Rapport de stage d'été 2A

Étude des *Simple Stochastic Games*

Auteur :
Yannis Juglaret

Encadrants :
David Auger
Pierre Coucheney
Yann Strozecki

du 17 juin au 26 juillet 2013

Table des matières

Introduction

Dans le cadre de ma formation à l'*École nationale supérieure de l'électronique et de ses applications* (ENSEA), j'ai pu effectuer l'année dernière un stage en entreprise d'une durée d'un mois. Cet été, j'ai voulu m'ouvrir à de nouveaux horizons en faisant un stage d'initiation à la recherche dans un laboratoire académique. En effet, je suis intéressé par la recherche et j'ai voulu vivre le quotidien d'un chercheur.

Durant mes années passées en classe préparatoire, j'ai été très motivé par les cours d'informatique fondamentale que j'ai pu suivre. Depuis, j'ai pu continuer mon apprentissage grâce aux cours de l'ENSEA et aussi par la lecture d'un certain nombre de livres dédiés à cette thématique. Pour autant, j'ai rarement eu l'occasion d'exercer mon esprit aux raisonnements théoriques propres à cette science, car mes projets à l'ENSEA se concentraient sur la résolution de problèmes plus concrets.

En cherchant parmi les laboratoires affiliés au *Centre national pour la recherche scientifique* (CNRS) qui exploraient ce domaine, j'ai découvert le laboratoire *Parallélisme, Réseaux, Systèmes, Modélisation* (PRiSM) dont les thèmes de recherche ont retenu mon attention. Après avoir contacté la personne responsable d'une des équipes du laboratoire, j'ai rapidement obtenu deux propositions de stage, et j'ai finalement retenu celle de M. Strozecki.

M. Strozecki fait partie de l'équipe *Algorithmique, combinatoire analytique et applications* (ALCAAP). Lui, M. Auger et M. Coucheney travaillent en ce moment autour de la théorie des jeux et plus particulièrement sur un modèle de jeu appelé les *Simple Stochastic Games* (SSG).

Ce modèle, qui a des applications en économie et en vérification, soulève des questions liées à la théorie de la complexité algorithmique, ce qui lui donne un lien avec les cours de l'ENSEA intitulés *Structures de données* (1A) et *Algorithmique* (3A IS) et dans une moindre mesure les options *Cryptologie* (2A), *Information quantique* (2A) et *Recherche dans les bases de données multimédia* (2A) que j'ai suivies.

Mes encadrants m'ont fourni quelques pistes à explorer une fois le modèle des SSG bien compris et tous les prérequis théoriques acquis. Le but de mon

stage était donc la découverte du monde de la recherche, l'acquisition de nouvelles connaissances, et l'approfondissement de ces pistes de réflexion.

Dans ce rapport, je présenterai d'abord le laboratoire PRiSM et l'équipe ALCAAP. Ensuite je ferai une présentation du domaine d'étude allant du plus général vers le plus spécifique. Enfin je préciserai les résultats qu'ont donné les pistes de travail qui m'ont été proposées.

Toutes les définitions présentées dans ce document sont inspirées de la littérature que l'on peut trouver autour de ces domaines mais restent personnelles. Il en va de même pour les noms choisis pour les objets définis, et pour les notations. On pourra donc retrouver les mêmes concepts sous des noms différents et avec des notations différentes dans d'autres sources d'information.

Chapitre 1

La vie au laboratoire PRiSM

1.1 Structure d'accueil

1.1.1 L'université

L'*Université de Versailles–Saint-Quentin-en-Yvelines* (UVSQ) est l'une des huit universités nouvelles qui ont été créées dans les années 1990. Elle est implantée sur sept agglomérations.

Dans mon cas, ce sont les travaux en sciences exactes, ingénierie et technologie qui m'ont attiré, mais l'université dispense aussi des formations dans de nombreux autres domaines tels que les sciences sociales, les sciences humaines, les sciences juridiques et politiques, ou encore la médecine.

Mon stage s'est déroulé au sein de l'UFR [1] des sciences de l'UVSQ, à Versailles, où se situe le laboratoire PRiSM.

1.1.2 Le laboratoire

Le laboratoire PRiSM est un laboratoire de recherche en informatique français affilié au CNRS (UMR [2] 8144). Les salariés sont essentiellement des chercheurs. En dehors de ceux-ci on compte un directeur, un directeur adjoint, un gestionnaire financier, trois secrétaires pour la recherche et deux administrateurs systèmes et réseaux. Neuf chercheurs ont un statut particulier de responsable d'équipe de recherche.

Il y a une quarantaine de chercheurs, majoritairement enseignants-chercheurs, pour une cinquantaine de doctorants. Ceux-ci sont répartis dans les équipes suivantes :

1. Unité de formation et de recherche.
2. Unité mixte de recherche.

— *Algorithmique, combinatoire analytique et applications* (ALCAAP) ;
— *Advanced Modelling of Adaptive Information Systems* (AMIS) ;
— *Architecture et parallèlisme* (ARPA) ;
— *Cryptologie et sécurité de l'information* (CRYPTO) ;
— *Évaluation des performances des réseaux informatiques* (EPRI) ;
— *Data Integration Mining* (DIM) ;
— *Calcul réparti et optimisation* (CaRO) ;
— *Secured and Mobile Information Systems* (SMIS).

1.2 Fonctionnement interne

1.2.1 L'équipe

L'équipe ALCAAP, que j'ai rejoint, travaille sur des thématiques issues des fondements de l'informatique comme la théorie des graphes, l'algorithmique, la théorie de la complexité ainsi que la combinatoire. Les SSG, sujet de mon stage, interviennent dans ce cadre.

Outre la recherche fondamentale dans ces domaines, l'équipe travaille aussi sur leurs applications pour l'industrie ou pour d'autres domaines scientifiques tels que la bioinformatique et la logique. Ainsi, par exemple, j'ai pu assister à des présentations sur les liens entre l'informatique et la chimie ou la biologie.

1.2.2 Outils utilisés

En rejoignant l'équipe j'ai été invité à utiliser les outils en place. J'ai ainsi eu accès à un dépot *git* sur lequel se trouvait un prototype développé en *Python* permettant d'effectuer des opérations sur les SSG.

1.2.3 git

Le logiciel *git* permet de gérer les différentes versions d'un programme au cours de son développement. De plus, il garde une trace des modifications effectuées par chacun des développeurs, ce qui permet d'avoir facilement un aperçu des évolutions et de pouvoir, si cela est nécessaire, revenir à un état antérieur stable.

1.2.4 Python

Python est un langage de programmation généraliste qui permet un développement rapide tout en fournissant une syntaxe agréable à lire. Ayant

déjà eu affaire à ce langage avant le stage, je n'ai eu qu'à m'imprégner du code déjà écrit avant de pouvoir commencer à utiliser le prototype.

Chapitre 2

De la théorie des jeux aux SSG

Dans ce chapitre je présenterai tout d'abord un sous-ensemble de la théorie des jeux avant de m'intéresser au cas particulier des SSG.

Notations

$|X|$: cardinal de l'ensemble X ;
$\mathbb{M}_{n,p}$: ensemble des matrices à n lignes et p colonnes ;
$d^+(v)$: nombre de successeurs du sommet v d'un graphe orienté.

2.1 Théorie des jeux à deux joueurs à information parfaite

La théorie des jeux fournit un cadre mathématique pour l'étude des jeux. Pour une présentation générale, on pourra consulter [NRTV07]. Ici je me restreindrai au cas des jeux à deux joueurs à information parfaite.

Cela signifie tout d'abord qu'au moment de la décision, chaque joueur connait :

— ses propres possibilités d'action ;
— les possibilités d'action de l'autre joueur ;
— les gains de chacun des joueurs en fonction des actions choisies ;
— le but du jeu pour les deux joueurs.

D'autre part, dans le cas d'une partie se jouant en plusieurs étapes, l'historique complet de la partie est accessible aux deux joueurs au moment de leur décision, qui peuvent donc jouer en fonction de celui-ci.

Dans ce document nous considérerons que les gains peuvent toujours être exprimés par des nombres réels, et que le but de chacun des joueurs

est toujours de maximiser son propre gain. Ce choix couvre bien entendu les jeux où il est question de points ou d'argent (les réels négatifs peuvent modéliser une perte), mais pas seulement. En effet il s'agit simplement de hiérarchiser et quantifier les préférences des joueurs quant à toutes les issues possibles du jeu.

Prenons par exemple le cas d'un jeu où il n'y a que les possibilités de victoire, égalité ou défaite. Il suffira alors de choisir de représenter chacune de ces issues possibles respectivement par 1, 0 et -1 pour refléter les préférences des joueurs par rapport à ces trois issues possibles.

2.1.1 Représentation standard

Nous allons maintenant formaliser ces idées en les représentant par des objets mathématiques. Les jeux que nous considérerons pour le moment sont des jeux en une seule manche. Les joueurs font leurs choix d'action chacun de leur côté, puis les révèlent en même temps et les gains de chacun sont déduits en conséquence.

Définition 1. *Un jeu (à deux joueurs à information parfaite) est un quadruplet $J = (A, B, g_A, g_B)$; où A et B sont deux ensembles, et g_A et g_B deux fonctions telles que $g_A : A \times B \to \mathbb{R}$ et $g_B : A \times B \to \mathbb{R}$. Les éléments de A et de B sont appelés des actions. Les éléments (a, b) de l'ensemble $A \times B$ sont appelés des instances du jeu J. Les fonctions g_A et g_B sont appelées fonctions de gain. À toute instance (a, b) du jeu J, elles associent chacune un unique réel $g_A(a, b)$ ou $g_B(a, b)$ qui est appelé gain.*

Cette définition est purement formelle, mais nous avons introduit des noms pour les objets mathématiques que nous manipulerons qui sont proches des concepts qu'ils représentent. Voyons donc maintenant comment ce modèle permet effectivement de représenter le concept de jeu en une manche dont nous avons l'habitude. Pour ce faire, nous appellerons nos deux joueurs Alice et Bob.

