^*(s)) + \sum_{s'} p(s'|\Pi^*(s)) V^*(s')$ for all $s = 1, .., n$. In particular, $y^* := V^*$ is feasible for (D) and because the pair (x^{Π^*}, y^*) satisfies the complementary slackness conditions, y^* is optimal for (D).

Now let us reverse the complementary slackness conditions. An optimal solution x^* to (P) can satisfy $x^*(a) > 0$ only if $V^*(s) = c(a) + \sum_{s'} p(s'|a) V^*(s')$. Let $\mathcal{A}^*$ be the set of all such actions and let us restrict our instance of SSP to those actions in $\mathcal{A}^*$. Because there is an optimal proper, deterministic and stationary policy Π^* for SSP and because such a policy must use only actions in $\mathcal{A}^*$, we know that there is a path from every state to the target state 0 in the support graph $G^* = (\mathcal{S}^*, \mathcal{A}^*, E^*)$ of this instance. We know from Lemma 1.6 that we can thus find a proper, deterministic and stationary policy Π in time $O(|\mathcal{S}^*| + |\mathcal{A}^*| + |E^*|)$. The pair (x^{Π}, y^*) satisfies the complementary slackness conditions and thus Π is optimal.

Unfortunately, we might never reach the precise value of V^* when iterating VI. However, we can build a proper deterministic and stationary policy Π_k at each step k of Value Iteration by considering an approximate version of the complementary slackness theorem. For all $k \geq 0$, we define $y_k := V_k$, and, for each action a, $\epsilon_a^k := V_k(s^{-1}(a)) - (c(a) + \sum_{s'} p(s'|a) V_k(s'))$. For $\epsilon \geq 0$, we define $\mathcal{A}_\epsilon^k$ the set of actions $a \in \mathcal{A}$ such that

$\epsilon_a^k \geq -\epsilon$ and we denote by SSP_ϵ^k the restriction of our SSP instance to the actions in $\mathcal{A}_\epsilon^k$. Let us denote by $\epsilon_k \geq 0$ the minimum value $\epsilon \geq 0$ such that SSP_ϵ^k admits a proper, deterministic and stationary policy Π^k. Observe that $\epsilon_k \in \{-\epsilon_a^k, a \in \mathcal{A}\}$. We can thus compute ϵ_k and Π_k in strongly polytime using Lemma 1.5 and Lemma 1.6. We will now prove that V^{Π_k} converges to V^* as k tends to infinity ($V^{\Pi_k}(s)$ is the value associated with Π_k when starting from s).

Let us first notice that $\epsilon_k \leq \max_s\{V_k(s) - V^*(s)\}$ (remember $V_k(s) \geq V^*(s)$). Indeed for $\epsilon = \max_s\{V_k(s) - V^*(s)\}$, we have $V^*(s) \leq V_k(s) \leq V^*(s) + \epsilon$ for all s and it follows that for any s and for any optimal policy Π^*, we have $c(\Pi^*(s)) + \sum_{s'} p(s'|\Pi^*(s))V_k(s') \leq c(\Pi^*(s)) + \sum_{s'} p(s'|\Pi^*(s))(V^*(s') + \epsilon) \leq c(\Pi^*(s)) + \sum_{s'} p(s'|\Pi^*(s))V^*(s') + \epsilon = V^*(s) + \epsilon$. It follows that $V_k(s) - (c(\Pi^*(s)) + \sum_{s'} p(s'|\Pi^*(s))V_k(s')) \geq V_k(s) - V^*(s) - \epsilon \geq -\epsilon$ and thus $\Pi^*(s) \in \mathcal{A}_\epsilon^k$. Hence Π^* is a proper deterministic and stationary policy of SSP_ϵ^k and the result follows. It implies in particular that ϵ_k tends to zero as k tends to infinity by Theorem 1.11.

Let us consider the pair (x^{Π_k}, y_k). x^{Π_k} is a solution of (P) but y_k might not be a feasible solution to (D) so it is not a primal/dual pair of solutions. However it almost satisfies the complementary slackness conditions. In particular we have $\sum_{a\in\mathcal{A}} c(a)x^{\Pi_k}(a) = \sum_{a\in\Pi_k} c(a)x^{\Pi_k}(a) \leq \sum_{a\in\Pi_k}(y_k(J-P)^T\mathbf{1}_a + \epsilon_k)x^{\Pi_k}(a) = y_k(J-P)^T x^{\Pi_k} + \sum_{a\in\Pi_k}\epsilon_k x^{\Pi_k}(a) = y_k\mathbf{1} + \epsilon_k\mathbf{1}^T x^{\Pi_k}$. It follows that, as k tends to infinity, $\sum_{a\in\mathcal{A}} c(a)x^{\Pi_k}(a)$ tends to the optimal value of (P). Indeed y_k tends to V^* by Theorem 1.11 (and $V^*\mathbf{1}$ is the optimal value of (D) and (P)), ϵ_k tends to zero by the discusion above, and $\mathbf{1}^T x$ is bounded for proper policies. Therefore x_k^Π tends to be an optimal solution of P, Π_k tends to be an optimal policy, and V^{Π_k} tends to V^*. We sum up the result in the following theorem.

Theorem 1.12

In each iteration k of Value Iteration, one can compute in strongly polynomial time a proper, deterministic and stationary policy Π_k such that V^{Π_k} tends to V^* as k tends to infinity.

1.4.2 Policy Iteration

An alternative to Value Iteration is to use a simplex algorithm to solve (P). In order to do so we need an initial basis. We can use Lemma 1.6 to find a proper deterministic and stationary policy Π. Then as we already observed, $x^\Pi = (I - P_\Pi)^{-T}\mathbf{1}$ is a non-degenerate feasible basic solution of (P). Because the basic solutions are non-degenerate, we can implement any pivot rule from this initial basic solution and the simplex algorithm will converge in a finite number of steps. This type of algorithm is often referred to as *simple policy iteration* in the litterature. This proves that simple PI terminates in a finite number of steps. Unfortunately, most pivot rules are known to be exponential in n and m in the worst case [75].

Theorem 1.13

Under Assumption 24, simple policy iteration converges in a finite number of steps.

In contrast with simple policy iteration, Howard's original policy iteration method [58] changes the actions of a (basic) policy in each state s for which there is an action in $\mathcal{A}(s)$ with negative *reduced cost*. We will prove now that this method converges under Assumption 24, by proving that the method iterates over proper, deterministic and stationary policies and that the cost is decreasing at each iteration. Given a proper, deterministic and stationary policy Π, $x^\Pi = (I - P_\Pi)^{-T}\mathbf{1}$ is the basic feasible solution of (P) associated with the basis $(I - P_\Pi)^T$. We define the *reduced cost* vector associated with c and Π as $\bar{c}^\Pi := c - c_\Pi(I - P_\Pi)^{-T}(J - P)^T$ following linear programming (in order not to overload the notations we consider c as a row vector in this section). Let us denote by $\mathcal{A}^<(\Pi)$ the set of actions a of $\mathcal{A}$ such that $\bar{c}^\Pi(a) < 0$. We know from linear programming that if $\bar{c}^\Pi(a) \geq 0$ for all a, then x^Π (and thus Π) is optimal. If $\mathcal{A}^<(\Pi) \neq \emptyset$, then we can swap actions in Π with actions in $\mathcal{A}^<(\Pi)$ for each state where such an action exists. Let us denote by Π' the resulting policy.

Proposition 29
Π' *is proper and* $c \cdot x^{\Pi'} < c \cdot x^\Pi$

Proof. We denote by y^Π the dual solution associated with Π i.e. $y^\Pi = c_\Pi(I - P_\Pi)^{-T}$. Assume for contradiction that Π' is not proper. Let $G_{\Pi'}$ be the support graph of this policy. Since Π' is not proper, there exists a non empty set of states that are not in $R^-_{G_{\Pi'}}(0)$. It implies that there is a set of vertices V in $V(G_{\Pi'})$ such that $0_\mathcal{S}, 0_\mathcal{A} \notin V$ and $\delta^+(V) = \emptyset$. We can choose for instance $V = V(G_{\Pi'}) \setminus R^-_{G_{\Pi'}}(0)$. Now we choose V minimal with this property. There exists an action a of $\mathcal{A}^<(\Pi)$ in V, otherwise vertices in V are not in $R^-_{G_\Pi}(0)$, a contradiction with Π being proper. Consider the graph G_a obtained by taking the subgraph of $G_{\Pi'}$ induced by the vertices in V that are reachable from a, by removing the edge between a and the unique state s with $a \in \mathcal{A}(s)$, and by adding an artificial state s_0 with a as its unique possible action. Let $\mathcal{A}_a$ be the set of actions in G_a. Note that by minimality of V, every vertex in G_a is in $R^-_{\Pi'}(s)$. Indeed if not we can change the set V by considering instead the vertices in G_a that do not have a path to s.

We can associate a s_0-SSP instance to G_a by considering s as the target state. Π' is a s_0-proper policy for this problem. Now let $x^{\Pi'}$ be the corresponding flux vector (in principle it is defined only on the actions in G_a but we extend the flux on the other actions by setting it to zero). We can interpret $x^{\Pi'}$ as a (non zero) transition cycle of the original problem (the flux is defined on the same set of actions and $x^{\Pi'}(a) = 1$). The vector $x^{\Pi'} \geq 0$ thus satisfies $(J - P)^T x^{\Pi'} = 0$. Now the reduced cost $\bar{c}^\Pi(a') = c(a') - c_\Pi(I - P_\Pi)^{-T}(J - P)^T\mathbf{1}_{a'}$ satisfies $\bar{c}^\Pi(a') \leq 0$ for all $a' \in \mathcal{A}_a$, by definition of Π and Π' (actions in Π have reduced cost 0 and actions in Π' that are not action from Π have a negative reduced cost). Also, as already observed, $\bar{c}^\Pi(a) < 0$. Let us analyze $cx^{\Pi'}$. We have $cx^{\Pi'} = \sum_{a' \in \mathcal{A}_a} c(a') \cdot x^{\Pi'}(a') = (\sum_{a' \in \mathcal{A}_a} \bar{c}^\Pi(a') \cdot x^{\Pi'}(a')) + c_\Pi(I - P_\Pi)^{-T}(J - P)^T x^{\Pi'}$.

Because $(J-P)^T x^{\Pi'} = 0$, we have $c\, x^{\Pi'} = \sum_{a' \in \mathcal{A}_a} \bar{c}^{\Pi}(a')\, x^{\Pi'}(a')$ but this is negative as $x^{\Pi'}(a') > 0$, $\bar{c}^{\Pi}(a') \leq 0$ for all $a' \in \mathcal{A}_a$, and $\bar{c}^{\Pi}(a) < 0$. Therefore $x^{\Pi'}$ is a negative cost transition cycle for our original instance, but this contradicts Assumption 24.

Now that we know that Π' is proper, we can define $x^{\Pi'}$ to be the network flux associated with Π'. We have $c\, x^{\Pi'} - c\, x^{\Pi} = c\, (x^{\Pi'} - x^{\Pi}) = (\bar{c}^{\Pi} + c_{\Pi}(I - P_{\Pi})^{-T}(J - P)^T)(x^{\Pi'} - x^{\Pi})$. But by feasibility of Π' and Π, we have $(J-P)^T x^{\Pi'} = (J-P)^T x^{\Pi} = \mathbf{1}$ and thus $c\, x^{\Pi'} - c\, x^{\Pi} = \bar{c}^{\Pi}\, (x^{\Pi'} - x^{\Pi}) = \bar{c}^{\Pi}\, x^{\Pi'}$ (as $\bar{c}^{\Pi}(a) = 0$ for all $a \in \Pi$ by definition of the current basis) . This latter term is negative as Π' is using at least one action in $\mathcal{A}^{<}(\Pi)$ and the actions in Π have reduced cost zero.

□

Because we have a finite number of proper, deterministic and stationary policies, we can conclude that Howard's policy iteration algorithm converges in a finite number of steps.

Theorem 1.14

Under Assumption 24, Howard's PI method converges in a finite number of steps.

Observe that it is important not to change actions which are not strictly improving. Indeed, in this case it is easy to build deterministic examples where Proposition 29 fails (see for instance Fig. 1.2). As for value iteration, prior to this work policy iteration was not known to converge in this setting. And again, as for VI, unfortunately Howard's Policy Iteration can be exponential in n and m [41].

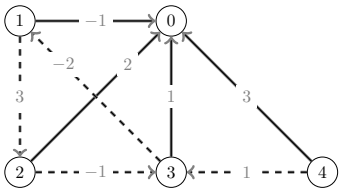


Figure 1.2 – A deterministic shortest path (with target state 0): the dark "actions" represent the current policy, and the dashed "actions" have non positive reduced cost ; changing all actions with non positive reduced cost yield a new policy which is not proper.

1.4.3 The Primal-Dual algorithm: a generalization of Dijkstra's algorithm

Primal-dual algorithms proved very powerful in the design of efficient (exact or approximation) algorithms in combinatorial optimization. Edmonds' algorithm for the weighted matching problem [39] is probably the most celebrated example. It is well-known that

for the deterministic shortest path problem, when the costs are nonnegative, the primal-dual approach corresponds to Dijkstra's algorithm [84]. We extend this approach to the SSP setting. Let us first recall the linear formulation of the problem and its dual :

$$\begin{array}{lll} \min & c^T x & \\ (J-P)^T x & = \mathbf{1} & \text{(P)} \\ x & \geq 0 & \end{array} \qquad \begin{array}{lll} \max & \mathbf{1}^T y & \\ (J-P)\ y & \leq c & \text{(D)} \end{array}$$

The primal-dual algorithm works as follows here. Consider a feasible solution $\bar{y}$ to (D) (initially $\bar{y} = 0$ is feasible if $c \geq 0$). Now let $\bar{\mathcal{A}} := \{a \in \mathcal{A} : \mathbf{1}_a^T\ (J-P)\ y = c_a\}$. We know from complementary slackness that $\bar{y}$ is optimal if and only if there exists $x \geq 0 : (J-P)^T x = \mathbf{1}$ and $x_a = 0, \forall a \notin \bar{\mathcal{A}}$ ((P) admits a finite optimum by Assumption 24). The problem can be rephrased as a so-called restricted primal (RP), where $J_{\bar{\mathcal{A}}}, P_{\bar{\mathcal{A}}}$ and $x_{\bar{\mathcal{A}}}$ are the restrictions of J, P, x to the row in $\bar{\mathcal{A}}$. We also give its corresponding dual problem (DRP).

$$\begin{array}{lll} \min & \mathbf{1}^T z & \\ (J_{\bar{\mathcal{A}}} - P_{\bar{\mathcal{A}}})^T x_{\bar{\mathcal{A}}} + z & = \mathbf{1} & \text{(RP)} \\ x_{\bar{\mathcal{A}}}, z & \geq 0 & \end{array} \qquad \begin{array}{lll} \max & \mathbf{1}^T y & \\ (J_{\bar{\mathcal{A}}} - P_{\bar{\mathcal{A}}})\ y & \leq 0 & \\ y & \leq \mathbf{1} & \\ & & \text{(DRP)} \end{array}$$

If there is a solution of cost 0 to (RP) then we have found an optimal solution to our original problem. Else, we use an optimal, positive cost solution $\underline{y}$ to (DRP) and we update the initial solution by setting $\bar{y} := \bar{y} + \epsilon \underline{y}$ with $\epsilon \geq 0$ maximum with the property that $\bar{y} + \epsilon \underline{y}$ remains feasible for (D), and we iterate. The algorithm is known to converge in a finite number of steps ((RP) being non degenerate, no anti-cycling rule is needed to guarantee finiteness here [84]) and this provides an alternative approach to the problem as long as we can also solve (RP) and (DRP).

Observe that (RP) can be interpreted as a SSP problem with action set $\bar{\mathcal{A}} \cup \{m+1, ..., m+n\}$, where actions $m+k$, for all $k = 1, ..., n$ is an artificial action associated with state k that lead to the target state 0 with probability one. The cost of actions in $\bar{\mathcal{A}}$ is zero while the cost of the artificial actions $m+1, ..., m+n$ is one. The primal-dual approach thus reduces the initial problem to a sequence of simpler 0/1 cost SSP problems. Note that (RP) is actually the problem of maximizing the probability of reaching state 0 using only actions in $\bar{\mathcal{A}}$. This problem is known in the AI community as MAXPROB [73]. Little is known about the complexity of this problem. We know though that it can be solved in weakly polynomial time because it fits into our framework and we can thus solve it using linear programming. We could also use Value Iteration, the simplex method or Policy Iteration as described in the previous subsections. Some simplex rules are known to be exponential in this setting [75]: the question of the existence of a strongly polynomial algorithm is thus wide open for this subproblem too and we believe that MAXPROB deserves attention on its own. Using Howard's policy iteration algorithm to solve the auxiliary problem, the primal-dual approach provides an alternative finite algorithm to solve SSP for nonnegative costs instances.

Theorem 1.15

When $c \geq 0$, the primal-dual algorithm can be initialized with $\bar{y} = 0$ and if the MAXPROB subproblems are solved using Howard's Policy Iteration (or any other simple Policy Iteration method), then it terminates in a finite number of steps.

We are investigating the complexity of this extension of Dijkstra's algorithm to the SSP. Observe that we do not need to impose that c is nonnegative to apply the primal-dual approach. In fact, one can use the standard trick of adding an artificial constraint $\sum_a x_a \leq M$ to the problem, with M "big" to find an initial dual solution and iterate the algorithm [84]. The structure of the subproblem changes but it can still be solved using the simplex method. This provides an alternative approach to Value Iteration and Policy Iteration in the general case too.

One might consider variants of the primal-dual algorithm where the updates of the dual solution do not follow the generic mechanism that guarantees finiteness for general LPs, but instead the updates are 'ad-hoc' and exploit the structure of the problem. It might still be possible to prove finiteness of the algorithm in such cases. For instance the so-called auction algorithm [14] introduced by Bertsekas to solve the (deterministic) shortest path problem can be seen as such an ad-hoc implementation of the primal-dual algorithm. The original version is pseudo-polynomial but it could be turned into a strongly polynomial time algorithm [18]. This might be an alternative route toward a strongly polynomial time algorithm for the stochastic shortest path problem.

1.5 Conclusion and Perspectives

In this chapter, we have introduced a new unifying framework for the stochastic shortest path problem. We have shown that the classic *flow decomposition theorem* extends naturally from network flows to network flux and we have exploited this result to prove that, in this setting, we can restrict to deterministic and stationary policies and that the standard iterative algorithms for Markov Decision Process, i.e. Value Iteration and Policy Iteration, converge. We have also introduced a new promising algorithm that can be seen as a generalization of Dijkstra's algorithm for the deterministic shortest path problem. Our goal is now to implement fast versions of these algorithms and to compare their practical performances on various real-world instances. While the implementation of Value Iteration and Policy Iteration does not seem to provide major numerical challenges (we are still testing the corresponding implementations), our first implementation of the generalization of Dijkstra's algorithm suffers from numerical instability. We are gaining expertise in this area by exchanging with experts of stabilization techniques. We should soon have a stable version of this algorithm implemented. Nevertheless a careful numerical evaluation of the different methods on significative instances is beyond the scope of the current chapter. We leave the corresponding project for future research.

This chapter leaves several fundamental questions unsolved. In particular, we do not provide new insight on the challenging question of whether the stochastic shortest path

problem (or total reward undiscounted MDPs) can be solved in strongly polynomial time. We conclude though with a few (possibly) simpler questions that, we believe, deserve some attention on their own and that might help addressing the former : Is it possible to solve MAXPROB in strongly polynomial time ? Can we bound the number of iterations of our variant of Dijkstra's algorithm by a polynomial ? by a polynomial in n and m ? Can the stochastic shortest path problem be solved in strongly polynomial time when the costs are nonnegative ?

Chapter 2

Golf Strategy Optimization for professional golfers performances estimation on the PGA Tour

2.1 Introduction

The PGA tour is a competitive Golf circuit which takes place mostly in North America. Every year, between 132 and 156 male professional golfers compete on about fifty events, including the U.S. Open, the PGA championship, the Masters Tournament and three of the four majors. In this chapter, our goal is to develop new methods to predict the scores of professional golfers during events of PGA. The methodology we develop can be used for both predictive analysis (anticipate results) and prescriptive analysis (help golfers to decide upon which skills to improve to maximize winning).

Golf is a sport where the player has to put a ball in a cup with the help of golf club in a minimum number of shots (see [59] for the official rules of golf in 2019). The field where the golfer plays is called a *golf course* (or *course* for short). A course consists of eighteen *holes*. A *hole* is composed of different type of grounds:

- the *tee* where the ball is placed at the beginning of the game where the grass is short. The golfer can use a tee [1] in order to raise the ball for his first shot;
- the *green* where the grass is the most closely mown and where the hole (*cup*) and a flag-stick (*pin*) are placed;
- the *fairway* is a part of the hole between the tee and the green where the grass is kept short;

[1] same name but it refers to a small piece of wood T-shaped

- different types of *rough* (depending of the height of the grass) which are the areas around the fairway. Usually, the further you get from the fairway, the higher the grass is;
- the *bunkers*, hollow areas filled with sand where it is difficult to hit from;
- different obstacles like water or trees.

The score of a golfer on a hole is the number of shots he made in order to put the ball in the cup, plus possible penalties (*e.g.* when the ball fall into the water, or goes *out-of-bound* of the course) [59].

There are two main types of competitions in Golf:

- The *Stroke Play* is a mode where all players compete again each other. There is no limitation on the number of players. Each player plays on the 18 holes (or 72 for professional tournament) and the final score of a player is obtained by adding up each score on the different holes. The winner is the player with the lower final score.
- The *Match Play* is a 2-player mode where the player who has the lower score on a hole wins 1 point. If there is a draw, each player wins 0.5 point. At the end of the eighteen holes, the golfer with the most points wins (note that if the difference of points is more than the number of remaining holes, the players do not play the last holes).

At the beginning of a specific hole, the golfer places the ball on the tee. His score on the hole is initialized to 0. The golfer shoots the ball from the tee ground to the cup. Each time the golfer shoots, he chooses a direction and a distance of shooting and his score is increased by 1, or by 2 if the ball falls into the water, or out-of-bound (when the ball goes in the water or out-of-bound, a new ball in placed at a specific position: typically the previous position for out-of-bound and the entry point for water obstacles). The problem which consists in choosing the 'right' directions and distances for each shot in order to minimize the final score on a specific hole is what we call the golfer's strategy optimization problem (or sometimes, for short, the golfer's problem). In this chapter, we assume that the golfer is not influenced by his previous shots or his position in the leader-board. Even if a professional golfer might take more risks if he is far behind in order to catch up with the leaders, or if a missed-shot streak could impact in a bad way the play of the golfer (while a series of good shots could have the opposite effect), we do not consider these aspects. This kind of impacts are not to be under-estimated, in golf like in other sports[100][74]. However, professional golfers do not often take such risks and they usually play regardless of their position in the leader-board.

With these assumptions, the golfer's problem can be modeled as an Markov Decision Process (MDP) and especially as a Stochastic Shortest Path Problem (SSP). We will detail this later, but for formal definition of SSP, we encourage the reader to refer to chapter 1.

Broadie was a precursor in assessing golfer's performances through advanced quantitative analysis. In [24], he introduced a new statistical measure called 'Strokes gained' in order to rank professional golfers in different parts of the game of Golf (typically for different distances, types of grounds...). In [6], Bansal and Broadie study the impact of the size of the hole on the putting [2], for either professional and amateur golfers with simulation techniques. Using the same techniques, Broadie and Ko in [26] study the impact of distance and direction errors on Golf scores. In [25], Broadie explains how to use Strokes gained in order to help golfers to improve their skills and performances.

Sugawara, Kawamura and Suzuki [98] used Q-Learning in order to optimize golfers' strategies, taking into account the course layout, and the golfers' skills using simulation models. Drappi and Co Ting Keh [36] predict the discrete probability distribution of a golfer's score, shot after shot, taking into account the golfer's personal skill, the difficulty and the conditions of the course, using learning and Bayesian techniques.

Markov Decisions processes have already been used in Golf. Lowell Heiny [67] used stochastic process and Markov chains in order to predict scores. He considered that golf could be modeled as an absorbing Markov chain, where the transition probabilities are given from Shotlink Database (see the introduction for details on Shotlink). As we try to infer the will of the player from the data, Lowell Heiny used directly 'gains' (a mark given for each shot) in order to build the transitions [67]. Prior to his work, Maher applied Markov chain theory to several sports: tennis, soccer, darts, golf and snooker [68].

Markov decision processes has been also used in many other sports. In 2013, Terroba, Kosters and al [102] used Markov Decision Processes in order to compute optimal strategies in tennis. They study video sequences in order to build the transition probabilities. In 2012, Trumbelj and al. [105] modeled Basket-Ball as a Markov Decision Process. There transition probabilities are built from a play-by-play NBA program, and statistics of the teams themselves. With the same idea, Routley [92] applied Markov decision processes algorithms in order to find the optimal strategy of a team in ice hockey. Pfeiffer et al. [85] in 2010 decomposed the different time period in table tennis matches in order to model this sport as a MDP. Hoffmeister and Rambau [55] [56] used a 2-scale approach with two types of MDP in order to model different sports, and especially beach volley-ball. The first one is the 'strategic' MDP, where the macro decisions of the game are taken, and the second one is needed to build the transition probabilities of the strategic MDP by simulation.

In this study we model the golfer's strategy optimization problem as a stochastic shortest path problem in the same spirit as [98] and [26] but with a different target as we will detail later. We need to define the states, the actions available in each state, the costs of these actions and the transition matrix of the SSP. In our model, the states are the different positions where the ball can stand (we discretize the hole). The actions are shots that the golfer can perform in a particular state: it is basically defined by a targeted direction and a targeted distance. The cost of an action is 1, or 2 if there is penalty (if the ball fall into the water or out-of-bound).

[2] shots on the green

The transition matrix contains the probabilities to end up in any state, given that we perform a particular action. Ideally, in order to have this kind of information for a particular golfer, he would have to shoot hundreds of balls from different distances, different directions and different grounds (fairway, rough and bunkers): both the intention of the shot and its realizations are needed to compute an empirical transition matrix. Beside, once the theoretical distribution of the players gathered (we define what it is later), we need a simulator in order to evaluate the true final position of impact of the ball given the weather conditions, the roll of the ball, the obstacle on the balla trajectory and so on. We developed a simple simulator that take into account some of these aspects under certain assumptions that we detail later in this chapter.

Shotlink is a huge database which collect millions of shots of professional golfers during international competitions [86]. This database gives access to a large number of information among which coordinates and distance to the pin before and after each shot, type of ground (fairway, rough, bunkers) from where the golfer has shot and a large number of other information. Unfortunately, we cannot access the intention of the player through Shotlink. One way to infer the intention of the player uses video sequences analysis in order to get the physical signs of potential targets. In 2006, Beetz et al. used this method in football [9]. We did not have access to such information so, instead we developed a new methodology to infer the intention of a golfer from general knowledge of professional golfer's strategies. We detail this methodology later in the chapter.

Once the SSP model is set, we use algorithms from chapter 1 to solve the golfer's strategy optimization problem. Now the originality of our approach comes from the fact that we do not compute the optimal strategy to help the professional golfers to improve their game. Instead, we use it to predict their score. In order to be able to predict golfers scores, we first compute the optimal strategy for a player. As the player is a professional golfer, we assume that he is playing (close to) his optimal strategy according to his personal skill, the course layout, etc... Thus, we are able to create a 'numerical clone' of a player. This clone can play thousands of holes which allows to build statistical results such as distribution of scores or probability of winning. The same type of techniques is used in [28] in the game of Go in order to train AI. Monte Carlo simulation were also used in [79] in order to compute the probability for a player to win a match in tennis.

Organization of the chapter

In the first part of this chapter we describe precisely how we model the golfer's strategy optimization problem as a SSP. Then we explain how we infer the intention of the golfer from the Shotlink database. In order to evaluate the accuracy of our assumptions and the quality of our model in general, we compared the prediction we make with our model to what has happened in reality. For the two main types of golf competitions (Stroke Play and Match Play), we developed ad-hoc methods [23] inspired by bootstrapping [4][40] to assess the performances of our approach. The numerical results and the validations methods are explained in the last part of this chapter.

We use the example of Phil Mickelson to illustrate the model. Then we present and

analyze our results for two competitions: one in stroke-play at Augusta National Golf 2017 and one in match-play at the Ryder-Cup 2018, before concluding and giving some perspectives.

2.2 Modeling the golfer's problem as a SSP

In this section, we describe in more details how we create an instance of SSP for a given player and a given hole. This requires to define states, actions, action costs and a transition matrix.

2.2.1 The States

In our model, a hole is a bounded rectangle region encapsulating tee, fairway, green and the main obstacles within an area where the golfer might end up while playing rationally. The states are pairs (position, ground) of the hole. We assume that the hole is flat and we discretize the hole into a grid of identical cells in order to restrict to a finite set of states (see figures 2.1 2.2 for an illustration).

The choice of discretization is a trade-off between precision and computation time. Indeed, the smaller the cells are, the better the precision is (we know more precisely where the ball stands) but the higher the computation time is (the model size increases). The discretization size is consequently a parameter of our model. For the numerical experiments, we considered a square size of one meter which offers a good trade-off.

Note that in our case the state space has to be finite in order to define a SSP instance, but alternative exist, such as non finite or continuous state spaces [37].

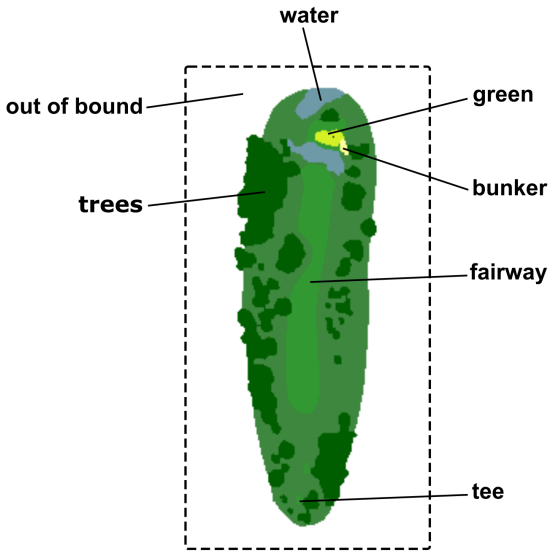


Figure 2.1 – The 15^{th} hole of Augusta National Golf Club

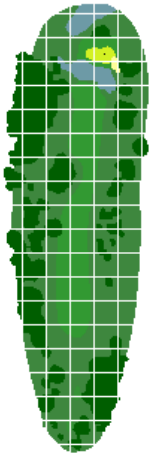


Figure 2.2 – An illustration of our discretization approach

The entries of our model for the holes are pictures of these holes (see example in figure 2.1). Each pixel of the pictures has a color, which represent a type of ground. The flag is represented in black. In the discretization, the ground type of a cell is basically the ground type of the majority of the pixels it contains. We also add a sink state (the pin).

2.2.2 The Actions

In each state, the actions correspond to the different possible shots. In our model, an action consists (except for green states that we describe later) in a targeted (flying) direction and distance. An action defines the target of the player.

Note that in real conditions, the golfer chooses a direction, a golf club and usually an intensity for his shot. But each club is used for specific distance, so our approach matches the decisions the golfer has to make in real conditions.

In the following, we actually treat differently the actions performed on non-green states, and those performed on the green.

Actions performed in non-green states

An action performed in a non-green state consists, as already discussed, in a distance and a direction. We discretize the distances and directions so as to define a finite set of actions. For the distances, the discretization depends on the distance of the shot.

Indeed, the discretization has to be greater if the golfer shoots far, and smaller if the golfer shoots closer: the golfer cannot target two points that are close when the distance of the shot is big, whereas it can be possible when the distance of shot is short. If the distance of the shot is greater than 30 meters, we assume that the player is able to target any multiple of 5 (meters). Under 30 meters, we assume that the golfer can target any multiple of 2 (meters). This limit of 30 meters corresponds to the *short game* which includes *chipping* and *short pitching*. In the short game, the golfer is usually more precise, so the discretization has to be smaller.

For the directions, we assume that the golfer can aim at any even angle (in degree from 0 to 360): this is also a parameter of our model which can be changed. This parameter has been chosen so as to have a targeted direction error lower than 3.5 meters for a distance of 100 meters, which seems reasonable given the fact that golfers aim 'visually'. The choice of discretization for distances and direction is different in nature from the choice of discretization for the states. For the states, the smaller the discretization is, the more precise the model is: there is no disadvantage to lower this discretization if we do not care about computational time complexity. Whereas for the actions, if the discretization is 'too small', we artificially define actions that a golfer cannot perform in real conditions: it would 'allow' the player to play more shots, even if the golfer is not able to play them. The discretization we took is based on the knowledge of professional golfer's play.

Actions performed in the green

For the states on the green, the layout of the field is quite different from the rest of the hole (assuming that the hole is flat is no longer reasonable). We would need another specific simulator, so we manage the actions on the green differently. We assume that the average number of shots the player has to shoot to put the ball in the cup from any point on the green is a function of the distance to the pin only, which is a very natural assumption, as studied by Tierney and Cook [103] (see figure 2.3).

