Travail préparatoire au mémoire

Samuel Dehouck

23 mai 2014

Table des matières

Chapitre 1

Introduction

Un système embarqué est un système autonome devant accomplir une série de tâches spécifiques. Ces systèmes embarqués se retrouvent partout dans notre quotidien : les voitures ou les téléphones portables en sont des exemples. De plus, ces systèmes sont généralement limités par leurs composantes : mémoire, batterie, ... Il est aussi important de savoir qu'une fois ces systèmes mis en service, il devient difficile d'effectuer des corrections dans le logiciel implémenté. Cela devient particulièrement problématique dans le cas de systèmes critiques (à bord d'un avion par exemple) car une simple erreur pourrait mener à une catastrophe. Il est donc important que les contrôleurs gérant ce genre de systèmes embarqués soient prouvés comme sans failles. Pour cela, une approche possible est d'essayer de synthétiser ces contrôleurs en se basant sur un modèle représentant l'environnement dans lequel le contrôleur fonctionnera. On peut voir cette synthèse comme un jeu dans lequel deux joueurs, le contrôleur et l'environnement, s'affrontent. Le but du contrôleur est de satisfaire une certaine propriété tandis que celui de l'environnement est de l'en empêcher. Un contrôleur est correct si la propriété est toujours satisfaite quelles que soient les actions de l'environnement. Dans ce document, nous nous intéresserons particulièrement à une catégorie de jeux : les jeux temporisés à coûts.

Chapitre 2

Etat de l'art

2.1 Automates temporisés

Afin de représenter des modèles avec un nombre fini de configurations et les analyser, les automates finis peuvent être utilisés efficacement. Cependant, pour représenter les comportements de modèles de systèmes temps réel (un avion par exemple), les automates finis ne suffisent plus. En effet, l'élement important de ces systèmes est l'évolution du temps, ce qui n'est pas exprimé par les automates finis. Ceux-ci peuvent donc être modifiés afin d'obtenir un modèle correspondant mieux à notre nouveau problème : les automates temporisés [2].

Un automate temporisé est un automate fini dont les états sont appelées "emplacements" avec un nombre fini d'horloges évoluant de manière continue. Ces horloges peuvent être remise à zéro (*reset*) par les transitions de l'automate. De plus, des contraintes de temps sont posées pour chaque transition et celles-ci doivent être respectées pour pouvoir prendre cette transition (*guard*). De plus, des contraintes sont aussi associées aux sommets qui doivent être respectées afin de pouvoir y rester (*invariant*).

Deux types de transitions existent :

- Discrètes : correspondent à un changement d'emplacement.
- Temporelles : correspondent à une consommation de temps dans un emplacement.

Voici un exemple simple d'automate temporisé :

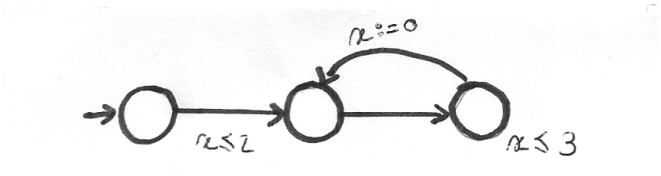

FIGURE 2.1 – Automate temporisé

2.2 Problème d'*accessibilité*

Le problème d'*accessibilité* est un problème de décision très étudié en vérification car il permet de détecter si des emplacements non sûrs sont atteints (par exemple dans le cas de processus concurrents, un emplacement correspondant à une violation de l'exclusion mutuelle). Il a été démontré que ce problème est PSPACE-complet pour des automates temporisés. L'outil *Uppaal Tiga* permet de manipuler des automates temporisés et vérifier des propriétés d'*accessibilité* en utilisant un algorithme basé sur le graphe de régions associé à l'automate de base.

Afin de pouvoir étudier le problème d'*accessibilité optimale* dans les automates temporisés, on peut leur ajouter une composante de coûts (ou poids). Ce dernier modèle, appelé automate temporisé à coûts[3], consiste en un automate temporisé dans lequel des coûts sont associés à chaque transition et à chaque emplacement.

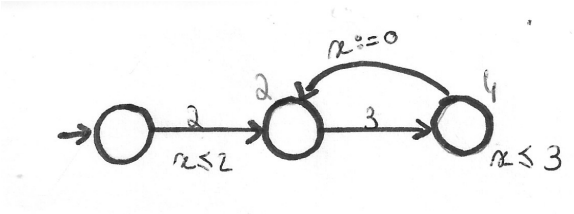

FIGURE 2.2 – Automate temporisé à coûts

Le problème d'*accessibilité optimale* revient donc à déterminer comment, à partir d'une certain emplacement, accéder à une autre en ayant le coût total (la somme des coûts des emplacements multipliés par le temps passé dans cet emplacement et des coûts des transitions prises) minimum. De plus, ce nouveau modèle permet de représenter des systèmes embarqués dans lesquels des consommations de ressources doivent être modélisées. Le problème d'*accessibilité optimale* dans un automate temporisé avec des coûts à la fois positifs et négatifs a été démontré comme étant PSPACE-complet[4].

2.3 Jeux temporisés à coûts

Les jeux sur des automates temporisés à coûts peuvent être définis de la manière suivante : deux joueurs s'affrontent, dont un seul est contrôlable, et se partagent l'ensemble des emplacements de l'automates. Le possesseur d'un emplacement va pouvoir décider de l'action à effectuer lorsque l'on s'y trouve. Le problème d'*accessibilité optimale dans des jeux* est de savoir si un joueur peut forcer l'accessibilité d'une place donnée tout en garantissant au maximum un certain coût fixé. Une solution en temps exponentiel à ce problème a été trouvée [1].

Dans le but de simplifier nos modèles, nous n'analyserons maintenant plus que des jeux sur des automates à coûts positifs.

Il a d'abord été prouvé que le coût optimal pour résoudre ces jeux est calculable sous la condition de non-zenoness stricte (chaque cycle dans l'automate à un coût différent de zero)[6]. Ensuite, l'existence d'une stratégie améliorante avec un coût limité a été prouvée indécidable dans un jeu avec 5 horloges[8]. Ce résultat a ensuite été raffiné à l'indécidabilité à partir de 3 horloges[5]. Pour l'instant, aucun résultat n'a été démontré pour des jeux avec 2 horloges.