A est l'ensemble des actions possibles pour Alice dans le jeu J, et B l'ensemble des actions possibles pour Bob dans ce même jeu. Ces ensembles représentent donc les coups permis à chacun des joueurs. Les instances de jeu $(a, b) \in A \times B$ représentent toutes les parties possibles en fonction des choix d'action des deux joueurs. Une fois que ceux-ci ont fait leur choix, l'issue de la partie est alors connue : Alice reçoit le gain $g_A(a, b)$ et Bob le gain $g_B(a, b)$.

Se placer dans le cadre d'un jeu à information parfaite revient ici à dire qu'au moment de leur décision, Alice et Bob ont connaissance des ensembles A et B et des fonctions g_A et g_B. Dès lors chacun peut faire des calculs sur ces fonctions pour faire son choix d'action.

Définition 2. *Soit $J = (A, B, g_A, g_B)$ un jeu quelconque. On définit les fonctions de coût pour Alice $c_A : A \times B \to \mathbb{R}$ et de coût pour Bob $c_B : A \times B \to \mathbb{R}$ comme étant les opposées des fonctions de gain. Ainsi $c_A = -g_A$ et $c_B = -g_B$.*

Cette nouvelle définition permet de définir ou d'étudier un jeu en termes de coûts plutôt que de gains. Ces deux représentations sont équivalentes, et maximiser un gain revient à minimiser un coût.

Définition 3. *Un jeu quelconque $J = (A, B, g_A, g_B)$ est appelé jeu à gains s'il satisfait :*

$$\forall (a, b) \in A \times B, g_A(a, b) \geq 0$$
$$\forall (a, b) \in A \times B, g_B(a, b) \geq 0$$

Il est appelé jeu à coûts s'il satisfait :

$$\forall (a, b) \in A \times B, g_A(a, b) \leq 0$$
$$\forall (a, b) \in A \times B, g_B(a, b) \leq 0$$

On parlera donc de jeux à gains lorsque les gains seront toujours positifs ou nuls et de jeux à coûts lorsque les coûts seront toujours positifs ou nuls.

Définition 4. *On appelle fonction moyenne de gain la fonction $g_{moy} = \frac{g_A + g_B}{2}$. À toute instance de jeu $(a, b) \in A \times B$ elle associe un unique réel $g_{moy}(a, b)$ appelé gain moyen.*

On définit de façon similaire la fonction moyenne de coût c_{moy} et le coût moyen $c_{moy}(a, b)$. L'étude de ces fonctions est intéressante pour savoir si une instance de jeu donnée représente un bon compromis pour les joueurs : dans certains jeux ceux-ci il peut être intéressant pour eux d'essayer de collaborer afin de rendre le gain moyen le plus grand possible, au lieu d'agir de façon égoïste.

2.1.2 Jeux matriciels

Dans le cas où A et B sont finis, on parle de jeux matriciels. En effet ces jeux peuvent être facilement représentés par deux matrices.

Définition 5. *Soit $J = (A, B, g_A, g_B)$ un jeu tel que A et B soient finis. Indexons ces deux ensembles d'une façon arbitraire fixée :*

$$A = \{a_i \mid 1 \leq i \leq |A|\}$$
$$B = \{b_j \mid 1 \leq j \leq |B|\}$$

On appelle matrice de gain pour Alice la matrice $G_A \in \mathbb{M}_{|A|,|B|}$ et matrice de gain pour Bob la matrice $G_B \in \mathbb{M}_{|A|,|B|}$ vérifiant :

$$\forall 1 \leq i \leq |A|, \forall 1 \leq j \leq |B|, (G_A)_{ij} = g_A(a_i, b_j)$$
$$\forall 1 \leq i \leq |A|, \forall 1 \leq j \leq |B|, (G_B)_{ij} = g_B(a_i, b_j)$$

On définit de manière analogue les matrices de coût $C_A \in \mathbb{M}_{|A|,|B|}$ et $C_B \in \mathbb{M}_{|A|,|B|}$ à partir des fonctions c_A et c_B. Les représentations matricielles permettent une vision concise du jeu, quand cela est possible.

Pour illustrer ces définitions voici un premier exemple de jeu très simple, qui est souvent utilisé pour introduire les concepts liés à la théorie des jeux : le dilemme du prisonnier.

FIGURE 2.1 – Matrices de côuts pour le dilemme du prisonnier

$$C_A = \begin{array}{c} \\ c_A \\ s_A \end{array} \begin{array}{c} \begin{array}{cc} c_B & s_B \end{array} \\ \begin{pmatrix} 4 & 1 \\ 5 & 2 \end{pmatrix} \end{array} \quad C_B = \begin{array}{c} \\ c_A \\ s_A \end{array} \begin{array}{c} \begin{array}{cc} c_B & s_B \end{array} \\ \begin{pmatrix} 4 & 5 \\ 1 & 2 \end{pmatrix} \end{array} \tag{2.1}$$

Exemple 1 (Dilemme du prisonnier)**.** *Deux prisonniers complices sont interrogés. Suivant que l'un, les deux, ou aucun ne se confesse, leurs peines seront différentes. Si les deux prisonniers se taisent, leur peine à chacun sera de 2 ans de prison faute de preuve. Si l'un d'eux se confesse et pas l'autre, il sera utilisé comme témoin au procès du deuxième qui écopera d'une peine de 5 ans, tandis que lui verra sa peine réduite à 1 an pour service rendu. Enfin si les deux se confessent ils écoperont chacun d'une peine de 4 ans.*

Modélisons ce dilemme sous la forme d'un jeu à deux joueurs. Chaque joueur possède deux actions possibles : la confession ou le silence, que nous allons respectivement représenter par les lettres c et s. Ainsi on pose $A = \{c_A, s_A\}$ et $B = \{c_B, s_B\}$. En choisissant directement le nombre d'années de prison comme coût, on obtient les matrices représentées sur la figure 2.1.

2.1.3 Stratégies

La stratégie d'un joueur caractérise la manière dont il fait ses choix d'actions en fonction de tous les paramètres qu'il a à sa disposition. Dans notre

cas de jeu en une manche, nous considérerons qu'un joueur peut uniquement s'aider, s'il le souhaite, d'une source de hasard pour faire ses choix. Dans la vraie vie, il pourrait s'aider par exemple de fuites d'information sur l'intention de son adversaire, mais nous ne le prendrons pas en compte ici.

Définition 6. *On munit l'ensemble A d'une tribu fixée pour former un espace mesurable. On note $\bar{A}$ l'ensemble des lois de probabilité sur cet espace mesurable. Une telle loi de probabilité $\bar{a} \in \bar{A}$ est appelée stratégie pour Alice.*

Définition 7. *Une stratégie $\bar{a} \in \bar{A}$ est dite pure s'il existe $a \in A$ tel que $\{a\}$ soit mesurable et $\mathbb{P}_{\bar{a}}(\{a\}) = 1$. Dans le cas contraire, elle est dite mixte.*

Une stratégie pure revient à jouer tout le temps la même action, par opposition aux stratégies mixtes qui doivent nécessairement faire intervenir une source de hasard pour être mises en œuvre. On définit de façon similaire les stratégies pour Bob et leur ensemble $\bar{B}$, avec là aussi la distinction entre les stratégies pures et les stratégies mixtes.

Dans le cas de jeux matriciels, on choisit la tribu des parties et on représente les stratégies $\bar{a} \in \bar{A}$ d'Alice et les stratégies $\bar{b} \in \bar{B}$ Bob par deux vecteurs $p = (p_1 \dots p_{|A|})^t$ et $q = (q_1 \dots q_{|B|})^t$ tels que :

$$\forall 1 \leq i \leq |A|, p_i = \mathbb{P}_{\bar{a}}(\{a_i\})$$
$$\forall 1 \leq j \leq |B|, q_j = \mathbb{P}_{\bar{b}}(\{b_j\})$$

Les espérances de gain pour Alice et pour Bob sont alors respectivement :

$$\mathbb{E}_{p,q}[g_A] = \sum_{i=1}^{|S_A|} \sum_{j=1}^{|S_B|} p_i q_j g_A(a_i, b_j)$$
$$\mathbb{E}_{p,q}[g_B] = \sum_{i=1}^{|S_A|} \sum_{j=1}^{|S_B|} p_i q_j g_B(a_i, b_j)$$

Par définition des matrices de gain, on a alors directement :

$$\mathbb{E}_{p,q}[g_A] = p^t.G_A.q$$
$$\mathbb{E}_{p,q}[g_B] = p^t.G_B.q$$

2.1.4 Équilibres de Nash et instances optimales

Définition 8. *Soit $J = (A, B, g_A, g_B)$ un jeu quelconque. Un équilibre de Nash pour le jeu J est une instance de jeu $(\hat{a}, \hat{b}) \in A \times B$ telle que :*

$$\forall a \in A, g_A(a, \hat{b}) \leq g_A(\hat{a}, \hat{b})$$
$$\forall b \in B, g_B(\hat{a}, b) \leq g_B(\hat{a}, \hat{b})$$

De manière informelle, un équilibre de Nash est une instance de jeu dans laquelle aucun des joueurs n'a d'intérêt à modifier son propre choix d'action, car cela diminuerait son gain. Dans toutes les autres configurations, au moins un des joueurs peut agir de façon égoïste et modifier son choix pour améliorer son propre gain, aux dépens du gain moyen. En ce sens, les équilibres de Nash sont les seules positions stables lorsque les joueurs agissent de façon égoïste.

En fonction du jeu considéré, il peut n'y avoir aucun équilibre de Nash, un seul, un nombre fini ou même un nombre infini. Dans le cas du dilemme du prisonnier, il y en a un unique : il s'agit du cas où les deux prisonniers se confessent. En effet leur coût à tous deux est alors de 4, et un joueur qui choisirait le silence amènerait son coût à 5, ce qui ne serait pas avantageux pour lui.