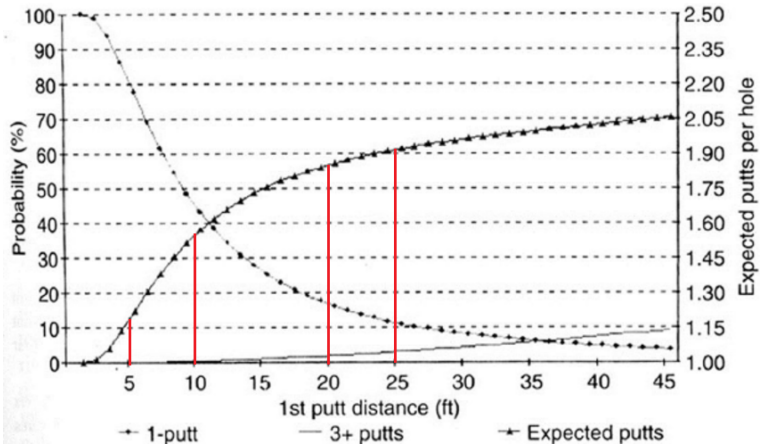


Figure 2.3 – Probabilities of 1 and more putts according to the distance to the pin on the green (figure from [103])

Figure 2.3 shows the probability of 1 putt, the probability of more than 3 putts and the expected number of putts to put the ball in the cup according to the distance to the cup. We can read for example, that the expected number of shots when the ball is around 10 feet far from the cup is around 1.54. We can also see that the probability to put the ball in the cup in one shot when the ball is 20 feet far from the cup is around 17%.

Consequently, for each state on the green, we define only one action, that leads to the cup with probability 1. The cost of this action is the average number of shots the player makes to put the ball in the cup. The cost's definition, which depends on the distance to the pin and the golfer himself is described in the following section. Note that we implicitly assume that it is impossible to enter the cup from outside the green, which is also quite reasonable (even though it may happen).

2.2.3 The Cost Function

As we defined differently the actions performed from the green and the actions which are not, we define differently their costs as well.

For each shot which is performed out of the green, the player score is increased by 1, or 2 if there is a penalty. A penalty arises when the ball falls into the water, or when the ball goes out-of-bound. In this case the next state is the previous position for out-of-bound and the entry point for water obstacles. Since we know the transition matrix of the model (described in the following section), we can define the cost of an action as the average of the costs of its realizations.

On the green, there is only one action per state. The cost of an action depends on the golfer, and the distance between the (middle of the) state and the pin. This cost represents the average number of shots the player makes to put the ball in the pin from this distance. From the Shotlink database, we know how many shots the player made

to reach the pin according to the distance on a certain period. Here is an example for Phil Mickelson in 2018 (see figure 2.4)

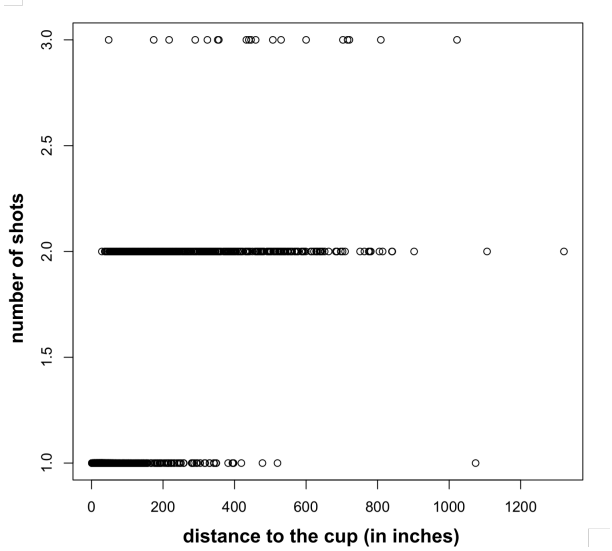


Figure 2.4 – Number of putts according to the distance (in inches) for Phil Mickelson in 2018

From this, we average over the number of shots with 'slices' of

- 1 foot large when the distance to the pin is below 10 feet
- 5 feet large when the distance to the pin is between 10 and 25 feet
- We average over all the remaining shots after 25 feet

For Phil Mickelson in 2018, we obtain:

Distance state/pin (in feet)	cost
< 1	1
< 2	1
< 3	1.005
< 4	1.04762
< 5	1.14583
< 6	1.26984
< 7	1.4375
< 8	1.36585
< 9	1.45238
< 10	1.57576
< 15	1.67949
< 20	1.81651
< 25	1.7963
≥ 25	1.98655

We can plot the average number of putts according to the distance to the pin (see figure 2.5).

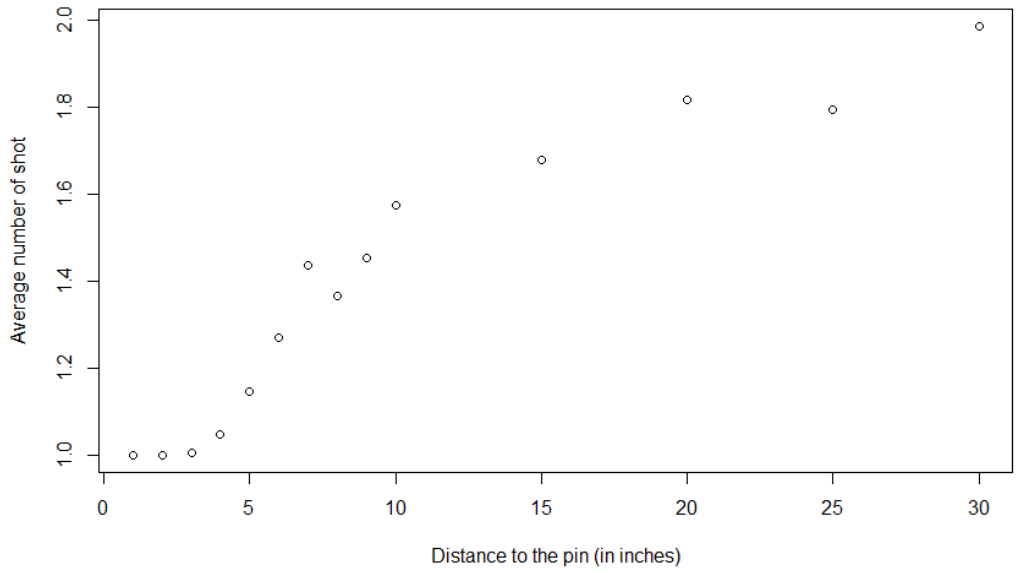


Figure 2.5 – Average number of putts according to the distance to the pin

Note that the empirical values we get are consistent with the graphic from Tierney and Cook (see figure 2.3).

We can see some statistical aberration between 7 and 8 feet: the average number of shots is greater when the player shots from 8 feet than from 7. We assume that these errors are due to the data. Since a green state is one square-meter large, we smooth this effect by picking 100 points at random in this state (which correspond to possible 'true' positions), compute the cost from each point and average over these 100 realizations.

Now that we defined the states, the actions from these states and the costs of these actions, let us define the probabilities: the transition matrix.

2.2.4 The Transition Matrix

To define the transition matrix, we have to know the probabilities to end up in a particular state, given that we performed a particular action. We build the transition matrix in two steps: we first define the player's theoretical distribution (of the impact point) around the target, and then simulate these results on the hole. The theoretical statistics of a player represent the intrinsic shot's deviations of the player, while the transition matrix also depends on the obstacles, the roll of the ball, etc...

Theoretical distributions

One possible way to get access to theoretical distribution of a player is to ask him perform all possible actions (distance and direction) hundreds of times in order to get the empirical deviations of his shots. Obviously this approach is unfeasible in practice for professional golfers of the PGA tour.

In our model we infer the theoretical distribution thanks to the Shotlink database. We describe the method we use in the next sections.

Simulation on the Hole

From the theoretical distribution of a golfer (a set of realizations of the impact of shots with the ground around the target point defined by the action) we simulate the shots on the hole we consider: we plug impact of the ball on the hole and we change the point of arrival if an obstacle is on the trajectory of the ball, or if the rules of golf say so.

We developed a simple simulator, assuming that:

- there is no rolling: the ball ends up exactly where it falls.
- the trees have infinite height: the golfer cannot shoot over a tree.
- the impact between the ball and the obstacles (trees) are very naive: when a ball hit a tree, it ends up at the earlier point in the trajectory before the tree.
- when the ball falls into the water or goes out-of-bound, the ball is repositioned where it was before shooting out-of-bound, and at the point of entrance for water obstacles. Then, a penalty occurs.

Our simulator is quite naive and can obviously be improved. However it is an independent piece of our model: it 'only' affects the probabilities of the transition matrix. We could easily plug in any more advanced simulator (e.g. game).

To sum up, from an action (a target point) if we have access to the distribution of impacts around the target, we can simulate these results on an actual hole and see in which cell of the hole (in which state) the ball would end up. Then we obtain an empirical probability distribution to end up in a state for a given action, which is exactly the transition matrix. Let us now describe how we can build the theoretical distribution.

2.3 Statistical Inference

In this section we will explain how we infer the theoretical distribution of a golfer from the Shotlink database. The main issue is to infer the aim of the player only from realizations of shots during professional competitions.

2.3.1 Shotlink Database

Shotlink is a huge database which collects shots of golfers during professional competitions [86] since 1983. Among all the quantitative information gathered by Shotlink, we can have access to:

- the name of the golfer who shoots;
- the name of the tournament, of the course, and the hole ;
- the type of ground before and after the shot;
- the coordinates of the ball before and after the shot;
- the distance to the pin before and after the shot;
- whether the shot is a recovery shot or not (*i.e.* if the players just wanted to reach a place easier to shoot from);
- the type of game: Approach the green (middle and long game), Around the green (short game), putting (on the green) or driving (from the tee).

We can get access to a lot of other informations (see Annex 3.6).

These information are our raw material to construct the theoretical distribution. In order to be able to relate the level of the players at a precise instant, we restrict to the data of one year. The main informations missing (but quite impossible to collect without assumptions) is the intention of the golfer when he shoots (that is, his original target). In order to do that, we distinguish two cases: the shots made from the tee and the others (off the tee).

2.3.2 Shots off the tee

The goal of the golfer is to put the ball in the cup. One natural way to infer the target is to assume that the golfer is targeting the pin directly (when the distance permits) for the shots off the tee.

In order to validate this assumption we need to gather all the shots played off the tee (at a distance where the pin is reachable). Then, we draw 'profiles' like so: for a shot, we have the coordinates of the point from which the shot has been made and the point of arrival of the same shot. We also have access to the coordinates of the pin of the hole on which the golfer shoots. We first draw at the origin all the starting points of the shots, as if the golfer shoots from the same point. Then on a vertical line we draw the points corresponding to the pins: ordinates the corresponding shots are the distance between the starting point of the shot to the pin. Finally, we draw the arrival points relatively to the corresponding pin position and we link with a line between the two. We give an example with two shots in figure 2.6.

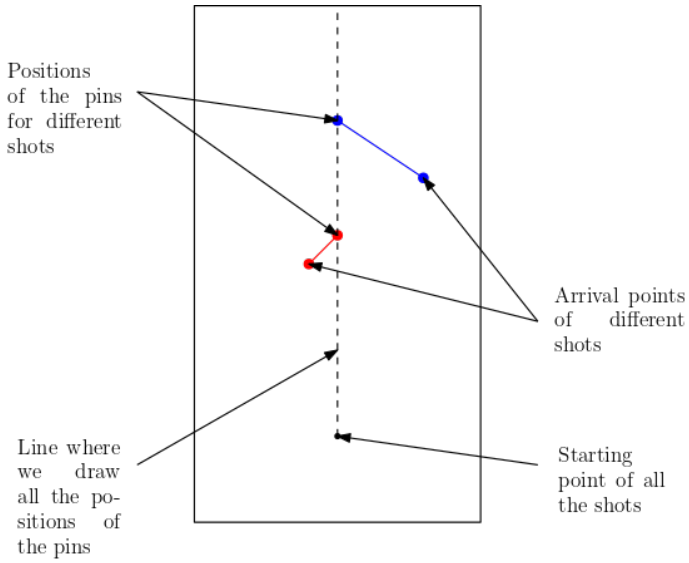


Figure 2.6 – Construction of a profile

We compute the profiles (the drawing described above) of the player for two types of games: Approach the green and Around the green, and for four types of ground: Fairway, Primary Rough, Intermediate Rough and Bunkers. The distances are in inches (figure 2.7, 2.8, 2.9, 2.10, 2.11, 2.12, 2.13, 2.14)

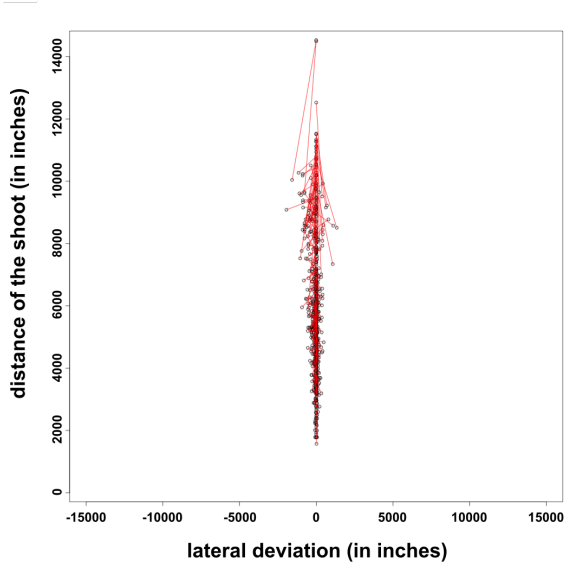


Figure 2.7 – Profile of Phil Mickelson for approach the Green on the Fairway

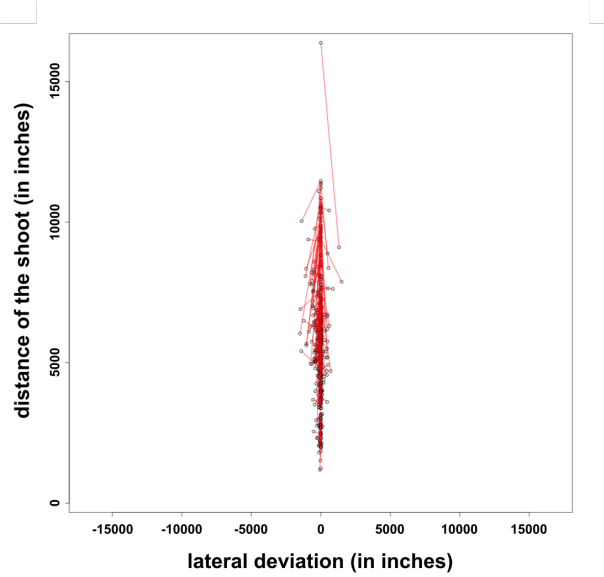


Figure 2.8 – Profile of Phil Mickelson for approach the Green on Primary Rough

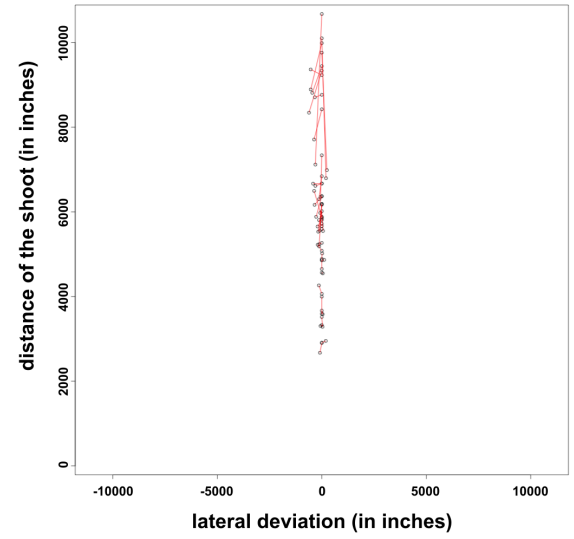


Figure 2.9 – Profile of Phil Mickelson for approach the Green on Intermediate Rough

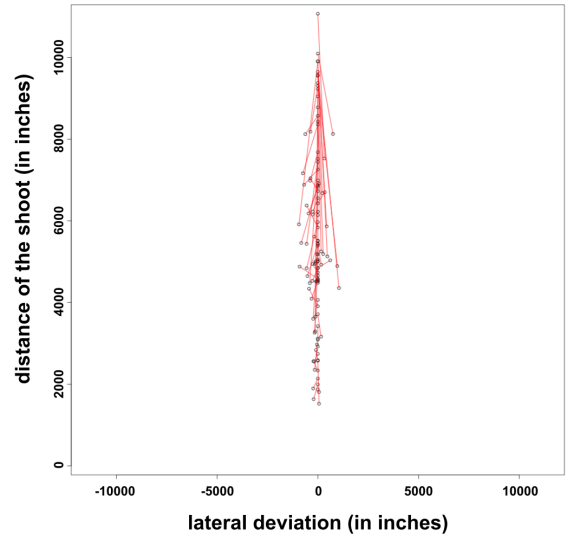


Figure 2.10 – Profile of Phil Mickelson for approach the Green on Bunkers

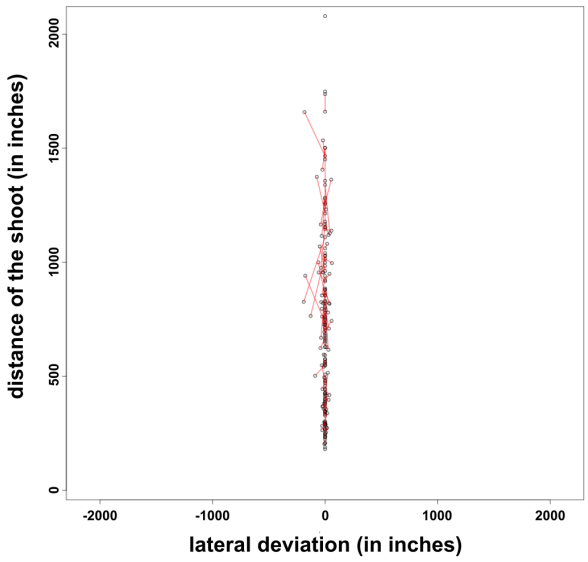


Figure 2.11 – Profile of Phil Mickelson for around the Green on the Fairway

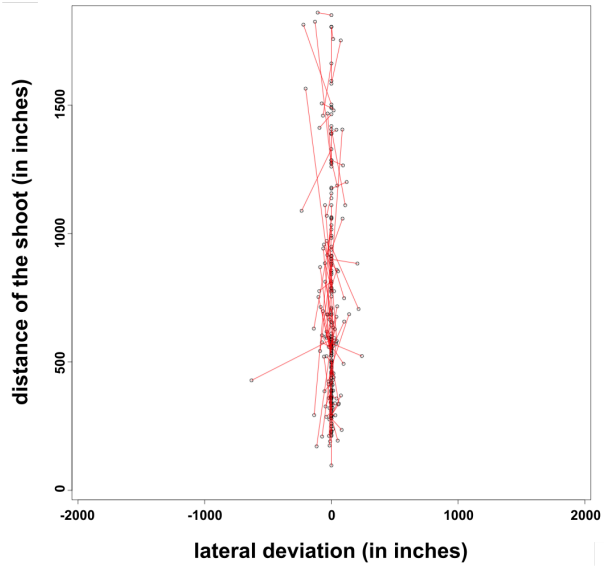


Figure 2.12 – Profile of Phil Mickelson for around the Green on Primary Rough

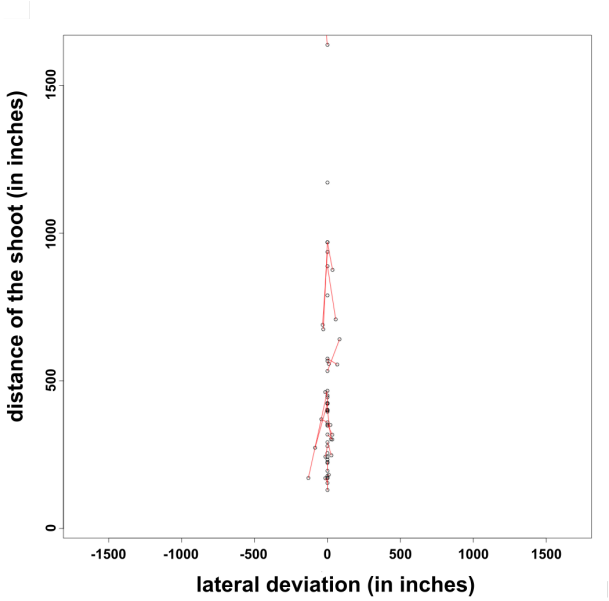


Figure 2.13 – Profile of Phil Mickelson for around the Green on Intermediate Rough

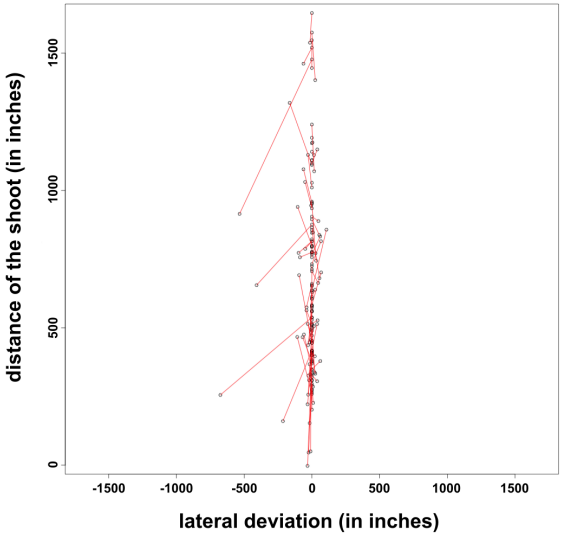


Figure 2.14 – Profile of Phil Mickelson for around the Green on Bunkers

We consider that:

- long game cover shots over 100 meters long, excluding driving
- middle game cover shots between 30 and 100 meters long
- short game cover shots below 30 meters long, excluding putting

We can observe 'visually' that in most cases, the pair of points (hypothetical target and arrival point) are close. For pairs that are remote, there are two possible explanations: either the golfer has completely missed his shot (unlikely for professional golfers, unlike he hit a tree), or the golfer was not targeting the flag. In both cases, we choose to erase the corresponding data. We erase these data according to two criteria: a flat distance and a percentage of the distance of the shot.a For the short game we deleted the shots that end up more than 5 meters away from the hypothetical target and more than 10% of the distance of the shot. For the middle game, we deleted the shots that end up more than 30 meters away from the hypothetical target and more than 10% of the distance of the shot. These parameters seem reasonable for professional golfers and can be modified.

A first approach would be to use these profiles as theoretical distribution directly: these profiles provide us pairs of (target, realization) which is enough to build the theoretical deviation of shots for a golfer. However, the problem of this approach is the lack of data for some type of ground. Indeed, from bunkers or intermediate rough, or for the

short games, there are some distances with almost no historical shot. The issue is that if, for instance, there is only one shot played for a particular distance d on a particular ground g. Then if we adopt this first approach, the model would consider that every time the golfer is targeting a point from distance d on ground g, the ball will end up in the only state for which we have a realization. This is a serious issue as we want to assess the general behavior of a player and not consider that a particular shot relates the average play. So we have to manage the lack of data for some distance/ground.

So the use of raw data alone is not satisfying. We have to 'regenerate' plausible additional data in order to cover all the distances in all types of grounds.

Creation of 'Stars' on the Fairway

In order to generate new plausible data, we want to take as reference the ground on which we have the greater number of data: the fairway. But already in the fairway, we might have the issue discussed above for some distance. So we start by regenerating new data for the fairway.

The main problem is to generate data for any distance. To be able to do so, we assume that there is a very simple linear dependency **on the fairway** between the distance of the shot and the deviation (depth and lateral deviation). That is, the deviation at distance d is twice the deviation at distance $\frac{d}{2}$ (figure 2.7, 2.11).

A star is drawn considering that all the shots are targeting a same artificial point: the aggregation point. From the shots of the profile, we align all the pins of the corresponding shot to the aggregation point and we multiply the coordinates of the arrival point of the corresponding point with a factor $\frac{d_a}{d_s}$ where d_a is the distance of distance of aggregation and d_s the distance of the shot (see figure 2.15).

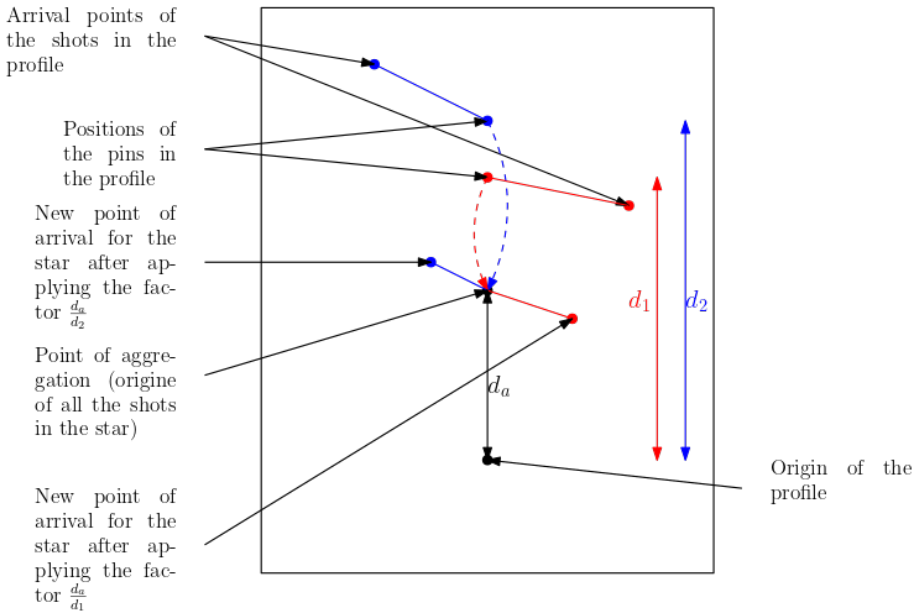


Figure 2.15 – Construction of a star

Thus on the fairway, we create stars for the three different types of game: long, middle and short game (see figures 2.16, 2.17, 2.18 for Phil Mickelson with the 2018's data). The distances of aggregation are defined as follow. We also remind the definition of short, middle and long game. Note that the short game does not include putting and that the long game does not include driving.

	Range (in m)	Distance of aggregation (in m)
Short Game	$[1, 30]$	10
middle Game	$]30, 100]$	55
Long Game	> 100	130

At this point, we want to test if our linear dependency between distance and deviation is reasonable enough. We consider that the linearity between the distance of the shot and both depth and lateral deviations is proven if both abscissa and ordinates of the points of the stars follow normal laws, as the number of realization is sufficiently big to assume that the central limit theorem is true.

We tested with the Shapiro-Wilk test if the normal assumption is reasonable. Shapiro-Wilk test is testing whether a set of real number is likely to be a realization of a normal distribution or not. It returns a p-value: if the p-value is greater than a threshold (usually 0.05), we can assume that the distribution follow normal law. The greater the p-value is, the most certain of the normality of the distribution we are. Here are the p-values of Shapiro-Wilk test for Phil Mickelson with the data from 2018:

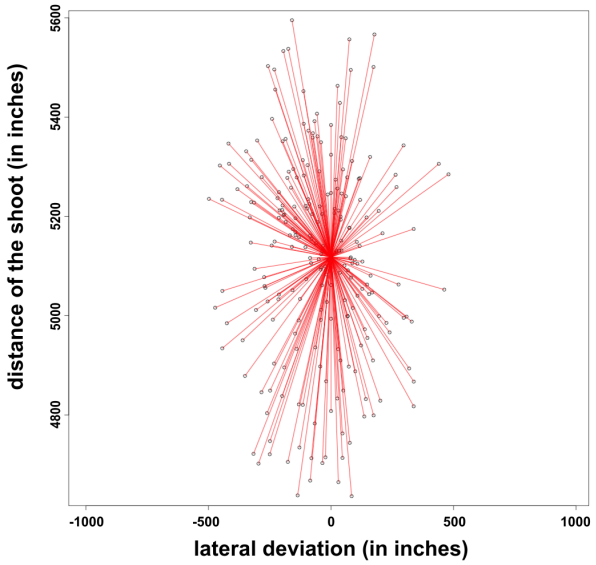


Figure 2.16 – Star of Phil Mickelson for long game

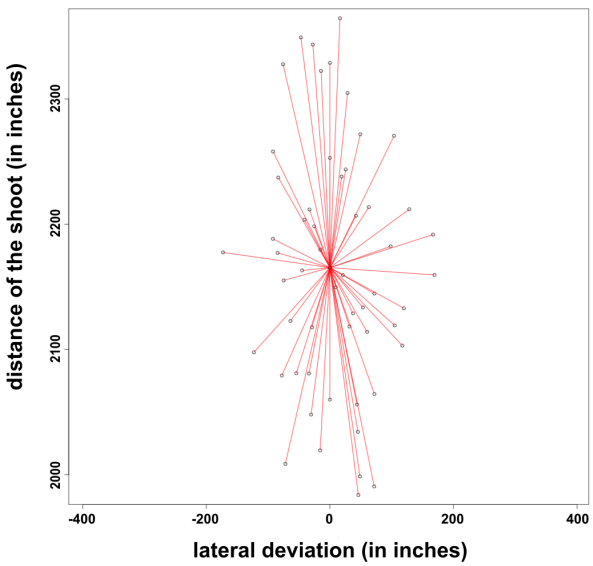


Figure 2.17 – Star of Phil Mickelson for middle game

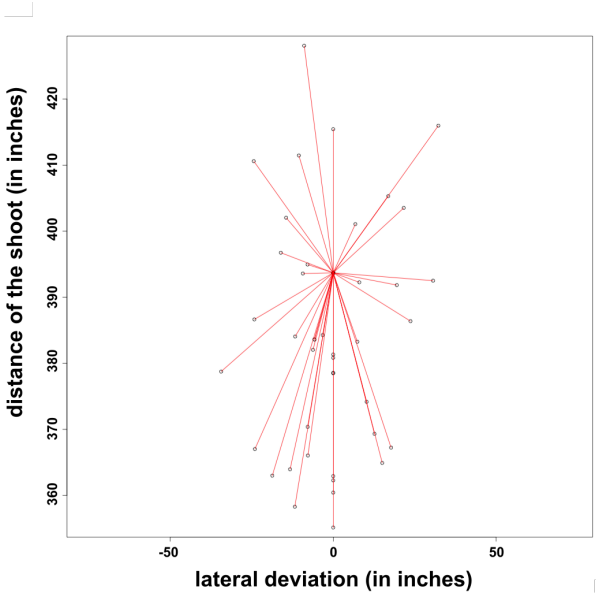


Figure 2.18 – Star of Phil Mickelson for short game

	Abscissa	Ordinates
Short Game	0.5838	0.2264
Middle Game	0.919	0.4791
Long Game	0.7074	0.06258

In the results above, if we take a threshold of 0.05, we consider that the distribution follows a normal law if the p-value is above 0.05. For the abscissa, the p-value is great enough to conclude about the normality of the distribution. This confirms that our assumption that players play the same way in every direction. For the ordinates of the short game, even if the p-value is smaller, we can reasonably conclude the same way. For the ordinates of the long game, however, the results are more questionable. On the one hand, the p-value is greater than 0.05. On the other hand, it is very close to the threshold we (arbitrarily) defined. This could mean that for the long game, the golfers may have different deviation before and after their target, which is empirically true if they play 'full shot' (with full intensity). In order to simplify our assumptions, we will still consider that the ordinates of the long game follow a normal law.

We obtain similar results with other players and different time periods. The linear dependency between the distance of shots and deviations are reasonable enough. Now we are able to regenerate data for all the distances, and for all grounds.

Regenerate data for all distances

With the previous results, we can reasonably assume that the abscissa and the ordinates of the stars follow normal distribution. As the stars were built applying a to the deviations of the shots a factor depending on the distance of the shot, we can reasonably assume that the dependency between the deviations (depth and lateral) is a linear function of the distance of the shots.

We now generate new plausible data on the fairway like so:

- as we assume that the coordinates of the points of the stars follow normal law, we compute the empiric parameters of the corresponding normal law (μ_d, σ_d), (μ_l, σ_l) (for the deviations in depth, and lateral deviation);
- for every distance of shots that we want to generate new plausible data, we simulate n_{shots} points of coordinates $(\mathcal{N}(\mu_d, \sigma_d), \mathcal{N}(\mu_l, \sigma_l))$ following normal laws;
- we multiply the coordinates of these points by $\frac{d_s}{d_a}$ as we assume there is a linear dependency between the deviations and the distances.

As we assumed in section 2.2.2 that for the long game and the middle game a golfer is able to play every 5 meters, and 2 meters for the short game, we simulate shots accordingly. The number of shots we simulate for each distance is a parameter of our model. For our numerical experiment, we took $n_{shots} = 20$. This parameter has been chosen so that the results are not sensitive to the random generation: the results that we detail later on the optimal strategies are the same with different random generations (figures 2.19 2.20 2.21 2.22).