Au vu de ces résultats sur l'indécidabilité, l'étude de stratégies a été restreinte au cas, plus simple, d'un automate temporisé à coûts avec une unique horloge. De plus, en vue de simplifier les notations, nous abrévieront jeux temporisé à coûts par *PTG* (*Priced Timed Games*).

2.4 Résolution de jeux temporisés à coûts

Le problème de *stratégies gagnantes à coût optimal* dans des jeux temporisés à coûts muni d'une horloge a été étudié [7]. Dans ce type de jeux un joueur cherche à minimiser le coût total et un second visant à le maximiser.

Le résultat principal de cette étude est que la fonction de coût optimal est calculable et affine par morceaux et qu'il existe toujours une strategie *positionnelle* et ε-optimale en chaque emplacement. Une stratégie est ε-optimale si elle s'écarte au plus de ε de la fonction de coût optimale.

L'idée générale de l'algorithme est de commencer par simplifier le jeu en un PTG dont l'horloge n'appartient qu'à l'intervalle [0,1]. Cela se fait en ajoutant une remise à zéro à chaque fois que l'horloge atteint 1. De plus, il est nécessaire de retenir la partie discrète de l'horloge. La seconde étape consiste à obtenir un [0,1]-PTG sans resets. Pour ce faire, des copies du PTG sont faites pour chaque transition ayant un reset qui lui est associé (une étape similaire sera expliquée plus en détail dans l'étude de l'algorithme principal de ce document).

Une fois cette simplification effectuée, il est maintenant possible de calculer des stratégies presque optimales. La méthode, assez complexe, est expliquée en détail dans [8] ainsi que les démonstrations des propriétés associées.

Cet algorithme dont la complexité était 3-EXPTIME a ensuite été amélioré dans [10] en un nouvel algorithme EXPTIME.

Chapitre 3

Algorithme d'itération de stratégies pour les jeux temporisés à coûts positifs

Cet algorithme a été proposé dans *A Faster Algorithm for Solving One-Clock Priced Timed Games* [9], et se compose de trois étapes ajoutant au fur et à mesure de la complexité : nous allons d'abord nous concentrer sur la résolution de jeux à coûts en utilisant le paradigme d'*iteration de stratégie*. Une fois cela fait, nous essayerons de résoudre des jeux temporisés à coûts simple ($SPTG$) et enfin nous terminerons par la résolution de jeux temporisés à coûts avec une seule horloge, des resets et des coûts positifs.

3.1 Jeux à coûts

3.1.1 Définitions

Un jeu à coûts G est caractérisé par :
- Un ensemble fini d'emplacement S de taille n
- Un ensemble fini de transitions A de taille m où A_k est l'ensemble des actions qui peuvent être prises dans l'emplacement k
- S_1 et S_2, les deux sous-ensembles de S représentant l'ensemble des emplacements appartenant respectivement au Joueur 1 et au Joueur 2
- A chaque action $j \in A$ est associé un coût $c_j \in \mathbb{R}^+ \cup \{\infty\}$ et une destination $d(j) \in S \cup \perp$ où $\perp$ est un emplacement terminal spécial
- A^i l'ensemble des actions partant de sommets appartenant à S_i

Ce type de jeu peut être vu comme un plus court chemin pour le Joueur 1 avec certaines transitions incontrôlables.

Une stratégie positionnelle σ_i pour le Joueur i est un choix d'action $\sigma_i(k)$ pour chaque emplacement k du Joueur i. Un profil de stratégies σ est un

choix de stratégie pour chaque joueur. Ce profil définit un chemin maximal $P_{k_0,\sigma}$ pour chaque $k \in S \cup \bot$ dont la longueur est notée $l(k_0, \sigma)$. De plus, ce chemin définit un *coût* $u(k, \sigma) \in \mathbb{R}^+ \cup \{\infty\}$ payé par le Joueur 1 au Joueur 2 :

$$u(k,\sigma) = \begin{cases} \infty & \text{si } l(k,\sigma) = \infty \\ 0 & \text{si } k = \bot \\ c_{\sigma(k)} + (u(d(\sigma(k)),\sigma) & \text{dans les autres cas} \end{cases}$$

Intuitivement, le *coût* $u(k, \sigma)$ vaut ∞ lorsque le chemin est lui-même infini, zéro lorsque la cible est l'emplacement k et la somme de tous les coûts du chemin (de manière récursive) dans les autres cas.

Voici un exemple de jeu à coûts :

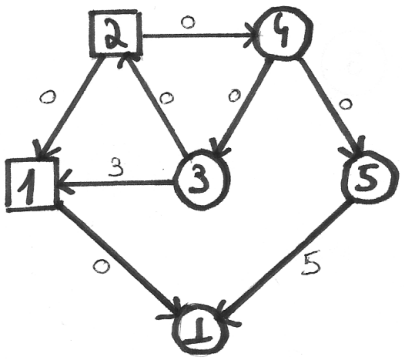

FIGURE 3.1 – Jeu temporisé à coûts

3.1.2 Valeurs, stratégies optimales et `ExtendedDijkstra`

La valeur inférieure est définie par $\underline{v}(k) = \max_{\sigma_2} \min_{\sigma_1} u(k, \sigma_1, \sigma_2)$.
La stratégie σ_2 est *optimale*, si pour tous les emplacements, on a $\sigma_2 \in \text{argmax}_{\sigma_2} \min_{\sigma_1} u(k, \sigma_1, \sigma_2)$

La valeur supérieure est définie par $\overline{v}(k) = \max_{\sigma_1} \min_{\sigma_2} u(k, \sigma_1, \sigma_2)$.
La stratégie σ_1 est *optimale*, si pour tous les emplacements, on a $\sigma_2 \in \text{argmin}_{\sigma_1} \min_{\sigma_2} u(k, \sigma_1, \sigma_2)$

Il a été prouvé [11] que les jeux à coûts ont des valeurs $v(k) := \underline{v}(k) = \overline{v}(k)$ et qu'il est possible de les calculer en utilisant une variante de l'algorithme de Dijkstra en regardant le jeu comme un problème de plus cours chemin.