Par ailleurs, la meilleure solution en termes de coût moyen pour les deux prisonniers est clairement qu'ils restent tous les deux silencieux, mais elle ne correspond pas à un équilibre de Nash. En effet dans cette configuration il suffit que l'un des deux prisonniers choisisse de se confesser pour qu'il diminue son coût, passant de 2 à 1.

Cet exemple simple montre qu'un équilibre de Nash, qui est un équilibre local, ne correspond pas dans le cas général au meilleur compromis pour les deux joueurs, qui réalise l'équilibre global en minimisant le coût moyen. Ce constat est formalisé par la notion de *Price of Anarchy* (PoA) qui quantifie cet écart entre équilibre local et équilibre global dans le pire des cas possibles.

Définition 9. *Soit $J = (A, B, g_A, g_B)$ un jeu à gains possédant au moins un équilibre de Nash. Soit $E \subseteq A \times B$ l'ensemble des équilibres de Nash du jeu J. On appelle* Price of Anarchy *(PoA) la quantité :*

$$PoA = \frac{\sup\limits_{(a,b)\in A\times B} g_{moy}(a, b)}{\inf\limits_{(\hat{a},\hat{b})\in E} g_{moy}(\hat{a}, \hat{b})}$$

Soit $J = (A, B, -c_A, -c_B)$ un jeu à coûts possédant au moins un équilibre de Nash. Soit $E \subseteq A \times B$ l'ensemble des équilibres de Nash du jeu J. On appelle Price of Anarchy *(PoA) la quantité :*

$$PoA = \frac{\sup\limits_{(\hat{a},\hat{b})\in E} c_{moy}(\hat{a}, \hat{b})}{\inf\limits_{(a,b)\in A\times B} c_{moy}(a, b)}$$

Dans les deux cas mis en évidence, le PoA quantifie la pénalité introduite sur le coût moyen en laissant les joueurs faire des choix égoïstes, c'est-à-dire en situation d'anarchie, par rapport à la situation où une autorité imposerait leurs choix aux joueurs pour atteindre le meilleur compromis. Cela se traduit par la diminution du gain moyen dans un jeu à gains, ou par l'augmentation du coût moyen dans un jeu à coûts. Le PoA est par définition toujours supérieur ou égal à 1.

Dans le cas du dilemme du prisonnier, qui est un jeu à coûts, le PoA est :

$$\frac{(\frac{4+4}{2})}{(\frac{2+2}{2})} = 2 \tag{2.2}$$

L'écart étant ainsi quantifié par le PoA, on peut dire que laisser les prisonniers faire des choix égoïstes les amènera à la seule situation stable qui se trouve être une situation deux fois pire, de leur point de vue et en termes de coût moyen, que la situation optimale où une autorité leur imposerait de garder le silence.

On définit de façon similaire les équilibres de Nash mixtes et le PoA mixte en autorisant aux joueurs les stratégies mixtes, et en étudiant alors les gains et coûts en termes d'espérance.

2.1.5 Actions dominantes et stratégies dominantes

Définition 10. *Une action d'Alice $\hat{a} \in A$ est dite dominante si elle vérifie :*

$$\forall a \in A, \forall b \in B, g_A(a,b) \leq g_A(\hat{a},b) \tag{2.3}$$

On définit de même les actions dominantes de Bob. Quelle que soit l'action jouée par l'adversaire, une action dominante aussure au joueur un gain supérieur à toutes les autres actions qu'il a à sa disposition. C'est donc une action particulièrement intéressante pour un joueur égoïste.

Définition 11. *Une instance de jeu $(\hat{a},\hat{b}) \in A \times B$ est dite à actions dominantes si $\hat{a}$ et $\hat{b}$ sont toutes les deux des actions dominantes.*

Un jeu possède une instance à actions dominantes dès lors que chacun des joueurs possède un choix d'action dominant. C'est le cas dans le dilemme du prisonnier. En effet, quoi que fasse l'autre prisonnier, le coût est moindre quand on se confesse (soit il sera de 1 plutôt que 2, soit de 4 plutôt que 5). Cela fait du dilemme du prisonnier un cas très particulier de jeu.

Une telle instance est toujours un équilibre de Nash. Cependant, comme on l'a vu avec le dilemme du prisonnier, elle n'aboutit pas nécessairement au meilleur compromis pour les deux joueurs.

La notion d'action dominante se généralise par les stratégies dominantes en autorisant les stratégies mixtes et en considérant les gains et les coûts par leur espérance.

2.1.6 Gains assurés

On suppose qu'Alice ne connait pas le choix d'action de Bob et réciproquement.

Cas des jeux matriciels

Fixons le choix d'Alice en prenant $1 \leq i_0 \leq |A|$. Alors Alice est assurée d'obtenir au moins un certain gain, qui est le gain obtenu si Bob joue de la façon la plus contraignante pour Alice, donc selon la stratégie minimisant $(G_A)_{i_0 j}$. Ainsi Alice est assurée que son gain sera au moins :

$$\min_{1 \leq j \leq |S_B|} (G_A)_{i_0 j} \tag{2.4}$$

Ne connaissant pas la stratégie de Bob, le meilleur gain qu'Alice puisse assurer est obtenu en choisissant la stratégie qui maximise cette quantité. En faisant ce choix, elle est ainsi assurée que son gain soit au moins :

$$\max_{1 \leq i \leq |S_A|} \min_{1 \leq j \leq |S_B|} (G_A)_{ij} \tag{2.5}$$

Fixons maintenant une stratégie pour Bob en choisissant $1 \leq j_0 \leq |S_B|$. Alors Bob est assuré que le gain d'Alice ne dépassera pas la quantité suivante :

$$\max_{1 \leq i \leq |S_A|} (G_A)_{i j_0} \tag{2.6}$$

Ne connaissant pas la stratégie d'Alice, le pire gain pour Alice que Bob puisse assurer est obtenu en minimisant cette valeur. En choisissant la stratégie adéquate, Bob est assuré que le gain d'Alice soit au plus :

$$\min_{1 \leq j \leq |S_B|} \max_{1 \leq i \leq |S_A|} (G_A)_{ij} \tag{2.7}$$

Ainsi en choisissant bien leurs stratégies, Alice peut assurer que son gain ne descende pas plus bas qu'un certain seuil, et Bob peut assurer que le gain d'Alice ne dépasse pas un autre seuil. Si les deux choisissent les stratégies assurant chacune des bornes, alors le gain d'Alice sera bien entendu compris entre les deux bornes, ce qui est formalisé par la propriété ci-dessous et la démonstration associée.

Propriété 1.

$$\max_{1\leq i\leq|S_A|}\min_{1\leq j\leq|S_B|}(G_A)_{ij}\leq\min_{1\leq j\leq|S_B|}\max_{1\leq i\leq|S_A|}(G_A)_{ij} \tag{2.8}$$

Démonstration. Supposons qu'Alice joue selon la stratégie i_0 et Bob selon la stratégie j_0 telles que :

$$\min_{1\leq j\leq|S_B|}(G_A)_{i_0j}=\max_{1\leq i\leq|S_A|}\min_{1\leq j\leq|S_B|}(G_A)_{ij} \tag{2.9}$$

$$\max_{1\leq i\leq|S_A|}(G_A)_{ij_0}=\min_{1\leq j\leq|S_B|}\max_{1\leq i\leq|S_A|}(G_A)_{ij} \tag{2.10}$$

Par définition du min (resp. du max) on a les inégalités suivantes :

$$\min_{1\leq j\leq|S_B|}(G_A)_{i_0j}\leq(G_A)_{i_0j_0} \tag{2.11}$$

$$(G_A)_{i_0j_0}\leq\max_{1\leq i\leq|S_A|}(G_A)_{ij_0} \tag{2.12}$$

On en déduit directement :

$$\begin{aligned}\max_{1\leq i\leq|S_A|}\min_{1\leq j\leq|S_B|}(G_A)_{ij}&=\min_{1\leq j\leq|S_B|}(G_A)_{i_0j} &(2.13)\\ &\leq(G_A)_{i_0j_0} &(2.14)\\ &\leq\max_{1\leq i\leq|S_A|}(G_A)_{ij_0} &(2.15)\\ &=\min_{1\leq j\leq|S_B|}\max_{1\leq i\leq|S_A|}(G_A)_{ij} &(2.16)\end{aligned}$$

□

Pour autant, Bob n'a pas forcément intérêt à choisir une stratégie qui minimise le gain d'Alice. Ce sera le cas dans une classe particulière de jeux appelés les jeux à somme nulle, présentés plus loin, où minimiser le gain d'Alice est équivalent à maximiser celui de Bob et réciproquement.

Cas général

On peut généraliser les idées précédentes dans le cas de jeux où les ensembles d'actions possibles sont infinis, mais le tout est moins intuitif car on ne peut raisonner qu'en termes de bornes inférieure et supérieure sans savoir si elles seront atteintes ou non, et si elles sont finies ou non.

Fixons le choix d'Alice en prenant $\hat{a}\in A$ quelconque. Alors quel que soit le choix $\hat{b}\in B$ de Bob, on a par définition de la borne inférieure :

$$g_A(\hat{a},\hat{b})\geq\inf_{b\in B}g_A(\hat{a},b)\in\mathbb{R}\cup\{-\infty\} \tag{2.17}$$

On obtient ainsi la limite du gain qu'Alice est assurée d'obtenir en jouant $\hat{a}$. En effet, contre $\hat{a}$, Bob ne pourra pas faire descendre le gain d'Alice en dessous de cette limite. Par ailleurs, quel que soit le choix $\hat{a}$ d'Alice, on a :

$$\inf_{b\in B} g_A(\hat{a}, b) \leq \sup_{a\in A} \inf_{b\in B} g_A(a, b) \tag{2.18}$$

On sait par conséquent qu'Alice ne pourra pas assurer un gain supérieur à cette nouvelle limite, car elle ne pourra pas faire monter son gain assuré au-delà de celle-ci.