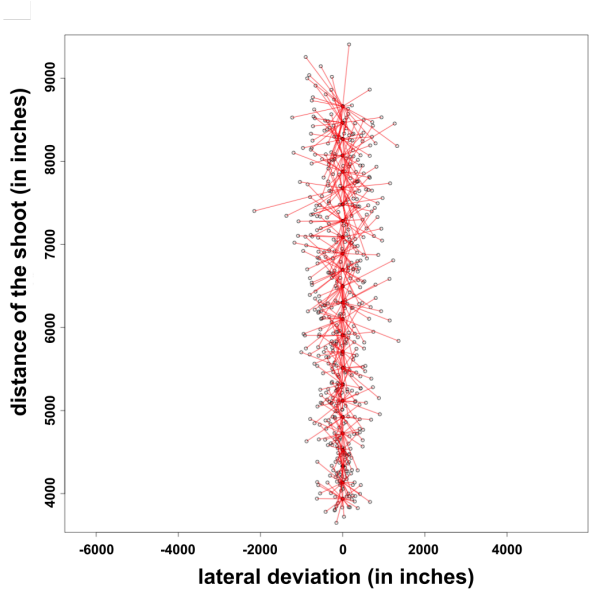


Figure 2.19 – Regeneration of plausible data for Phil Mickelson for the long game

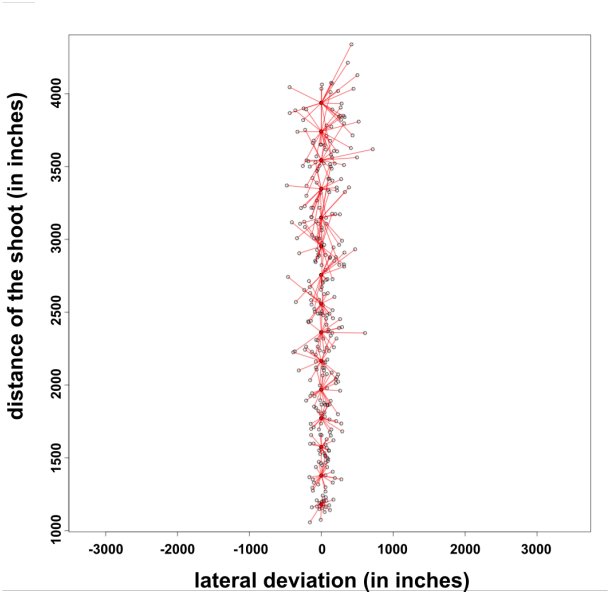


Figure 2.20 – Regeneration of plausible data for Phil Mickelson for the middle game

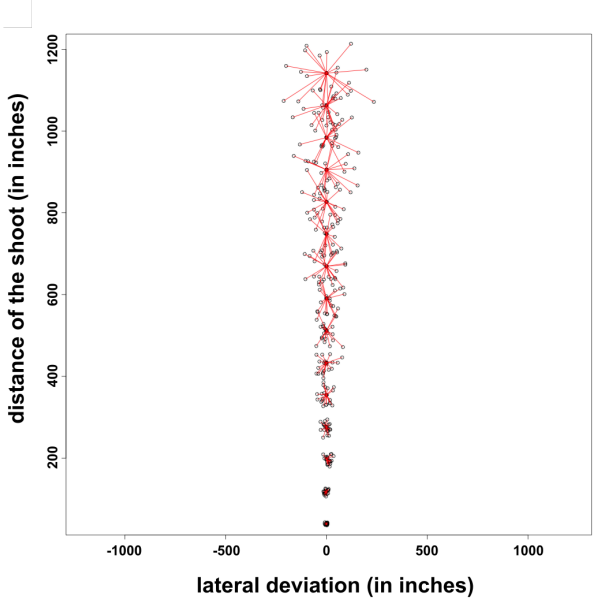


Figure 2.21 – Regeneration of plausible data for the short game

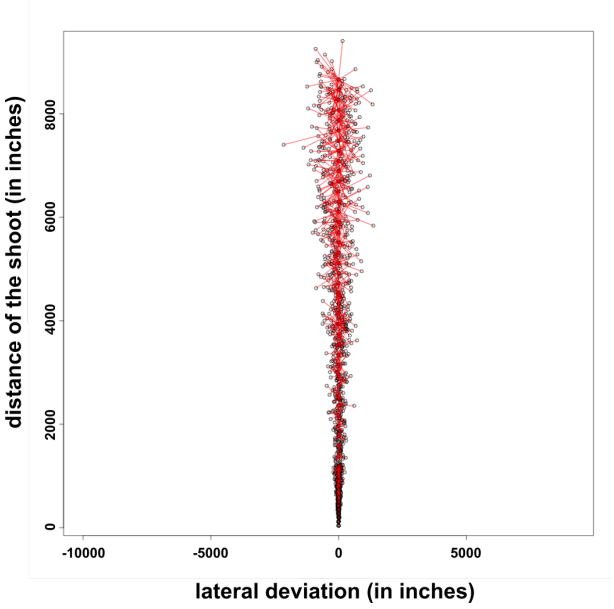


Figure 2.22 – Full generated profile of Phil Mickelson on the Fairway

For the short game, we regenerate $n_{shots} = 20$ shots from 1 to 30 meters every 2 meters. For the middle game, from 30 to 100 meters every 5 meters. For the long game, in order to compute the maximal distance to which the golfer can play, we compute a ratio between his maximum distance on the drive and the maximal distance in the long game. Indeed, the maximal distance of each player for the long game depends on his intrinsic power. These assumption are consistent with the ability for the players to target different points described in section 2.2.2. The number of shots regenerated has been chosen large enough for the model not to be sensitive to the data without increasing too much the computational running time.

Regenerate data for all the grounds

In the previous section we succeed in regenerating data in order to get the theoretical statistics on the fairway. We are now about to regenerate shots for all the types of ground.

We will now assume that for a player, the ratio between the deviation on the fairway and on the bunkers/rough is constant. This is a first natural assumption to consider.

All we have to do now is to compute this constant factor between the fairway and the bunkers on the one hand, and between the fairway and the rough on the other hand, and to apply this factor to regenerate data with modified parameters for the normal laws. In order to be more realistic, we discriminated upon two cases: the case where the ball ends up before the target and the case where it ends up after the target. Indeed, on the rough and in the bunkers, it is more likely that the ball ends up before the target, because of the difficulty to have a clean shot on these grounds. So we compute the empirical probability for the ball to end up before the target (easy to do, as we have the coordinates of the pin, and the ones of arrival points), and we generate the right proportion of shots with the right constant coefficient. The coefficients of deviation are computed as the mean of the deviations of shots. We obtain the following generated profiles (figures 2.23 2.24 2.25 2.26 2.27 2.28 2.29 2.30):

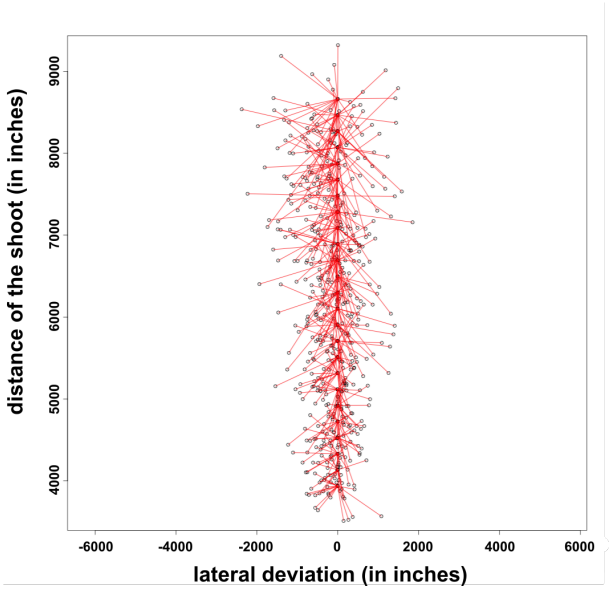


Figure 2.23 – Regeneration of plausible data for Phil Mickelson for the long game on the rough

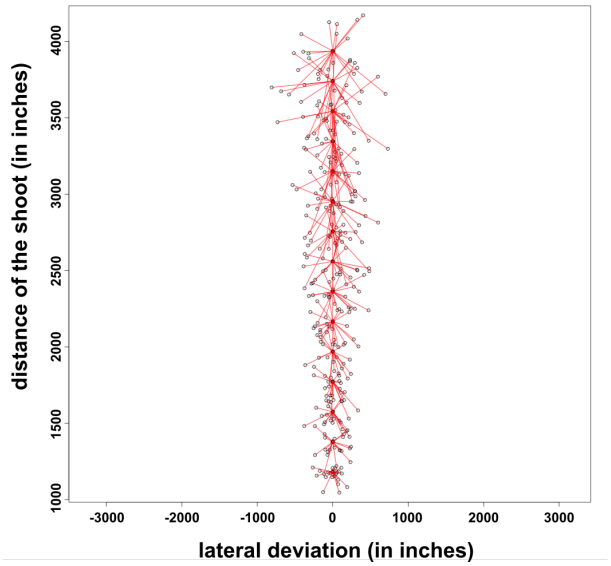


Figure 2.24 – Regeneration of plausible data for Phil Mickelson for the middle game on the rough

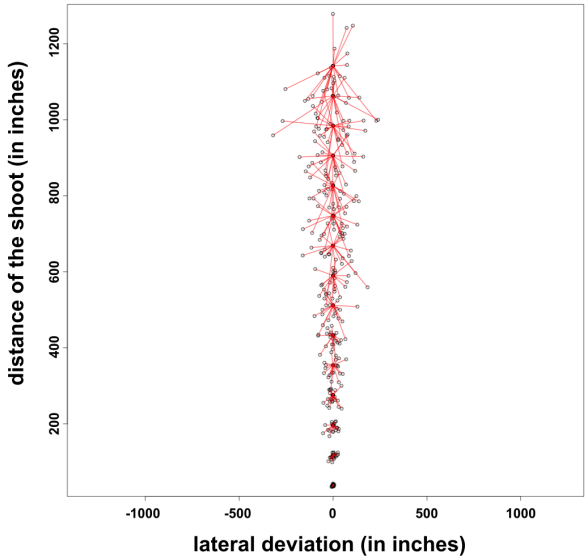


Figure 2.25 – Regeneration of plausible data for Phil Mickelson for the short game on the rough

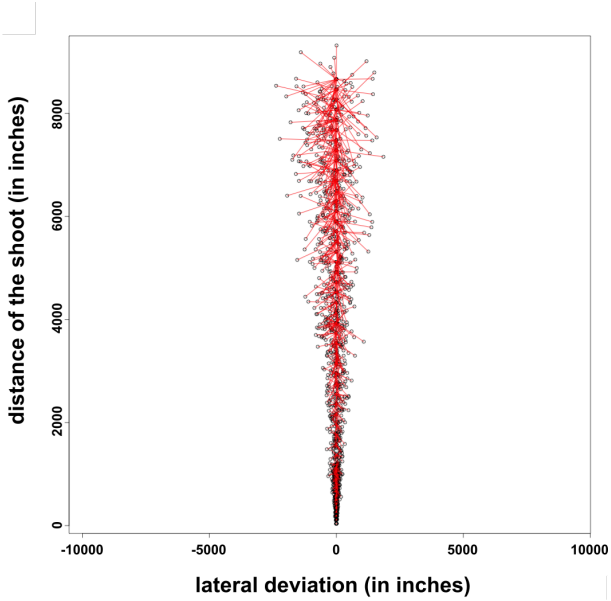


Figure 2.26 – Full generated profile of Phil Mickelson on the Rough

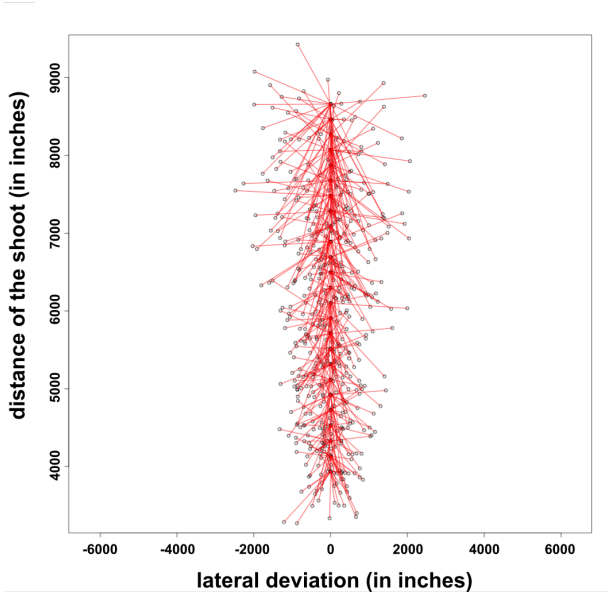


Figure 2.27 – Regeneration of plausible data for Phil Mickelson for the long game on bunkers

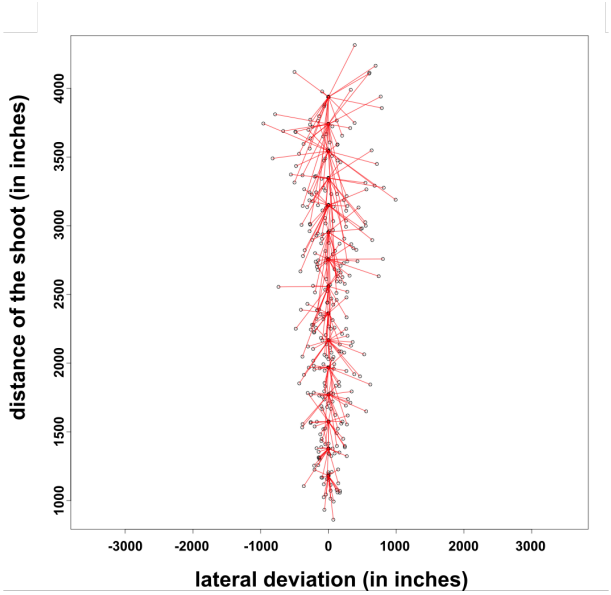


Figure 2.28 – Regeneration of plausible data for Phil Mickelson for the middle game on bunkers

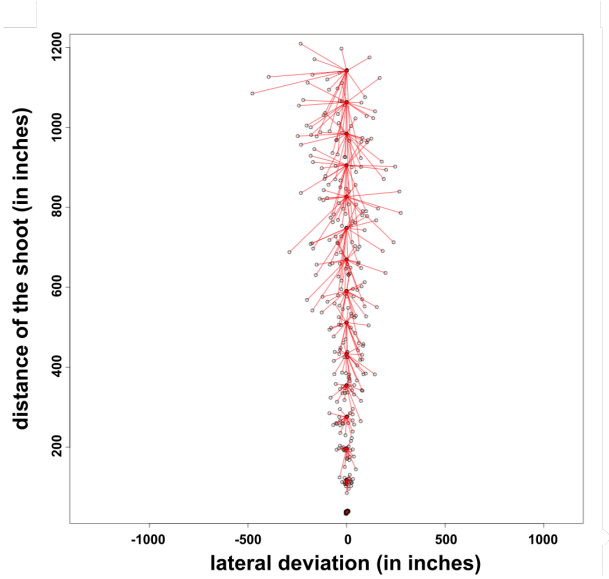


Figure 2.29 – Regeneration of plausible data for Phil Mickelson for the short game on bunkers

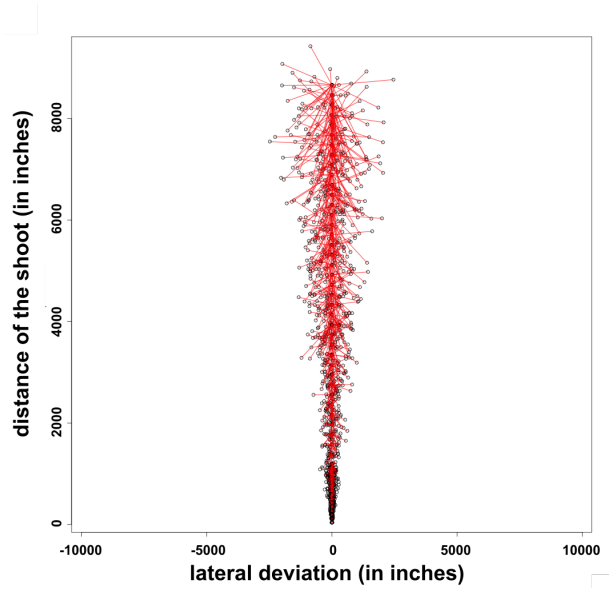


Figure 2.30 – Full generated profile of Phil Mickelson on the Bunkers

2.3.3 The Driving

For the driving, it is not reasonable to assume that the player is targeting the hole: in many cases the golfer cannot reach the green in one shot because of the too big distance between the teeing ground and the flag. However, we have access to the location of the tee for each hole and to the coordinates of the arrival points for shots on the tee. We can plot the scatter plot of the corresponding arrival points, considering that all the shots have been played from the same point. Let us analyze this scatter plot of the shots on the tee (figure 2.31).

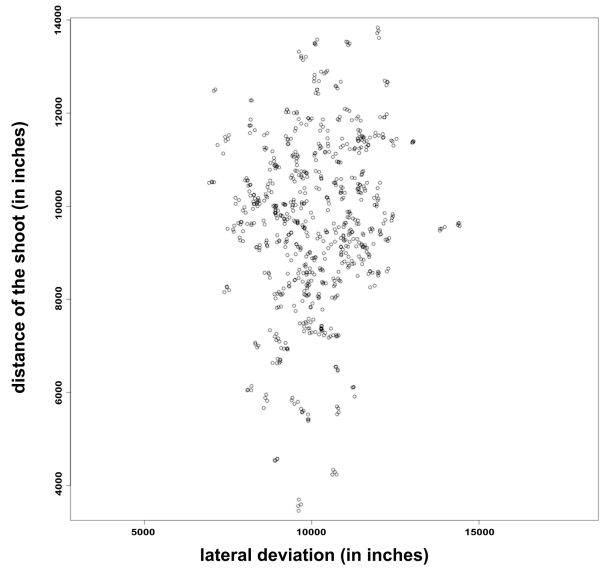


Figure 2.31 – Driving of Phil Mickelson in 2018

If either the abscissa and the ordinates of this scatter plot follow normal law, this would mean that the same point has been targeted (relatively to the starting point), so we could apply the same techniques than before and generate new plausible data for all distances. We test this assumption with a Shapiro-Wilk test:

	Abscissa	Ordinates
Driving	0.0007353	$5.345.10^{-8}$

Even with a small threshold, it is not reasonable to assume that abscissa and ordinates of the driving follow normal laws. This would mean that for each shot, the relative targets are not the same. This result was predictable, as the layout of the holes in the database are different.

Assumptions on the target for driving

In stroke play mode, the golfers are playing the entire course four times (four *rounds*). On every hole, we do have four realizations of their first shot on the tee. We assume that the player are following the same strategy for the drive during the whole competition: the golfers are targeting the same point in each round.

We begin by eliminating the shots on the holes that the player did not play four times (the golfer can concede). The idea here is to define the target of the four realizations as the centroid of all the shots that was not 'missed'. We consider that among the four distances of shots, the greater is the one wanted by the golfer. We assume that if a shot's arrival point is 'too far' from the longest shot (we defined a threshold of 50 meters) then there was likely an obstacle, so this shot was not intended by the golfer. We assume that the direction of the shots are never missed. Indeed, the distance of a shot can be quite easily shorten because of an obstacle when the direction is very rarely missed, and when it is missed, the distance of the shot is missed too (so the shot will not be taken into account).

To sum up, we define the target of the four realizations of the four rounds of a competition as the point where:

- direction is the mean of the four directions of the shots
- the distance is the mean of the shots that end up less than 50 meters from the longest shot

Just as before, we can plot the profile of Phil Mickelson with 2018's data for the driving (see figure 2.32), just like we did before (see figure 2.6):

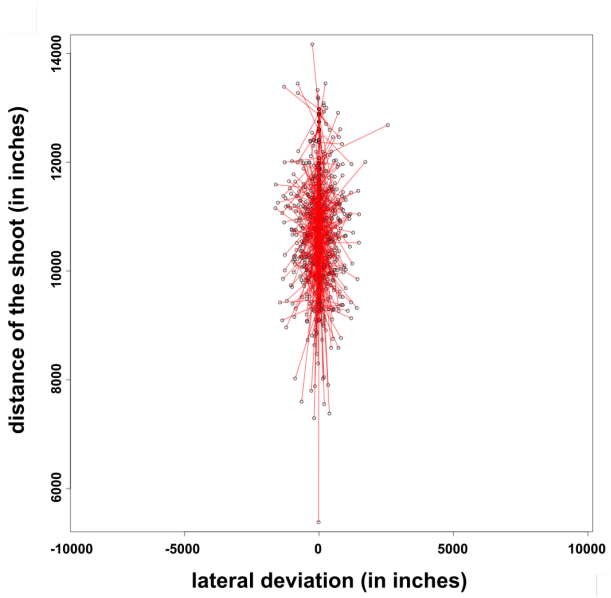


Figure 2.32 – Driving of Phil Mickelson in 2018 after statistical inference

Now that we have a potential target for each shot, we can create a star by applying the same techniques as before (see figure 2.15. We obtain figure 2.33:

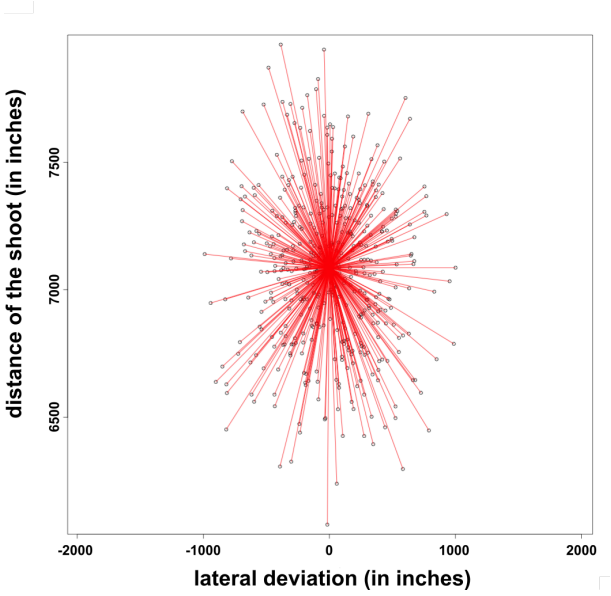


Figure 2.33 – Star for the Drive (Phil Mickelson, 2018)

We test with a Shapiro-Wilk test if the abscissa and the ordinates follow normal law:

	Abscissa	Ordinates
Driving	0.8769	0.7563

According to the results of the tests, we can reasonably assume that the two coordinates follow normal distributions, so we can re-generate data for all the distances. We generate data every 5 meters, accordingly to section 2.2.2: we consider that a professional golfer is able to discriminate two targets if they are at least 5 meters away one from the other. The lower bound is the upper bound of the long game on the fairway (which was defined with a ratio between the longest shot of the drive and the longest shot on the fairway) so that the is a continuity between the different sections of distances. The upper bound is define as the distance for which 95% of the shots on the drive are above. This maximum distance of shot for the driving has been chosen so that the players are discriminated according to the maximum power of their shots but also to erase the statistical aberrations (typically when the layout of the hole makes the ball roll to a distance that could not been reached on another hole).

We obtain this generated profile of Phil Mickelson for the driving:

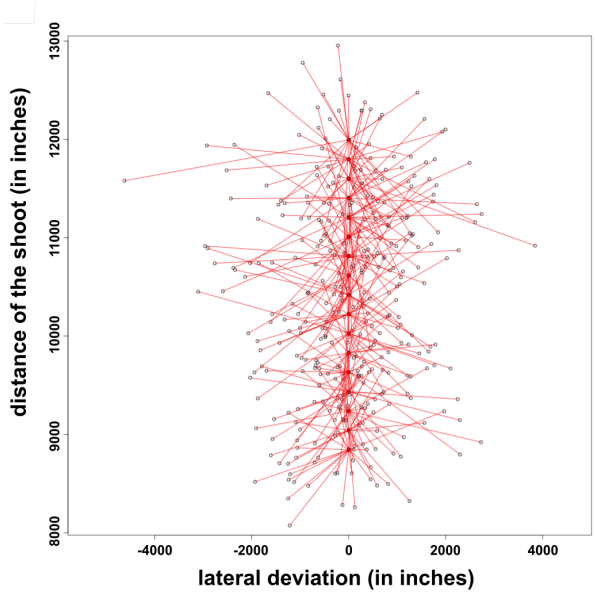


Figure 2.34 – Regenerated data for Phil Mickelson on the tee in 2018)

Now that we managed in creating theoretical distributions for a golfer, we can simulate these on a specific hole as described in section 2.2.4 to create the transition matrix. The states, the actions, the costs of these actions and the transition matrix have been defined, so the SSP model is complete.

2.4 Results and Validation

Let us consider one specific player and one specific hole. We have seen that we can create an instance of SSP for the golfer's problem on this hole. We can solve the corresponding instance with standard algorithms: in practice we solved the instance with Value Iteration, but Policy Iteration and our new Dijkstra like algorithm (see chapter 1 for more details) have been implemented too. The optimal solution of this instance is a deterministic and stationary policy. It provides the optimal actions that the golfer has to perform in order to minimize the expected number of shots to put the ball in the pin. To represent a policy, we draw in the cells composing the state, the direction of the shot, and the distance of the shot in the bottom left corner. We also write the expected number of shots needed to reach the pin in the top left corner (see figures 2.35 2.36 2.37 2.38).

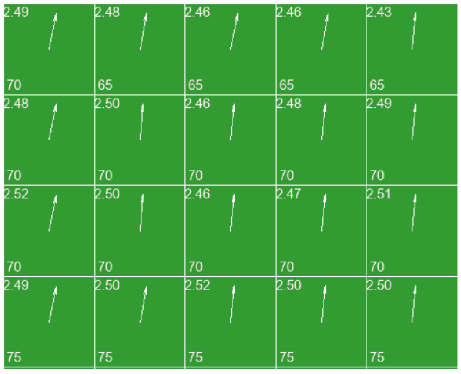


Figure 2.35 – Part of an optimal policy of Phil Mickelson one the 15^{th} hole of Augusta National Golf Club. Each square represent a state, the optimal action on the corresponding state is represented as an arrow in the center with a distance in the bottom left corner

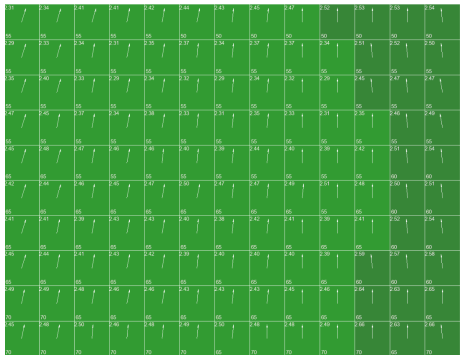


Figure 2.36 – Part of an optimal policy of Phil Mickelson one the 15^{th} hole of Augusta National Golf Club with a larger scale

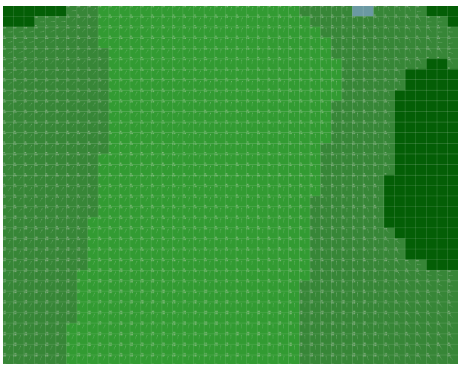


Figure 2.37 – Part of an optimal policy of Phil Mickelson one the 15^{th} hole of Augusta National Golf Club

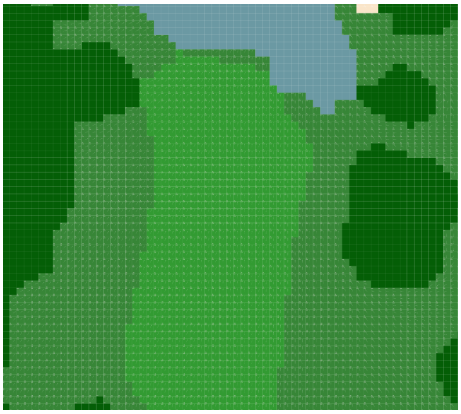


Figure 2.38 – Part of an optimal policy of Phil Mickelson one the 15^{th} hole of Augusta National Golf Club

Our goal is to predict the scores of professional golfers on the PGA Tour. We will create a *digital clone* of the player. We assume that the professional golfers play there optimal strategy, taking into account his personal skill and the hole's layout. Then the optimal strategy we find matches, with this assumption, the behavior of the player we consider, and we can artificially make the clone play as many times as we want to with Monte-Carlo simulations. A simulation consists on following the optimal policy from the tee to the pin:

- we initialize the golfer's score to 0;
- we start from the state whose location corresponds to the tee's location;
- we perform the optimal action on this state;
- from the transition matrix, we know the states in which we can end up, with a corresponding probability distribution;
- we add 1 on the score, or 2 if a penalty occurs;
- we pick randomly one realization (*i.e.* one state) according to the transition matrix;
- we iterate until a state on the green is reached;
- from figure 2.4 we know the distribution of number of shots the golfer makes to put the ball in the pin from the corresponding state on the green. We pick randomly one realization, *i.e.* one number of putt, and we add it to the score.

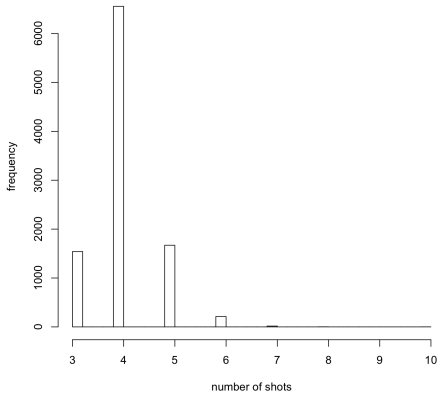


Figure 2.39 – Histogram of expected number of shots for 10000 simulation of Phil Mickelson on the 1^{st} hole of Augusta National Golf Club)

So we are able to have an estimation of the scores of a player on a specific hole. We assume that all the holes of a course are independent, meaning that the player plays identically on the first and the eighteenth hole: his strategy is not influenced by his current score. As discussed in the introduction of this chapter, this assumption is questionable considering the risks that the golfers could take, but in practice this assumption is reasonable for professional golfers. Depending on the mode of the competition, we can either compute the expected score of the competition (Stroke-play) or the expected probability of winning (Matcha-play). We take in the following sections two examples of competition: the 2017' Augusta competition for Stroke-play and the 42^{th} Ryder Cup in 2018 for Match-play.

2.4.1 Stroke-Play

During a stroke-play competition, each golfer plays the whole course four times. The score of the player on a round is the sum of the scores over the 18 holes.

Simulation

In order to evaluate the distribution of the score of a golfer in a stroke-play competition, we simulate his game thanks to his numerical clone on each hole and we add up his score over the 18 holes. We repeat this a large number of time (10000 in our numerical experiments). This provides us an approximation of the histogram of scores of the player. Let us call this histogram H_S. (see figure 2.41 2.40 for an example with Phil Mickelson with his 2017 statistics on Augusta National Golf Club).