Voici une exécution de l'algorithme sur l'exemple :

Algorithme 1 : `ExtendedDijkstra`

$(v(\perp), v(1),..., v(n)) \leftarrow (0, \infty,..., \infty)$;
tant que $S \neq \emptyset$ **faire**
  $(k, j) \leftarrow \underset{k \in S, j \in A_k}{\text{argmin}} \; c_j + v(d(j)))$;
  si $k \in S_1$ *or* $|A_k| = 1$ **alors**
    $v(k) \leftarrow c_j + v(d(j))$;
    $\sigma(k) \leftarrow j$;
    $S \leftarrow S \backslash \{k\}$;
  sinon
    $A_k \leftarrow A_k \backslash \{j\}$;
  fin
  retourner (v, σ);
fin

Etape	$\sigma(1)$	$\sigma(2)$	$\sigma(3)$	$\sigma(4)$	$\sigma(5)$	$v(\perp)$	$v(1)$	$v(2)$	$v(3)$	$v(4)$	$v(5)$	Action
0						0	∞	∞	∞	∞	∞	
1	$\perp$					0	0	∞	∞	∞	∞	$S \backslash \{1\}$
2	$\perp$					0	0	∞	∞	∞	∞	$A_2 \backslash \{2 \rightarrow 1\}$
3	$\perp$		1			0	0	∞	3	∞	∞	$S \backslash \{3\}$
4	$\perp$		1	3		0	0	∞	3	3	∞	$S \backslash \{4\}$
5	$\perp$	4	1	3		0	0	3	3	3	∞	$S \backslash \{2\}$
6	$\perp$	4	1	3	$\perp$	0	0	3	3	3	5	$S \backslash \{5\}$

3.1.3 Equilibre de Nash et `StrategyIteration`

Si $\sigma_1 \in argmin_{\sigma_1} u(k, \sigma_1, \sigma_2)$, $\forall k \in S$, on dit que σ_1 est la meilleure réponse à σ_2. Inversement, σ_2 est la meilleure réponse à σ_1 si $\sigma_2 \in argmax_{\sigma_2}$ $u(k, \sigma_1, \sigma_2)$, $\forall k \in S$. Un profil de stratégies (σ_1, σ_2) est un *équilibre de Nash* si chaque stratégie est la meilleure réponse à l'autre. De plus, si un profil de stratégies σ est un équilibre de Nash, alors $v(k) = u(k, \sigma)$ (le profil est donc optimal).

Définissons, pour un profil de stratégies σ, la notion de valuation :

$$\nu(k, \sigma) = (u(k, \sigma), l(k, \sigma)) \; \forall k \in S$$

où, pour rappel, $u(k, \sigma)$ est le payoff associé à l'emplacement k et $l(k, \sigma)$ la longueur du chemin partant de k et respectant le profil de stratégies σ. Une transition j d'un emplacement k est un *échange améliorant* pour le Joueur 1 si en prenant cette transition à la place de celle appartenant à la stratégie σ_1, on obtient une valuation strictement inférieure (la première composante étant la plus importante).L'échange est dit *fortement améliorant* si le coût du nouveau chemin est inférieur à l'ancien. De même, pour les transitions

du Joueur 2 mais un échange est améliorant si la nouvelle valuation est supérieure à l'ancienne et strictement améliorant si son coût est supérieur.

L'ajout de la longueur pour déterminer si un échange est améliorant est importante afin d'éviter les cycles de longueur infinie dont le coût est nul.

Un profil de stratégies $\sigma = (\sigma_1, \sigma_2)$ est dit optimal si aucun des deux joueurs n'a d'échange améliorant. Cela peut être prouvé en montrant que (σ_1, σ_2) est un équilibre de Nash puisqu'alors $v(k) = u(k, \sigma)$.

Définissons un ensemble d'actions $B \subseteq A$ tel que $|B \cap A_k| \leq 1, \forall k \in S$, pour $B \cap A \neq \emptyset$, on note $j(k, B)$ l'action de l'emplacement k appartenant à B. De plus, soit un profil de stratégie σ, on définit $\sigma[B]$ par :

$$\sigma[B](k) = \begin{cases} j(k, B) & \text{si } B \cap A_k \neq 0 \\ \sigma(k) & \text{dans les autres cas} \end{cases} \tag{3.1}$$

Une transition $j \in A$ est dite *faiblement améliorante* pour un joueur si elle n'est pas un échange améliorant pour l'autre joueur. $B \subseteq A$ est un *ensemble améliorant pour le Joueur i* s'il existe un échange améliorant $j \in B$ pour le Joueur i et que pour tout $j \in B$, j est faiblement améliorant pour le Joueur i.

Soit donc un profil de stratégies $\sigma = (\sigma_1, \sigma_2)$ pour lequel le Joueur 2 n'a pas d'échange améliorant. Soit $B^1 \subseteq A^1$ un ensemble améliorant pour le Joueur 1 et soit $\sigma' = (\sigma_1[B], \sigma_2)$ où σ_2 est n'importe quelle stratégie du Joueur 2. Alors $\nu(k, \sigma') \leq \nu(k, \sigma)\ \forall k \in S$ et est strictement inégal si $\sigma'(k)$ est un échange améliorant pour le Joueur 1.

Ce dernier résultat nous permet de définir `StrategyIteration` : l'algorithme de recherche d'un optimum local et donc global puisque lorsqu'un profil de stratégies n'a plus d'échange améliorant, celui-ci est optimal. Dans cet algorithme, le Joueur 1 sélectionne à chaque tour un échange améliorant et le Joueur 2 y répond de la meilleure manière qu'il puisse faire.