Fixons maintenant le choix de Bob en prenant $\hat{b} \in B$ quelconque. Alors quel que soit le choix $\hat{a} \in A$ d'Alice, on a par définition de la borne supérieure :

$$g_A(\hat{a}, \hat{b}) \leq \sup_{a\in A} g_A(a, \hat{b}) \in \mathbb{R} \cup \{+\infty\} \tag{2.19}$$

On obtient ainsi la limite de gain qu'Alice ne pourra pas dépasser si Bob joue $\hat{b}$. En effet, contre $\hat{b}$, Alice ne pourra pas faire monter son gain au-delà de celle-ci. Par ailleurs, quel que soit le choix $\hat{b}$ de Bob, on a :

$$\sup_{a\in A} g_A(a, \hat{b}) \geq \inf_{b\in B} \sup_{a\in A} g_A(a, b) \tag{2.20}$$

On sait par conséquent que Bob ne ne pourra pas assurer qu'Alice ait un gain inférieur à cette nouvelle limite.

Nous avons ainsi exhibé deux limites sur le gain d'Alice : en choisissant bien leurs stratégies, d'une part Alice peut s'assurer d'avoir un gain dans $]-\infty; \sup_{a\in A} \inf_{b\in B} g_A(a, b)[$, et d'autre part Bob peut s'assurer que le gain d'Alice soit dans $]\inf_{b\in B} \sup_{a\in A} g_A(a, b); +\infty[$.

Pour autant, Bob n'a pas forcément intérêt à choisir une stratégie qui minimise le gain d'Alice. Ce sera le cas dans une classe particulière de jeux appelés les jeux à somme nulle, présentés plus loin, où minimiser le gain d'Alice est équivalent à maximiser celui de Bob et réciproquement.

Propriété 2.

$$\sup_{a\in A} \inf_{b\in B} g_A(a, b) \leq \inf_{b\in B} \sup_{a\in A} g_A(a, b) \tag{2.21}$$

Démonstration. Par définition de la borne supérieure, on a :

$$\forall \hat{a} \in A, \forall \hat{b} \in B, g(\hat{a}, \hat{b}) \leq \sup_{a\in A} g(a, \hat{b}) \tag{2.22}$$

D'où en prenant la borne inférieure des deux côtés de l'égalité :

$$\forall \hat{a} \in A, \inf_{b \in B} g(\hat{a}, b) \leq \inf_{b \in B} \sup_{a \in A} g(a, b) \tag{2.23}$$

Finalement par définition de la borne supérieure :

$$\sup_{a \in A} \inf_{b \in B} g(a, b) \leq \inf_{b \in B} \sup_{a \in A} g(a, b) \tag{2.24}$$

□

2.1.7 Stratégies mixtes et pures

De plus :

$$\max_{p} \min_{q} \mathbb{E}_{p,q}[g_A] = \min_{q} \max_{p} \mathbb{E}_{p,q}[g_A] \tag{2.25}$$

On peut ainsi toujours atteindre ce que l'on appelle un équilibre de Nash mixte en utilisant des stratégies mixtes.

Démonstration :

programmes linéaires duaux (todo)

2.1.8 Jeux à somme nulle

Un jeu est dit à somme nulle si la somme des gains des joueurs est nulle. Dans un jeu à deux joueurs cela revient à dire que le gain du premier joueur est toujours l'opposé de celui du deuxième. Autrement dit :

$$\forall (a, b) \in A \times B, g_A(a, b) + g_B(a, b) = 0$$

Dans ce cas le jeu est entièrement défini par la donnée de (A, B, g_A). Si l'on autorise les stratégies mixtes, la somme des espérances de gain d'Alice et de Bob est évidemment nulle par linéarité de l'espérance.

Exemple : le jeu pierre-feuille-ciseaux (PFC)

Dans le jeu PFC, chaque joueur a trois stratégies possibles : pierre, feuille ou ciseaux. Si les deux joueurs choisissent la même, alors il y a égalité : les deux ont un gain nul. Sinon, la stratégie d'un des joueurs est gagnante face à son adversaire : la pierre bat les ciseaux, les ciseaux battent la feuille et la feuille bat la pierre. Le gain du gagnant est alors 1 et celui du perdant -1.

On obtient donc la représentation matricielle suivante :

$$G_A = \begin{array}{c} \\ p_A \\ f_A \\ c_A \end{array} \begin{array}{c} \begin{array}{ccc} p_B & f_B & c_B \end{array} \\ \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \end{array} \quad G_B = \begin{array}{c} \\ p_A \\ f_A \\ c_A \end{array} \begin{array}{c} \begin{array}{ccc} p_B & f_B & c_B \end{array} \\ \begin{pmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{pmatrix} \end{array}$$

Le jeu est à somme nulle, comme on peut le vérifier en sommant ces deux matrices. Dans le cas du jeu PFC avec des stratégies pures on vérifie directement que :

$$-1 = \max_{1\leq i\leq |A|} \min_{1\leq j\leq |B|} (G_A)_{ij} < \min_{1\leq j\leq |B|} \max_{1\leq i\leq |A|} (G_A)_{ij} = 1$$

Tandis qu'avec des stratégies mixtes on a :

$$\max_p \min_q \mathbb{E}_{p,q}[g_A] = \min_q \max_p \mathbb{E}_{p,q}[g_A] = 0$$

Preuve. Il n'y a pas besoin de la programmation linéaire pour le prouver ici. En effet :

$$\begin{aligned} \mathbb{E}_{p,q}[g_A] &= p^t.G_A.q \\ &= \begin{pmatrix} p_1 & p_2 & p_3 \end{pmatrix} \cdot \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} \\ &= \begin{pmatrix} p_1 & p_2 & p_3 \end{pmatrix} \cdot \begin{pmatrix} q_3 - q_2 \\ q_1 - q_3 \\ q_2 - q_1 \end{pmatrix} \\ &= p_1(q_3 - q_2) + p_2(q_1 - q_3) + p_3(q_2 - q_1) \\ &= q_1(p_2 - p_3) + q_2(p_3 - p_1) + q_3(p_1 - p_2) \end{aligned}$$

Si l'on fixe la stratégie d'Alice en posant $\hat{p} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}^t$ alors on constate que quelle que soit la stratégie q de Bob, on aura toujours $\mathbb{E}_{\hat{p},q}[g_A] = 0$. D'où :

$$\max_p \min_q \mathbb{E}_{p,q}[g_A] \geq \min_q \mathbb{E}_{\hat{p},q} = 0$$

Si l'on fixe la stratégie de Bob en posant $\hat{q} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}^t$ alors on constate que quelle que soit la stratégie p d'Alice, on aura toujours $\mathbb{E}_{p,\hat{q}}[g_A] = 0$. D'où :

$$\min_q \max_p \mathbb{E}_{p,q}[g_A] \leq \max_p \mathbb{E}_{p,\hat{q}} = 0$$

Or on peut montrer facilement grâce à un raisonnement similaire à celui sur les stratégies pures que :

$$\max_p \min_q \mathbb{E}_{p,q}[g_A] \leq \min_q \max_p \mathbb{E}_{p,q}[g_A]$$

Donc finalement :

$$0 \leq \max_p \min_q \mathbb{E}_{p,q}[g_A] \leq \min_q \max_p \mathbb{E}_{p,q}[g_A] \leq 0$$

D'où :

$$\max_p \min_q \mathbb{E}_{p,q}[g_A] = \min_q \max_p \mathbb{E}_{p,q}[g_A] = 0$$

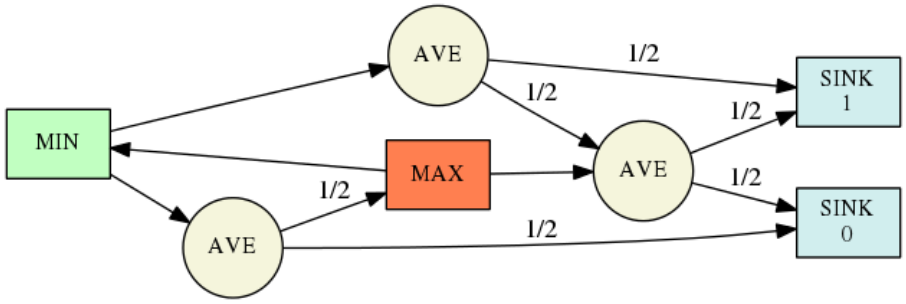
□

2.2 Les *Simple Stochastic Games*

La recherche autour des jeux stochastiques a été initiée par L. Shapley dans [Sha53]. Plus tard, A. Condon dans [Con92] a introduit le modèle des *Simple Stochastic Games*, un cas particulier du précédent, qui est en lui-même suffisamment intéressant pour être étudié en profondeur.

2.2.1 Définition

Figure 2.2 – Exemple de plateau de jeu



Définition 12. *On appelle plateau de jeu de SSG tout graphe orienté* $G = (V, E)$ *muni d'une partition* $(V_{MAX}, V_{MIN}, V_{AVE}, V_{SINK})$ *de l'ensemble des sommets* V *et d'une fonction de valeur* $val : V_{SINK} \to \mathbb{R}$*, vérifiant :*

$$\forall v \in V_{MAX} \cup V_{MIN} \cup V_{AVE}, d^+(v) = 2 \tag{2.26}$$
$$\forall v \in V_{SINK}, d^+(v) = 0 \tag{2.27}$$

La figure 2.2 donne un exemple de plateau de jeu et une représentation graphique de celui-ci. Ici on a indiqué la classe du sommet à l'intérieur de celui-ci, accompagné de sa valeur dans le cas des sommets SINK.

Définition 13. *On appelle SSG l'association d'un plateau de jeu $G = (V, E)$ et d'un sommet $d \in V$ appelé sommet de départ.*

Il y a deux joueurs que nous appellerons désormais MAX et MIN (respectivement Alice et Bob), ainsi qu'une source de hasard qui peut être considérée comme un joueur indépendant appelé AVE qui tirerait ses décisions à pile ou face.

Un unique pion est placé sur le sommet de départ d. Ensuite, ce pion se déplace de sommet en successeur. C'est à chaque fois l'un des joueurs qui choisit sur quel successeur ira le pion : chaque sommet est contrôlé soit par MAX, soit par MIN, soit par AVE. La répartition des sommets entre les trois joueurs est fixée au départ par la partition de l'ensemble V et est donc conservée tout au long de de la partie.