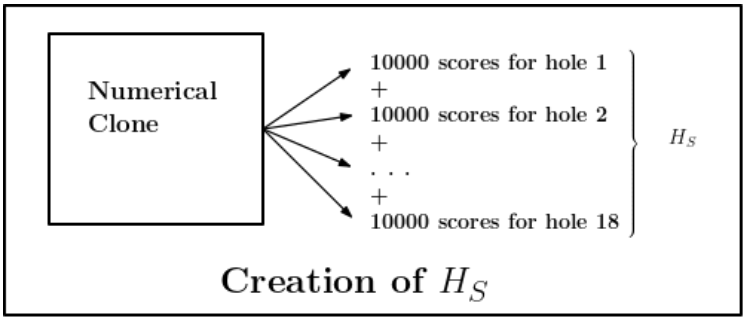


Figure 2.40 – Outline of the creation of H_S

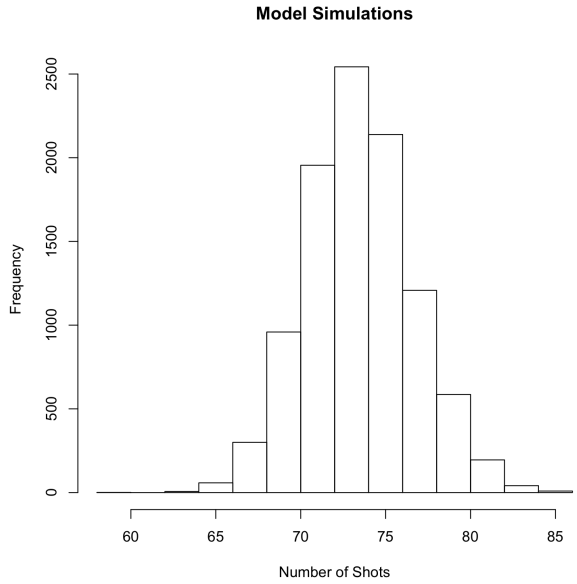


Figure 2.41 – Histogram H_S of the scores of the numerical clone of Phil Mickelson on Augusta National Golf Club

Now that we have score distribution of a player on an entire course, we would like to know if this score distribution is accurate. We know the score that the golfer has made during the same competition in the same year:

	Round 1	Round 2	Round 3	Round 4
Hole 1	4	5	3	4
Hole 2	3	4	4	3
Hole 3	4	3	6	6
Hole 4	2	3	3	4
Hole 5	5	5	4	4
Hole 6	4	3	4	3
Hole 7	4	3	4	5
Hole 8	5	5	6	5
Hole 9	4	4	5	4
Hole 10	5	3	4	4
Hole 11	5	5	4	5
Hole 12	3	3	3	3
Hole 13	4	4	5	4
Hole 14	4	5	4	4
Hole 15	5	5	4	4
Hole 16	2	4	3	2
Hole 17	4	5	4	4
Hole 18	4	4	4	4

However, it seems difficult to compare a distribution of scores with only 4 realizations (one per round). We would like to create new artificial 'realization' from the one we have *i.e.* use bootstrapping techniques. As we consider that all the rounds are independent, we can create another (artificial) realization by considering the score on the first round, and switch the score of the first hole with the one of the first hole on 2nd round. We can generalize this by considering that on each hole there is a discrete empirical distribution of scores which corresponds to the four realizations of his four rounds. Then we can regenerate 'realizations' for an entire course by picking at random one of the four realizations for each hole. Thus, instead of having only one realization for the four rounds, we have 4^{18} artificial realizations. As we assume that the strategy of the player is not modified neither by the number of the hole nor by the number of the round, we consider that the artificial realizations we get are plausible. Let us create 10000 artificial realizations and create the corresponding histogram of scores that we call H_E (see figure 2.42 for a outline of the creation of H_E and see 2.43 for an example with Phil Mickelson)

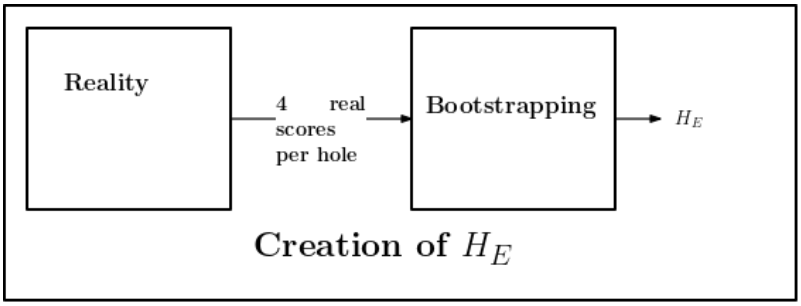


Figure 2.42 – Outline of the creation of H_E

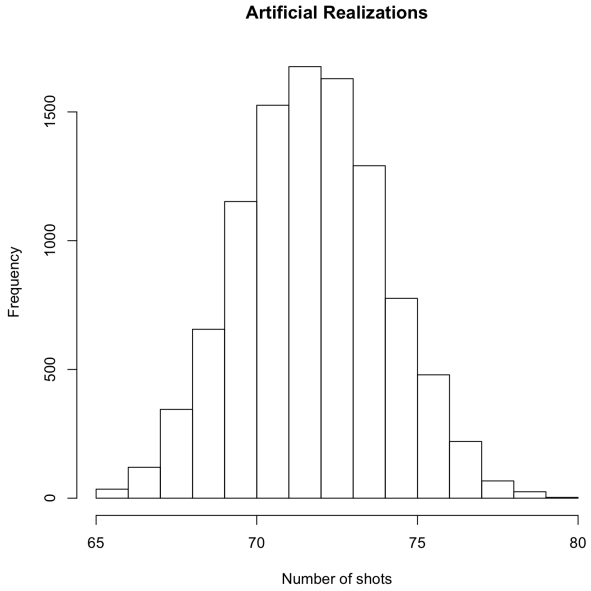


Figure 2.43 – Histogram H_E of the artificial realizations of Phil Mickelson on Augusta National Golf Club

Now the main question is to know how to compare the histogram of the artificial realizations with the histogram of simulations from our model. The simple study of the mean and the standard deviation of them is not enough even if it can be interesting to look at them:

	Mean	Standard Deviation
Simulation	72.2162	2.2848
Empirical	72.4974	2.5465

The means and the standard deviations are quite 'similar'. The next sections describe the method we created to compare the simulations from our model and what happened in reality.

2.4.2 Match-Play

In Match Play, two golfers play against each others. Each hole is played once by the two players. The winner of a hole (the player who put the ball in the cup in the least number of shots) earns one point. If there is a tie, each golfer earns half a point. At the end of the 18 holes, the winner is the player with the most number of points. Thus, for each hole we can get an empirical probability of winning a point for each player by comparing the scores of each player on every hole and see who gets the most points at the end of the 18 holes (see figure 2.44 2.50).

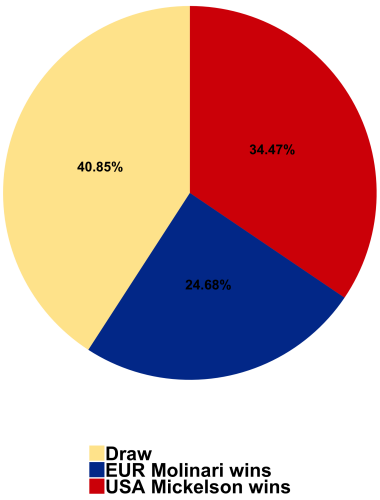


Figure 2.44 – Probability of victory of Phil Mickelson and Francesco Molinari on the 3^{rd} hole of Golf National

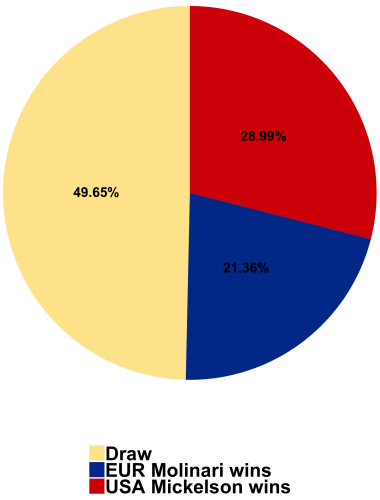


Figure 2.45 – Probability of victory of Phil Mickelson and Francesco Molinari on the 15^{th} hole of Golf National

As we have this information on each hole, we can also, by simulating the play of each player on the entire course, compute the probability of winning more than 9 points over the 18 holes, *i.e.* the probability of victory of a golfer on the course (see figure 2.46):

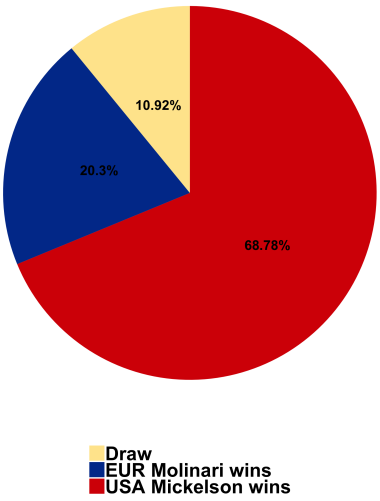


Figure 2.46 – Probability of victory of Phil Mickelson and Francesco Molinari on Golf National

This match between Phil Mickelson and Francesco Molinari took place during the 42^{th} Ryder Cup near Paris. Unlike Augusta Golf National which is a Stroke-play competition, the Ryder cup is a Match-play competition (this is why we take this competition as reference for Match-Play). Here are the results:

	Phil Mickelson	Francesco Molinari
Hole 1	5	4
Hole 2	3	3
Hole 3	5	4
Hole 4	5	5
Hole 5	5	4
Hole 6	3	4
Hole 7	5	4
Hole 8	3	3
Hole 9	Win	Conceded
Hole 10	4	4
Hole 11	2	3
Hole 12	5	4
Hole 13	4	4
Hole 14	5	4
Hole 15	4	4
Hole 16	Conceded	Win
Hole 17	Not played	Not played
Hole 18	Not played	Not played

When a player is sure to win the match before the end of the 18 holes, the players do not play the remaining holes (it was the case for the two last holes here, where Francesco Molinari was sure to win).

Unlike for the Stroke Play, we have only one realization (and not four). The validation of the accuracy of our winning probabilities is a big issue. We will see in the next section how we manage to give a way to evaluate our model.

2.4.3 Validation

In the previous sections, we managed in giving results for the two main modes in golf: the Stroke Play and the Match Play. For Stroke Play, we give a discrete score distribution for the players, and for Match we give probability of winning for both players. Now the main issue is to evaluate the accuracy of our forecasts. The main difficulty is the few number of realizations (4 per hole for Stroke Play, and only 1 for the Match Play).

Validation for Stroke-Play

For Stroke Play competitions, we have 4 realizations per hole (4 rounds on the 18 holes). One first idea of our validation is to use bootstrapping methods in order to regenerate 'artificial realizations' based on the only 4 we get (figure 2.43). Let us call this histogram H_E. We can compare H_E with the simulations from our model (figure 2.41), which we call H_S. Even if looking at the mean and the standard deviation of H_S and H_E is a first approach, it is not enough to conclude (what is a reasonable difference between the means ? and the standard deviations ?).

We would like to define a distance between two histograms. A well known distance is the Kolmogorov distance. Let F_1 and F_2 be two distribution functions, and X the space of there distribution function. The Kolmogorov distance between F_1 and F_2 is defined as:

$$\sup_{x \in X} |F_1(x) - F_2(x)|$$

We want to know if the realizations we have could have been easily simulated with our model. The idea is to take 4 simulations on each hole from our model and consider that the corresponding simulations are the scores that the player could have done during 4 artificial rounds. We can bootstrap others realizations from them just like we did for H_E. We get an histogram of realizations based on simulations from our model. Let us repeat this process n times. We get n histograms $(H_{E_1}, .., H_{E_n})$.

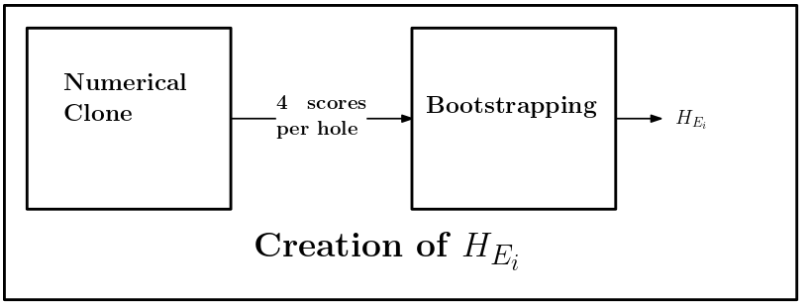


Figure 2.47 – Outline of the creation of H_{E_i}

Let F_S, F_E, and $(F_{E_1}, .., F_{E_n})$ be the empirical distribution functions of H_S, H_E, and $(H_{E_1}, .., H_{E_n})$. We can compute the Kolmogorov distances $(d_{K_1}, ..d_{K_n})$ between F_S and the $(F_{E_1}, .., F_{E_n})$. Let H_K be the histogram of the (d_{K_i}) for $i = 1..n$ (see figure 2.48). Let d_E be the Kolmogorov distance between F_E and F_S. In the following we will slightly abuse notation and talk about Kolmogorov distance between two histogram: it is in fact Kolmogorov distance between the corresponding distribution functions. For Phil Mickelson in 2017, here are the histogram H_K and d_E (figure 2.49).

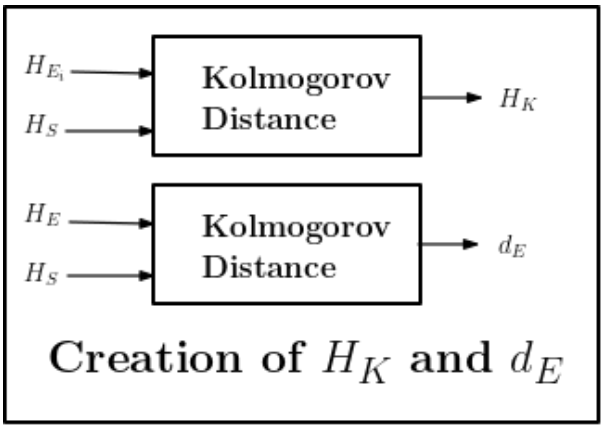


Figure 2.48 – Outline of the creation of H_K and d_E

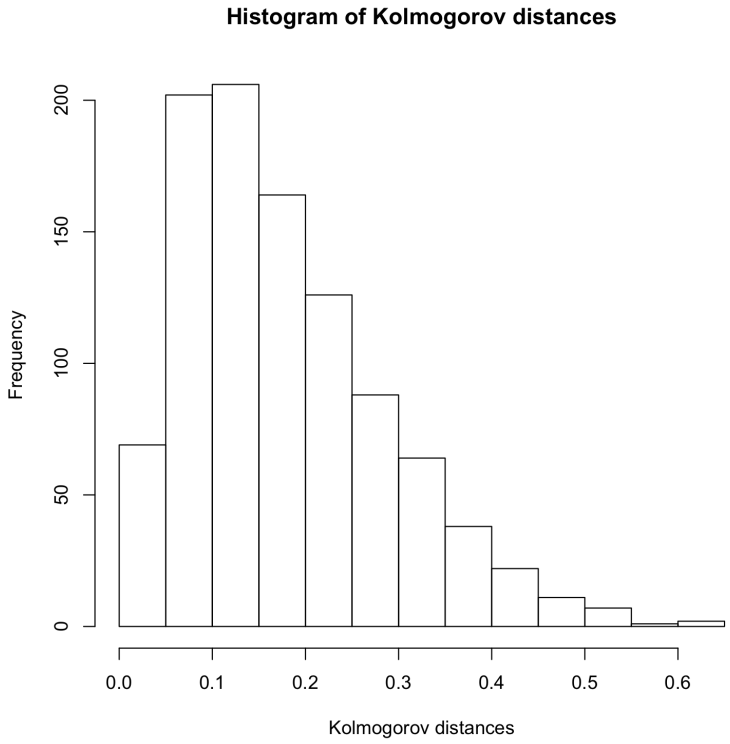


Figure 2.49 – Histograma H_K of Kolmogorov distances for Phil Mickelson in 2017

d_E	0.2727

We assume that if d_E is one likely realization of H_K then the realization H_E can be easily simulated by picking at random 4 realizations from our model. Looking at the quantiles of H_K, we have:

Quantiles	77%	78%
Value	0.2686	0.2746

From the quantiles, and under our assumptions, we can conclude it is likely for d_E to be a realization of H_K. Thus, the realization that Phil Mickelson score in 2017 in Augusta National Golf Club can likely be simulated with our model. Indeed, the Kolmogorov distance between H_E and H_S is one plausible realization of H_K. Here are the results for other players for the same year (2017) on Augusta competition.

Name	d_E	Quantiles
Rory McIlroy	0.2423	$[73\%, 74\%]$
Francesco Molinari	0.1248	$[37\%, 38\%]$
Justin Thomas	0.1701	$[52\%, 53\%]$

Note that the reader can reproduce the results thanks to our program in C++ and the R scripts given. For these players, we can also conclude that what happened in reality can plausibly be simulated with our model.

Validation for Match-Play

For the validation of Match Play, we adopt the same kind of method. Contrary to Stroke Play where each hole is played four times, in Match Play each hole is played only once by the two players involved. Let us consider the match Phil Mickelson against Francesco Molinari in the Ryder Cup 2018. We would like to know whether or not what happened could have been 'easily' simulated with our model.

We begin with simulating each digital clone of the two players involved on the 18 holes of Golf National course. Each simulation gives a winner (or a draw). By repeating this operation, we obtain a histogram H_S which relates of the probability of victory for both players on the entire golf course as seen before (figure 2.46).

We want to compare this histogram with the reality (what happened in the Ryder Cup). The problem is that we have only one realization for it. The idea here is the same as before: we have to use bootstrap techniques in order to create artificially new 'realizations'. We can see that during the Ryder Cup 2018 between Phil Mickelson and Francesco Molinari, only the first 16 holes have been played. We can create new realizations by picking at random 18 holes among the 16 played holes during the Ryder Cup. By comparing the scores of the two players on these 18 holes, we have a winner, or a draw. By repeating this operation 10000 times, we obtain a histogram H_E which related of the probability of winning for both player, and a probability of draw (figure 2.50).

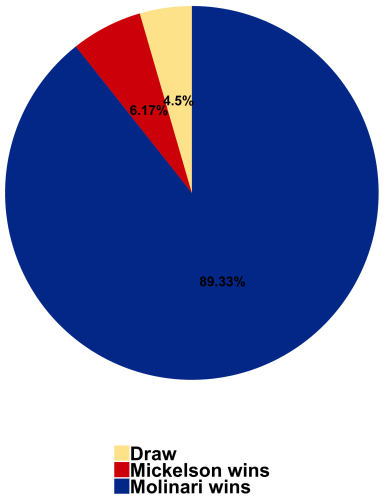


Figure 2.50 – Bootstrapping for Mickelson-Molinari

Obviously, since Molinari has won 7 holes over 16 and Mickelson has won only 3 holes over 16, it is predictable that picking at random 18 holes among these 16 holes makes Molinari wins in most of the cases. Because of the particularity of the unique realization we have, we cannot compare H_E and H_S (figure 2.46 and 2.50) directly.

However, we can pick at random 18 holes among the first 16 played and simulate with our model clones of both players. By doing this n times, we get n histograms $(H_{E_i})_{i\in\{1,..,n\}}$. These histograms represent possible realization from our model if we consider only the 16 first holes. Just like for the Stroke Play, we can compute the Kolmogorov distances d_i between H_S and the (H_{E_i}) for all $i \in \{1, .., n\}$. For $n = 1000$ we get the following histogram, that we call H_K of Kolmogorov distances (figure 2.51):

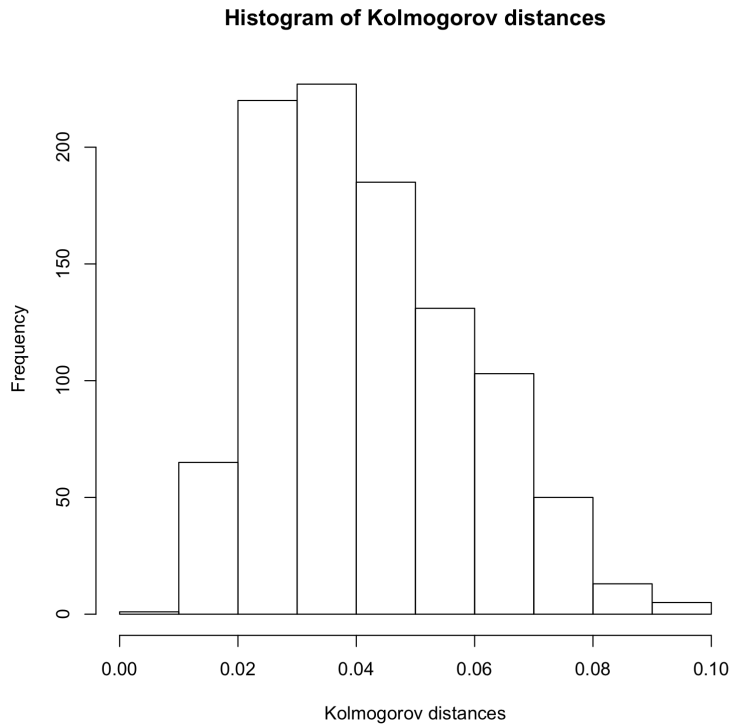


Figure 2.51 – histogram H_K of Kolmogorov distances between H_S and the H_{E_i}

By computing the Kolmogorov distance d_E between H_E and H_S we get $d_E = 0.4229$. So it is unlikely we could have predict the realization of the Ryder Cup with our model, under our assumption. Indeed, d_E cannot be a realization of H_K.

However, we have seen that what happened between Mickelson and Molinari during this Ryder Cup is very particular. Molinari won most of the holes and the two last holes has not been played. Let us take another example where all the holes have been played: Justin Thomas versus Rory McIlroy. By applying the exact same method, we obtain the following histogram of Kolmogorov distance (figure 2.52), and d_E:

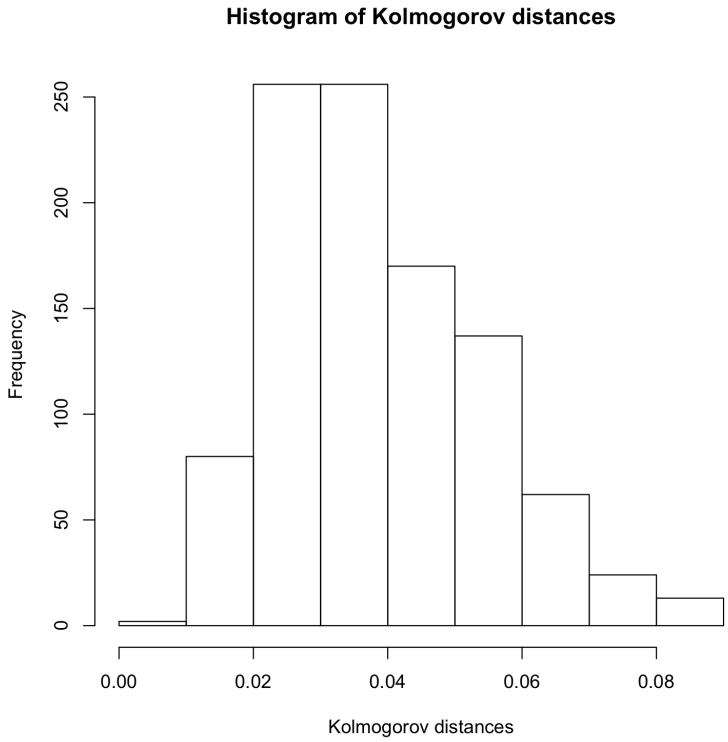


Figure 2.52 – histogram H_K of Kolmogorov distances between H_S and the H_i

d_E	0.04211

Looking at the quantiles of the previous histogram, we get:

Quantiles	55%	56%
Value	0.04145	0.0.4224

In this case, we conclude that our model can predict what happened, as d_E is a plausible value of H_K. Obviously the fact that our model cannot predict the very particular cases is a big limit of our model.

Computational techniques and remarks

Our computational experiments were developed in C++ and in R. Data analysis and treatment is done with R, while the optimization and the construction of the models are done in C++. We give full access to our code so that the readers can replicate the results.

The 'statistics' of a player are built using R from the Shotlink database. All the assumptions we made can be changed as parameters of the scripts. Once the script is

ran, a file of general name "FirstName_LastName.stats" is created. This file is needed for the construction of the model.

The construction of the model, and the optimization has been written in C++. This program can build a SSP model from a ".stats" file (which can be created with our R scripts), and a ".data" file. This latter file describes the topology of the hole we consider. A ".data" file can be created thanks to our program from a picture of a hole. Once the model is created, the program can find an optimal solution thanks to Value Iteration, Policy Iteration or our new Dijkstra-like algorithm (see chapter 1). Our program can also solve generic SSP instance, using standard input files describing the states, actions, action costs, and transition matrix of the instance.

Even if Value Iteration, Policy Iteration and our Dijkstra-like algorithm can be used to solve an instance thanks to our program, all the results in this chapter were found with Value Iteration. Our program has been optimized in order to have good computational performances with Value Iteration: the internal memory management (cache memory) and the construction of our model's data structure were designed to speed-up parallel computing, which lower drastically the computational running time of Value Iteration (from several minutes to a few seconds). We mainly used arrays in order to store our data. As Value Iteration is iterating over all states, the way we store the state in the memory is very important and has drastic impact regarding computational running time (the change of our data divided by 20 the computational running time of Value Iteration).

Regarding the computational time, most of the time is used to load the model, and to create the SSP instance from the data (the Shotlink database and a picture of the hole). The solution time has been optimized as explained above, and take no more than a few second, while loading the model is done in few minutes. Obviously, the value of the parameters has an big impact on the computational running time (discretization of the hole, discretization of directions...). For our choice of parameters, we can create 10000 simulation of a player on the 18 holes of a course in about 6 hours (with a coreI7 4 cores 2.2GHz processor and 16Go 1600MHz DDR3 memory). A tiny part of which is dedicated to the optimization (about 1%).

Contrariwise, our model construction is not adapted to the execution of our Dijkstra algorithm, which explained (in part) the running time of this algorithm with our program.

2.5 Conclusion and Perspectives

We created a model for the golfer's problem using Markov Decision Processes, and in particular the Stochastic Shortest Path problem. The states, the actions and the cost of these actions were quite easy to define, while the transition probabilities were harder to model. We used the Shotlink database, which gather thousands of shots of professional golfers during international competitions in order to create theoretical distribution of the players. The main difficulty was to infer the intention of the players. Under certain assumptions, we succeeded in defining targets for the golfers for each

shot, and we checked the relevance of this assumptions with statistical tests. Once the transition matrix defined, we were able to apply standard algorithms of the SSP to solve the problem exactly and get the optimal strategy of the player, taking account his personal skills, and the simplified physics of the hole he plays on. As we assume that the professional golfers play their optimal strategy, we created 'digital clones' of players and simulate the play of the golfers thousands times on several courses. Depending on the mode (stroke-play or match-play), we were able to have a prediction of the scores, or a winning probability.

The results and their validation is a big issue of this chapter. We had to find ad hoc methods in order to compare our results and what happened in reality. The few number of realizations prevent us to use standard statistical methods (chi 2 test, p-values). We managed in creating artificially new realizations thanks to bootstrap techniques.

As we build our model, we had a lot of assumptions. Obviously these assumptions are questionable.

In order to build the transition matrix, we simulated the behavior of the physics of the ball. The assumptions we made are very simplistic: no rolling, infinite height for the trees, simplified impacts between the ball and the obstacles... The simulator could be improved to be more realistic. The improvement of this simulator would not have major consequences on the model: this is an independent piece which would change the transition matrix but would certainly impact the computational time for model creation.

The assumption we made regarding the intention of the players are also questionable. We consider that the golfers always target the pin if they are not on the tee (and if they are at reach). Although it is reasonable most of the cases, when the par is high (par 5 for example), or when the shape of the hole is elbowed (a so called dog-leg in golf), the golfers would rather target the middle of the fairway to insure to reach the green faster.

When we create the theoretical statistics of the golfers, we had a lot of parameters to tune: thresholds, percentage of shots we cut, statistical tests acceptance... These parameters were chosen based on knowledge of golfers. We also test our model on different year, and these parameters could have to be modified in order to pass the statistical tests. This lack of robustness is also a quite big limit of our model.

The tests we chose are also assumptions: we assume that the distribution of shots is realistic if it follows a normal law, which is questionable. The linearity between the deviation and the distance, the constant factor between the deviations on the different grounds are too.

Chapter 3

On Stochastic Games and MAX-PROB

3.1 Introduction

As explained in chapter 2, we came to the SSP problem by studying strategy optimization in Golf: a golfer has to put a ball in a hole in a minimum number of shots taking into account the topology of the field, the weather conditions and their personal skills among other things. The 'golfer's problem' can be modeled as a SSP under certain assumptions (see chapter 2 for more details). Consequently, solving SSP can provide golfers a 'best strategy' to reach the hole in an minimum expected number of shots.

Modeling the golfer's problem as a SSP only considers 'competing against the golf course' and does not include the other players into the equation. This is appropriate when modeling 'Stroke play' competitions where players try to score as low as possible on a large number of holes (72) to beat *the field* (that is, the rest of the players). But there are other competitions where two players compete against each others in another form of game called 'Match play' (like for the Ryder Cup, for example). In this mode, both players play potentially every holes of the same golf course against each other. On each hole, the player with the smaller score earns 1 point ($\frac{1}{2}$ if there is a draw). The winner is the player with the most points at the end of the 18 holes (it might end earlier if the difference of points is more than the number of remaining holes to play).

As Stroke Play competitions can be modeled as SSP, Match Play competitions can be modeled as Stochastic Shortest Path Games (SSPG), which are natural game extensions of SSP where two players control the states of the system.

A SSPG is basically a SSP where the states are partitioned into two sets that are controlled respectively by a MIN and a MAX player with antagonist objectives. The goal of the MIN player is to find a *strategy* (a choice of action for each state controlled by this player and each period of time) to reach the target state with minimum expected cost while the MAX player wants to find a strategy to maximize the expected cost. in

general, MIN might be tempted to keep looping in the system if negative cost transition cost cycle exist, and vice versa MAX might be tempted to keep looping in the system if positive cost cycle exists. So technical conditions are required to define the problem formally. in this chapter we focus mainly on the special case of SSPG with termination inevitable (where all pairs of strategies lead to the target state with probability 1).

SSPG are a special case of (zero-sum) stochastic games introduced originally by Shapley for discounted problems [96] but whose definition has been extended later to undiscounted problems (for a comprehensive treatment of stochastic games, see for instance [12] and [43]). **SSPG with termination inevitable** are special cases of BWR-games (two-person zero-sum stochastic mean payoff games with perfect information) with total effective payoff [22]. In particular, because stochastic shortest path games with inevitable termination have mean payoff (average cost per time period) of value 0 from any starting state as the game ends with probability one, it follows from Theorem 27 in [22] and Von Neuman Minimax theorem for zero-sum games [78] that: (i) there exists (at least) a pair of *uniformly*[1] deterministic and stationary strategies for both players which forms a Nash Equilibrium (i.e. no player can benefit from deviating from his strategy) and (ii) that the corresponding strategy for MIN minimizes the maximum expected total cost over all possible strategy for MAX and the strategy for MAX maximizes the minimum expected total cost under all possible strategy for MIN. The stochastic shortest path game is the problem of finding such a pair of strategies. One well known iterative algorithm to solve exactly the SSPG is known as Strategy Iteration [51, 30, 31] which can be interpreted as a extended version of Policy Iteration for SSPG (we detail this later).

During a Match-Play competition in Golf, the first player teeing on a hole is the player who won a hole last (initially it is selected at random). Let us call him MIN, and MAX his opponent. This is then the player whose ball is the further from the flag who plays first. When a golfer shoots, he knows the position of his ball, the position of his opponent's ball and the current difference of scores between him and his opponent. This naturally defines a state space: for a specific hole, we consider the set $\mathcal{S}$ with element s of the form $s = (p_{MIN}, p_{MAX}, \delta)$ where p_{MIN} (resp. p_{MAX}) is the position of the ball of MIN (resp. of MAX) on the discretized hole (typically 2D-coordinates) and $\delta \in \mathbb{Z}$ is the current relative difference of scores between MIN and MAX. Let us call $\sigma_{MIN} \in \mathbb{Z}^+$ the current score of MIN and $\sigma_{MAX} \in \mathbb{Z}^+$ the current score of MAX, we have $\delta = \sigma_{MIN} - \sigma_{MAX}$. The initial state is $s_0 = (P_{tee}, P_{tee}, 0)$ where P_{tee} is the position of the tee. Even if the relative difference of scores cannot be bounded *a priori*, in practice beyond a certain point difference, usually, the late runner "gives" the hole to the opponent, so we can easily define a difference $D \in \mathbb{Z}^+$ and restrict to states with $\delta \in \{-D, -D+1, .., D-1, D\}$ (for professional golfers, D is rarely more than 2 or 3). A state $s = (p_{MIN}, p_{MAX}, \delta)$ is controlled by MIN if the distance from p_{MIN} to the flag is greater than the distance between p_{MAX} and the flag, and is controlled by MAX otherwise. The actions available in a state are the shots of the player who controls this state. Each time MIN is playing from a state $s = (p_{MIN}, p_{MAX}, \delta)$, his ball ends up

[1] i.e. the policy is the same for any starting state

in another position of the hole p'_{MIN} and the new state is $s' = (p'_{MIN}, p_{MAX}, \delta')$ with $\delta' = \delta + 1$ (or $\delta' = \delta + 2$ if a penalty occurs, see [59] for details of penalties in Golf, unless $\delta' > D$ and then it goes to the target node that we define later with a cost 1). Similarly, when MAX plays from s, the ball ends up in another position p'_{MAX}, and the new state is $s'' = (p_{MIN}, p'_{MAX}, \delta'')$ with $\delta'' = \delta - 1$ (or $\delta'' = \delta - 2$), unless $\delta'' < -D$ and then it goes to the sink node with a cost -1. These transitions induce no cost except when reaching the sink node. Let us call *flag states* the states whose general form is $(P_{flag}, P_{flag}, \delta^f)$ where P_{flag} is the position of the flag. The game stops when a flag state $(P_{flag}, P_{flag}, \delta^f)$ is reached: MIN wins if $\delta^f < 0$, MAX wins if $\delta^f > 0$ and there is a draw if $\delta^f = 0$. In order to match exactly a SSPG instance, we define a sink node (called 0) which can be reached only from flag states $(P_{flag}, P_{flag}, \delta^f)$ (and the states where $\delta < -D$ or $\delta > D$ as we have seen before) with a probability 1 and a cost 1 if $\delta^f > 0$, -1 if $\delta^f < 0$ and 0 if $\delta^f = 0$. The cost of a pair of policies $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ represents the points when MIN follows Π_{MIN} and MAX follows Π_{MAX}.