Algorithme 2 : `StrategyIteration(G,`σ`)`

tant que $\exists$ *un ensemble améliorant* $B^1 \subseteq A^1$ *pour le Joueur 1 respectant* σ **faire**
  $\sigma \leftarrow \sigma[B^1]$;
  tant que $\exists$ *un ensemble améliorant* $B^2 \subseteq A^2$ *pour le Joueur 2 respectant* σ **faire**
    $\sigma \leftarrow \sigma[B^2]$;
  fin
fin
retourner $(u(\sigma), \sigma)$;

3.2 Jeux temporisés à coûts simple

3.2.1 Définitions

Un jeu temporisé à coûts simple (SPTG = Simple Priced Timed Games) G est défini par :

- Un ensemble fini d'emplacement S de taille n
- Un ensemble fini de transitions A de taille m où A_k est l'ensemble des actions qui peuvent être prises dans l'emplacement k
- S_1 et S_2, les deux sous-ensembles de S représentant l'ensemble des emplacements appartenant respectivement au Joueur 1 et au Joueur 2
- A chaque emplacement $i \in S$ est associé un taux $r_i \in \mathbb{R}^+$
- A chaque action $j \in A$ est associé un coût $c_j \in \mathbb{R}^+ \cup \{\infty\}$ et une destination $d(j) \in S \cup \{\perp\}$ où $\perp$ est un emplacement terminal spécial
- Une horloge x comprise dans l'intervalle [0,1]

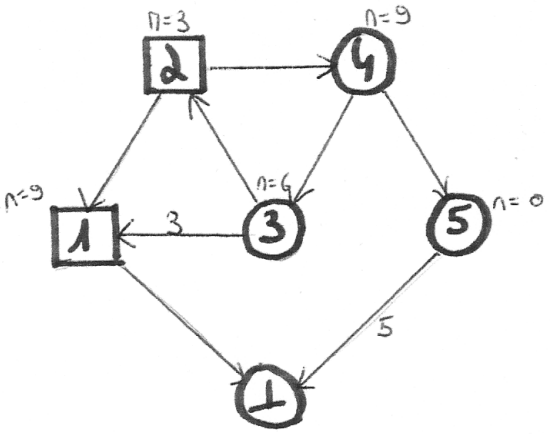

FIGURE 3.2 – SPTG

Un SPTG se joue de la manière suivante : un pion est placé dans un emplacement à l'instant initial. Chaque joueur décide de l'action à effectuer lorsque le pion se situe dans un sommet lui appartenant. La configuration du jeu est donc un couple $(k, x) \in S \times [0, 1]$ où x est le temps courant. Le joueur va donc devoir attendre un temps $\delta \in [0, 1 - x]$ puis prendre une transition $j \in A_k$.

Une *exécution* du jeu est donc une séquence de configurations successives commençant en (k_0, x_0) :

$$\rho \;=\; (k_0, x_0) \xrightarrow{j_0,\delta_0} (k_1, x_1) \xrightarrow{j_1,\delta_1} ... \xrightarrow{j_{t-1},\delta_{t-1}} (k_t, x_t)$$

Le jeu termine lorsque $k_t = \perp$. Le coût de l'exécution, payé par le Joueur 1 au Joueur 2 est donné par :

$$cost(\rho) \;= \sum_{i=0}^{t-1} (\delta_i r_{k_i} + c_{j_i})$$

où δ_i est le temps passé dans l'emplacement k_i avant la configuration suivante, r_{k_i} est le taux associé au sommet k atteint dans la configuration i et c_{j_i} le coût de la transition $j_i \in A_k$ prise dans pour sortir du sommet k. Bien sûr, si ρ est une exécution infini, $cost(\rho) = \infty$.

Une *stratégie positionnelle* pour le Joueur i est une relation définie par :

$$\pi_i : S_i \times [0, 1] \to A \cup \{\lambda\}$$

où λ est une action spéciale représentant un délai. Pour tout $k \in S$, à l'instant $x \in [0, 1)$, si $\pi_i(k, x) = \lambda$ alors il doit exister un $\delta > 0$ tel que pour tout $0 \leq \varepsilon < \delta$, $\pi_i(k, x + \varepsilon) = \lambda$. Soit $\delta_{\pi_i}(k)$ le délai d'attente dans l'emplacement k en respectant la stratégie π_i.

Lorsque le pion est dans l'emplacement $k \in S_i$ à l'instant $x \in [0, 1]$, le Joueur i suit la stratégie π_i s'il attend un temps δ_{π_i} avant de bouger le pion en respectant $\pi_i(k, x+\delta_{\pi_i})$. Un profil de stratégies π est un choix de stratégie pour chacun des deux joueurs.

La *fonction de valeur* d'un profil de stratégies est définie par :

$$v_k^\pi = cost(\rho_{k,x}^\pi)$$

où $\rho_{k,x}^\pi$ est l'exécution commençant dans la configuration (k, x).

Pour une stratégie π_2 fixée (ou π_1 dans le second cas), la *meilleure réponse* pour le Joueur 1 (Joueur 2), à l'emplacement k, est donnée par :

$$v_k^{\pi_1}(x) = \inf_{\pi_1} \; v_k^\pi(x)$$
$$v_k^{\pi_2}(x) = \sup_{\pi_2} \; v_k^\pi(x)$$

On peut alors définir les fonctions de *valeur inférieure* et *supérieure* :

$$\underline{v}_k^\pi(x) = \sup_{\pi_2} \; v_k^{\pi_1}(x)$$
$$\overline{v}_k^\pi(x) = \inf_{\pi_1} \; v_k^{\pi_2}(x)$$

Il a été prouvé [8], que ces jeux admettent une fonction de valeur $v_k(x) = \underline{v}_k^{\pi}(x) = \overline{v}_k^{\pi}(x)$.

Une strategie π_i est *optimale à partir de* x pour le Joueur i si, pour tout emplacement k et tout $x' \in [x, 1]$, on a $v_k^{\pi_i}(x') = v_k(x')$. Une stratégie est *optimale* si elle est optimale à partir de l'instant 0.

Un profil de stratégies $\pi = (\pi_1, \pi_2)$ est un *équilibre de Nash à partir de l'instant* x si chaque stratégies est une meilleure réponse à l'autre à partir de x et on a alors, pour ce profil de stratégie : $v_k^{\pi}(x') = v_k(x') \ \ \forall x' \in [x, 1]$. De plus, il peut être démontré que, pour tout SPTG, il existe un profil de stratégies optimal.