Le jeu se termine lorsque le pion arrive sur un sommet n'ayant pas de successeur. Un tel sommet est appelé SINK, et porte une certaine valeur : c'est le montant que devra payer le joueur MIN au joueur MAX lorsque la partie se termine sur ce sommet (s'il est négatif, c'est le joueur MAX qui devra payer la valeur absolue du montant au joueur MIN). Par ailleurs dans le cas où le jeu tournerait indéfiniment, on considère alors par convention que le joueur MIN ne paie rien au joueur MAX, ce qui revient à atteindre un SINK de valeur nulle.

Les gains des joueurs étant ainsi définis, un SSG est bien un jeu à somme nulle. Le but du joueur MAX va donc être de faire terminer le jeu sur un sommet de type SINK de valeur la plus grande possible, et celui du joueur MIN de faire terminer le jeu sur un sommet de valeur la plus petite possible, d'où leurs noms respectifs.

On note $V_{POS} = V_{MAX} \cup V_{MIN} \cup V_{AVE}$ l'ensemble des sommets non terminaux. La définition du plateau de jeu requiert que chacun de ces sommets ait exactement deux successeurs.

Dans le modèle considéré ici, nous nous restreindrons à des plateaux de jeu ayant exactement deux sommets terminaux notés V_0 et V_1 de valeur respective 0 et 1, et les transitions effectuées par le joueur AVE auront toujours pour distribution de probabilités $(\frac{1}{2}, \frac{1}{2})$. Cette restriction est sans

conséquences car on peut facilement ramener la résolution d'un SSG quelconque à celle d'un SSG respectant ces limitations (voir [AM09] de plus amples détails).

2.2.2 Intérêt

D'autres modèles de jeux ont été mis en avant et étudiés. On peut citer en particulier les *Parity Games* et les *Mean-Payoff Games*.

En fait, les SSG constituent un modèle plus général que ceux-ci, au sens où tous ces jeux peuvent se ramener facilement à un SSG de taille similaire. Pour être plus précis, il existe un algorithme nécessitant un temps polynômial par rapport à la taille du jeu de base qui donne un SSG de taille polynômiale par rapport à la taille du jeu de base.

Par ailleurs si l'on introduit du hasard dans ces modèles, donnant les *Stochastic Parity Games* (SPG) et les *Stochastic Mean-Payoff Games* (SMPG), alors les trois modèles deviennent équivalents [1] comme cela est montré dans [AM09]. L'intérêt de travailler sur l'un de ces modèles, en l'occurence les SSG, est démultiplié par ce résultat.

L'avantage des SSG sur les SPG ou les SMPG réside dans leur simplicité, comme l'indique leur nom : le déroulement et la fin d'une partie sont simples à expliquer.

2.2.3 Stratégies déterministes

Une stratégie pour le joueur MAX détermine la position suivante du pion en fonction de l'historique de la partie, quand celle-ci arrive sur un sommet MAX. Ainsi on peut la représenter mathématiquement par une application :

$$\sigma : (\bigcup_{k\in\mathbb{N}} {V_{POS}}^k) \times V_{MAX} \to V_{POS} \tag{2.28}$$

Une stratégie est dite positionnelle si elle vérifie la propriété suivante :

$$\forall h \in \bigcup_{k\in\mathbb{N}} {V_{POS}}^k, \forall h' \in \bigcup_{k\in\mathbb{N}} {V_{POS}}^k, \forall v \in V_{MAX}, \sigma(h,v) = \sigma(h',v) \tag{2.29}$$

Autrement dit la stratégie ne dépend que de la position courante du pion.

1. i.e. on peut passer d'un jeu d'un modèle à un jeu d'un autre modèle de taille polynômiale par rapport à celle du jeu initial, avec un algorithme nécessitant un temps polynômial.

Dans ce cas on note plus simplement :

$$\sigma : V_{MAX} \to V_{POS} \tag{2.30}$$

On définit de façon similaire les stratégies quelconques et positionnelles pour le joueur MIN, notées τ.

Valeur à stratégies fixées

À cause des sommets de type AVE, pour des stratégies MAX et MIN fixées le résultat ne sera pas nécessairement le même à chaque partie. En conséquence, tant que la partie n'a pas été jouée, il ne peut être exprimé qu'en termes probabilistes. Ainsi pour comparer deux stratégies d'un même joueur, on compare les espérances de gain du joueur en question sous chacune de ces stratégies.

On définit la valeur $val_{\sigma,\tau}(v)$ d'un sommet $v \in V$ sous les stratégies $(\sigma, \tau) \in S_{MIN} \times S_{MAX}$ comme étant l'espérance du gain du joueur MAX pour une partie commençant sur ce sommet, où le joueur MAX joue selon la statégie σ et le joueur MIN selon la stratégie τ par :

$$val_{\sigma,\tau}(v) = \mathbb{E}_{\sigma,\tau}[g_A^v] \tag{2.31}$$

Avec le modèle simplifié où les sommets SINK sont V_0 et V_1, le joueur MAX a un gain nul si la partie ne termine pas ou termine sur V_0, et un gain unitaire si la partie se termine en V_1. Ainsi, si on note « $v \to V_1$ » l'évènement « à partir du sommet v on rejoint le sommet V_1 », on a l'égalité suivante :

$$val_{\sigma,\tau}(v) = \mathbb{P}_{\sigma,\tau}(v \to V_1) \tag{2.32}$$

Valeur assurée

Si on fixe la stratégie σ du joueur MAX, le joueur MAX assure un certain gain noté $val_\sigma(v)$. Il s'agit du gain obtenu quand le joueur MIN joue de la meilleure façon possible :

$$val_\sigma(v) = \min_{\tau \in S_{MIN}} v_{\sigma,\tau} \tag{2.33}$$

On définit la meilleure réponse $\tau(\sigma)$ à σ comme étant la stratégie de MIN atteignant cette limite. On a donc :

$$val_\sigma(v) = v_{\sigma,\tau(\sigma)} \tag{2.34}$$

De même si on fixe la stratégie τ du joueur MIN, le joueur MIN assure un certain gain noté $val_\tau(v)$ vérifiant :

$$val_\tau(v) = \max_{\sigma \in S_{MAX}} v_{\sigma,\tau} \tag{2.35}$$

On définit la meilleure réponse $\sigma(\tau)$ à τ comme étant la stratégie de MAX atteignant cette limite. On a donc :

$$val_\tau(v) = v_{\sigma(\tau),\tau} \tag{2.36}$$

2.2.4 Calcul des valeurs des sommets à stratégies fixées

On fixe la stratégie σ de MAX et la stratégie τ de MIN. On veut connaître la valeur des sommets, c'est-à-dire l'espérance par rapport aux choix faits par AVE du gain de MAX pour une partie commençant sur le sommet considéré.

Dans ce cas il n'y a plus de choix mais que du hasard : le jeu se ramène à une chaîne de Markov. Pour calculer les valeurs il suffit de résoudre un système linéaire ce qui se fait en temps polynômial par rapport au nombre d'inconnues, c'est-à-dire au nombre de sommets de type AVE.

2.2.5 Représentation matricielle

Si on n'autorise à chaque joueur que les stratégies positionnelles, un SSG peut se représenter comme un jeu matriciel. La matrice de gain associée est de taille exponentielle par rapport à la taille du problème et donne l'espérance de gain en fonction des stratégies de chacun des deux joueurs.

Si Alice est le joueur MAX et Bob le joueur MIN, alors à chaque nœud MIN (resp. MAX) le joueur a deux choix, et ainsi :

$$|S_A| = 2^{|V_{MAX}|} \tag{2.37}$$

$$|S_B| = 2^{|V_{MIN}|} \tag{2.38}$$

Chapitre 3

Évolution de la complexité en fonction du nombre de circuits

Après l'étude bibliographique autour des SSG, j'ai pu avec l'aide de mes encadrants m'intéresser à une question pour le moment ouverte. Je présenterai dans cette section les résultats obtenus au cours de mon stage.

Un circuit de longueur n dans un graphe orienté $G = (V, E)$ est une suite de sommets $(c_0, \ldots, c_n) \in V^{n+1}$ avec $c_0 = c_n$ telle que $\forall 0 \leq i \leq n-1, (c_i, c_{i+1}) \in E$. Autrement dit, en parcourant un circuit on avance de successeur en successeur jusqu'à revenir au point de départ. Le circuit est dit élémentaire s'il ne passe pas deux fois par le même sommet, c'est-à-dire qu'en dehors de c_0 et c_n tous les sommets sont deux à deux distincts.

Comme nous le verrons dans un premier temps, il est facile de résoudre un SSG dans le cas où le graphe associé ne présente aucun circuit. La question que nous nous sommes posé est alors la suivante : peut-on trouver des algorithmes efficaces dans le cas où il y a des circuits en petit nombre ?

Notations

Pour un graphe orienté donné $G = (V, E)$ nous utiliserons les notations suivantes :

S_v : ensemble des successeurs d'un sommet v ;
$E \leftarrow E \setminus \{(u, v)\}$: on retire l'arête de G reliant u à v ;
$E \leftarrow E \cup \{(u, v)\}$: on ajoute à G une arête reliant u à v.

De plus nous supposerons prédéfinies les fonctions suivantes :
— NvGraphe() : crée un nouveau graphe vide ;
— NvSommet(G, t) : crée un nouveau sommet dans G, de type $t \in$

{AVE,MAX,MIN} ;
— NvSommet($G, SINK, v$) : crée un nouveau sommet de type SINK dans G et lui associe la valeur v ;
— TypeSommet(u) : retourne le type du sommet u :
— ValSink(u) : récupère la valeur associée au sommet $u \in V_{SINK}$;
— Succ($u, G, \neq v$) : récupère le successeur dans G de u qui n'est pas v (suppose que u a exactement deux successeurs) ;
— NvVecteurValeurs(G) : crée un nouveau vecteur de valeurs indexé par les sommets de G ;
— TriTopologique(G) : retourne un tri topologique du DAG G, comme expliqué dans la section dédiée.