Despite the fact that Match Play competitions are well described by SSPG, the size of the state space is too large for implementing strategy iteration. Indeed, the state space size is $n^2 * (2D + 1)$ where n is the number of possible locations on the hole (depending on the discretization we choose), and D the bound on the difference of scores between the players. With regards to the computational performances in chapter 2 where the state space size was only n, it is hopeless to run similar algorithms like strategy iteration on such big instances: this is why we used other techniques in chapter 2 to model Match Play competitions which we believe provide a decent heuristic. In [55] and [56], Hoffmeister and Rambau used MDPs for strategy optimization in beach Volleyball. They also face too large instances that could not be solved exactly. They succeed in solving heuristically the problem by simulating strategies of MDP instead of solving it. Such an approach would certainly be of interest here too. However, it is interesting to study Stochastic Shortest Path Games for themselves. In this chapter though, we put the focus on the theory of SSPG and in particular on the question of existence of polynomial time algorithms.

Finding an optimal solution of SSPG, *i.e.* a Nash Equilibrium, is a problem in $NP \cap coNP$, and the question whether it is in P or not is open [30] (we refer to chapter 1 for definitions of P, NP and $coNP$). In particular, the question of the existence of a LP-formulation with a polynomial number of variables and constraints (a polyhedron whose extreme points are solutions for the SSPG), even in a extended space, is open. Even if we know that some *'naive'* formulations do not work for a special case of SSPG called the Simple Stochastic Games (that we define later formally)[31], the question has been neither answered positively nor negatively.

In this chapter we focus on this open question and we try to bring new insights and some directions for further investigations. Our goal is to analyze first which subproblems of SSPG are known to be polynomial, and in particular to identify polynomial size extended formulations for those or to prove that none exist. The idea would then be to extend the conclusion when possible to the general case.

There have been tremendous developments in the field of extended formulations

lately. In his survey [61], Kaibel gathers the combinatorial problems for which there exists compact extended formulation (meaning that the number of inequalities of the extended formulation is a polynomial function of the data of the original problem). Spanning trees, permutahedron, disjunctive programming are examples for which such extended formulations exist. However, there are some theoretical limitation regarding extended formulations. Rothvoss proves in [91] that extended formulations of some 0/1 polytopes need exponential number of inequalities [91]. Fiorini et al. in [44] prove that for the Traveling Salesman Problem (TSP), a LP-formulation need an exponential number of constraints. Even more striking is the fact that some polynomial problems like matching do not have a polynomial extended formulation [91].

In this chapter, we first define formally the SSPG with termination inevitable, then the special case of stopping Simple Stochastic Games (SSG). Later in the document we study another special case, the Robust Shortest Path Problem with termination inevitable (RSP), for which we present a ILP-formulation that can be extended to SSPG, which is the main contribution of this chapter. Finally, after considering instances which are both stopping SSG and RSP with termination inevitable for which we find a condition for the existence of an optimal solution that, we believe, could be extended to stopping SSG instances, we analyze the complexity of these different problems, and of MAXPROB. This last problem has already been defined in chapter 1 and arise naturally in this chapter too.

3.2 Stochastic Games

An instance of a SSPG is defined by a tuple $(\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ where $(\mathcal{S} := \mathcal{S}_{MIN} \cup \mathcal{S}_{MAX}, \mathcal{A} = \mathcal{A}_{MIN} \cup \mathcal{A}_{MAX}, J, P, c)$ is an instance of SSP and $\mathcal{S}_{MIN} \cap \mathcal{S}_{MAX} = \{0\}$. We again assume w.l.o.g. that each action is available in exactly one state. A Min player controls the actions in the states $\mathcal{S}_{MIN}$, while a Max player controls the actions in the states $\mathcal{S}_{MAX}$. A (positional) *strategy* for player Min is a mapping $\Pi_{MIN} : \mathcal{S}_{MIN} \mapsto \mathcal{A}_{MIN}$, and a (positional) *strategy* for player Max is a mapping $\Pi_{MAX} : \mathcal{S}_{MAX} \mapsto \mathcal{A}_{MAX}$. The pair (Π_{MIN}, Π_{MAX}) is called a (positional) *strategy profile*. A strategy profile $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ induces a policy $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ for the SSP instance defined by $(\mathcal{S} := \mathcal{S}_{MIN} \cup \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$ and we define the value of the game for the pair $\Pi = (\Pi_{MIN}, \Pi_{MAX})$, from an initial state i, as the value of the SSP solution associated with Π i.e. $J_\Pi(i)$.

From now on, we assume that **termination is inevitable**, meaning that any strategy profile (Π_{MIN}, Π_{MAX}) induces a policy Π that is proper for the corresponding SSP instance.

We denote by Σ_{MIN} the set of all (positional) strategies for player MIN and by Σ_{MAX} the set of all (positional) strategies for player MAX. SSPG with termination inevitable are a special case of BWR-Games with total effective payoff [22]. Because all policies are proper, for all initial state i, the mean payoff version of the game starting in state i has value zero. It then follows from Theorem 27 in [22] that there exists

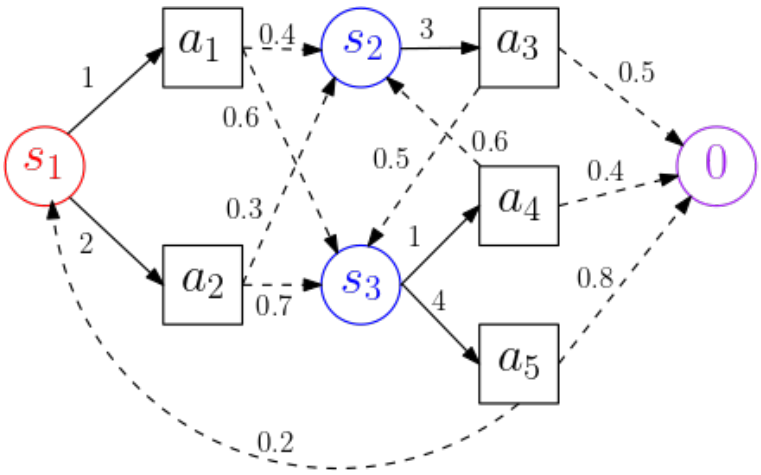


Figure 3.1 – Example of an SSPG instance with $I = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ with $\mathcal{S}_{MIN} = \{s_1, 0\}$ (in red) , $\mathcal{S}_{MAX} = \{s_2, s_3, 0\}$ (in blue), $\mathcal{A}_{MIN} = \{a_1, a_2\}$, $\mathcal{A}_{MAX} = \{a_3, a_4, a_5\}$, $c = (1, 2, 3, 1, 4)$, $J = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$, $P = \begin{pmatrix} 0 & 0.4 & 0.6 \\ 0 & 0.3 & 0.7 \\ 0 & 0 & 0.5 \\ 0 & 0.6 & 0 \\ 0.2 & 0 & 0 \end{pmatrix}$

a *(uniform) Nash Equilibrium in positional strategy* i.e. there exists a strategy profile (Π_{MIN}, Π_{MAX}) such that $J_{(\Pi_{MIN}, \Pi'_{MAX})}(i) \leq J_{(\Pi_{MIN}, \Pi_{MAX})}(i) \leq J_{(\Pi'_{MIN}, \Pi_{MAX})}(i)$ for all $i \in \mathcal{S} \setminus \{0\}$, for all $\Pi'_{MIN} \in \Sigma_{MIN}$ and all $\Pi'_{MAX} \in \Sigma_{MAX}$. We say that Π_{MIN} is the *best response* to strategy Π_{MAX} and Π_{MAX} is the *best response* to strategy Π_{MIN}. Now by Von Neumann's minimax theorem for zero-sum games, we know that such a Nash Equilibrium (Π_{MIN}, Π_{MAX}) satisfies for all $i \in \mathcal{S} \setminus \{0\}$

$$\begin{aligned} J_{(\Pi_{MIN}, \Pi_{MAX})}(i) &= \min_{\Pi'_{MIN} \in \Sigma_{MIN}} \max_{\Pi'_{MAX} \in \Sigma_{MAX}} J_{(\Pi'_{MIN}, \Pi'_{MAX})}(i) \\ &= \max_{\Pi'_{MAX} \in \Sigma_{MAX}} \min_{\Pi'_{MIN} \in \Sigma_{MIN}} J_{(\Pi'_{MIN}, \Pi'_{MAX})}(i) \end{aligned}$$

The stochastic shortest path game with inevitable termination is the problem of finding such a Nash equilibrium.

We can represent an instance of SSPG with a graph $G = (V = (V_{\mathcal{S}_{MIN}} \cup V_{\mathcal{S}_{MAX}}) \cup (V_{\mathcal{A}_{MIN}} \cup V_{\mathcal{A}_{MAX}}), A)$. The representation is the same as the representation of SSP instances in chapter 1, but now the state nodes are colored according to the partition $\mathcal{S}_{MIN}$, $\mathcal{S}_{MAX}$ of $\mathcal{S}$, in particular the red nodes are controlled by player MIN, and the blue nodes are controlled by player MAX. As node 0 is 'controlled' neither by MIN nor MAX, it is colored in purple. An example is given in figure 3.1. The most famous algorithm to solve SSPG is *Strategy Iteration*.

Strategy Iteration

Even if no polynomial time algorithm is known for SSPGs, we can solve this problem with the so called *strategy iteration algorithm* [54, 90]. This iterative algorithm is very close to policy iteration for SSP: both players will improve their strategies step by step until a Nash equilibrium is found. For the following definition, we remind that c_Π and P_Π are the subvector and submatrix of c and P respectively, whose lines correspond to the actions of Π. We first define the reduced cost of an action.

Definition 30 (reduced cost)
Let $\mathcal{S} = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ be an instance of SSPG with inevitable termination.

For all action $a \in \mathcal{A}$, and all deterministic and stationary policy Π, the reduced cost of a is defined by:

$$\bar{c_\Pi}(a) = c(a) - c_\Pi(I - P_\Pi)^{-T}(J - P)^T \mathbb{1}_a$$

where $\mathbb{1}_a$ is a vector of m lines with 1 in the a^{th} position, and 0 elsewhere.

The reduced cost $\bar{c_\Pi}(a)$ represent the relative gain to choose a with regards to Π. If $\bar{c_\Pi}(a) < 0$, including a in the current policy should be interesting for MIN whereas if $\bar{c_\Pi}(a) > 0$ it should be interesting for MAX to do so. Then we can define improving strategies for MIN and MAX like so.

Definition 31 (improving strategy)
Let $\mathcal{S} = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ be an instance of SSPG with inevitable termination and $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ be a pair of deterministic and stationary policies for MIN and MAX. Let $j \in \{MIN, MAX\}$ and $B_j \subseteq \mathcal{A}_j$ such that:

- $\exists a \in B_j$, $\bar{c_\Pi}(a) < 0$ *and* $\forall a \in B_j$, $\bar{c_\Pi}(a) \leq 0$ *if* $j = MIN$
- $\exists a \in B_j$, $\bar{c_\Pi}(a) > 0$ *and* $\forall a \in B_j$, $\bar{c_\Pi}(a) \geq 0$ *if* $j = MAX$
- $\forall s \in \mathcal{S}$, , $\sum_{a \in B_j} J(a, s) \leq 1$

We denote by an improving set with regards to player j such a set.
We denote by $\Pi[B_j]$ the strategy such that:

- $\forall s \in \mathcal{S}$ *if* $\exists a \in B_j$ *such that* $J(a, s) = 1$ *then* $\Pi[B_j](s) = a$
- $\Pi[B_j](s) = \Pi(s)$ *otherwise*

We call $\Pi[B_j]$ an improving strategy with regards to player j

When a player does not have improving strategies anymore, we say that he has a *best response*. In the next definition, we define formally a best response.

Definition 32 (best response)
Let $I = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, C)$ be an instance of SSPG with inevitable termination, $\Pi_{MIN} \in \Sigma_{MIN}$ and $\Pi_{MAX} \in \Sigma_{MAX}$. $\Pi^*_{MAX} \in \Sigma_{MAX}$ is a best response to Π_{MIN} if and only if

$$J_{(\Pi_{MIN}, \Pi^*_{MAX})} = \max_{\pi_{MAX} \in \Sigma_{MAX}} J_{(\Pi_{MIN}, \pi_{MAX})}$$

Similarly, $\Pi^*_{MIN} \in \Sigma_{MIN}$ is a best response to Π_{MAX} if and only if

$$J_{(\Pi^*_{MIN}, \Pi_{MAX})} = \min_{\pi_{MIN} \in \Sigma_{MIN}} J_{(\pi_{MIN}, \Pi_{MAX})}$$

Definition 33 (Nash equilibrium)
A pair of strategies $(\Pi_{MIN}, \Pi_{MAX}) \in \Sigma_{MIN} \times \Sigma_{MAX}$ is a Nash Equilibrium if and only if Π_{MIN} is a best response to Π_{MAX} and Π_{MAX} is a best response to Π_{MIN}.

Lemma 3.1

A strategy $\Pi_{MIN} \in \Sigma_{MIN}$ is a best response to a strategy $\Pi_{MAX} \in \Sigma_{MAX}$ if and only if there is no improving strategy with regard to player MIN. Similarly, a strategy $\Pi_{MAX} \in \Sigma_{MAX}$ is a best response to a strategy $\Pi_{MIN} \in \Sigma_{MIN}$ if and only if there is no improving strategy with regard to player MAX.

Proof. We know from chapter 1 that a strategy is optimal for SSP if and only if there is no action of negative reduced cost. Since fixing Π_{MIN} or Π_{MAX} leads to a (simplified) SSP problem for MIN or MAX, we have symmetrically $\Pi_{MIN} \in \Sigma_{MIN}$ is a best response to a strategy $\Pi_{MAX} \in \Sigma_{MAX}$ if and only if there is no action of negative reduced cost (no improving strategy with regards to MIN), and $\Pi_{MAX} \in \Sigma_{MAX}$ is a best response to a strategy $\Pi_{MIN} \in \Sigma_{MIN}$ if and only if there is no action of positive reduced cost (no improving strategy with regards to MAX). □

Lemma 3.2

Let $I = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, C)$ be an instance of SSPG and let $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ be a pair of deterministic and stationary strategies such that Π_{MIN} is a best response to Π_{MAX}. Let Π'_{MAX} be an improving strategy with regards to MAX, and Π'_{MIN} a best response to Π'_{MAX}. Then

$$J_{(\Pi'_{MIN}, \Pi'_{MAX})} \geq J_{(\Pi_{MIN}, \Pi_{MAX})}$$

and the inequality is strict for at least one state.

Proof. Π_{MIN} is a best response to Π_{MAX} so there is no improving set with regards to MIN (lemma 3.1). Thus, for all $a \in \mathcal{A}_{MIN}$, $\bar{c_\Pi}(a) \geq 0$ so in particular, $\bar{c_\Pi}(a) \geq 0$ for all $a \in \Pi'_{MIN}$.

Π'_{MAX} be an improving strategy with regards to player MAX so there exists $B_{MAX} \subseteq \mathcal{A}_{MAX}$ which is an improving set with regards to MAX. It follows that $\bar{c_\Pi}(a) \geq 0$ for all $a \in B_{MAX}$ and there exists $a \in B_{MAX}$ such that $\bar{c_\Pi}(a) > 0$ (by definition of an improving strategy). Let us see $\Pi' = (\Pi'_{MIN}, \Pi'_{MAX})$ as a strategy of the SSP instance related

with I. We have $c_{\Pi}^{-}(a) \geq 0$ for all $a \in \Pi'$, and there is at least one action $a \in \Pi'$ such that $c_{\Pi}^{-}(a) > 0$. It follows from chapter 1 that $J_{(\Pi'_{MIN},\Pi'_{MAX})} = J_{\Pi'} \geq J_{\Pi} = J_{(\Pi_{MIN},\Pi_{MAX})}$, and the inequality is strict for at least one state.

□

We now define *strategy iteration algorithm*:

Algorithm 3 Strategy Iteration

Input : a deterministic and stationary policy $\Pi = (\Pi_{MIN}, \Pi_{MAX})$
$k = 1$
$\Pi^k_{MIN} \leftarrow \Pi_{MIN}$
$\Pi^k_{MAX} \leftarrow \Pi_{MAX}$
$\Pi^k \leftarrow (\Pi^k_{MIN}, \Pi^k_{MAX})$
loop
  while $\exists B_{MIN} \subseteq \mathcal{A}_{MIN}$ such that $\Pi^k[B_{MIN}]$ is an improving strategy with regards to MIN **do**
    $\Pi^k \leftarrow \Pi^k[B_{MIN}]$
    Define Π^k_{MIN}, Π^k_{MAX} such that $\Pi^k = (\Pi^k_{MIN}, \Pi^k_{MAX})$
  end while
  if $\exists B_{MAX} \subseteq \mathcal{A}_{MAX}$ such that $\Pi^k[B_{MAX}]$ is an improving strategy with regards to MAX **then**
    $\Pi^{k+1} \leftarrow \Pi^k[B_{MAX}]$
    Define Π^k_{MIN}, Π^k_{MAX} such that $\Pi^k = (\Pi^k_{MIN}, \Pi^k_{MAX})$
    $k \leftarrow k + 1$
  else return Π^k
  end if
end loop

Let us apply this algorithm on the previous example of SSPG instance I (figure 3.1) with an initial deterministic and stationary policy $\Pi^1 = (\Pi^1_{MIN}, \Pi^1_{MAX})$ with $\Pi^1_{MIN}(s_1) = a_2$, $\Pi^1_{MAX}(s_2) = a_3$, and $\Pi^1_{MAX}(s_3) = a_4$. We represent a deterministic and stationary policy by coloring in red (resp. in blue) the arc leading to the chosen action in the states of $\mathcal{S}_{MIN}$ (resp. $\mathcal{S}_{MAX}$).

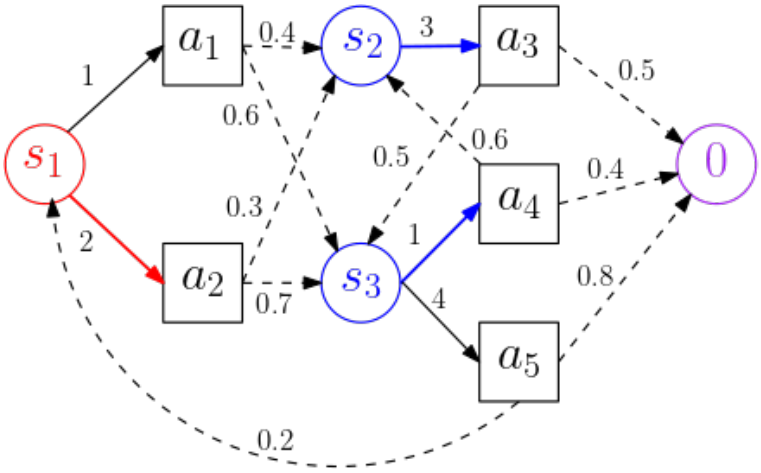


Figure 3.2 – Graphical representation of I with policy Π^1

We begin with computing the reduced cost of the actions in $\mathcal{A}_{MIN}$. From Definition 30, we have $c_{\Pi^1}^-(a_1) = -0.9$ and $c_{\Pi^1}^-(a_2) = 0$. We define $B_{MIN} = \{a_{MIN}\}$, and $\Pi^1[B_{MIN}]$ is an improving strategy with regards to MIN by definition. According to the strategy iteration algorithm, we define $\Pi^2 = \Pi^1[B_{MIN}]$:

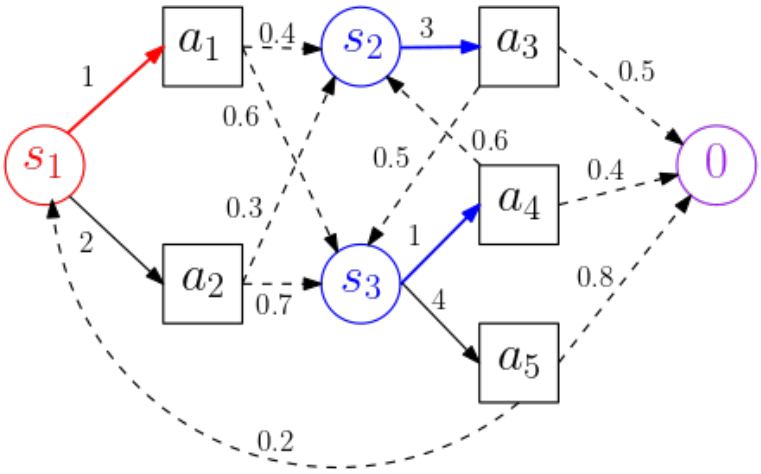
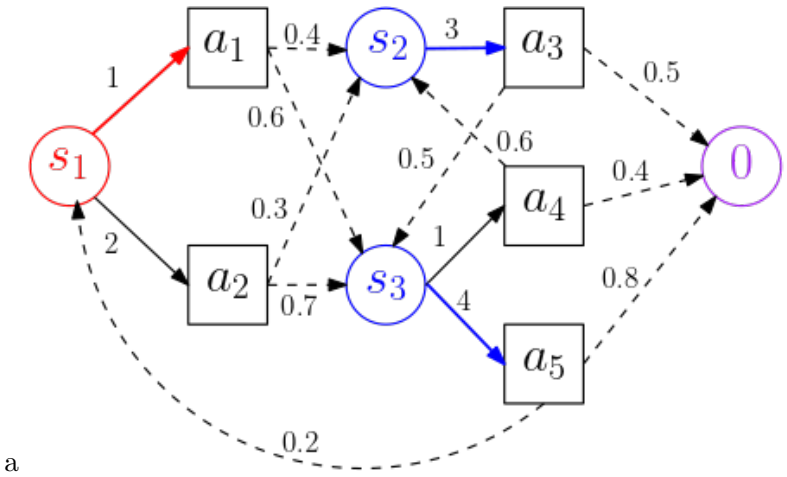


Figure 3.3 – Graphical representation of I with policy Π^2

There is no more possible improvement for player MIN, so we continue by computing the reduced cost of the actions in B_{MAX} with policy Π_{MAX}. We have $c_{\Pi^2}^-(a_3) = 0$, $c_{\Pi^2}^-(a_4) = 0$ and $c_{\Pi^2}^-(a_5) = 4$. We define $B_{MAX} = \{a_5\}$ and again, $\Pi^2[B_{MAX}]$ is an improving strategy with regards to MAX by definition of an improving strategy. We define $\Pi^3 = \Pi^2[B_{MAX}]$:



a

Figure 3.4 – Graphical representation of I with policy Π^3

By computing the reduced cost of the actions of $\mathcal{A}$, we obtain

$$(c_{\bar{\Pi}_3}(a_1), c_{\bar{\Pi}_3}(a_2), c_{\bar{\Pi}_3}(a_3), c_{\bar{\Pi}_3}(a_4), c_{\bar{\Pi}_3}(a_5)) = (0, \frac{81}{84}, 0, -4, 0)$$

so there is no improving strategy. The algorithm terminates with $\Pi^3 = (a_1, a_3, a_5)$.

Theorem 3.3

Strategy iteration algorithm terminates after a finite number of iterations N and $\Pi^N = (\Pi^N_{MIN}, \Pi^N_{MAX})$ is a Nash equilibrium.

Proof. At each iteration $k \in [\![1, n]\!]$, At the end of the 'while' loop, Π^k_{MIN} is a best response to Π^k_{MAX} (there is no more improving set with regards to MIN. At the end of the 'if', Π_{MAX} is an improving strategy with regards to MAX. Then from lemma 3.2:

$$J_{(\Pi^{k+1}_{MIN}, \Pi^{k+1}_{MAX})} \geq J_{(\Pi^k_{MIN}, \Pi^k_{MAX})}$$

and the inequality is strict for at least one state.

It follows that a pair of strategies does not appear twice, and since there is a finite number of pairs of strategies, the algorithm terminates in a finite number of iteration, let us say $N \in \mathbb{Z}^+$. At the end of the algorithm the pair of strategy $(\Pi^N_{MIN}, \Pi^N_{MAX})$ is returned, and there is no improving strategy with regards to MIN nor MAX. It follows from lemma 3.1 that Π^N_{MIN} is a best response to Π^N_{MAX} and that Π^N_{MAX} is a best response to Π^N_{MIN}. So by definition, $(\Pi^N_{MIN}, \Pi^N_{MAX})$ is a Nash Equilibrium. □

Note that each iteration of the strategy iteration algorithm can be interpreted as a policy iteration algorithm fixing Π^k_{MAX} and improving Π^k_{MIN} (and hence finding the best response to Π^k_{MAX}); and one iteration of a policy iteration algorithm fixing Π^k and improving Π^k_{MAX}. So the worst case complexity of strategy iteration is exponential in

the number of states and actions as the worst case of policy iteration is exponential [75] (even in the case of termination inevitable [104]).

3.3 Special cases of SSPG with termination inevitable

As mentioned before, the existence of a LP-formulation for SSPG is a long standing open question. Our idea is to dig into special cases of SSPG with termination inevitable: stopping Simple Stochastic Games (stopping SSG) and Robust Shortest Path with termination inevitable (RSP with termination inevitable), in order to find a LP-formulation for these potential 'simpler' cases and hopefully generalize these formulations to SSPG with termination inevitable.

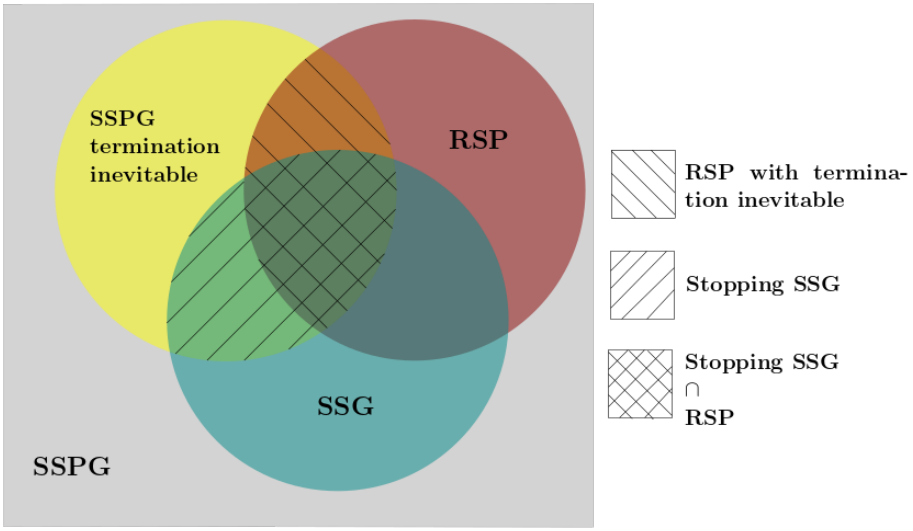


Figure 3.5 – Inclusions of SSPG, SSG and RSP

Note that more general version of stopping SSG and RSP with termination inevitable exists as we detail later, which are special case of SSPG (but without termination inevitable).

3.3.1 Simple Stochastic Games (SSG)

Condon introduced Simple Stochastic Games in 1992 [30]. We follow her definition: a *simple stochastic game* is a game defined on a directed graph with three types of vertices, called *max*, *min* and *average* vertices. There are two special sink vertices named 0_{MIN}-*sink* and 0_{MAX}-*sink* and all but these two sink vertices have exactly two neighbors (possibly identical). The game is played by two players, a MIN and a MAX player. A

pebble is initially placed on a given node and the pebble is moved on the edges of the graph as follows: in a *max* (resp. *min*) node, MAX (resp. MIN) chooses a neighbor to which the pebble is moved, and in a average node, the pebble is moved to one of the two neighbors with equal probability $\frac{1}{2}$. MAX wins if the pebble reaches the 0_{MAX}-sink and otherwise MIN wins (if the pebble reaches the 0_{MIN}-sink or if no sink is reached). Condon [31] proved that deciding if the probability that MAX wins is greater than $\frac{1}{2}$ is in $NP \cap coNP$ and she conjectured that the problem is in P. She also proved that one can restrict attention to *stopping* simple stochastic games as far as polynomial solvability is concerned, i.e. simple stochastic games that end in a sink node with probability one for any possible choice of strategies of the players.

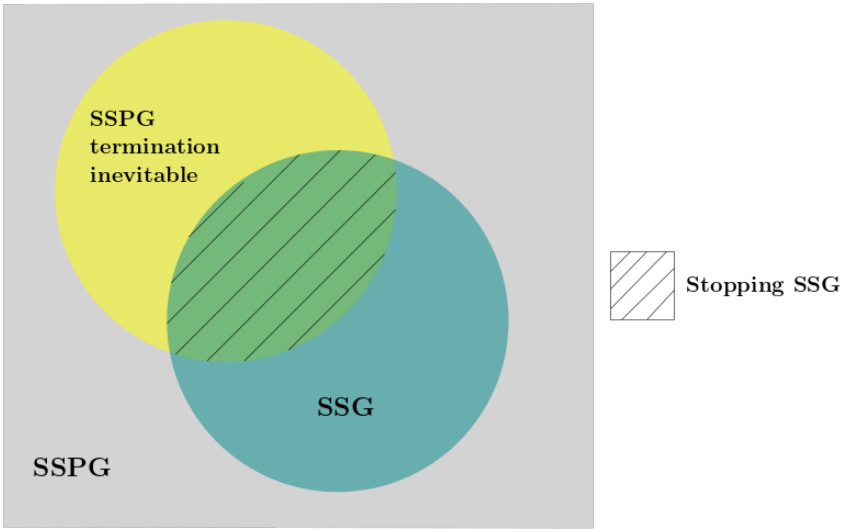


Figure 3.6 – Inclusions SSG and SSPG with termination inevitable

In our representation, we color the MIN vertices in red, the MAX vertices in blue and the *average* vertices in green. We also distinguish the edges as follows: the regular edges represent the choices for the two players and the dashed edges represent the two possible stochastic transitions from an *average* vertex (with probability $\frac{1}{2}$).

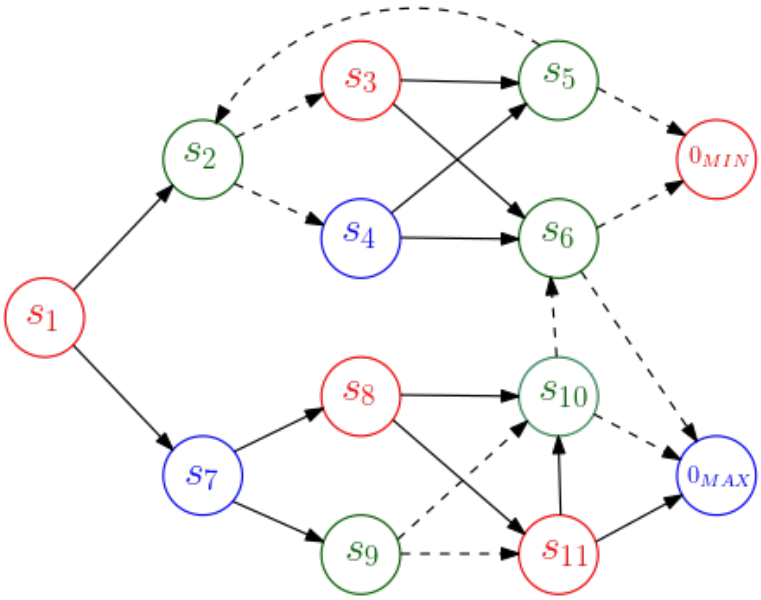


Figure 3.7 – Example of a SSG instance

A SSG instance is a tuple $(\mathcal{S} = (\mathcal{S}_{MIN} \cup \mathcal{S}_{MAX} \cup \mathcal{S}_{avg}), E)$ where $\mathcal{S}_{MIN}$ is a finite set of states controlled by MIN, $\mathcal{S}_{MAX}$ is a finite set of states controlled by MAX and $\mathcal{S}_{avg}$ is a finite set of states not controlled by any player. $E \subseteq S \times S$ is a set of 'edges'. The edges can be partitioned into $E = \cup_{s \in \mathcal{S}} E(s)$ with $E(s) = \{e = (s_1, s_2) \in E | s_1 = s\}$. We have $|E(s)| \leq 2$. There are two special target sink states $0_{MIN} \in \mathcal{S}_{MIN}$ and $0_{MAX} \in \mathcal{S}_{MAX}$: $E(0_{MIN}) = E(0_{MAX}) = \emptyset$, and one starting state $s_0 \in \mathcal{S}$. A deterministic and stationary policy for MIN (resp. for MAX) is a mapping $\Pi_{MIN} : S_{MIN} \to E$ (resp. $\Pi_{MAX} : S_{MAX} \to E$) such that $\Pi_{MIN}(s) \in E(s)$ (resp. $\Pi_{MAX}(s) \in E(s)$). Once a pair of policies $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ is set, the states form a Markov chain, with two types of transitions: deterministic one from $\mathcal{S}_{MIN}$ and $\mathcal{S}_{MAX}$ that are given by Π ; and stochastic transitions from the states of $\mathcal{S}_{avg}$ whose transition probability to the two neighbors are $\frac{1}{2}$. The SSG decision problem is deciding whether there exists a policy for player MAX that guarantee that the probability to end up in sink 0_{MAX} (given by the stationary distribution of the absorbing Markov chain) is greater than $\frac{1}{2}$.