3.2.2 Résolution de SPTGs

L'idée générale de l'algorithme qui sera utilisé pour résoudre des SPTGs, est de commencer à l'instant 1 et de remonter l'intervalle de temps afin de construire les fonctions de valeurs pour chaque emplacement. Pour ce faire, il est nécessaire de trouver un profil de stratégies π optimal à partir de l'instant x (dans la première itération, le jeu n'est qu'un jeu à coûts et les fonctions sont donc relativement faciles à déterminer), puis de fonctionner par induction pour déterminer un nouveau profil π' optimal à partir de $x' < x$.

La seule manière de faire écouler du temps est d'attendre, on peut donc modéliser le jeu à l'instant x' par un jeu à coûts dans lequel on aurait ajouté de nouvelles actions λ_k à chaque emplacement k (représentant donc cette attente $x - x'$). Une fois cela fait, il suffit ensuite de déterminer, grâce à ce nouveau jeu le plus petit $x' < x$ après lequel on est certain que le profil π' est optimal et de trouver les fonctions de valeurs entre x' et x.

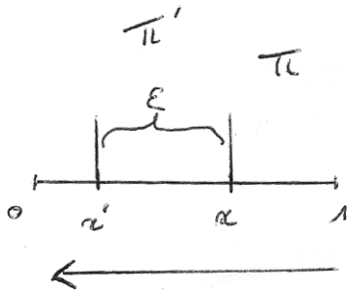

FIGURE 3.3 – Recherche du plus petit x'

A partir d'un SPTG G $= (S_1, S_2, A_{k,k\in S}, c, d, r)$, un instant $x \in [0, 1)$, et un $\varepsilon \geq 0$ (représentant l'attente entre deux jeux à coûts), on définit le jeu à coûts $G^{x,\varepsilon} = (S_1, S_2, A_{k,k\in S}, c^{x,\varepsilon}, d')$ par :

$$\forall k \in S : \quad A'_k = A - k \cup \{\lambda_k\}$$
$$\forall j \in A'_k : \quad c_j^{x,\varepsilon} = \begin{cases} v_k(x) + \varepsilon r_k & \text{si } j = \lambda_k \\ c_j & \text{dans les autres cas} \end{cases}$$
$$\forall j \in A'_k : \quad d'(j) = \begin{cases} \bot & \text{si } j = \lambda_k \\ d(j) & \text{dans les autres cas} \end{cases}$$

Pour des soucis d'écriture, nous simplifierons $G^{x,\varepsilon}$ par G^x et $c^{x,\varepsilon}$ par c^x.

Dans G^x, considérons le chemin $P_{k,\sigma} = (k, k_0, .., k_t)$ commençant en k et utilisant les actions du profil de stratégies σ. Lorsque ce chemin est fini et que la dernière action est λ_k, utilisons une notation pour représenter le taux de l'avant-dernier emplacement de ce chemin :

$$r(k, \sigma) = \begin{cases} r_{k_{t-1}} & \text{si } \sigma(k_{t-1}) = \lambda_{k_{t-1}} \\ 0 & \text{dans les autres cas} \end{cases}$$

Pour chaque $x \in (0, 1]$, soit $\sigma^x = (\sigma_1^x, \sigma_2^x)$ un profil de stratégies pour G^x pour lequel aucun joueur n'a d'échange améliorant ; ce profil existe et est optimal. De plus, ce profil est optimal pour $G^{x,0}$ et $u(k, \sigma^x, G^{x,0}) = v_k(x)$.

Montrons maintenant le lien entre le SPTG G et le jeu à coûts G^x, pour un $x \in (0, 1]$: soit π un profil de stratégies pour G optimal à partir de x et soit $x' < x$. Si $\pi(x'') = \sigma^x$ pour tout $x'' \in [x', x)$, alors $v_x^\pi(x') = v_k(x) + (x - x')r(k, \sigma^x)$ pour tout $k \in S$.

Afin de pouvoir déterminer le prochain point auquel des actions discrètes sont faites, il faut d'abord définir la fonction suivante, pour toute action $j \in A$ et pour tout $x \in (0, 1]$:

$$f_{j,x}(x'') := c_j + u(d(j), \sigma^x, G^{x,0}) + (x - x'')r(d(j), \sigma^x, G^x)$$

Par définition, nous savons que σ^x est optimal pour $G^{x,x-x''}$ quand x'' est proche de x. Nous allons maintenant essayer de réduire x'' autant que possible tout en gardant σ^x optimal. Pour rappel, σ^x est optimal si aucun des joueurs n'a d'échange améliorant. Utilisons maintenant la notion de valuation définie dans la partie concernant les jeux à coûts : bien que les payoffs $u(k, \sigma^x, G^{x,\varepsilon})$ changent pour différents ε, la longueur des chemins, elle, ne varie pas. Il n'y a donc pas d'échange améliorant si et seulement si :

$$\forall k \in S_1, j \in A_k : \ (f_{\sigma^x(k),x}(x''), l(k, \sigma^x, G^{x,0})) \geq (f_{j,x}(x''), 1 + l(k, \sigma^x, G^{x,0}))$$
$$\forall k \in S_2, j \in A_k : \ (f_{\sigma^x(k),x}(x''), l(k, \sigma^x, G^{x,0})) \leq (f_{j,x}(x''), 1 + l(k, \sigma^x, G^{x,0}))$$

Soit $x' < x$ la première intersection des deux lignes $f_{\sigma^x(k),x}(x'')$ et $f_{j,x}(x'')$, pour $k \in S$ et $j \in A_k \backslash \sigma^x$. Pour $x' \leq x'' \leq x$, les deux équations ci-dessus

sont satisfaites mais ne le sont pas lorsque $x'' \leq x'$ donc x' est le plus petit instant à partir duquel σ^x est optimal.

`NextEventPoint`(G^x) est défini par :

$$max\{0\} \cup \{x' \in [0,x) | \exists k \in S, j \in A_k : f_{j,x}(x') = f_{\sigma^x(k),x}(x') \wedge f_{j,x}(x) \neq f_{\sigma^x(k),x}(x)\}$$

Pour résumer, soit x' =`NextEventPoint`(G^x), et soit π un profil de stratégies optimal à partir de x. Alors le profil de stratégies π', définit par :

$$\pi'(k,x'') = \begin{cases} \sigma^x(k) & \text{si } x'' \in [x',x) \\ \pi(k,x'') & \text{dans les autres cas} \end{cases}$$

est optimal à partir de x', et $v_k(x'') = v_k(x) + r(k,\sigma^x)(x - x'')$, pour $x'' \in [x,x)$ et $k \in S$.