3.1 Jeu sans circuit

Un graphe sans circuit est communément appelé DAG pour *Directed Acyclic Graph*. Nous nous intéressons ici à ce type très particulier de graphe, pour lequel on mettra en évidence un algorithme de résolution RésDAG.

3.1.1 Tri topologique

Un tri topologique d'un DAG est une énumération de ses sommets dans laquelle n'importe quel sommet apparait avant tous ses successeurs. Il s'agit d'une opération classique qui se réalise en faisant un parcours en profondeur du DAG, donc en un temps linéaire par rapport au nombre de sommets.

Plus formellement, dans un graphe orienté $G = (V, E)$ sans circuit, un tri topologique $t = (t_i)_{1 \leq i \leq |V|}$ vérifie :

$$\forall v \in V, \exists 1 \leq i \leq |V| : t_i = v$$
$$\forall 1 \leq i \leq |V|, \forall 1 \leq j \leq |V|, (t_i, t_j) \in E \rightarrow i < j$$

3.1.2 Algorithme de résolution

L'algorithme RésDAG commence par traiter les sommets les plus loins dans le tri topologique puis s'occupe de remonter les valeurs. On sait que les successeurs du sommet courant ont déjà été traités grâce au parcours dans l'ordre inverse du tri topologique.

Propriété 3. *L'algorithme* RésDAG *termine et à sa sortie, le vecteur de valeurs calculé est bien défini et optimal.*

Algorithme 1 : RésDAG - Résolution d'un SSG sans circuit

Entrée : un SSG sans circuit $G = (V, E)$;
un tri topologique $t = (t_i)_{1 \leq i \leq |V|}$ de G.
Sortie : un vecteur de valeurs optimal pour G.

```
1  val ← NvVecteurValeurs(G)
2  pour chaque s dans V_SINK fais
3  |  val[s] ← ValSink(s)
4  fait
5  pour chaque v dans t parcouru en ordre inverse fais
6  |  si v ∈ V_MAX alors
7  |  |  val[v] ← max_{w∈S_v} val[w]
8  |  sinon si v ∈ V_MIN alors
9  |  |  val[v] ← min_{w∈S_v} val[w]
10 |  sinon si v ∈ V_AVE alors
11 |  |  val[v] ← (Σ_{w∈S_v} val[w]) / |S_v|
12 |  finsi
13 fait
14 retourne val
```

Démonstration. La terminaison est évidente car t comporte un nombre fini $|V|$ d'éléments : il y a exactement |V| tours dans la boucle principale.

On montre ensuite que le vecteur calculé est bien défini et optimal en montrant qu'à la sortie de l'algorithme, tous les sommets ont une valeur bien définie et les équations locales d'optimalité sont satisfaites pour ceux qui sont non-terminaux.

Comptons les tours de boucle de 1 à $|V|$, et notons v_i le sommet traité lors du i-ème tour de boucle. Alors pour tout i on a $v_i = t_{|V|-i+1}$. On déduit directement des propriétés du tri topologique que :

$$\forall v \in V, \exists 1 \leq i \leq |V| : v_i = v$$
$$\forall 1 \leq i \leq |V|, \forall 1 \leq j \leq |V|, (v_i, v_j) \in E \rightarrow i > j$$

Donc d'une part tous les sommets sont parcourus, et d'autre part les successeurs d'un sommet quelconque ont été parcourus dans un tour antérieur.

On montre alors par récurrence la propriété suivante : à la fin du i-ème tour de boucle, les sommets $v_1, \ldots, v_i$ portent des valeurs bien définies, qui satisfont les équations locales d'optimalité quand ces sommets sont non-terminaux.

Celle-ci est vrai pour le premier tour. En effet le sommet v_1 n'a pas de successeur, sans quoi ce successeur serait placé après lui dans l'ordre topologique, ce qui est contradictoire. Donc v_1 est un sommet terminal, donc de type SINK, et porte déjà une valeur bien définie.

Supposons la propriété vraie pour un certain $1 \leq i \leq |V| - 1$, et plaçons-nous à la fin du tour de boucle $i + 1$.

Si v_{i+1} est terminal, la propriété pour $i + 1$ se déduit directement du fait qu'elle soit vraie pour i et qu'en tant que sommet de type SINK, v_{i+1} porte déjà une valeur bien définie. Donc la propriété est vraie pour $i + 1$.

Supposons alors v_{i+1} non terminal. Soient j et k tels que les deux successeurs de v_{i+1} soient v_j et v_k. On a vu plus haut que l'on a alors $j < i + 1$ et $k < i + 1$ grâce aux propriétés du tri topologique. Donc par hypothèse de récurrence, ces deux sommets portent des valeurs bien définies. Donc v_{i+1} reçoit lui-même une valeur bien définie, laquelle satisfait bien l'équation locale d'optimalité car c'est ainsi qu'elle a été fixée ligne 4, 7 ou 10. Donc la propriété est vraie pour $i + 1$.

On déduit de tout cela qu'à la fin de l'algorithme, les sommets $v_1, \ldots, v_{|V|}$ ont une valeur bien définie et les équations locales d'optimalité sont satisfaites. Comme l'énumération est exhaustive, on aboutit à un vecteur de valeurs bien défini et optimal. □

Propriété 4. *L'algorithme* RésDAG *s'exécute en un temps* $\Theta(|V|)$.

Justification. On a $|V_{SINK}| \leq |V|$ tours de la première boucle, qui n'exécute que des opérations élémentaires et prend donc un temps $\Theta(|V_{SINK}|)$.

On a $|V|$ tours de la deuxième boucle. Puisque pour tout $v \in V$, $|S_v| \leq 2$, les calculs de minimum, maximum ou moyenne prennent chacun un temps $\Theta(1)$, d'où un temps $\Theta(|V|)$ pour cette boucle.

Finalement le temps d'exécution global est en $\Theta(|V|)$. □

3.2 Jeu avec un seul circuit

On considère maintenant des graphes où il n'y a qu'un seul circuit et on met en évidence un algorithme de résolution RésCircUniq.

3.2.1 SSG simple à un seul circuit

On propose de s'intéresser d'abord à un cas particulier de SSG à un seul circuit : il s'agit du cas où les seuls nœuds non-terminaux sont les nœuds du circuit. De plus, on autorise n'importe quelle valeur fractionnaire comprise

entre 0 et 1 pour les sommets de type SINK. On verra que l'on peut toujours se ramener à ce cas particulier pour résoudre le problème sur un SSG quelconque ayant un seul circuit.

On notera $(c_i)_{0\leq i\leq n}$ les sommets du circuit avec $c_0 = c_n$, $(v_i)_{0\leq i\leq n-1}$ leurs valeurs respectives, et $(s_i)_{0\leq i\leq n-1}$ les valeurs des sommets de type SINK qu'ils ont respectivement pour successeur.

Cas d'un circuit de nœuds AVE

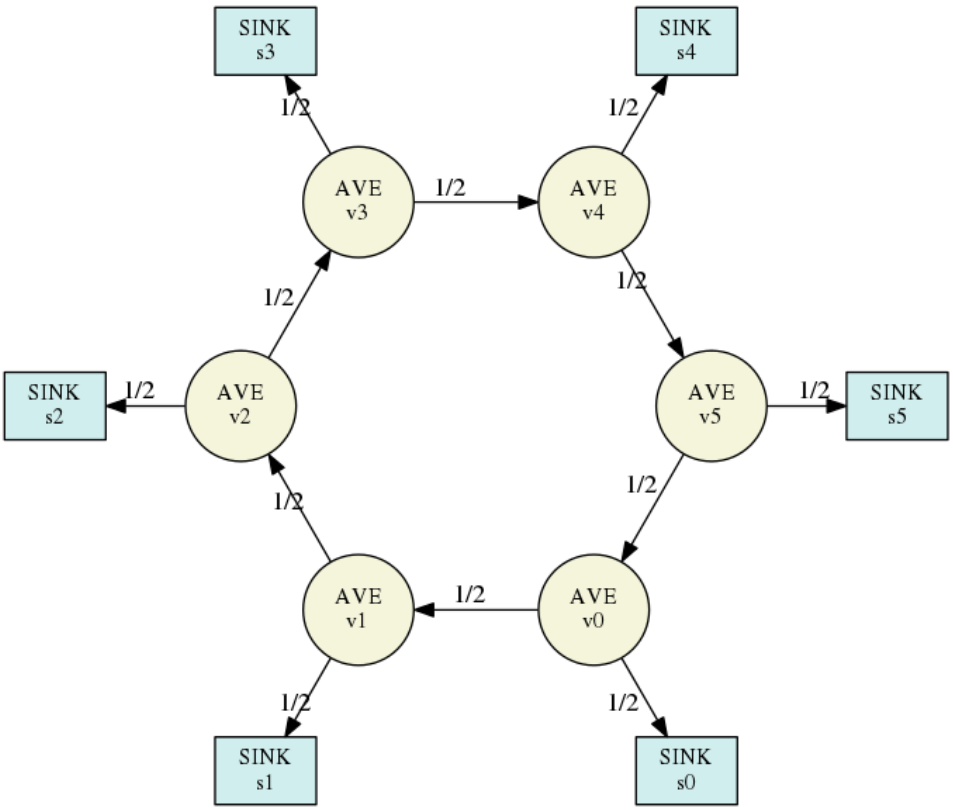


FIGURE 3.1 – Exemple de graphe simple à circuit unique de nœuds AVE

On s'intéresse ici à un cas particulier de ce cas particulier : celui où tous les nœuds du circuit sont de type AVE. Le type de graphe en question est représenté sur la figure 3.1 avec les notations précédemment définies.