From now on, we will consider only *stopping SSG*: we can see a stopping SSG instance as a special case of SSPG with termination inevitable. A stopping SSG instance can be seen as a tuple $(\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, \mathcal{A}_{avg}, J, P, c)$ where $((\mathcal{S}_{MIN} \cup \mathcal{S}_{avg}), \mathcal{S}_{MAX}, (\mathcal{A}_{MIN} \cup \mathcal{A}_{avg}), \mathcal{A}_{MAX}, J, P, c)$ is an instance of SSPG with termination inevitable. Moreover, for all state $s \in \mathcal{S}$, $\sum_{a \in \mathcal{A}} J(a, s) \leq 2$, meaning that there is at most two actions available in each state. All actions $\mathcal{A}_{MIN} \cup \mathcal{A}_{MAX}$ are deterministic, the restriction of P to the actions of $\mathcal{A}_{MIN} \cup \mathcal{A}_{MAX}$ contains only 0 and 1. The restriction of P to the actions of A_{avg} contains only $0, 1$ and $\frac{1}{2}$. There is a special state $0_{MIN} \in \mathcal{S}_{MIN}$ (resp. $0_{MAX} \in \mathcal{S}_{MAX}$) in which there is one only action that lead to 0 with probability

1 and cost 0 (resp. with probability 1 and cost 1). The cost of all the other actions is 0, and cannot lead to state 0. Here the goal of MIN and MAX are pretty clear: since the only action that has positive cost is the outgoing action from 0_{MAX}, MIN wants to reach state 0_{MIN} and MAX want to reach state 0_{MAX}. A deterministic and stationary policy for MIN (resp. for MAX) is mapping $\Pi_{MIN} : \mathcal{S}_{MIN} \mapsto \mathcal{A}_{MIN}$ (resp. $\Pi_{MAX} : \mathcal{S}_{MAX} \mapsto \mathcal{A}_{MAX}$). Solving a stopping SSG instance is deciding whether or not there exist a deterministic and stationary policy Π^*_{MAX} such that for all deterministic and stationary Π_{MIN} of MIN, we have $J_{(\Pi_{MIN},\Pi^*_{MAX})} \geq \frac{1}{2}$. Note that as the only action which has a non-zero cost is the (only) action available in 0_{MAX}, the cost of a pair of strategies $J_{(\Pi_{MIN},\Pi_{MAX})}$ is exactly the probability to reach 0_{MAX} following Π_{MIN} and Π_{MAX}.

Note that we artificially put the nodes of $\mathcal{S}_{avg}$ under control of player MIN. We could have put them under control of player MAX without any consequences on the model, while only one action is available in nodes of $\mathcal{S}_{avg}$.

A SSG is a special case of SSPG. The representation of the previous instance of SSG as an SSPG is given by figure 3.8.

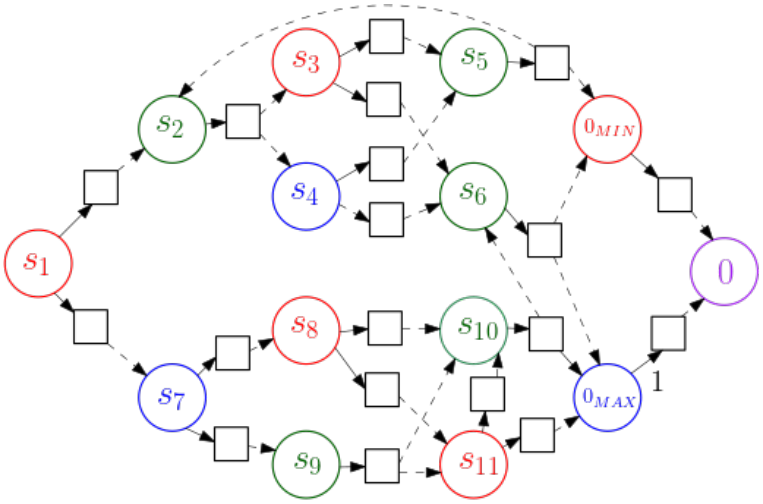


Figure 3.8 – Representation of the instance of SSG 3.7 as a SSPG

The existence of a (polynomial) LP formulation for stopping SSG (*i.e.* a linear program whose optimal solution would solve the SSG decision problem) is open [30].

Condon [31] proves that some 'naive' formulations do not work. She considers for instance the following linear program:

$$
\begin{array}{lcll}
\min & & \sum_{s\in\mathcal{S}_{MAX}} v(s) - \sum_{s\in\mathcal{S}_{MIN}} v(s) & \\
v(i) & \geq & v(j) & \forall i \in \mathcal{S}_{MIN} \text{ and } (i,j) \in E \\
v(i) & \leq & v(j) & \forall i \in \mathcal{S}_{MAX} \text{ and } (i,j) \in E \\
v(i) & = & \frac{1}{2}(v(j)+v(k)) & \forall i \in \mathcal{S}_{Avg} \text{ and } (i,j),(i,k) \in E \\
v(i) & \geq & 0 & \forall i \in \mathcal{S} \\
v(0_{MIN}) & = & 0 & \\
v(0_{MAX}) & = & 1 &
\end{array}
\qquad (P_{Condon})
$$

The idea behind this linear program is to mimic the dual of the standard network flow formulation for the deterministic shortest path (for more details on network flow formulation see) when there are only 0-cost edges. Indeed, if we consider only the *min* vertices, the previous linear program becomes:

$$
\begin{array}{lcll}
\max & & \sum_{s\in\mathcal{S}_{MIN}} v(s) & \\
v(i) & \leq & v(j) & \forall i \in \mathcal{S}_{MIN} \text{ and } (i,j) \in E \\
v(i) & \geq & 0 & \forall i \in \mathcal{S}_{MIN} \\
v(0_{MIN}) & = & 0 &
\end{array}
\qquad (D_{SP})
$$

and a feasible solution of (D_{SP}) is a potential over the nodes in the special case of 0-cost edges. Thus, we would like $v(i)$ to be the probability, starting from $i \in \mathcal{S}$, to reach the sink 0_{MIN}: if $v(i) = 0$, the probability of reaching 0_{MIN} is 0 so from i, we would reach 0_{MAX} surely (as termination is inevitable for stopping SSG).

However, this linear program is not a formulation for stopping SSG. Let us consider the following instance of SSG (we keep Condon's representation for convenience, erasing all the actions).

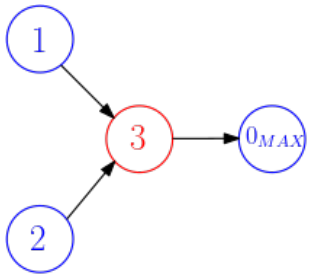


Figure 3.9 – Counter-Example to D_{SP} being a LP formulation of stopping SSGs (taken from [30])

On this example, in each node, there is only one strategy for each player, and this strategy leads to 0_{MAX} with probability 1. Thus, we would like to have $v(i) = 1$ for $i = 1, 2, 3$ (which is feasible for (P_{Condon})), and in this case, the cost function in (P_{Condon}) is 1. However, if we define $v(i) = 0$ for $i = 1, 2, 3$, then it is another feasible solution to (P_{Condon}), and the cost is 0, hence the solution to the SSG does not coincide with the optimal solution to (P_{Condon}). The previous linear program, and the counter example

are from [31]. In this article, Condon presents algorithms to solve the SSG decision problem, and a quadratic formulation. She also proves that some naive algorithms such that a modified version of Hoffman-Karp algorithm, or Pollatschek Avi-Itzhak algorithm do not work. She also proved that one can restrict attention to stopping SSG as far as polynomial solvability is concerned. The main remaining question is the existence of a polynomial time algorithm to solve the stopping SSG decision problem.

Strategy Iteration can be applied to solve stopping SSG (as it is a special case of SSPG with termination inevitable), but we know that strategy iteration algorithm is exponential in the worst case in the number of states and actions even in this case as already discussed [104]. Our original idea was to start studying polynomial time solvable special cases of SSPG with termination inevitable. This is the case of the Robust Shortest Path with termination inevitable (RSP with termination inevitable)[17]. It is the deterministic version of SSPG with termination inevitable. It can also be seen as a natural variation of Stochastic Shortest Path, where 'nature' does not choose randomly but as to 'hurt' as much as possible. For example in the golfer's problem, solving the RSP gives an upper bound for the number of shots the golfer has to shoot in order to put the ball in the hole.

3.3.2 Robust Shortest Path

In [17], Bertsekas defines the RSP as a non-symmetrical game on a directed graph $G = (\mathcal{S} \cup \{0\}, A)$ (0 is a special sink node), where player MIN controls the decisions on nodes $\mathcal{S} \cup \{0\}$ while an antagonist player MAX controls the destination of these decisions. From each node $s \in S$, MIN chooses an edge (or action) a from a finite set $A(s) \subseteq A$ of available actions in s. Then MAX chooses the destination $s' \in \mathcal{S} \cup \{0\}$ from a subset of nodes that we call $\mathcal{S}^+(a, s)$. Then a cost $c(s, a, s')$ is incurred. Bertsekas defines a policy for MIN to be proper if MAX cannot make MIN loop in the graph without being able to reach 0. A policy which is not proper is said to be improper. The RSP is the problem of finding a proper policy of minimum cost (the cost of a proper policy is the sum of the incurred costs until reaching the destination 0). He proves that such a policy exists if (i) there exists at least one proper policy and (ii) every improper policy makes player MIN loop in a positive cost cycle. Bertsekas could also relax the latter assumption and prove that an optimal policy exists even if every improper policy makes player MIN loop in a non-negative cost cycle. Regarding the computational methods, Bertsekas proves that three iterative algorithms converge to an optimal exact solution: Value Iteration, Policy Iteration and a Dijkstra-like algorithm when the costs are non negative. We explain a version of Value Iteration and the Dijkstra-like algorithm in this framework later in the chapter. Bertsekas also proves that both Value Iteration and Dijkstra converge in polynomial time, which proves that RSP is in P. We encourage the reader to refer to [17] for more details.

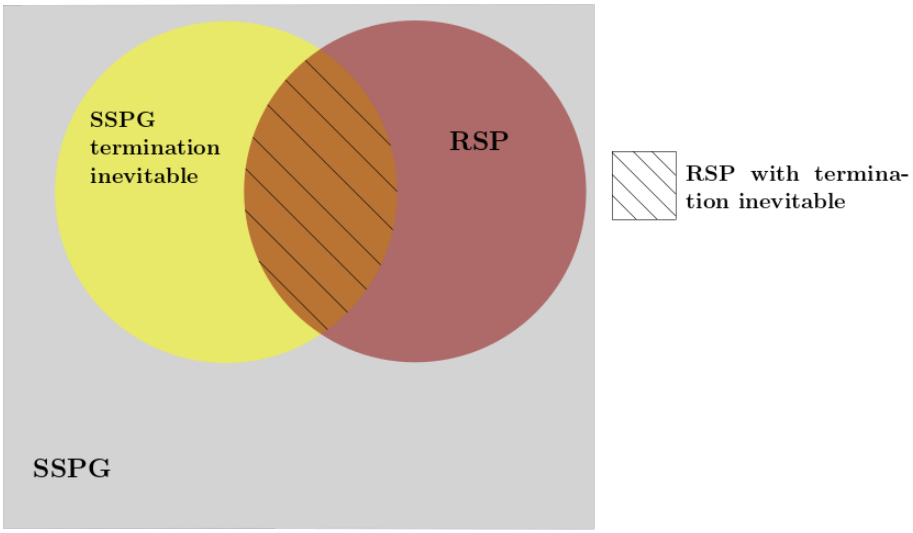


Figure 3.10 – Inclusion of RSP and SSPG with termination inevitable

If we assume that all policies are proper, we can focus on instances of RSP said to be *with termination inevitable*. An instance of RSP with inevitable termination can be seen as a SSPG instance with inevitable termination $(\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ where P is a 0/1 matrix. The states in $\mathcal{S}_{MIN}$ are controlled by MIN and the states in $\mathcal{S}_{MAX}$ are controlled by MAX. A strategy for MIN (resp. for MAX) is a mapping $\Pi_{MIN} : \mathcal{S}_{MIN} \mapsto \mathcal{A}_{MIN}$ (resp. $\Pi_{MAX} : \mathcal{S}_{MAX} \mapsto \mathcal{A}_{MAX}$). Note that for the instances of RSP with termination inevitable, the two assumptions of Bertsekas for the problem to have an optimal solution are trivially true.

As RSP is a special case of SSPG, we could adopt the graphical representation of SSPG, but for convenience we erase the deterministic actions to replace them with simple edges. Hence, we can represent a RSP instance with a graph $G = (V = V_{MIN} \cup V_{MAX}, A)$ such that each node in V_{MIN} (resp. in V_{MAX}) represent a state in $\mathcal{S}_{MIN}$ (resp. $\mathcal{S}_{MAX}$). Each action $a \in \mathcal{A}$ is represented by an arc $a = (s, s')$ such that s is the unique state such that $J(a, s) = 1$ and s' is the unique state such that $P(a, s') = 1$ if it exists, and $s' = 0$ otherwise. We will call this representation the graphical representation of RSP instance. We will slightly abuse notation and from now, a state $s \in \mathcal{S}$ will either denote the state or the corresponding node in V, and an action $a \in \mathcal{A}$ will either denote the action or the corresponding arc in A.

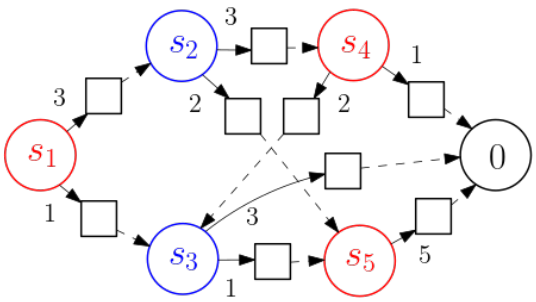


Figure 3.11 – Graphical representation of a RSP as a SSPG

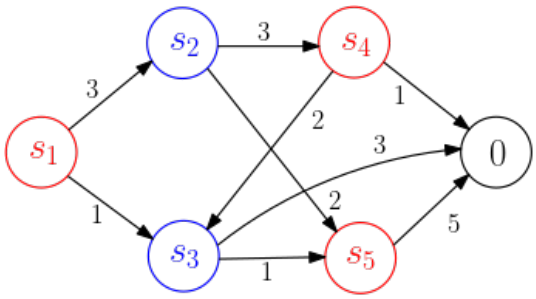


Figure 3.12 – Simplified graphical representation of a RSP

At this point, RSP with termination inevitable can be seen with two different points of view. On the one hand, it can be seen as a special case of SSPG with termination inevitable. A solution is thus a pair of strategies $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ for MIN and MAX which forms a Nash Equilibrium, *i.e.* for which we have, in particular:

$$J_{(\Pi_{MIN},\Pi_{MAX})}(i) = \min_{\Pi'_{MIN}\in\Sigma_{MIN}} \max_{\Pi'_{MAX}\in\Sigma_{MAX}} J_{(\Pi'_{MIN},\Pi'_{MAX})}(i) \text{ for all } i \in \mathcal{S} \setminus \{0\}$$

On the other hand, from the formulation above, the RSP can be seen as an optimization problem for MIN (just like Bertsekas does in [17]). In this vision, an optimal solution is a policy for player MIN of minimum cost (the cost of a policy $\Pi_{MIN} \in \Sigma_{MIN}$ is $\max_{\Pi'_{MAX}\in\Sigma_{MAX}} J_{(\Pi_{MIN},\Pi'_{MAX})}$). We know that if $\Pi = (\Pi_{MIN}, \Pi_{MAX})$ is a Nash equilibrium, then Π_{MIN} is an optimal solution for the corresponding optimization problem.

If we consider only instances with termination inevitable, the solutions to the optimization problem are Nash equilibrium. Indeed in this case $\max_{\Pi'_{MAX}\in\Sigma_{MAX}} J_{(\Pi'_{MIN},\Pi'_{MAX})}$ is finite for all $\Pi'_{MIN} \in \Sigma_{MIN}$.

Definition 34
Let I be an instance of RSP seen as an optimization problem for MIN, and $\Pi_{MIN} \in \Sigma_{MIN}$ any solution of I (a strategy for MIN). Let $s \in \mathcal{S}$, we define the potential of s with respect to Π_{MIN} as:

$$y^{\Pi_{MIN}}(s) = \max_{\Pi'_{MAX}\in\Sigma_{MAX}} J_{(\Pi_{MIN},\Pi'_{MAX})}(s)$$

$y^{\Pi_{MIN}}$ represents the cost from s to 0 if we follow Π_{MIN} and that MAX plays one of his best response to Π_{MIN}. If Π_{MIN} is optimal, $y^{\Pi_{MIN}}$ are said to be the optimal potentials.

Note that, if we consider only instance with termination inevitable, there is no pair of policies for MIN and MAX such that the agent could 'loop' in the system. This implies that the graph that represents instances of RSP with termination inevitable is acyclic.

In our representation, we color the chosen edges by MIN in red and by MAX in blue. We focus on deterministic policies and hence there is only one edge chosen for each state. Moreover, as the graph representing RSP with termination inevitable is acyclic, the subgraph induced by the selected edges is a 0-anti-arborescence. We give two examples of deterministic and stationary policies with the previous instance (figure 3.13).

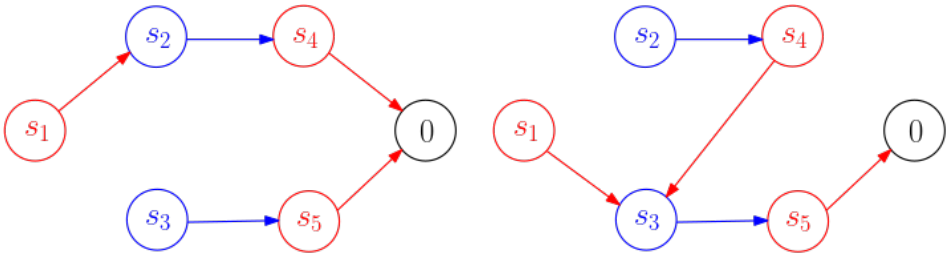


Figure 3.13 – Two examples of deterministic and stationary policies

Computational methods

We know that the Robust Shortest Path Problem is in P [17]. We present two algorithms from [17] a Bellman-like algorithm and Dijkstra-like one.

Bellman-like Algorithm

Let $I = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ be an instance of RSP **with termination inevitable** and $G_I = (V = V_{MIN} \cup V_{MAX}, A)$ its graphical representation. As G_I is acyclic, there exists a topological order $\mathcal{O}$ over the states (see the preliminaries for more details). We can apply a slightly modified version of Bellman algorithm for shortest path problem. Just like for Bellman algorithm, we will set and update *labels* y for each state v, that represent the potentials (here we see RSP as an optimization problem for MIN, so the optimal cost of a path is the cost incurred from v to 0 following the optimal policy for MIN that we are looking for). As 0 is a sink node, we can always assume that 0 is at the end of all topological order $\mathcal{O}$ and we set $y(0) = 0$. Then we update each node v one by one, following the reverse order of $\mathcal{O}$:

- if $v \in V_{MIN}$, $y(v) = \min_{a=(v,v') \in A} c(a) + y(v')$
- if $v \in V_{MAX}$, $y(v) = \max_{a=(v,v') \in A} c(a) + y(v')$

To determine the strategy from the potentials, we only need to remember for which arc the min (resp. the max) has been obtained for the nodes in V_{MIN} (resp. in V_{MAX}). We can prove inductively that it returns an optimal solution (an optimal strategy for MIN).

Let us illustrate the previous algorithm on the following instance of RSP with the following topological order $\mathcal{O} = (s_1, s_2, s_4, s_3, s_5, 0)$ (figure 3.14).

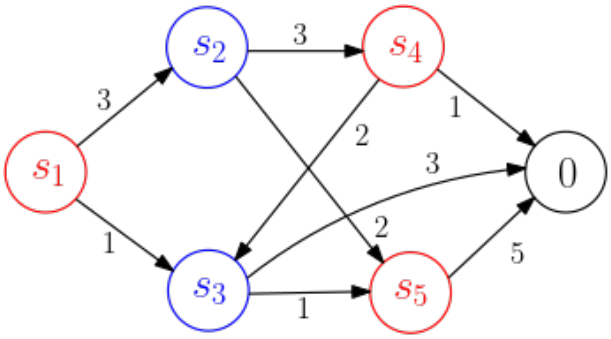


Figure 3.14 – Example of a RSP instance

Here are the different steps that lead to the optimal strategy Π^*. We represent the labels y as a vector $y = (y(s_1), y(s_2), y(s_3), y(s_4), y(s_5), y(0))$

Iterations	labels y
1	$y = (\infty, \infty, \infty, \infty, \infty, 0)$
2	$y = (\infty, \infty, \infty, \infty, 5, 0)$
3	$y = (\infty, \infty, 6, \infty, 5, 0)$
4	$y = (\infty, \infty, 6, 1, 5, 0)$
5	$y = (\infty, 7, 6, 1, 5, 0)$
6	$y = (7, 7, 6, 1, 5, 0)$

We give an optimal strategy in the following figure (figure3.15).

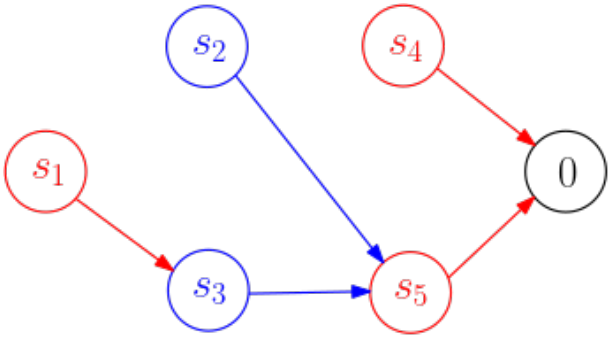


Figure 3.15 – Optimal solution of a RSP

This algorithm runs in $O(m)$ (where m the number of arcs): we explore each arc exactly once.

Dijkstra-like Algorithm

When the costs are non negative and even if termination is not inevitable, we can apply a Dijkstra-like algorithm. This algorithm is taken from [17]. We slightly adjusted the algorithm to make it fit our framework: we symmetrize the points of view, and compute both strategies at the same time. Here again, we consider RSP as an optimization problem for MIN, so our goal is to find a policy of minimum cost for MIN.

Just like for Bellman-Like algorithm, we define *labels* y over the states. At the end of the algorithm, $y(v)$ represents the optimal potential of v. We initialize $y(0) = 0$ and $y(v) = \infty$ for all $v \in \mathcal{S} \setminus \{0\}$. We define two sets Y and Z that we set to: $Y = \emptyset$ and $Z = \{0\}$ initially. At each step of the algorithm, Y represents the nodes for which the labels are the optimal potentials, and Z represent the nodes for which the labels have a finite value, but are maybe not the optimal potentials yet. The algorithm iterates until Z is empty.

Each iteration runs as follows (it generalizes naturally the standard Dijkstra's algorithm):

- we take from Z the state z^* in V_{MIN} of minimal label. As $z^* \in Z$, the label of z^* has finite value.

- as z^* has minimum label, we can insure that its potential is optimal [2]. So we place it in Y.

- we update all the labels of the states $u \in Z \cap V_{MAX}$: if all the potentials of the neighbors of u are optimal. In this case the label of u is updated choosing the path incurring a maximum cost (best choice for MAX).

- we update all the labels of the states $v \in Z \cap V_{MIN}$ if one choice induces a smaller cost. If we update, we now know that the label of u is finite, but may be not represent an optimal potential (so we place u in Z)

We give the outline of a Dijkstra-like algorithm. For all $u \in V$ we denote by $\delta^+(u)$ the outgoing arcs from u in G. (and by $\delta^-(u)$ the incoming arcs to u in G).

[2] as will be seen later

Algorithm 4 Dijkstra-like Algorithm

while $Z \neq \emptyset$ **do**
  Define z^* such that $y(z^*) = \min_{z \in Z \cap V_{MIN}} y(z)$
  $Z \leftarrow Z \setminus \{z^*\}$
  $Y \leftarrow Y \cup \{z^*\}$
  for all $u \notin Y$ and $u \in V_{MAX}$ **do**
    if $\delta^+(u) \subseteq Y$ **then**
      $y(u) \leftarrow \max_{a \in \delta^+(u)} y(v) + c(u, v)$
      $Y \leftarrow Y \cup \{u\}$
    end if
  end for
  for all $u \notin Y$ and $u \in V_{MIN}$ **do**
    $U(u) = \{a = (u, v) \in \delta^+(u) | v \in Y, z^* \in \delta^+(a)\}$
    if $U(u) \neq \emptyset$ AND $y(u) > \min_{a=(u,v) \in U(u)} y(v) + c(a)$ **then**
      $y(u) \leftarrow \min_{a=(u,v) \in U(u)} y(v) + c(a)$
      $Z \leftarrow Z \cup \{u\}$
    end if
  end for
end while

Let us illustrate this algorithm on the previous example. At each iteration, we give the value of Y, Z, the value of z^* and y at the end of the iteration (figure 3.16).

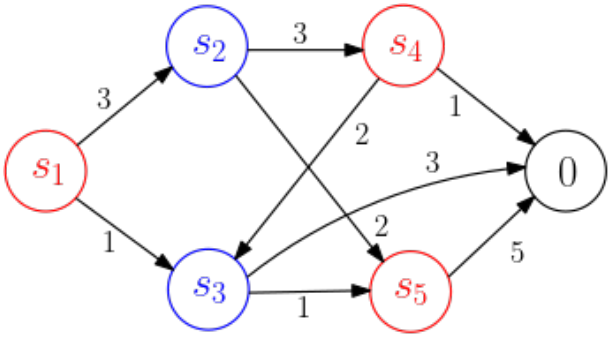


Figure 3.16 – Example of a RSP instance

Iterations	labels y	Y	Z	z^*
0	$y = (\infty, \infty, \infty, \infty, \infty, 0)$	$\emptyset$	$\{0\}$	
1	$y = (\infty, \infty, \infty, 1, 5, 0)$	$\{0\}$	$\{4, 5\}$	0
2	$y = (\infty, \infty, \infty, 1, 5, 0)$	$\{0, 4\}$	$\{5\}$	4
3	$y = (7, 7, 6, 1, 5, 0)$	$\{0, 4, 5, 2, 3\}$	$\{1\}$	5
4	$y = (7, 7, 6, 1, 5, 0)$	$\{0, 4, 5, 2, 3, 1\}$	$\emptyset$	1

Once again, by looking for which arc the *min* or the *max* has been reached for y, we

can find the optimal strategy (figure 3.17).

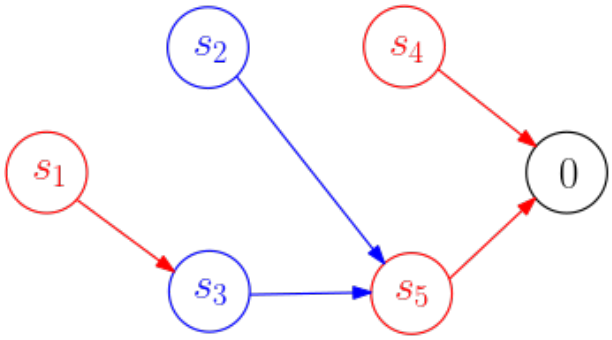


Figure 3.17 – An optimal solution

This algorithm runs in $O(n^2m)$ in worst case, as it terminates with at most $n+1$ iterations of nm operations in the worst case [17] (n is the total number of states, and m the total number of actions/edges).

Proof. (sketch) We can note first that at each iteration (an iteration is the operations inside the outer while loop), there is one node that leaves Z, and this node is never coming back to Z. Thus, the number of iterations is bounded by n, where n is the number of nodes. In fact, the algorithm terminates in exactly n_1+1 iterations, where $n_1 = |V_{MIN}|$ and at the end, $Y = V$. We refer to [17] for the proof.

Then, let us prove that at the end of each iteration, for all $v \in V_{MIN} \cap Y$ and for all $v' \in V_{MIN} \cap (V \setminus Y)$, we have $y(v) \geq y(v')$, meaning that the labels of the nodes in $V_{MIN} \cap Y$ are smaller (one by one) than the labels of the nodes in $V_{MIN} \cap (V \setminus Y)$.

We use induction on the number of iterations. At the end of the first iteration, we have $V_{MIN} \cap Y = \{0\}$ and $y(0) = 0$. Since all the costs are non-negative and from the updates of the labels in the algorithms, we have $y(v) \geq 0$ for all $v \in V$. So the assertion holds for the first iteration. Let us assume that the assertion holds for the iteration $k-1$. Let us call y^k the labels at the start of iteration k and $\tilde{y}^k$ the labels at the end of iteration k. By definition of z^*, we have, for all $v \in V_{MIN} \cap (Y \cup \{z^*\})$ (whose labels do not change during an iteration) and for all $v' \in V_{MIN} \cap (V \setminus (Y \cup \{z^*\})$,

$$\tilde{y}^k(v) = y^k(v) \leq y^k(z^*) \leq y^k(v)$$

Then, for all $v' \in V_{MIN} \cap (V \setminus (Y \cup \{z^*\}))$, the labels are updated according to

$$\tilde{y}^k(v') = \min\left[y^k(v'), \min_{a=(v',w)\in U(v')} y(w) + c(a)\right]$$

Because $c(a) \geq 0$ for all $a \in \mathcal{A}$, and $z^* \in \delta^+(a)$ for all $a \in U(v')$, we have:

$$\tilde{y}^k(v') \geq \min\left[y^k(v'), y^k(z^*)\right]$$

$$\tilde{y}^k(v') \geq y^k(v)$$
$$\tilde{y}^k(v') \geq \tilde{y}^k(v)$$

which proves the assertion. Now, as at the end of each iteration, the nodes in $V_{MIN} \cap Y$ has smaller labels than the other nodes, and as the costs are non negative, we can prove by induction that at the end of each iteration, $y(v)$ represents the cost of an optimal path from v to 0 using only nodes in Y. The main idea is that as the nodes in $V_{MIN} \cap Y$ have smaller potentials, and the costs non-negative, it would not be worth to use a path that goes in and out Y. At the end of the algorithm, as $Y = V$, the returned labels are optimal potentials. We let the reader refer to the complete proof in [17]. □

Note that this algorithm still works for instances of RSP that contains cycles [17].

3.3.3 ILP-Formulations for RSP and SSPG

RSP with termination inevitable is in P, but no LP-formulation is known for this problem. The RSP is very close to the deterministic shortest path problem, so we tried to 'mimic' known LP-formulations for shortest path problems.

We know that for the $(s-t)$-deterministic shortest path problem, one linear programming formulation consists in looking for a feasible flow from s to t of minimum cost. Once an optimal solution is found, the optimal flow defines a path from s to t of minimum cost. For the RSP with termination inevitable, we want a shortest path from each $v \in V_{MIN}$ to 0, and a longest path from each $v \in V_{MAX}$ to 0. Thus, we define $|V|$ feasible flows from each $s \in V$ to 0 (constraint (1)). Constraint (5) defines potentials from the flows. Constraint (2) insure that all these paths are consistent: let $s, s' \in V$ be two states, and p, p' the optimal paths from s to 0 and from s' to 0, respectively. Then if p pass through s', then the sub-path of p from s' to 0 and p' have to be the same. In order to insure that the path from each $s \in V_{MIN}$ (resp. $s \in V_{MAX}$) have minimum cost (resp. maximum cost), we mimic the dual of the linear programming formulation for deterministic shortest path. We have seen in the introduction of this thesis that this dual uses potentials, and that the inequalities between potentials define the minimization of the path. We slightly adapt these constraints in order to define shortest and longest path from the states in V_{MIN} and V_{MAX}, respectively.