Voici l'algorithme résolvant des SPTGs en utilisant les différentes parties d'algorithme que nous avons énoncé jusqu'ici :

Algorithme 3 : `SolveSPTG(G)`

$(v(1), (\pi_1(1), \pi_2(1))) \leftarrow$ `ExtendedDijkstra`(G')
$x \rightarrow 1$
tant que $x > 0$ **faire**
  $((u(k,\sigma^x, G^{x,0}) + \varepsilon r(k,\sigma^x), (\sigma_1,\sigma_2) \leftarrow$
  `StrategyIteration`$(G^x, (\pi_1(x), \pi_2(x))))$
  $x' \leftarrow$ `NextEventPoint`(G^x)
  pour tous les $k \in S$ *and* $x'' \in [x',x)$ **faire**
    $v_k(x'') \leftarrow v_k(x) + r(k,\sigma^x)(x - x'')$
    $\pi_1(k,x'') \leftarrow \sigma_1(k)$
    $\pi_2(k,x'') \leftarrow \sigma_2(k)$
  fin
  $x \leftarrow x''$
fin
retourner $v, (\pi_1, \pi_2))$

Détaillons maintenant les différentes étapes de l'exécution de l'algorithme sur le SPTG donné en exemple. Commençons par résoudre le jeu à coûts à l'instant 1.

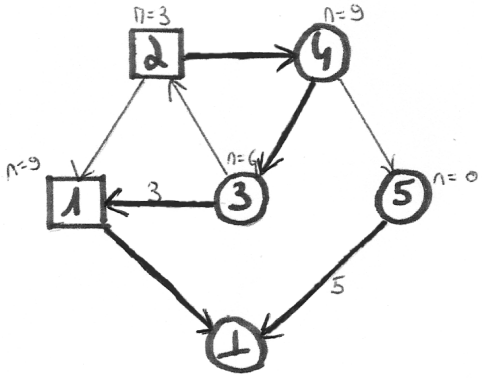

FIGURE 3.4 – Etape 1 : résolution du jeu à coûts à l'instant 1

Une fois ces coûts déterminés pour l'instant 1, nous pouvons ajouter les actions λ et commencer la résolution.

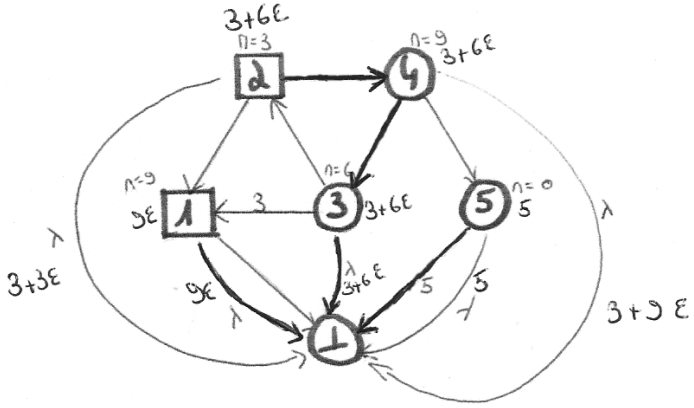

FIGURE 3.5 – Etape 2 : recherche du premier ε

Pour déterminer l'intervalle ε durant lequel cette stratégie est optimale nous devons résoudre l'équation suivante représentant le choix de stratégie, en fonction de l'intervalle ε dans l'emplacement 4 :

$$3 + 3\varepsilon = 5$$

Ce qui nous donne $\varepsilon = 1/3$. En effet, avant cela, prendre l'autre transition devient intéressant.

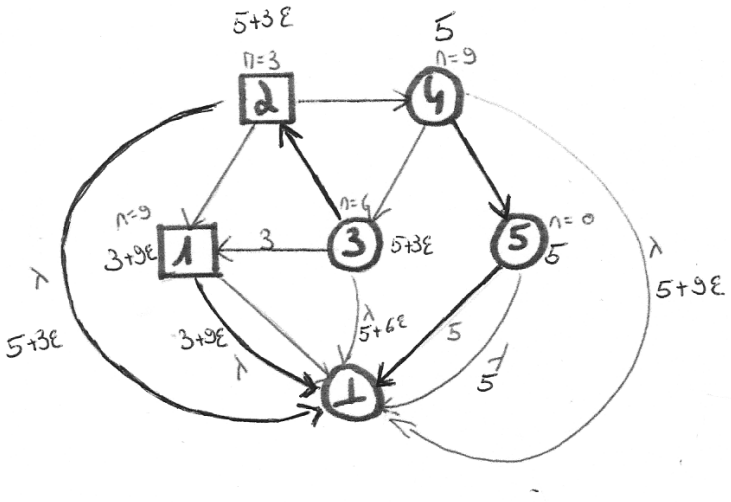

FIGURE 3.6 – Etape 3 : recherche du prochain ε depuis l'instant 2/3

Le prochain ε est déterminé par le choix de stratégie dans l'emplacement 2 :

$$5 + 3\varepsilon = 3 + 9\varepsilon$$

Ce qui nous donne $\varepsilon = 1/3$.

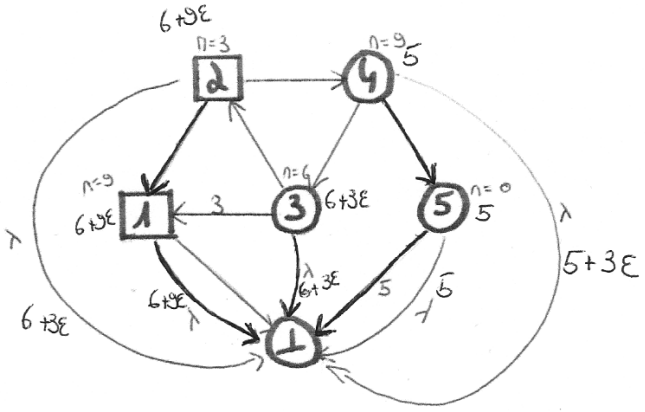

FIGURE 3.7 – Etape 4 : recherche du prochain ε depuis l'instant 1/3

Nous pouvons maintenant voir qu'il n'y a pas d'intersection entre les

différents choix de stratégie entre l'instant 0 et 1/3. Nous pouvons donc déterminer l'ensemble des fonctions de valeurs pour chaque emplacement.