Dans ce cas, les valeurs peuvent être calculées directement à partir des valeurs des sommets terminaux. En effet le système d'équations est le suivant :

$$
\begin{aligned}
v_0 &= \frac{s_0 + v_1}{2} \\
v_1 &= \frac{s_1 + v_2}{2} \\
&\cdots \\
v_{n-2} &= \frac{s_{n-2} + v_{n-1}}{2} \\
v_{n-1} &= \frac{s_{n-1} + v_0}{2}
\end{aligned}
$$

En déroulant les équations on obtient l'égalité :

$$
\begin{aligned}
v_0 &= \frac{s_0}{2} + \frac{s_1}{4} + \cdots + \frac{s_{n-1}}{2^n} + \frac{v_0}{2^n} \\
&= \sum_{i=0}^{n-1} \frac{s_i}{2^{i+1}} + \frac{v_0}{2^n}
\end{aligned}
$$

D'où :

$$
\begin{aligned}
v_0 &= \frac{1}{1 - \frac{1}{2^n}} \sum_{i=0}^{n-1} \frac{s_i}{2^{i+1}} \\
&= \frac{2^n}{2^n - 1} \sum_{i=0}^{n-1} \frac{s_i}{2^{i+1}} \\
&= \frac{\sum_{i=0}^{n-1} s_i 2^{n-1-i}}{2^n - 1} \\
&= \frac{\sum_{j=0}^{n-1} s_{n-1-j} 2^j}{2^n - 1}
\end{aligned}
$$

Remarquons que dans le cas où les sommets SINK sont réduits à 0 et 1, les valeurs peuvent s'écrire plus simplement comme le rapport entre un nombre entier écrit sur n bits et la valeur maximale d'un nombre écrit sur n bits :

$$
\begin{aligned}
v_0 &= \frac{(s_0 s_1 \dots s_{n-1})_2}{2^n - 1} \\
v_1 &= \frac{(s_1 s_2 \dots s_{n-1} s_0)_2}{2^n - 1} \\
v_2 &= \frac{(s_2 s_3 \dots s_{n-1} s_0 s_1)_2}{2^n - 1} \\
&\dots \\
v_{n-1} &= \frac{(s_{n-1} s_0 s_1 \dots s_{n-2})_2}{2^n - 1}
\end{aligned}
$$

Ce résultat permet donc notamment de voir que dans un SSG où les sommets SINK sont 0 et 1 uniquement, on peut générer n'importe quelle valeur s'écrivant sous la forme $\frac{x}{2^n-1}$ avec $x \in \mathbb{N}$ et $0 \leq x \leq 2^{n-1}$: il suffit d'assigner aux sommets SINK les valeurs des bits de x dans le bon ordre.

Ainsi, si on a affaire à un circuit de la forme précédente, les valeurs se calculent très facilement. Dans la suite nous supposerons donc que l'on dispose d'un algorithme ValCircBouclé. Cet algorithme prend en entrée un SSG simple à un circuit, qui n'a pas nécessairement que des sommets AVE dans son circuit.

Il retourne en un temps $\Theta(|V|)$ le vecteur de valeurs obtenu lorsque les joueurs MIN et MAX font partout le choix de rester sur le circuit. En effet dans ce cas le calcul revient au calcul sur un circuit AVE que nous venons de détailler.

Cas général du SSG simple à un circuit

Propriété 5. *L'algorithme 3 termine pour tout graphe ayant un seul circuit.*

Justification. En retirant une des arêtes de l'unique circuit de G à la ligne 3, nous avons transformé G en DAG. Puisque RésDAG termine pour tout DAG, et que ValCircBouclé termine pour tout graphe simple à un circuit, l'algorithme RésCircUniq termine pour tout graphe ayant un seul circuit. □

Propriété 6. *À la fin de l'algorithme 3, le vecteur de valeurs calculé est optimal.*

Démonstration. On distingue deux cas.

1^{er} **cas :** l'algorithme termine dans l'un des tours de boucle.

Algorithme 2 : RésCircUniq - Résolution d'un SSG à circuit unique

Entrée : un SSG simple à un circuit $G = (V, E)$;
l'unique circuit $(c_i)_{1 \leq i \leq n}$ de G.
Sortie : un vecteur de valeurs optimal.

```
1  pour i de 0 à n − 1 fais
2      si c_i ∈ V_MAX ou c_i ∈ V_MIN alors
3          E ← E \ {(c_i, c_{i+1})}
4          val ← RésDAG(G, TriTopologique(G))
5          E ← E ∪ {(c_i, c_{i+1})}
6          si c_i ∈ V_MAX et val[c_i] ≥ val[c_{i+1}] alors
7              retourne val
8          sinon si c_i ∈ V_MIN et val[c_i] ≤ val[c_{i+1}] alors
9              retourne val
10         finsi
11     finsi
12 fait
13 ValCircBouclé (G, c) retourne val
```

Dans ce cas l'algorithme termine soit à la ligne 7, soit à la ligne 9. Notons i_0 la valeur de i au tour de boucle final et G_0 le DAG obtenu à partir de G en retirant l'arête (c_{i_0}, c_{i_0+1}). L'appel à RésDAG calcule un vecteur de valeurs optimal pour G_0, donc puisque les équations locales d'optimalité pour G_0 sont vérifiées en tout sommet on a :

$$\forall \begin{cases} 1 \leq i \leq n-1 \\ i \neq i_0 \end{cases}, \begin{cases} c_i \in V_{MAX} \rightarrow val[c_i] = max(val[s_i], val[c_{i+1}]) \\ c_i \in V_{MIN} \rightarrow val[c_i] = min(val[s_i], val[c_{i+1}]) \\ c_i \in V_{AVE} \rightarrow val[c_i] = \frac{val[s_i]+val[c_{i+1}]}{2} \end{cases}$$
$$val[c_{i_0}] = val[s_{i_0}]$$

Seul c_{i_0} a un ensemble de successeurs différent dans les graphes G et G_0, donc à part pour ce sommet les équations locales d'optimalité sont inchangées par le passage de G_0 à G, et par conséquent pour tout $i \neq i_0$ les équations locales d'optimalité pour G sont vérifiées par le vecteur calculé. On montre alors que l'équation locale d'optimalité est également vérifiée pour c_{i_0} dont les successeurs sont s_{i_0} et c_{i_0+1} :

— si l'algorithme termine à la ligne 7 alors puisque l'on est entré dans la condition on a $c_{i_0} \in V_{MAX}$ et $val[c_{i_0}] \geq val[c_{i_0+1}]$, or on a vu ci-dessus que $val[c_{i_0}] = val[s_{i_0}]$, par suite :

$$c_{i_0} \in V_{MAX} \wedge val[c_{i_0}] = max(val[s_{i_0}], val[c_{i_0+1}])$$

— si l'algorithme termine à la ligne 9 alors puisque l'on est entré dans la condition on a $c_{i_0} \in V_{MIN}$ et $val[c_{i_0}] \leq val[c_{i_0+1}]$, or on a vu ci-dessus que $val[c_{i_0}] = val[s_{i_0}]$, par suite :

$$c_{i_0} \in V_{MIN} \wedge val[c_{i_0}] = min(val[s_{i_0}], val[c_{i_0+1}])$$

2^{eme} **cas :** l'algorithme termine après la boucle.

On montre tout d'abord que dans ce cas, il n'existe pas de stratégie optimale dans laquelle le joueur choisit de ne pas rester sur le circuit.

Soit σ une stratégie de MAX vérifiant :

$$\exists 1 \leq i \leq n-1 : c_i \in V_{MAX} \wedge \sigma(c_i) \neq c_{i+1}$$

Et montrons qu'une telle stratégie n'est pas optimale dans G. Soit i_0 tel que $\sigma(c_{i_0}) \neq c_{i_0+1}$. Autrement dit, $\sigma(c_i) = s_i$. On note G_0 le graphe G après avoir retiré l'arête (c_i, c_{i+1}). Puisque $\sigma(c_i) = s_i$, σ est une stratégie (valide) de MAX dans G_0 et le vecteur de valeurs associé est identique, donc :

$$val^{G_0}_{\sigma,\tau(\sigma)} = val^{G}_{\sigma,\tau(\sigma)}$$

Plaçons-nous dans le tour de boucle où $i = i_0$. On a alors calculé le vecteur de valeurs optimal de G_0, noté val^{G_0}.

Si σ n'est pas optimale dans G_0, alors le vecteur de valeurs associé ne satisfait pas les équations locales d'optimalité dans G_0, donc dans G non plus. En conséquence, la stratégie σ n'est pas optimale dans G.

Supposons alors que σ est optimale dans G_0. Par unicité du vecteur optimal de valeurs, on a :

$$val^{G_0}_{\sigma,\tau(\sigma)} = val^{G_0}$$

Or puisque l'algorithme n'a pas fini au tour où $i = i_0$, et puisque $c_i \in V_{MAX}$, cela signifie que :

$$val^{G_0}(c_i) < val^{G_0}(c_{i+1})$$

Donc le vecteur de valeurs associé à $(\sigma, \tau(\sigma))$ ne vérifie pas l'équation locale d'optimalité en c_i dans G. Par conséquent la stratégie σ n'est pas optimale dans G.

Par contraposée, une stratégie optimale σ de MAX dans G vérifie nécessairement :

$$\forall 0 \leq i \leq n-1 : c_i \notin V_{MAX} \vee \sigma(c_i) = c_{i+1}$$

Qui s'écrit de façon équivalente :

$$\forall 0 \leq i \leq n-1 : c_i \in V_{MAX} \rightarrow \sigma(c_i) = c_{i+1}$$

Or il n'y a qu'une seule stratégie MAX positionnelle vérifiant cette propriété. Comme on sait qu'il existe nécessairement une stratégie MAX positionnelle optimale, c'est celle-ci.

Par un raisonnement similaire on peut montrer qu'il y a une et une seule stratégie MIN positionnelle optimale τ dans G, à savoir celle qui vérifie :

$$\forall 0 \leq i \leq n-1 : c_i \in V_{MIN} \rightarrow \tau(c_i) = c_{i+1}$$

Le vecteur de valeurs associé à ces deux stratégies est celui calculé par ValCircBouclé, qui est par conséquent nécessairement optimal. □

Propriété 7. *L'algorithme* RésCircUniqSimple *s'exécute en un temps* $O(|V|^2)$.