Let $I = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}, J, P, c)$ be a instance of RSP, and $G = (V = V_{MIN} \cup V_{MAX}, A)$ be the graph representation of this instance. For all $u \in V$, we denote by $N(u)$ the neighborhood of v, i.e. $N(u) = \{v \in V | \exists a \in A, a = (u, v)\}$. We define the following polyhedron:

$$
\begin{array}{lcll}
x^s(\delta^+(v)) - x^s(\delta^-(v)) & = & \begin{cases} 1 & \text{if } v = s \text{ and } v \neq 0 \\ 0 & \text{otherwise} \end{cases} & \forall s, v \in V \quad (1) \\
x^u_a - x^s_a & \geq & 0 & \forall a \in \delta^+(u), \forall u, s \in V \quad (2) \\
y^v - y^u & \leq & c_a & \forall a = (u, v) \in A, u \in V_{MIN} \quad (3) \\
y^v - y^u & \geq & c_a & \forall a = (u, v) \in A, u \in V_{MAX} \quad (4) \\
\sum_{a \in A} x^s_a c_a & = & y^s & \forall s \in V \quad (5)
\end{array}
\qquad (P^I_{RSP})
$$

Proposition 35
There is a one-to-one correspondence between the feasible solutions of (P^I_{RSP}) *and the optimal solutions of* I *(considering RSP as an optimization problem for* MIN*).*

Proof. Let us define $x' \in \{0,1\}^m$: $\forall a \in \mathcal{A}, x'_a = \max_{s\in\mathcal{S}} x^s_a$. As $\sum_{a\in\delta^+(v)} x^s_a \leq 1$ for all $s, v \in \mathcal{S}$ (constraint (1)) and $x^u_a \geq x^s_a \ \forall a \in \delta^+(u) \ \forall u, s \in \mathcal{S}$ (constraint (2)), since $x^s \in \{0,1\}^m$, we know that $supp_G(x') = \{a \in A | x'_a > 0\}$ is a 0-anti-arborescence. Reciprocally, from a 0-anti-arborescence $\bar{A}$ we can generate $\bar{x}^s$ for all $s \in \mathcal{S}$. Indeed, let us call p_s the unique path in $\bar{A}$ from s to 0. Then for all $a \in \mathcal{A}$, $\bar{x}^s_a = 1$ if $a \in p_s$ and 0 otherwise. Then the constraints (1) and (2) are satisfied. Then constraints (1) and (2) define a pair of policies for MIN and MAX, which is a solution of the RSP instance with termination inevitable.

Finally, once the x^s are defined, from constraint (5), y^s is exactly the cost of the path defined by x^s from s to 0. Thus, if constraints (3) and (4) are verified, we can define Π_{MIN}, Π_{MAX} such that Π_{MAX} is a best response to Π_{MIN}, and Π_{MAX} have minimum cost. Thus, Π_{MIN} is an optimal solution of I.

□

We also give another integer linear programming formulation for RSP, whose idea comes from Jannick Matuschke. The idea is to define variables $y \in \mathbb{R}^n$ (n is still the total number of states) that represent potentials. We also define variables $z \in \{0,1\}^{|\mathcal{A}_{MAX}|}$, that represent the choice of actions for MAX: for all $a \in \mathcal{A}_{MAX}$, $z_a = 1$ if MAX chooses a, and 0 otherwise. Once again, we 'mimic' the dual of the linear programming formulation for the deterministic shortest path problem. Let us consider the following linear programming formulation, where M is a constant to be defined:

$$
\begin{array}{lllll}
\max & \sum_{v\in V_{MIN}} y_v & & & (1) \\
& y^v - y^u & \leq & c_a \qquad \forall (u,v) = a \in \mathcal{A}_{MIN} & (2) \\
& y^v - y^u & \leq & z_a c_a + (1-z_a)M \quad \forall (u,v) = a \in \mathcal{A}_{MAX} & (3) \\
& \sum_{\substack{a=(u,v)\\ u\in V_{MAX}}} z_a & = & 1 & (4) \qquad (P^I_{RSP\_JM}) \\
& y^t & = & 0 & (5) \\
& y & \in & \mathbb{R}^n & (6)
\end{array}
$$

Proposition 36
Let (y^*, z^*) *be an optimal solution of* $(P^I_{RSP\_JM})$*. Then* y^* *are optimal potentials: there exists* $\Pi^*_{MIN} \in \Sigma_{MIN}$ *an optimal policy for* MIN *such that* $y^{\Pi^*_{MIN}} = y^*$*.*

Proof. Once z^* is fixed, the strategy for MAX is set. For all $(u,v) = a \in \mathcal{A}_{MAX}$ chosen by MAX, constraints (2) are in fact equalities, and are true for the action not chosen by MAX (for which $z_a = 0$), if we take M sufficiently large (such a M is easy to find by solving both a shortest path and a longest path on the RSP instance, and taking the

difference between the two). The problem becomes then a 'classic' deterministic shortest path problem for MIN, whose solution is a 0-anti-arborescence. □

The same kind of approach can be generalize to find an integer linear programming formulation for SSPG with termination inevitable. The main idea is the same: to insure that we have a deterministic strategy profile and then thanks to the dual of the linear programming formulation of SSP, insure the optimality of this pair of strategies.

Proposition 37
Let $I = (\mathcal{S}_{MIN}, \mathcal{S}_{MAX}, \mathcal{A}_{MIN}, \mathcal{A}_{MAX}, J, P, c)$ be an instance of SSPG, and $G_I = (V, A)$ his graphic representation. Let us consider the following linear program with $\mathcal{A}^-(v) = \{a \in \mathcal{A} | P(a, v) > 0\}$.

$$
\begin{array}{lcll}
(J-P)^T x^s & = & \mathbb{1}_s & \forall s \in \mathcal{S} \quad (1) \\
y^v - y^u & \leq & c_a & \forall u \in \mathcal{S}_{MIN}, \forall v \in \mathcal{S}, \forall a \in \mathcal{A}(u) \cap \mathcal{A}^-(v) \quad (2) \\
y^v - y^u & \geq & c_a & \forall u \in \mathcal{S}_{MAX}, \forall v \in \mathcal{S}, \forall a \in \mathcal{A}(u) \cap \mathcal{A}^-(v) \quad (3) \\
x^s_a & \leq & M z_a & \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \quad (4) \\
\sum\limits_{a \in \delta^+(s)} z_a & = & 1 & \forall s \in \mathcal{S} \quad (5) \\
c^T x^s & = & y^s & \forall s \in \mathcal{S} \quad (6)
\end{array}
\qquad (P^I_{SSPG})
$$

Then there is a one-to-one correspondence between the feasible solutions of P^I_{SSPG} and the optimal solutions of I.

Proof. The proof is quite equivalent as the proof of proposition 35. The constraints (1), (4) and (5) insure that $\cup_{s \in \mathcal{S}} x^s$ induces a deterministic strategy profile. Indeed, the variables z over the actions define the choices of MIN and MAX (an action a chosen by one of the player is such that $z_a = 1$, and 0 otherwise. Then constraint (6) defines exactly the variables y from the flux vectors $(x^s)_{s \in \mathcal{S}}$, and the constraints (2) and (3) insure the optimality of the solution. □

We find a integer linear programming formulation for both RSP with termination inevitable and SSPG with termination inevitable. However, these formulation are not integral, even in the simpler case of RSP with termination inevitable.

Proposition 38
Linear relaxation of (P^I_{RSP}) is not integral.

Proof. Let us consider the following instance of RSP with termination inevitable (in fact, a shortest path instance as there is no MAX player: in particular a Nash equilibrium is a shortest path arborescence) (figure 3.18).

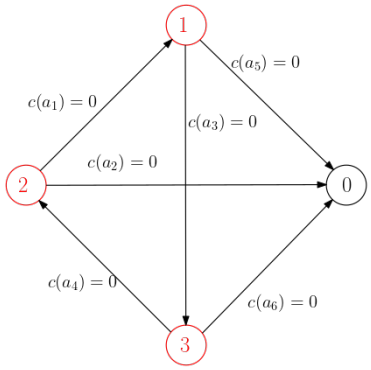


Figure 3.18 – Example showing that the extreme points of the linear relaxation of (P^I_{RSP}) are not integer

As we have only 0 cost edges, any 0-anti-arborescence is an optimal solution of the RSP with termination inevitable (seen as an optimization problem for MIN).

In order to prove that the extreme points of the linear relaxation of (P^I_{RSP}) are not integer, we show that there is a weight function over (P^I_{RSP}) and a fractional solution of better weight that the weight of any feasible integer solution.

Let us define $w^1, w^2, w^3 \in \mathbb{Z}^m$ and the objective function $\sum_{s\in\{1,2,3\}} w^s x^s$:

- $w^1 = (100, 0, 0, 0, 1, 100)$
- $w^2 = (0, 1, 0, 100, 100, 0)$
- $w^3 = (0, 100, 100, 0, 0, 1)$

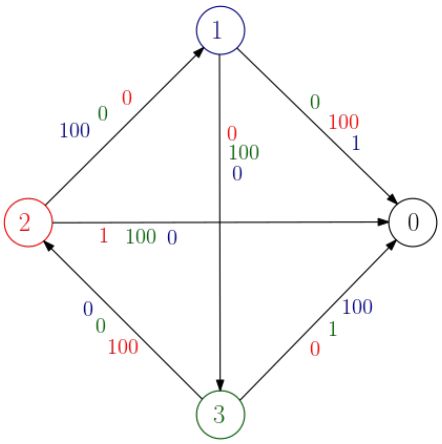


Figure 3.19 – Representation of w^1 (in blue), w^2 (in red) and w^3 (in green)

The solution $x^1 = (0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0)$, $x^2 = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 0, \frac{1}{2})$, $x^3 = (\frac{1}{2}, 0, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, and $y = (0, 0, 0, 0)$ is a solution to the linear relaxation (figure 3.20).

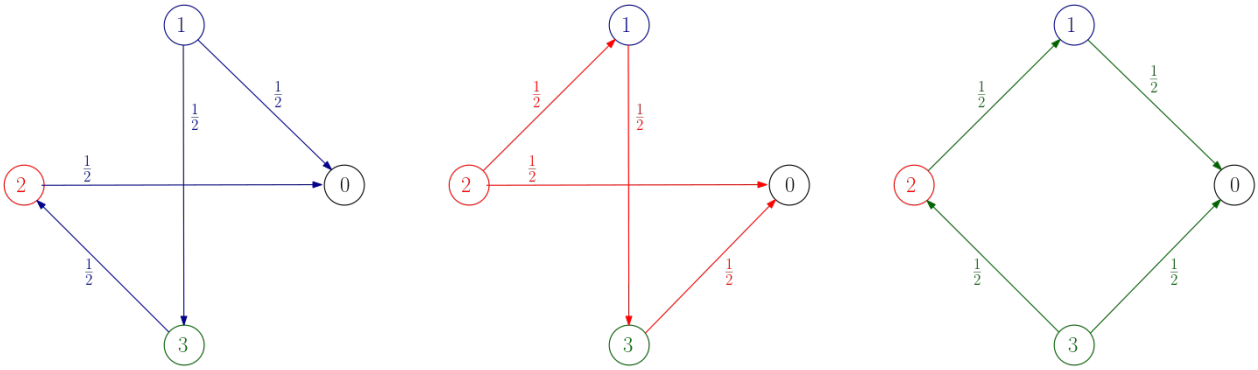


Figure 3.20 – From left to right: representation of x^1, x^2 and x^3

This solution has weight $\frac{1}{2} + \frac{1}{2} + \frac{1}{2} = \frac{3}{2}$. However, the optimal integer solution is the following (figure 3.21):

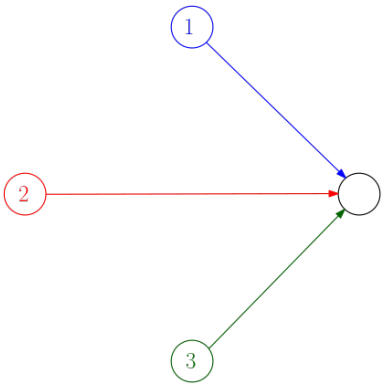


Figure 3.21 – The (unique) optimal integer solution

This solution corresponds to $x^1 = (0, 0, 0, 0, 1, 0)$, $x^2 = (0, 1, 0, 0, 0, 0)$, $x^3 = (0, 0, 0, 0, 0, 1)$, $y = (0, 0, 0, 0)$. Indeed, as we are looking for an integral solution, the solution is an 0-anti-arborescence, and it is easy to see that any other 0-anti-arborescence induces a cost greater than 100. The value of this solution is 1+1+1 = 3. Thus, there exists a fractional solution of better weight than any integer solution, which proves the proposition.

□

Now we know that the extreme points of the polyhedron of (P^I_{RSP}) are not integer. However it could be interesting to study the convex hull of the integer solutions of this polyhedron. We have started to study this integer hull with tools like Porta on examples

and the results are not encouraging to carry on in this direction. Indeed, we strongly believe that, even thought we did not formalize proofs yet, that this linear programming formulation encapsulates NP-hard problems. We also could dig into alternative directions, like checking integer non emptiness only.

It is interesting to note that our counter-example does not involve nodes controlled by MAX. This means that it is an instance of deterministic shortest path. We can conclude that the following is not a linear programming formulation for the shortest path

$$\begin{array}{lcll} x^s(\delta^+(v)) - x^s(\delta^-(v)) & = & \begin{cases} 1 & \text{if } v = s \text{ and } v \neq 0 \\ 0 & \text{otherwise} \end{cases} & \forall s \in V \\ x^u_a - x^s_a & \geq & 0 & \forall a = (u,v), u, s \in \mathcal{S}, v \in N(u) \\ y^u - y^v & \leq & c_a & \forall a = (u,v) \in A, u \in V \\ \sum_{a \in A} x^s_a c_a & = & y^s & \forall s \in V \end{array} \tag{3.1}$$

The previous counter example show that (P^I_{RSP}) is not a LP-formulation of the deterministic shortest path problem.

3.4 Instances of RSP ∩ SSG

We now restrict our attention to even simpler instances in our quest for LP formulation, that is, instances of RSP with termination inevitable which are also instances of stopping Simple Stochastic Games. We call the set of these instances stopping RSP∩SSG. An instance of stopping RSP∩SSG can be seen as an instance of stopping SSG where there is no average node.

Formally, an instance of stopping RSP∩SSG is a graph $G = ((V = V_{MIN} \cup V_{MAX}), A)$ where V_{MIN} is a finite set of nodes controlled by MIN and V_{MAX} is finite set of nodes controlled by MAX. A is a set of arcs such that for all $v \in V$, $|\delta^+(v)| \leq 2$. Just like for SSG, a (deterministic and stationary) policy for MIN (resp. for MAX) is a mapping $\Pi_{MIN} : V_{MIN} \to A$ (resp. $\Pi_{MAX} : V_{MAX} \to A$) such that $\Pi_{MIN}(s) \in \delta^+(s)$ for all $s \in V_{MIN}$ (resp. $\Pi_{MAX}(s) \in \delta^+(s)$ for all $s \in V_{MAX}$). There are two special sink nodes: 0_{MIN} and 0_{MAX}. We focus on instances where G is acyclic, so a pair of policies for MIN and MAX define an disjoint union of a 0_{MIN}-anti-arborescence and a 0_{MAX}-anti-arborescence. Thus, each node $v \in V$ is connected exclusively to 0_{MIN} or 0_{MAX}. From a start node $v_0 \in V$ and a pair of policies $\Pi = (\Pi_{MIN}, \Pi_{MAX})$, we say that MIN wins if v_0 is connected to 0_{MIN} in the corresponding disjoint union of anti-arborescences, and that MAX wins if s_0 is connected to 0_{MAX} in this union.

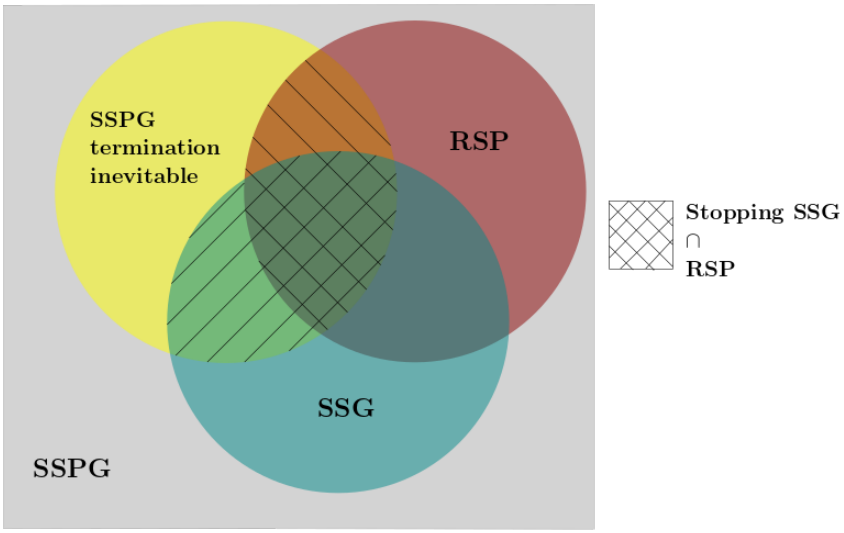


Figure 3.22 – instances of Stopping RSP ∩ RSP with inevitable termination

We give a simple algorithm to solve such instances: let G be an instance of stopping RSP∩SSG and let $\mathcal{O}$ be a topological order over the nodes of G (which exists since G is acyclic). Solving such instances boils down to *labeling* the nodes with 0 or 1, depending on if from it, MIN wins whatever the strategy of MAX is (then we label with 0 and with 1 otherwise). At the beginning of the algorithm, we can mark 0_{MIN} with 0 and 0_{MAX} with 1. As 0_{MIN} and 0_{MAX} are sinks, they can be assumed to be at the end of any topological order. Let us consider the nodes one by one, in opposite order of $\mathcal{O}$. As I is an instance of SSG, each nodes has at most 2 neighbors, and can be either min or max. We label the current node according the following possible cases, according to the labels of his neighbors:

- if the current node has only one neighbor, we label the current node with the label of its neighbor
- if the current node has exactly two neighbors, then we distinguish two cases:
  - the current node is min: if at least one of its neighbor's label is 0, we label the current node 0, and 1 otherwise
  - the current node is max: if both its neighbor's labels are 0, we label the current node 0, and 1 otherwise

Proposition 39
Once all the nodes are labeled, we can conclude who wins according to the label of the start node: if it is labeled 0*,* MIN *wins and* MAX *wins otherwise (then the label is* 1*).*

Proof. We prove this proposition by induction on the number of nodes. If there are only two nodes 0_{MIN} and 0_{MAX}, as we label 0_{MIN} with 0 and 0_{MAX} with 1, if $v_0 = 0_{MIN}$ then min wins and max wins if $v_0 = 0_{MAX}$ by definition.

Let us assume that the proposition is true for any graph composed of $n-1$ nodes. Let us consider a graph G_n composed of n nodes. As G_n is acyclic, we can define a topological order $\mathcal{O}$ over the nodes of G_n. Let v be the first node in $\mathcal{O}$, meaning that v has no incoming arc (and at most 2 outgoing arc). By induction, we can label the nodes of $G_n \setminus \{v\}$ such that from the nodes labeled 0, MIN wins and MAX wins from the nodes labeled 1. Then the proposition is true for all the nodes except v. Let us label v according to the different cases described above.

Let us assume that $v \in V_{MIN}$ (the same kind of proof can be made if $v \in V_{MAX}$).

If v has only one neighbor v', then if v' is labeled 0, v is labeled with 0 and from v, MIN wins as v' is connected to 0_{MIN}. Similarly, if v' is labeled 1, v is labeled with 1 and from v, MAX wins since v' is connected to 0_{MAX}.

If v has two neighbors then if at least of its neighbors v' is labeled 0, we set $\Pi_{MIN}(v) = (v, v')$ and from v, MIN wins. If all the neighbors of v are labeled 1, then we set $\Pi_{MIN}(s)$ to any of its outgoing arc and MAX wins from v. □

Let us consider the following linear program, where $v_0 \in V$ is the start node. We denote by $N^1 \subseteq V$ the subset of V of nodes which have only 1 neighbor, and $N^2 \subseteq V$ the subset of V of nodes which have exactly 2 neighbors.

$$
\begin{array}{lcll}
\max y_{v_0} & & & \\
y_i & \leq & y_j + y_k & \forall i \in V_{MAX} \cap N^2, \forall (i,j),(i,k) \in A \quad (1) \\
y_i & \leq & y_j & \forall i \in (V_{MAX} \cap N^1) \cup V_{MIN}, \forall (i,j) \in A \quad (2) \qquad (P^I_{RSP \cap SSG}) \\
y_{0_{MIN}} & = & 0 & (3) \\
y_{0_{MAX}} & = & 1 & (4)
\end{array}
$$

In this linear program, y_i represent the label of node i. The inequalities (1) and (2) force to label a node in V_{MAX} to 0 if its neighbors are labeled 0 and to label a node in V_{MIN} to 0 if at least one of its neighbor is 0. The objective function force to label a *max* node to 1 if at least one of its neighbor is 1, and a *min* node to be labeled to 1 if both of its neighbors is 1. Thus, if the optimal value of $P_{RSP \cap SSG}$ is 0, MIN wins from v_0.

Note that if the optimal solution to the linear relaxation of $P_{RSP \cap SSG}$ is positive, then there exists a $0-1$ solution with $y_s = 1$. Indeed, let y be an optimal solution with $y_s = \epsilon > 0$. Let $\bar{y}$ be a $0-1$ vector over the nodes defined by $\forall i \in V, \bar{y}(i) = \lceil y_i \rceil$. By definition of the constraints of $P_{RSP \cap SSG}$, $\bar{y}$ is a feasible solution of $P_{RSP \cap SSG}$, of value 1. Thus, if the optimal value of $P_{RSP \cap SSG}$ is positive, we know that there exist a solution of value 1, and MAX is winning from s.

We think that the same logic might be applied to solve SSG. Indeed, even if we do not have a LP-formulation, we have defined a linear program, for which we can decide which player wins according to the optimal value, and a threshold (here the threshold is 0: if we have a optimal solution of positive cost, then we know that MAX wins). Even if we did not formalize it, we could define the same kind of linear program for SSG, and decide who wins according to whether or not the optimal solution has a value greater than $\frac{1}{2}$ (in this case the threshold would be $\frac{1}{2}$).

Finally, the instance of $RSP \cap SSG$ are easy to solve. What happen if we introduce stochasticity in this problem ? If we re-introduce the average nodes, we come back to SSG. What if the transitions for player MAX were stochastic, with possibly different probabilities than $\frac{1}{2}$? This is MAXPROB, so let us go back to this problem.

3.5 The MAXPROB Problem

As seen in chapter 1, MAXPROB is a special case of SSP. There are only two actions a_{MIN} and a_{MAX} that can lead to 0 with probability 1. These actions are available in only two states: respectively 0_{MIN} and 0_{MAX}. The costs of all actions are 0, except for a_{MAX} which has cost 1. Thus, minimizing the expected cost of a proper policy is equivalent to maximizing the probability to reach state 0_{MIN}, as the only action that has non zero cost is a_{MAX}, and a_{MAX} is taken at most once.

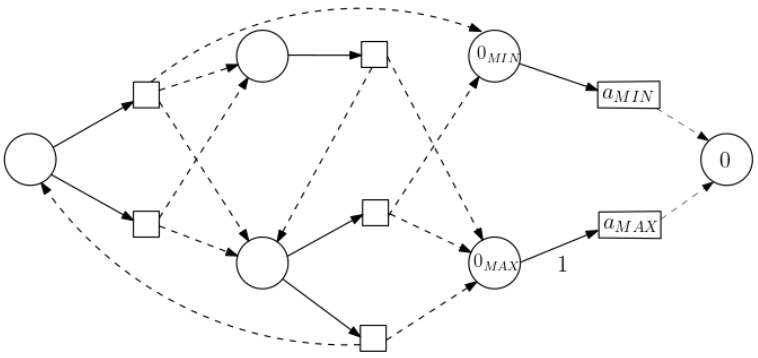


Figure 3.23 – Example of MAXPROB. All the actions have cost 0 except for a_{MAX} which has cost 1

We refer to the first chapter for a formal definition of MAXPROB. As MAXPROB is a special case of SSP, there exists a LP formulation so MAXPROB can be solved in weakly polynomial time.

Proposition 40
If there exists a strongly polynomial algorithm to solve an instance of MAXPROB without transition cycle, then there exists a strongly polynomial algorithm to solve general instances.

Proof. Let $I = (\mathcal{S}, \mathcal{A}, J, P, c)$ be an instance of MAXPROB with one only transition cycle $\tilde{x} \geq 0$ (we can assume that there is only one transition cycle without loss of generality), and G_I his graphical representation (see chapter 1 for more details on graphical representation of SSP). We remind the readers that a transition cycle is a solution $x \neq 0$ to $(J - P)^T x = 0$. We know that one can find a transition cycle in strongly polynomial time (see chapter 1 for more details).

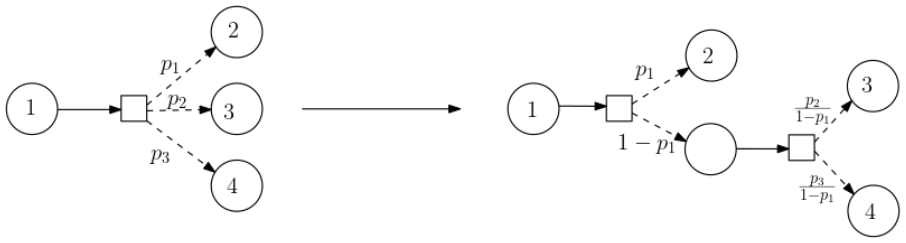
As all the costs are 0 (except for a_{MAX}), the cost of a transition cycle $\tilde{x}$ is $c^T\tilde{x} = 0$, as $\tilde{x}(a_{MAX}) = 0$ for all transition cycle $\tilde{x}$.

Let $\tilde{\mathcal{S}} = \{s \in \mathcal{S} | P^T\tilde{x} > 0\}$ be the subset of reachable states if we perform actions a such that $\tilde{x}(a) > 0$. Let $\tilde{G}_I$ be the graph in which all nodes corresponding to the states of $\tilde{\mathcal{S}}$ have been contracted in one node $s_{\tilde{x}}$ and $\tilde{I}$ the corresponding SSP instance. Then $\tilde{G}_I$ has no transition cycle. Let us assume that we can find an optimal (deterministic and stationary) policy $\tilde{\Pi}$ of $\tilde{I}$ in strongly polynomial time. Let us call $s' \in \mathcal{S}$ the unique state such that $J(\tilde{\Pi}(s_{\tilde{x}}, s') = 1$, Let us now define Π^*, a policy for instance I, such that:

$$\Pi^*(s) = \begin{cases} \tilde{\Pi}(s) \text{ if } s \in \mathcal{S} \setminus \tilde{\mathcal{S}} \\ a \text{ such that } \tilde{x}(a) > 0 \text{ if } s \in \tilde{S} \setminus \{s'\} \\ \tilde{\Pi}(s_{\tilde{x}}) \text{ if } s = s' \end{cases}$$

As $\tilde{\Pi}$ is proper, and by definition of a transition cycle we know that Π^* is proper. Then, as the cost of the transition cycle is 0, we have $J^{\Pi^*}(s) = J^{\tilde{\Pi}}(s_{\tilde{x}})$ for all $s \in \tilde{S}$. We conclude that Π^* is an optimal policy for I, which proves the proposition. □

We tried to answer the question of the existence of a strongly polynomial algorithm. For this quest, we can assume w.l.o.g that each action can lead to at most two states. We can easily generate an equivalent model with at most n^2 states and nm actions by repeating the following transformation:



In the LP-formulation of SSP, it results that we can assume there are at most three variables per inequality. With the help of László A. Végh, who is a specialist of strongly polynomial algorithms in combinatorial optimization, we tried to understand first to extend the results of [53] on linear programs with two variables per inequality. Even if we could not manage to extend the idea to find a strongly polynomial algorithm to solve MAXPROB. Our investigations were productive: László A. Végh found an error in the paper, and later corrected it. We understood from these first investigations that the quest of strongly polynomial time algorithm for MAXPROB was way too challenging and this is why we focused our research on games extensions of SSP where we though we had a better grip (even though as describe above, the open problems are very hard, but at least we could give some hopefully interesting directions).

3.6 Conclusion and Perspectives

In this chapter we have studied variations of the Stochastic Shortest Path Problem, and in particular, we have to studied the existence of LP-formulations for some game extensions. The most natural 2-player stochastic game extension of the SSP is the Stochastic Shortest Path Game and even though we could not find a LP formulation for the problem, we could formulate what we believe is the first ILP formulation of the problem, which opens new ways to look at the problem from a polyhedral perspective and this is we believe a first contribution (it could provide an alternative to strategy iteration for instance which could be interesting to evaluate computationally). Even though we have started to investigate the integer hull exploiting tools like Porta, our first experiments suggested that hard inequalities would be needed and even though we did not formalize the proof yet, we strongly believe that the formulation encapsulate NP-hard problems which refrained us from pursuing in this direction. Therefore we restricted attention to simpler sub-problems for which existence of LP-formulation was still open. Our main contribution is a simple LP model that has the property that if the solution is positive, then it has an integer solution of value one (which provides a simple and original LP approach to answer the problem). We believe that this offers an interesting direction for further investigation for the SSG problem. Maybe no LP-formulation exist but there might still be a LP model that has the property that $z_{LP} > \frac{1}{2}$ if and only if there exists a deterministic strategy that reaches 0 with probability more that $\frac{1}{2}$.

By adding stochasticity to this last special case, we come back to a well known problem we have encountered in the first chapter: MAXPROB. For this last problem, we prove that we can focus only on instances without transition cycle. It is interesting to see that even if MAXPROB is a special case of SSP, we do not know if solving MAXPROB is easier than solving SSP.

Conclusion, Limites et Perspectives de Recherche

Dans ce manuscrit de thèse, nous nous sommes intéressés au problème de l'optimisation de stratégie dans le sport et plus particulièrement à l'optimisation de la stratégie des joueurs de golf. Comme nous l'avons vu, les joueurs professionnels peuvent participer à deux types de compétitions : les compétitions en Stroke Play et les compétitions en Match Play. Durant une compétition en Stroke Play, tous les joueurs en lice jouent les dix-huit trous du parcours, le gagnant est le joueur ayant obtenu le plus petit score à l'issu de celui-ci. Concernant les compétitions en Match Play, deux joueurs s'affrontent sur chaque trou se départageant un point par trou, le vainqueur étant le joueur ayant le plus de points à la fin du parcours.

Dans le premier chapitre, nous avons redéfini et élargi le cadre d'étude du plus court chemin stochastique. Ce problème avait été défini et largement étudié dans un cadre assez éloigné de notre communauté scientifique. Nous avons considéré le PCCS d'un point de vue polyédral, plus proche de nos domaines de compétences. Nous avons réussi à alléger les hypothèses d'existence d'une solution optimale (les cycles de transition de coût nul sont désormais autorisés) et prouvé que les algorithmes classiques de résolution préexistants (Value Iteration, Policy Iteration) convergeaient bien dans notre nouveau cadre d'étude. De plus, notre approche nous a permis d'appliquer des algorithmes, bien connus de notre communauté, au problème du plus court chemin stochastique. Cela nous a mené à définir et à prouver la convergence d'un nouvel algorithme de résolution basé sur l'algorithme du primal-dual qui peut s'apparenter à la version stochastique de l'algorithme de Dijkstra pour les plus courts chemins déterministes. Même si nous n'avons pas obtenu de résultats importants concernant la complexité théorique ou pratique de cet algorithme, cela nous a conduit à étudier un sous-problème connu de la communauté Intelligence Artificielle : MAXPROB. Ce problème est en fait un cas particulier de PCCS qui ne pouvait pas être résolu tel quel avec les algorithmes classiques de résolution des PCCS, notamment à cause de la présence potentielle de cycles de transition de coût nul. Avec l'allègement des conditions d'existence de solutions optimales, nous avons pu faire entrer MAXPROB dans le cadre classique d'étude. Malheureusement la complexité temporelle des algorithmes de résolution connus demeure exponentielle dans

le pire cas, même dans ce cas particulier. Finalement, nous n'avons pas pu déterminer si MAXPROB était plus simple que le cas général des PCCS ou si il renfermait sa complexité.

La compréhension profonde de ce dernier problème aurait été doublement intéressante. D'abord d'un point de vue algorithmique : MAXPROB apparait naturellement quand on applique l'algorithme primal-dual au cas particulier du programme linéaire formulant les PCCS. La résolution efficace de ce problème pourrait potentiellement permettre de résoudre efficacement les PCCS. Sachant que l'existence d'algorithmes fortement polynomiaux pour résoudre les PCCS est une question ouverte très importante dans le domaine, une des pistes de recherche serait donc de trouver d'abord un algorithme fortement polynomial pour MAXPROB. D'autre part, ce problème étant un cas particulier des PCCS (les coûts ne valent que 0 et 1, on peut supposer qu'il n'y a pas de cycle de transition...), il est intéressant de l'étudier comme un problème 'élémentaire' afin de connaître l'essence de la difficulté des PCCS.