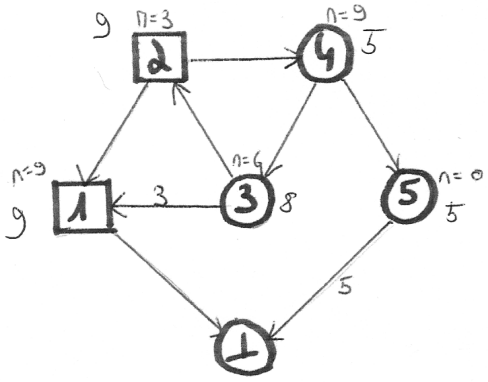

FIGURE 3.8 – Etape 5 : valeurs à l'instant 0

Voici les graphes représentants les fonctions de valeurs en chaque emplacement :

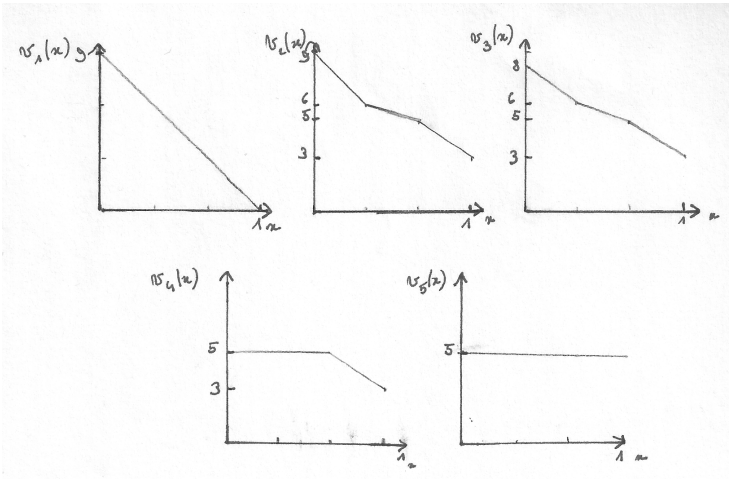

FIGURE 3.9 – Fonctions de valeur

3.2.3 Terminaison et complexité de l'algorithme

Afin de prouver que le nombre de points où des actions discrètes sont effectuées n'est pas infini et pour trouver la complexité de l'algorithme, nous avons besoin de définir plusieurs outils utilisés dans la preuve.

Soit n le nombre d'emplacements de G, N le nombre de taux distincts incluant 0 pour le taux de l'emplacement terminal $\perp$. Supposons que ces taux sont ordonnés $r_1 < r_2 < ... < r_N$. Soit

$$count(\sigma, i, l, r) = |\{k \in S_i | l(k, \sigma) = l \wedge r(k, \sigma) = r\}|$$

Pour chaque profil de strategie σ pour les jeux à coûts G^x, pour $x \in (0, 1]$, la *matrice de potentiel* $P(\sigma) \in \mathbb{N}^{n \times N}$ est définie par :

$$P(\sigma)_{l,r} = count(\sigma, 2, l, r) - count(\sigma, 1, l, r)$$

Pour ordonner ces matrices, les entrées dont les taux sont les plus faibles sont les plus importantes. Viennent ensuite les entrées avec une faible longueur. On note $P(\sigma) \prec P(\sigma')$ si et seulement s' il existe l et r tels que :

- $P(\sigma)_{l',r'} = P(\sigma')_{l',r'}$ pour tout $r' < r$ et $1 \leq l' \leq n$;
- $P(\sigma)_{l',r} = P(\sigma')_{l',r}$ pour tout $l' < l$;
- $P(\sigma)_{l,r} = P(\sigma')_{l,r}$.

De plus, à partir de cette construction, si un profil σ est optimal pour $G^{x,0}$ pour un certain $x \in (0, 1]$ et que $j \in A^i$ est un échange améliorant pour le Joueur i, alors $P[\sigma'] \prec P[\sigma]$.

Supposons une variante de l'algorithme `SolveSPTG` dans laquelle la fonction `StrategyIteration` ne fait qu'un échange améliorant pour chaque joueur. On utilise ensuite le profil optimal σ^x ainsi obtenu pour résoudre le jeu à coûts suivant $G^{x'}$ où $x' =$ `NextEventPoint`(G^x). Une fois que le profil de stratégies en $x = 1$ est trouvé, on peut voir que pour nouveau profil obtenu par un échange améliorant fait diminuer la matrice de potentiel. On a donc que le nombre total de profils de stratégie dans G^x est $\prod_{k \in S}(|A_k| + 1)$.

Une matrice de potentiel peut toujours être construite de la façon suivante : un ensemble on choisit d'abord un sous-ensemble non vide de colonnes $C \subseteq \{1, ..., N\}$ comprenant les colonnes dans lesquelles au moins un emplacement est controlé par un des deux joueurs c'est-à-dire dans lesquelles il existe r tel que $count(\sigma, 1, l, r) \neq 0$ ou $count(\sigma, 1, l, r) \neq 0$. Il existe, au maximum, $2^N - 1 \leq 2^{n+1}$ manières de le faire. Ensuite, on assigne des emplacements aux ensembles d'entrées $S_{l,r}$ ($S_{l,r}$ contient les emplacements à une distance l de $\perp$ et a le r-ème plus petit taux). Si $S_{l,r} \neq 0$, alors par définition, $S_{l',r} = \emptyset \; \forall l' < l$. Ce qui nous permet d'ajouter les emplacements de manière ordonnée.