Justification. Dans le pire des cas, on a $|V|$ tours de la boucle principale qui prennent chacun un temps $\Theta(|V|)$ en raison de l'appel à TriTopologique et de celui à RésDAG, d'où un temps d'exécution en $\Theta(|V|^2)$ pour cette boucle.

Puis a lieu l'appel à ValCircBouclé qui prend un temps $\Theta(|V|)$.

Donc dans le pire des cas, l'algorithme s'exécute en un temps $\Theta(|V|^2)$. □

3.2.2 Résolution pour un graphe quelconque ayant un seul circuit

Quand il n'y a qu'un circuit, on peut toujours se ramener au cas précédents où les sommets n'appartenant pas au circuit sont terminaux, le traiter, et répercuter les résultats obtenus sur le graphe initial. C'est ce que fait l'algorithme RésCircUniq.

Théorème 1. *L'algorithme* RésCircUniq *retourne un vecteur de valeurs optimal pour le graphe* G.

Démonstration. Notons G_0 l'état de G à la fin de la première boucle. On a retiré de G toutes les arêtes faisant partie de l'unique circuit, donc G_0 est un DAG. Donc RésDAG retourne un vecteur de valeurs *val* optimal pour G_0.

Le rajout des arêtes du circuit ne modifie les successeurs que pour les sommets appartenant au circuit, donc les équations locales d'optimalité sont les mêmes dans G et dans G_0 pour tous les sommets n'appartenant pas au

Algorithme 3 : RésCircUniq - Résolution d'un graphe à circuit unique

Entrée : un SSG à un circuit $G = (V, E)$;
l'unique circuit $(c_i)_{0 \leq i \leq n}$ de G.
Sortie : un vecteur de valeurs optimal.

pour i de 0 à $n-1$ **fais**
  $E \leftarrow E \setminus \{(c_i, c_{i+1})\}$
fait
$val \leftarrow$ RésDAG(G, TriTopologique(G))
pour i de 0 à $n-1$ **fais**
  $E \leftarrow E \cup \{(c_i, c_{i+1})\}$
fait
$G' \leftarrow$ NvGraphe()
pour i de 0 à $n-1$ **fais**
  $c'_i \leftarrow$ NvSommet(G', TypeSommet(c_i))
  $v_i \leftarrow$ Succ($c_i, G, \neq c_{i+1}$)
  $s'_i \leftarrow$ NvSommet($G', SINK, val[v_i]$)
fait
$c'_n \leftarrow c'_0$
pour i de 0 à $n-1$ **fais**
  $E \leftarrow E \cup \{(c'_i, s'_i)\}$
  $E \leftarrow E \cup \{(c'_i, c'_{i+1})\}$
fait
$val' \leftarrow$ RésCircUniqSimple(G')
pour i de 0 à $n-1$ **fais**
  $val[c_i] \leftarrow val'[c'_i]$
fait
retourne val

circuit. Donc pour tous ces sommets, les équations locales d'optimalité sont vérifiées par val dans G.

À l'issue de l'appel de RésCircUniqSimple, le vecteur val' vérifie les équations locales d'optimalité dans G', donc étant donné les valeurs auxquelles ont été initilisés les sommets SINK on a :

$$\forall 1 \leq i \leq n-1, \begin{cases} c'_i \in V_{MAX} \rightarrow val'[c'_i] = max(val[v_i], val'[c'_{i+1}]) \\ c_i \in V_{MIN} \rightarrow val'[c'_i] = min(val[v_i], val'[c'_{i+1}]) \\ c_i \in V_{AVE} \rightarrow val'[c'_i] = \frac{val[v_i]+val'[c'_{i+1}]}{2} \end{cases}$$

Donc après la boucle finale, puisque :

$$\begin{gathered} c'_i \in V_{MAX} \leftrightarrow c_i \in V_{MAX} \\ c'_i \in V_{MIN} \leftrightarrow c_i \in V_{MIN} \\ c'_i \in V_{AVE} \leftrightarrow c_i \in V_{AVE} \end{gathered}$$

On obtient :

$$\forall 1 \leq i \leq n-1, \begin{cases} c_i \in V_{MAX} \rightarrow val[c_i] = max(val[v_i], val[c_{i+1}]) \\ c_i \in V_{MIN} \rightarrow val[c_i] = min(val[v_i], val[c_{i+1}]) \\ c_i \in V_{AVE} \rightarrow val[c_i] = \frac{val[v_i]+val[c_{i+1}]}{2} \end{cases}$$

Donc les équations locales d'optimalité sont également vérifiées pour les sommets appartenant au circuit, et par conséquent le vecteur de valeurs retourné est optimal pour G.

□

Propriété 8. *L'algorithme* RésCircUniq *s'exécute en un temps* $O(|V| + n^2)$ *où* n *est la taille de l'unique circuit du SSG considéré.*

Justification. Les deux opérations coûteuses sont l'appel à RésDAG sur G et celui à RésCircUniqSimple sur G'.

Le premier prend un temps $\Theta(|V|)$ et le second un temps $O(|V'|^2)$ avec $|V'| = 2n$.

Donc globalement, on a un temps d'exécution en $O(|V| + n^2)$. Cette expression pourra bien sûr être simplifiée si on dispose de plus d'informations sur le graphe considéré, et que l'on sait par conséquent quel est le terme dominant entre $|V|$ et n^2. □

Conclusion

Ce stage aura été formateur et enrichissant sur plusieurs points.

Pour commencer, sur le plan des connaissances. En un laps de temps plutôt restreint, j'ai pu acquérir un nombre important de nouvelles connaissances par rapport à la théorie des jeux, la programmation linéaire, les chaînes de Markov et autres processus de décision de Markov.

Je suis particulièrement satisfait d'avoir pu, grâce à mon travail sur les SSG, entraîner mon esprit aux principaux types de raisonnements auxquels je pense que l'on a affaire en informatique théorique : d'une part la compréhension du modèle mathématique mis en œuvre, d'autre part l'écriture d'algorithmes et surtout de leurs preuves, et enfin la mise en œuvre concrète de ces derniers sur ordinateur.

Sur le plan technique, j'ai pu travailler avec le langage Python, que je connaissais déjà avant le stage, et me familiariser avec l'outil git. Ce dernier permet de synchroniser le travail des différents collaborateurs d'un projet, et est largement utilisé à travers le monde. Aussi y avoir été confronté sera un atout, et je pourrai utiliser des solutions similaires pour mes projets à venir.

J'ai également pu découvrir me faire une idée du métier de chercheur. Je me suis ainsi familiarisé entre autres avec la lecture de publications, la réflexion personnelle sur un sujet inédit et l'écriture d'un document scientifique.

Sur le plan humain, j'ai pu cotoyer des chercheurs et des doctorants et profiter de leurs points de vue.

En dépit de sa courte durée ce stage m'aura donc été très bénéfique, et j'aurai appris beaucoup de choses.

Annexe : programmation linéaire

Dans cette annexe je présenterai la programmation linéaire et les résultats de base qui sont nécessaires en théorie des jeux.

Problèmes considérés

La programmation linéaire est un outil mathématique de recherche opérationnelle utilisé pour maximiser (ou minimiser) une fonction linéaire d'un certain nombre de variables, dites variables de décision. On cherche les valeurs optimales à donner à ces variables pour maximiser (ou minimiser) notre objectif, connaissant les contraintes qu'elles doivent vérifier qui sont des équations ou inéquations linéaires.

Ainsi la formulation mathématique d'un problème de programmation linéaire prend la forme suivante :

Maximiser (ou minimiser)

$$z = c_1 x_1 + ... + c_n x_n \tag{3.1}$$

sous les contraintes

$$a_{11} x_1 + ... + a_{1n} x_n \ (\leq, =, \geq) \ b_1 \tag{3.2}$$
$$... \tag{3.3}$$
$$a_{m1} x_1 + ... + a_{mn} x_n \ (\leq, =, \geq) \ b_m \tag{3.4}$$
$$x_1 \geq 0, ..., x_n \geq 0 \tag{3.5}$$

On peut utiliser les matrices suivantes pour exprimer le problème en termes matriciels :

$$x = (x_1, ..., x_n)^t \tag{3.6}$$
$$c = (c_1, ..., c_n)^t \tag{3.7}$$
$$b = (b_1, ..., b_m)^t \tag{3.8}$$
$$A = (a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n} \tag{3.9}$$

Dualité

Un problème de programmation linéaire est dit primal s'il est de la forme suivante :

Trouver $\mathbf{x}$ qui maximise

$$z = \mathbf{c}^t \mathbf{x} \tag{3.10}$$

sous les contraintes

$$A\mathbf{x} \leq \mathbf{b} \tag{3.11}$$
$$\mathbf{x} \geq 0 \tag{3.12}$$

Son problème dual est alors :

Trouver $\mathbf{y}$ qui minimise

$$z = \mathbf{b}^t \mathbf{y} \tag{3.13}$$

sous les contraintes

$$A^t \mathbf{y} \geq \mathbf{c} \tag{3.14}$$
$$\mathbf{y} \geq 0 \tag{3.15}$$

(todo : résultats)

Bibliographie

[NRTV07] N. Nisan, T. Roughgarden, É. Tardos, V. Varizani. *Algorithmic Game Theory.* Cambridge University Press, 2007.

[TVK10] R. Tripathi, E. Valkanova, É. Tardos, V. Varizani. *On Strategy Improvement Algorithms for Simple Stochastic Games.* 2010.

[Con92] A. Condon. *The Complexity of Stochastic Games.* Information and Computation, 1992.

[Sha53] L. Shapley. *Stochastic Games.* Proceedings of the National Academy of Sciences, USA, 1953.

[AM09] D. Andersson, P. B. Miltersen. *The Complexity of Solving Stochastic Games on Graphs.* 2009.

[Bai96] G. Baillargeon. *Programmation linéaire appliquée.* Les éditions SMG, 1996.