Dans le deuxième chapitre, nous avons détaillé la modélisation du "problème du golfeur" en PCCS. Après avoir défini les états, les actions et le coût de ces actions de manière assez naturelle, il nous a fallu définir la matrice de transition qui relate à la fois de la topologie du terrain mais aussi du niveau intrinsèque du joueur. Grâce à la base de données Américaine Shotlink [86], nous avons eu accès à des informations relatives aux coups des joueurs professionnels durant des compétitions internationales, comme par exemple la position de la balle avant et après le coup ou le type de terrain (fairway, rough, bunkers, green...) sur lequel le joueur a tiré. Ces données nous ont permis de créer des statistiques intrinsèques au joueur qui relatent de la déviation de ses coups. Nous avons ensuite simulé ces statistiques sur le trou considéré afin de créer la matrice de transition. Une fois l'instance de PCCS créée à partir des données du terrain et du joueur, nous avons résolu à l'optimal cette instance afin de déterminer la stratégie optimale du joueur. Les joueurs que nous avons considéré étant professionnels, nous avons supposé qu'ils jouaient leur stratégie optimale, ce qui nous a permis de créer un 'clone numérique' du joueur. En simulant ce clone sur les différents trous d'un parcours, nous pouvons établir une distribution du nombre de coups que le joueur jouera pour mettre la balle dans les dix-huit trous du parcours s'il s'agit d'une compétition en Stroke Play ou une probabilité de victoire face à son adversaire s'il s'agit d'une compétition en Match Play.

Tout au long de cette modélisation, nous avons fait des hypothèses que l'on peut classer en deux catégories : celles concernant la simulation du jeu de Golf et celles liées aux statistiques du joueur. Pour les premières, nous avons du faire des simplifications : on considère que le trou est plat, on ne prend pas en compte le roulement de la balle et la gestion des interactions entre la balle et les obstacles est simpliste. Il est clair que pour avoir un simulateur plus réaliste il serait intéressant d'intégrer (entre autres) tous ces aspects. Cependant ces hypothèses ne remettent pas en cause notre démarche globale et ces améliorations peuvent être apportées indépendamment de la partie optimisation du programme. Concernant les hypothèses faites afin de construire le profil théorique des joueurs, elles ont principalement été faites à l'aide de connaissances 'métier' des golfeurs.

L'intention des joueurs de viser le trou quand ils ne sont pas sur le tee et que le trou est atteignable est une hypothèse qui est discutable, la linéarité de la dépendance entre la déviation des coups et la distance de celui-ci, ainsi que le facteur constant reliant les déviations sur le fairway et le rough ou les bunkers sont des hypothèses qui nous ont parues naturelles à considérer dans un premier temps et peuvent sûrement être améliorées afin d'être plus réalistes. Une autre limite de notre modèle concerne la définition des paramètres nécessaires au traitement des données. Nous avons défini des paramètres afin de supprimer des données quand elles étaient aberrantes. Au niveau de la validation des résultats, notre principale difficulté a été de trouver une méthode satisfaisante sachant que le nombre de réalisations effectives était très limité. Ainsi, il est difficile de savoir à partir d'un faible échantillon si la viabilité du modèle est contestable ou si il s'agit uniquement de variance statistique. Ce manque de réalisation nous a également empêché d'utiliser des méthodes de validations statistiques plus classiques, comme le test du chi-2. Nous avons dû créer une méthode de validation ad-hoc qui a permis de déterminer si la réalisation que nous avons aurait pu être facilement simulée par notre modèle. Cette validation est bien sûr perfectible et des modèles de validations plus fins pourraient être développés afin d'avoir des critères plus objectifs relatant de la qualité du modèle. Concernant les résultats, on peut constater qu'on peut facilement simuler des scénarios probables, mais on ne pourra pas (ou presque pas) simuler des scénarios rares même avec une faible probabilité. Ce dernier point est une limite importante du modèle : dans un but de prédiction, il est dommage de ne pouvoir anticiper que les scénarios très probables.

Dans le troisième et dernier chapitre, nous nous sommes intéressés à une autre modélisation possible pour les compétitions en Match Play. Sachant que dans ce mode deux joueurs s'affrontent sur chaque trou avec des buts antagonistes, il parait naturel de vouloir modéliser ce problème comme un jeu de plus court chemin stochastique, qui est l'extension naturelle du PCCS à deux joueurs. Même si, au vu des expériences computationnelles du chapitre 2, la construction et la résolution de telles instances ne pourraient être faites en un temps raisonnable, nous nous sommes attelés à étudier théoriquement les jeux stochastiques à deux joueurs. Nous avons étudié les jeux de plus court chemin stochastique afin d'essayer de trouver des formulations programmation linéaire décrivant le problème, comme il en existe pour le PCCS. Le problème initial étant complexe, notre démarche a été de s'intéresser à des cas particuliers potentiellement plus simples. Nous avons donc étudié les jeux stochastiques simples, ainsi que le problème de plus court chemin robuste. Nous avons réussi à trouver une formulation linéaire en nombres entiers pour le plus court chemin robuste, qui peut se généraliser aux JPCCS.

Ce dernier chapitre ne comporte pas de résultats théoriques amajeurs. Il présente cependant un état de l'art sur ce que l'on connaît actuellement sur les JPCCS, ainsi que quelques résultats préliminaires. Nous espérons que cela pourra donner des idées de pistes de réflexion pour des recherches futures. Une des pistes qui nous apparait intéressante concerne le programme linéaire que nous avons défini pour les instances qui sont à la fois des instances de jeux stochastiques simples et de plus courts chemins robustes. Pour ces instances, nous avons défini un programme linéaire qui ne caractérise

pas directement les solutions, mais telle que s'il existe une solution de coût strictement positif, alors il existe une solution de coût 1. Dans ce cas, nous pouvons définir une solution optimale à partir de la solution de coût 1. Nous pensons que le même type de raisonnement pourrait se généraliser aux instances de jeux stochastiques simples.

En plus d'apports théoriques, cette thèse comporte un aspect applicatif important. Le modèle décrit théoriquement dans le chapitre 2 pour formaliser le problème du golfeur en un PPCS a été implémenté en C++ et en R 'from scratch'. La partie de traitement des données, ainsi que la validation statistique a été faite en R, alors que la construction du modèle et l'optimisation a été faite en C++. L'intégralité du code est publique, afin que les résultats que nous avons trouvé soient reproductibles. Nous nous sommes également risqués à appliquer notre modèle de prédiction à la Ryder Cup 2018 qui a eu lieu en France. Avant la compétition nous avons créé un site [70] qui rassemble nos prévisions. Tout au long de la compétition, nous avons mis à jour le site afin de donner des statistiques en temps réel. Le site est toujours disponible à l'adresse suivante : `http://www.golfoptimization.com/`. Les prévisions ont pu légèrement évoluer au vu des modifications que nous avons apporté au modèle depuis un an.

Annex: The Shotlink Database

Tour Descrip	Year	Tourn #	Player #	Course #	Permanent T	Player First	Player Last N	Round	Tournament	Course Name	Hole	Hole Score	Par Value	Yardage	Shot	Shot Type(S	# of Strokes	From Locatic	From Locatic	To Location(S	To Location(I	Distance	Distance to H	In the Hole F	Around the (	1st Putt Flag	Distance to H	Time	Lie	Elevation	Slope	X Coordinate	Y Coordinate	Z Coordinati	Distance fro	Distance fro	Date	Left/Right	Strokes Gair	Strokes Gair	Recovery Shot
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	1	5	4	436	1	S	1	Tee Box		Primary Roug	Right Rough	10327	15804	N	N	N	5496	1241	Good	With	Level	10,734.3430	9,531.0870	84.1040	1104	645	10/13/2016	R	-.091	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	1	5	4	436	2	S	1	Primary Roug	Right Rough	Green	Unmapped	4988	5496	N	N	N	530	1246	N/A	With	Level	10,839.1510	9,933.2770	90.4280	245	159	10/13/2016	R	.076	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	1	5	4	436	3	S	1	Green	Unmapped	Green	Unmapped	501	530	N	N	Y	29	1250	Good	With	Level	10,836.8030	9,974.9770	91.1650	12	483	10/13/2016	L	.070	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	1	5	4	436	4	S	1	Green	Unmapped	Green	Unmapped	59	29	N	N	N	27	1252	Good	With	Level	10,837.8700	9,979.7810	91.1710	0	531	10/13/2016	L	-.994	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	1	5	4	436	5	S	1	Green	Unmapped			27	27	Y	N	N	0	1252	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.017	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	2	4	3	240	1	S	1	Tee Box		Primary Roug	Right Rough	7440	7200	N	Y	N	377	1257	Good	With	Level	10,220.1910	10,540.5780	100.6370	302	673	10/13/2016		-.204	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	2	4	3	240	2	S	1	Primary Roug	Right Rough	Green	Unmapped	288	377	N	N	N	100	1300	N/A	With	Level	10,220.5770	10,516.5450	100.2310	99	398	10/13/2016	R	-.228	Around the G	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	2	4	3	240	3	S	1	Green	Unmapped	Green	Unmapped	125	100	N	N	Y	23	1303	Good	With	Level	10,215.9240	10,507.2330	100.3310	20	332	10/13/2016	R	-.479	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	2	4	3	240	4	S	1	Green	Unmapped			23	23	Y	N	N	0	1303	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.007	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	3	3	4	424	1	S	1	Tee Box		Primary Roug	Right Rough	10891	15552	N	N	N	4913	1308	Good	With	Level	10,128.2850	11,095.6910	116.0430	558	114	10/13/2016	R	-.055	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	3	3	4	424	2	S	1	Primary Roug	Right Rough	Green	Unmapped	4908	4913	N	N	N	106	1313	N/A	With	Level	9,986.7990	11,479.1970	129.9760	106	780	10/13/2016	L	.555	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	3	3	4	424	3	S	1	Green	Unmapped			106	106	Y	N	Y	0	1321	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.558	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	4	3	4	407	1	S	1	Tee Box		Fairway	Left Fairway	10257	14652	N	N	N	4413	1324	Good	With	Level	10,141.5150	12,433.9000	117.8220	38	456	10/13/2016	L	.156	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	4	3	4	407	2	S	1	Fairway	Left Fairway	Green	Unmapped	4392	4413	N	N	N	47	1329	Good	With	Level	10,065.8910	12,791.7670	131.3960	43	137	10/13/2016	L	.714	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	4	3	4	407	3	S	1	Green	Unmapped			47	47	Y	N	Y	0	1335	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.126	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	5	4	5	538	1	S	1	Tee Box		Fairway	Left Fairway	11838	19260	N	N	N	7117	1339	Good	With	Level	10,875.8120	12,561.1480	115.5010	232	8	10/13/2016	L	.464	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	5	4	5	538	2	S	1	Fairway	Left Fairway	Green	Unmapped	7393	7117	N	N	N	514	1349	Good	With	Level	11,419.9960	12,272.3860	123.4350	347	825	10/13/2016	L	.065	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	5	4	5	538	3	S	1	Green	Unmapped	Green	Unmapped	551	514	N	N	Y	33	1354	Good	With	Level	11,379.6940	12,250.4390	122.1670	25	383	10/13/2016	R	.047	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	5	4	5	538	4	S	1	Green	Unmapped			33	33	Y	N	N	0	1357	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.035	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	6	4	4	417	1	S	1	Tee Box		Primary Roug	Left Rough	11069	14868	N	N	N	3929	1359	Good	With	Level	11,276.1650	11,213.8320	105.4680	878	451	10/13/2016	L	-.023	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	6	4	4	417	2	S	1	Primary Roug	Left Rough	Green	Unmapped	4172	3929	N	N	N	246	1404	N/A	With	Level	11,104.2240	10,911.6670	102.1820	5	249	10/13/2016	L	.148	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	6	4	4	417	3	S	1	Green	Unmapped	Green	Unmapped	238	246	N	N	Y	9	1408	Good	With	Level	11,109.3850	10,930.7960	101.9950	5	179	10/13/2016	L	-.120	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	6	4	4	417	4	S	1	Green	Unmapped			9	9	Y	N	N	0	1408	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.001	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	7	3	3	212	1	S	1	Tee Box		Green	Unmapped	7148	7452	N	N	N	632	1415	Good	With	Level	11,272.2210	10,602.3020	101.5540	537	181	10/13/2016	L	-.037	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	7	3	3	212	2	S	1	Green	Unmapped	Green	Unmapped	648	632	N	N	Y	25	1421	Good	With	Level	11,285.8340	10,550.0650	101.9110	23	622	10/13/2016	R	.145	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	7	3	3	212	3	S	1	Green	Unmapped			25	25	Y	N	N	0	1424	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.011	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	8	4	4	360	1	S	1	Tee Box		Primary Roug	Left Rough	9215	12600	N	N	N	3184	1426	Good	With	Level	12,001.0200	9,936.9520	89.3350	548	65	10/13/2016	L	-.090	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	8	4	4	360	2	S	1	Primary Roug	Left Rough	Green	Unmapped	3484	3184	N	N	N	311	1433	N/A	With	Level	12,246.0260	9,783.0720	113.6510	148	326	10/13/2016	R	.021	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	8	4	4	360	3	S	1	Green	Unmapped	Green	Unmapped	261	311	N	N	Y	65	1437	Good	With	Level	12,229.3770	9,797.0510	113.0050	60	279	10/13/2016	L	-.348	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	8	4	4	360	4	S	1	Green	Unmapped			65	65	Y	N	N	0	1437	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.289	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	9	5	5	557	1	S	1	Tee Box		Primary Roug	Right Rough	10697	20340	N	N	N	10048	1441	Good	With	Level	11,603.9210	9,257.2680	82.1540	814	400	10/13/2016	R	-.739	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	9	5	5	557	2	S	1	Primary Roug	Right Rough	Fairway	Right Fairwa	2577	10048	N	N	N	7570	1448	N/A	With	Level	11,563.7000	9,046.3200	81.8420	86	352	10/13/2016	R	.211	Approach the	Yes
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	9	5	5	557	3	S	1	Fairway	Right Fairwa	Primary Roug	Right Rough	8077	7570	N	Y	N	622	1450	Good	With	Level	11,263.7020	8,443.7780	81.5180	418	757	10/13/2016		-.259	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	9	5	5	557	4	S	1	Primary Roug	Right Rough	Green	Unmapped	815	622	N	N	N	193	1454	N/A	With	Level	11,257.6650	8,511.4410	81.6560	94	145	10/13/2016	R	-.316	Around the G	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	9	5	5	557	5	S	1	Green	Unmapped			193	193	Y	N	Y	0	1456	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.810	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	10	3	4	422	1	S	1	Tee Box		Primary Roug	Left Rough	10022	15192	N	N	N	5453	1502	Good	With	Level	11,121.1510	9,406.6680	84.3570	968	331	10/13/2016	L	-.133	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	10	3	4	422	2	S	1	Primary Roug	Left Rough	Green	Unmapped	5492	5453	N	N	N	36	1506	N/A	With	Level	11,455.5930	9,719.0930	89.1880	0	679	10/13/2016	R	1.117	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	10	3	4	422	3	S	1	Green	Unmapped			36	36	Y	N	Y	0	1518	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.045	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	11	3	3	182	1	S	1	Tee Box		Fairway	Left Fairway	5039	6228	N	Y	N	1234	1520	Good	With	Level	11,338.9570	10,068.5560	86.3010	277	222	10/13/2016		-.517	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	11	3	3	182	2	S	1	Fairway	Left Fairway	Green	Unmapped	1258	1234	N	N	N	22	1525	Good	With	Level	11,410.2330	10,145.4200	87.9990	4	159	10/13/2016	L	.538	Around the G	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	11	3	3	182	3	S	1	Green	Unmapped			22	22	Y	N	Y	0	1525	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.006	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	12	4	4	391	1	S	1	Tee Box		Fairway	Left Fairway	10553	14220	N	N	N	3675	1531	Good	With	Level	11,495.1080	11,555.0040	107.3000	114	375	10/13/2016	L	.194	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	12	4	4	391	2	S	1	Fairway	Left Fairway	Green	Unmapped	3908	3675	N	N	N	230	1536	Good	With	Level	11,609.4170	11,859.8200	116.6560	17	359	10/13/2016	L	-.079	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	12	4	4	391	3	S	1	Green	Unmapped	Green	Unmapped	226	230	N	N	Y	11	1539	Good	With	Level	11,603.6500	11,841.9400	115.9590	12	383	10/13/2016	L	-.140	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	12	4	4	391	4	S	1	Green	Unmapped			11	11	Y	N	N	0	1540	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.001	Putting	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	13	4	4	458	1	S	1	Tee Box		Primary Roug	Left Rough	10341	16776	N	N	N	6489	1546	Good	With	Level	11,091.4610	11,314.6890	102.7550	1330	631	10/13/2016	L	-.122	Off the Tee	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	13	4	4	458	2	S	1	Primary Roug	Left Rough	Fairway	Right Fairwa	5355	6489	N	Y	N	1149	1551	N/A	With	Level	10,847.9750	10,940.7890	97.3750	80	124	10/13/2016		-.246	Approach the	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	13	4	4	458	3	S	1	Fairway	Right Fairwa	Green	Unmapped	1178	1149	N	N	N	25	1554	Good	With	Level	10,808.9890	10,850.7510	99.0290	3	748	10/13/2016	R	.516	Around the G	No
PGA TOUR	2017	10	1810	552	464	Phil	Mickelson	1	Safeway Ope	Silverado Res	13	4	4	458	4	S	1	Green	Unmapped			25	25	Y	N	Y	0	1558	Good	With	Level	.0000	.0000	.0000	0	0	10/13/2016		.011	Putting	No

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[2] R. Alterovitz, T. Simon, and K. Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In M. P. W. Burgard et al. (Eds.), editor, *Robotics: Science and Systems III (Proc. RSS 2007)*, pages 233–241, 2008.

[3] K. Arulkumaran, A. Cully, and J. Togelius. Alphastar: An evolutionary computation perspective. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 314–315, New York, NY, USA, 2019. Association for Computing Machinery.

[4] K. B. Athreya. Bootstrap of the mean in the infinite variance case. *Ann. Statist.*, 15(2):724–731, 06 1987.

[5] D. Avis and V. Chvátal. *Notes on Bland's pivoting rule*, pages 24–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[6] M. Bansal and M. Broadie. A simulation model to analyze the impact of hole size on putting in golf. In *Simulation Conference, 2008. WSC 2008. Winter*, pages 2826–2834, Dec 2008.

[7] H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. *CoRR*, abs/1504.05140, 2015.

[8] N. Bäuerle and U. Rieder. *Markov Decision Processes with Applications to Finance: Markov Decision Processes with Applications to Finance*. Springer Science & Business Media, 2011.

[9] M. Beetz, N. v. Hoyningen-Huene, J. Bandouch, B. Kirchlechner, S. Gedikli, and A. Maldonado. Camera-based observation of football games for analyzing multi-agent activities. In *Proceedings of the Fifth International Joint Conference on*

Autonomous Agents and Multiagent Systems, AAMAS '06, pages 42–49, New York, NY, USA, 2006. ACM.

[10] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[11] R. BELLMAN. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[12] R. Bellman. In A. Neyman and S. Sorin, editors, *Stochastic Games and Applications*, volume 570 of *NATO Science Series C: Mathematical and Physical Sciences*. Springer, 2003.

[13] F. Bendali and A. Quilliot. Réseaux stochastiques. *Revue française d'automatique, d'informatique et de recherche opérationnelle*, 24(2):167–190, 1990.

[14] D. P. Bertsekas. An auction algorithm for shortest paths. *SIAM Journal on Optimization*, 1(4):425–447, 1991.

[15] D. P. Bertsekas. *Dynamic programming and optimal control. Volume I*. Athena Scientific optimization and computation series. Belmont, Mass. Athena Scientific, 2005.

[16] D. P. Bertsekas. *Dynamic programming and optimal control. Volume II*. Athena Scientific optimization and computation series. Belmont, Mass. Athena Scientific, 2012.

[17] D. P. Bertsekas. Robust shortest path planning and semicontractive dynamic programming. *CoRR*, abs/1608.01670, 2016.

[18] D. P. Bertsekas, S. Pallottino, and M. G. Scutellà. Polynomial auction algorithms for shortest paths. *Computational Optimization and Applications*, 4(2):99–125, Apr 1995.

[19] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, Aug. 1991.

[20] D. P. Bertsekas and H. Yu. Stochastic shortest path problems under weak conditions, 2016.

[21] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[22] E. Boros, K. M. Elbassioni, V. Gurvich, and K. Makino. Markov decision processes and stochastic games with total effective payoff. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, pages 103–115, 2015.

[23] V. Brault and J. Chevalier. Private Communication, 2018/12.

[24] M. Broadie. Assessing golfer performance on the pga tour. *Interfaces*, 42(2):146–165, Mar. 2012.

[25] M. Broadie. *Every Shot Counts: Using the Revolutionary Strokes Gained Approach to Improve Your Golf Performance and Strategy*. Gotham Books, Penguin Group, New York, NY, USA, 2014.

[26] M. Broadie and S. Ko. A simulation model to analyze the impact of distance and direction on golf scores. In *Winter Simulation Conference*, WSC '09, pages 3109–3120. Winter Simulation Conference, 2009.

[27] A. Charnes. Optimality and degeneracy in linear programming. *Econometrica*, 20(2):160–170, 1952.

[28] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. 01 2008.

[29] J. X. Chen. The evolution of computing: Alphago. *Computing in Science Engineering*, 18(4):4–7, July 2016.

[30] A. Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.

[31] A. Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory, volume 13 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.

[32] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. 1951.

[33] E. V. Denardo. On Linear Programming in a Markov Decision Problem. *Management Science*, 16(5):281–288, 1970.

[34] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[35] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959.

[36] C. Drappi and L. C. Ting Keh. Predicting golf scores at the shot level. *Journal of Sports Analytics*, Preprint(Preprint):1–9, 2018. Exported from https://app.dimensions.ai on 2019/03/07.

[37] F. Dufour and T. Prieto-Rumeau. Approximation of markov decision processes with general state space. *Journal of Mathematical Analysis and Applications*, 388(2):1254 – 1267, 2012.

[38] J. H. Eaton and L. A. Zadeh. Optimal pursuit strategies in discrete-state probabilistic systems. *J. Basic Eng.*, 84(1):23–29, Mar. 1962.

[39] J. Edmonds. Maximum matching and a polyhedron with (0,1) vertices. *J. Res. Nat. Bur. Standards*, 69:125–130, 1965.

[40] B. Efron. The bootstrap and modern statistics. *Journal of the American Statistical Association*, 95(452):1293–1296, 2000.

[41] J. Fearnley. Exponential lower bounds for policy iteration. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer Berlin Heidelberg, 2010.

[42] E. A. Feinberg and J. Huang. The value iteration algorithm is not strongly polynomial for discounted dynamic programming. *Oper. Res. Lett.*, 42(2):130–131, Mar. 2014.

[43] J. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer, 1996.

[44] S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and R. D. Wolf. Exponential lower bounds for polytopes in combinatorial optimization. *J. ACM*, 62(2):17:1–17:23, May 2015.

[45] L. R. Ford. Network flow theory. 1956.

[46] O. Friedmann. An exponential lower bound for the parity game strategy improvement algorithm as we know it. In *Proceedings of the 24th LICS*, pages 145–156, 2009.

[47] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, Jun 1981.

[48] M. Guillot and G. Stauffer. The stochastic shortest path problem: A polyhedral combinatorics perspective. *European Journal of Operational Research*, 2018.

[49] T. D. Hansen. *Worst-case Analysis of Strategy Iteration and the Simplex Method*. PhD thesis, Aarhus University, 2012.

[50] T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1:1–1:16, Feb. 2013.

[51] T. D. Hansen, P. B. Miltersen, and U. Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1:1–1:16, Feb. 2013.

[52] O. Hernández-Lerma and J.-B. Lasserre. The linear programming approach. In E. Feinberg and A. Shwartz, editors, *Handbook of Markov Decision Processes*, volume 40 of *International Series in Operations Research & Management Science*, pages 377–407. Springer US, 2002.

[53] D. S. Hochbaum. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM J. Comput.*, 23(6):1179–1192, Dec. 1994.

[54] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[55] S. Hoffmeister and J. Rambau. Strategy optimization in sports : A two-scale approach via markov decision problems, October 2015.

[56] S. Hoffmeister and J. Rambau. Sport strategy optimization in beach volleyball? how to bound direct point probabilities dependent on individual skills. In *MathSport International 2017 Conference*, 2017.

[57] A. Hordijk and L. C. M. Kallenberg. Linear programming and markov decision chains. *Management Science*, 25(4):352–362, 1979.

[58] R. A. Howard. *Dynamic programming and Markov processes*. The MIT press, New York London, Cambridge, MA, 1960.

[59] IGF. Shotlink intelligence program, 2016.

[60] A. Jaśkiewicz and A. S. Nowak. Stochastic games with unbounded payoffs: Applications to robust control in economics. *Dynamic Games and Applications*, 1(2):253–279, Jun 2011.

[61] V. Kaibel. Extended formulations in combinatorial optimization. *Optima*, 85, 04 2011.

[62] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 302–311, New York, NY, USA, 1984. ACM.

[63] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[64] J. E. Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[65] L. G. KHACHIYAN. A polynomial algorithm in linear programming. *Doklady Academii Nauk SSSR*, 244:1093–1096, 1979.

[66] V. Klee and G. J. Minty. How good is the simplex algorithm. 1970.

[67] H. E. L. and H. R. Lowell. Stochastic model of the 2012 pga tour season. *Journal of Quantitative Analysis in Sports*, 10(4), 2014.

[68] M. Maher. Stochastic modelling of sport. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 207–208, Sep. 2012.

[69] A. S. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, 1960.

[70] L. Martin. Private Collaboration for a website creation, 2018/09.

[71] G. S. Matthieu Guillot. Golf strategy optimization for professional golfers's performances estimation on the pga tour. Math&Sport International Conference, 2017.

[72] G. S. Matthieu Guillot. Golf strategy optimization for professional golfers's performances estimation on the pga tour. EURO Conference, 2018.

[73] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[74] E. McAuley and V. V. Tammen. The effects of subjective and objective competitive outcomes on intrinsic motivation. *Journal of Sport and Exercise Psychology*, 11(1):84–93, 1989.

[75] M. Melekopoglou and A. Condon. On the complexity of the policy improvement algorithm for markov decision processes. *ORSA Journal on Computing*, 6(2):188–192, 1994.

[76] R. Merton. An intertemporal capital asset pricing model. *Econometrica*, 41(5):867–887, 1973.

[77] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[78] J. v. Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100:295–320, 1928.

[79] P. Newton and K. Aslam. Monte carlo tennis. *SIAM Review*, 48(4):722–742, 2006.

[80] K. C. Nguyen, T. Alpcan, and T. Basar. Stochastic games for security in networks with interdependent nodes. In *2009 International Conference on Game Theory for Networks*, pages 697–703, May 2009.

[81] J. R. Norris. *Markov chains.* Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998.

[82] A. S. Nowak. On stochastic games in economics. *Mathematical Methods of Operations Research*, 66(3):513–530, Dec 2007.

[83] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991.

[84] C. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall, 1982.

[85] M. Pfeiffer, H. Zhang, and A. Hohmann. A markov chain model of elite table tennis competition. *International Journal of Sports Science & Coaching*, 5(2):205–222, 2010.

[86] PGA Tour. Shotlink intelligence program, 2016.

[87] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[88] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[89] A. Quilliot. personal communication, 2017.

[90] S. S. Rao, R. Chandrasekaran, and K. P. K. Nair. Algorithms for discounted stochastic games. *Journal of Optimization Theory and Applications*, 11(6):627–637, Jun 1973.

[91] T. Rothvoß. Some 0/1 polytopes need exponential size extended formulations. *Mathematical Programming*, 142(1):255–268, Dec 2013.

[92] K. Routley and O. Schulte. A markov game model for valuing player actions in ice hockey. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, UAI'15, pages 782–791, Arlington, Virginia, United States, 2015. AUAI Press.

[93] B. Schölkopf, J. Platt, and T. Hofmann. *Multi-Robot Negotiation: Approximating the Set of Subgame Perfect Equilibria in General-Sum Stochastic Games*, pages 1001–1008. MITP, 2007.

[94] D. Seese. Groetschel, m., l. lovasz, a. schrijver: Geometric algorithms and combinatorial optimization. (algorithms and combinatorics. eds.: R. l. graham, b. korte, l. lovasz. vol. 2), springer-verlag 1988, xii, 362 pp., 23 figs., dm 148,-. isbn 3–540–13624-x. *Biometrical Journal*, 32(8):930–930, 1990.

[95] L. S. Shapley. Stochastic games. *Proceedings of National Academy of Science*, 39(10):1095–1100, 1953.

[96] L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[97] S. Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20(2):7–15, 1998.

[98] S. Sugawara, H. Kawamura, and K. Suzuki. Skill-based simulation model for optimizing strategy in golf. In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pages 1591–1596, July 2013.

[99] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[100] Y. Tanaka and H. Sekiya. The relationships between psychological/physiological changes and behavioral/performance changes of a golf putting task under pressure. *International Journal of Sport and Health Science*, 8:83–94, 2010.

[101] F. Teichteil-Königsbuch. Stochastic safest and shortest path problems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 1825–1831. AAAI Press, 2012.

[102] A. TERROBA, W. KOSTERS, J. VARONA, and C. S. MANRESA-YEE. Finding optimal strategies in tennis from video sequences. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(06):1355010, 2013.

[103] D. Tierney and R. Coop. A bivariate probability model for putting proficiency. *Science and Golf*, 1998.

[104] R. Tripathi, E. Valkanova, and V. A. Kumar. On strategy improvement algorithms for simple stochastic games. *Journal of Discrete Algorithms*, 9(3):263 – 278, 2011. Selected papers from the 7th International Conference on Algorithms and Complexity (CIAC 2010).

[105] E. Trumbelj and P. Vraar. Simulating a basketball match with a homogeneous markov model and forecasting the outcome. *International Journal of Forecasting*, 28(2):532 – 542, 2012.

[106] D. J. White. A survey of applications of markov decision processes. *The Journal of the Operational Research Society*, 44(11):1073–1096, 1993.

[107] P. Wolfe. A technique for resolving degeneracy in linear programming. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):205–211, 1963.

[108] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[109] Y. Ye. A new complexity result on solving the markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005.

[110] Y. Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36(4):593–603, 2011.

[111] G. J. Yue Yang. An efficient implementation of shortest path algorithm based on dijkstra algorithm. *GEOMATICS AND INFORMATION SCIENCE OF WUHAN UNIVERSITY*, 24(3):208, 1999.

Un parcours de golf est composé de dix-huit trous. Sur chaque trou, le problème du golfeur est de déplacer la balle d'un point de départ prédéfini jusqu'au drapeau en un minimum de coups. Sous certaines hypothèses, ce problème peut se modéliser comme un problème de plus court chemin stochastique (PCCS). Le problème du PCCS est un processus de Markov particulier dans lequel un agent évolue dynamiquement dans un ensemble fini d'états. En chaque état, l'agent choisit une action, induisant un coût, qui le mène en un autre état en suivant une distribution de probabilité connue. Il existe également un état 'puits' particulier dans lequel, une fois atteint, on reste avec une probabilité un et un coût de zéro. Le but de l'agent est, depuis un état initial, d'atteindre l'état puits en un coût moyen minimal. Dans un premier chapitre, nous étudions de manière théorique le problème du PCCS. Après avoir redéfini un cadre d'étude dans lequel nous avons affaibli les hypothèses d'existence d'une solution optimale, nous avons prouvé que les algorithmes classiques de résolution convergent dans ce nouveau cadre. Nous avons également défini un nouvel algorithme de résolution basé sur l'algorithme primal-dual. Dans le deuxième chapitre, nous détaillons la modélisation du problème d'optimisation de stratégies au golf en un problème de PCCS. Grâce à la base de données Shotlink, nous définissons des 'clones numériques' de joueurs que nous pouvons faire jouer artificiellement sur différents parcours de golf afin de prédire les scores des joueurs. Nous avons appliqué ce modèle à deux compétitions : le master d'Augusta en 2017 et la Ryder Cup en 2018. Dans un troisième et dernier chapitre, nous étudions l'extension naturelle à deux joueurs du problème du PCCS : les jeux de plus courts chemins stochastiques. Nous étudions particulièrement les formulations programmation linéaire de ces jeux et de deux cas particuliers de ceux-ci.

A golf course consists of eighteen holes. On each hole, the golfer has to move the ball from the tee to the flag in a minimum number of shots. Under some assumptions, the golfer's problem can be modeled as a stochastic shortest path problem (SSP). SSP problem is a special case of Markov Decision Processes in which an agent evolves dynamically in a finite set of states. In each state, the agent chooses an action that leads him to another state following a known probability distribution. This action induces a cost. There exists a 'sink node' in which the agent, once in it, stays with probability one and a cost zero. The goal of the agent is to reach the sink node with a minimum expected cost. In the first chapter, we study the SSP problem theoretically. We define a new framework in which the assumptions needed for the existence of an optimal policy are weakened. We prove that the most famous algorithm still converge in this setting. We also define a new algorithm to solve exactly the problem based on the primal-dual algorithm. In the second chapter we detail the golfer's problem model as a SSP. Thanks to the Shotlink database, we create 'numerical clones' of players and simulate theses clones on different golf course in order to predict professional golfer's scores. We apply our model on two competitions: the master of Augusta in 2017 and the Ryder Cup in 2018. In the third chapter, we study the 2-player natural extension of SSP problem: the stochastic shortest path games. We study two special cases, and in particular linear programming formulation of these games.