En commençant par $l = 1$ et $r = min\ C$, on va ajouter, à chaque étape, un emplacement depuis S_1 ou S_2 à $S_{l,r}$ puis mettre à jour l et r d'une des trois manières suivantes :

– Ne rien faire, des emplacements doivent encore être insérés.
– Passer à la ligne suivante : il n'y a plus d'emplacements qui doivent être insérés dans $S_{l,r}$ mais certains dans $S_{l,r+1}$.
– Passer au début de la colonne suivante de C : il n'y a plus d'emplacements qui doivent être insérés dans la colonne l.

Il y a n étapes de ce type et il existe au plus six choix pour chaque étape. Le nombre de manières d'insérer les emplacements est, au maximum, 6^n et le nombre de matrices est donc, au maximum, 12^n.

Le nombre total de points auxquels des actions discrètes sont effectuées dans un SPTG à n emplacements est $L(G) \leq \min\{12^n, \prod_{k \in S}(|A_k|+1)\}$. L'algorithme `SolveSPTG` résout n'importe quel SPTG en temps $O(m \cdot \min\{12^n, \prod_{k \in S}(|A_k|+1)\}$.

3.3 Jeux temporisés à coûts

Un jeu temporisé à coûts, comme décrit dans l'état de l'art, est un SPTG dans lequel des contraintes temporelles doivent être respectées pour pouvoir prendre une transition et dans lequel on a ajouté des resets sur certaines transitions. Dans le type de jeu étudié plus loin, les contraintes ne sont associées qu'aux transitions et non aux emplacements. En effet, il est aisé de passer d'une version à l'autre.

3.3.1 Définitions

Un jeu temporisé à coûts ou PTG (Priced Timed Game) est un jeu $G = (S_1, S_2, A_k, d, r_k, I_j, R)$ où
– S_i for $i \in \{1, 2\}$: l'ensemble des emplacements contrôlés par le Joueur i.
– A_k pour $k \in S$: l'ensemble des actions pouvant être prises depuis l'emplacement k.
– $c_j \in \mathbb{R}^+ \cup \{\infty\}$ pour $j \in A$: le coût de l'action j.
– $d(j) \in S \cup \{\perp\}$ pour $j \in A$: la destination de l'action j.
– $r_k \in \mathbb{R}^+$ pour $k \in S$: le taux de l'emplacement k.
– I_j pour $j \in A$: l'*intervalle d'existence* (la contrainte temporelle) de l'action j durant lequel celle-ci est disponible.
– $R \subseteq A$: l'ensemble des resets.

Un (n, m, r, d)-PTG représente la classe des PTGs ayant n emplacements dont r sont des destinations de resets, m actions et d fins d'intervalle distinctes.

Définissons $e(I)$ l'ensemble des fins des intervalles d'existence et $M = max_{j \in A} e(I_j)$. Il n'existe donc plus d'actions disponibles après M.

Une *configuration* d'une exécution est un couple $(k, x) \in S \times [0, M]$. Une *exécution* est donc définie par une séquence :

$$\rho \;=\; (k_0, x_0) \xrightarrow{j_0,\delta_0} (k_1, x_1) \xrightarrow{j_1,\delta_1} ... \xrightarrow{j_{t-1},\delta_{t-1}} (k_t, x_t)$$

où, pour tout $i \geq 0, x_i + \delta_i \in I_j$ et si $j \in R$ alors $x_{i+1} = 0$.

La notion de stratégie positionnelle est définie de manière identique à celle utilisée pour des SPTGs.

Soit une exécution $\rho^{\tau}_{k,x}$ commençant en (k, x) et respectant le profil de stratégies τ. La fonction de valeur est définie par :

$$v^{\tau}_k(x) = cost(\rho^{\tau}_{k,x})$$

La meilleure réponse, la fonction de valeur inférieure et supérieure sont définies par :

$$v^{\tau_1}_k(x) = \inf_{\tau_1} \; v^{\tau}_k(x)$$
$$v^{\tau_2}_k(x) = \sup_{\tau_2} \; v^{\tau}_k(x)$$
$$\underline{v}^{\tau}_k(x) = \sup_{\tau_2} \; v^{\tau_1}_k(x)$$
$$\overline{v}^{\tau}_k(x) = \inf_{\tau_1} \; v^{\tau_2}_k(x)$$

Il a été démontré ([8]) que pour tout PTG G, il existe une fonction de valeur $v_k(x) = \underline{v}_k(x) = \overline{v}_k(x)$ et puisqu'il est possible de se rapprocher des fonctions de valeurs même en utilisant des stratégies positionelles, nous nous restreindrons à celles-ci.
Une stratégie π_i est ε-*optimale* pour le Joueur i pour $\varepsilon \geq 0$ si :

$$\forall k \in S, x \in [0, M] : \; |v^{\pi_i}_k - v_k(x)| \; \leq \varepsilon$$

De plus, il a aussi été démontré [8] que les stratégies optimales n'existent pas toujours. Voici un exemple de jeu pour lequel il n'existe pas de stratégie optimale.

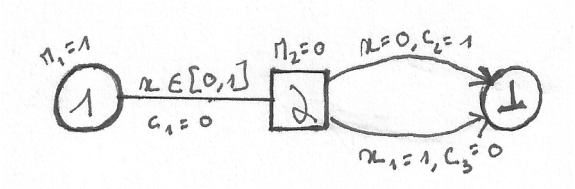

FIGURE 3.10 – Jeu sans stratégie optimale

où les fonctions de valeurs sont :

$$v_1(x) = 0$$
$$v_2(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{dans les autres cas.} \end{cases}$$

3.3.2 Réduction du PTG

Afin de résoudre un PTG, il est nécessaire de réduire le jeu à un SPTG. La première étape consiste à supprimer les resets en étendant le jeu.

Soit un (n, m, r, d)-PTG G, la résolution de ce jeu peut être réduite à la résolution de $r+1$ $(n, m, 0, d)$-PTGs. L'idée est que si une exécution utilise deux resets menant au même emplacement, alors une même configuration sera atteinte deux fois et on aura donc un cycle infini puisqu'on s'est limité à des stratégies positionnelles.

On peut alors construire un nouveau PTG G' composé de $r+1$ sous-jeux identiques dans lesquels, si une transition ayant un reset est prise, la transition mène à l'emplacement correspondant dans le sous-jeu suivant. Pour $j \in A$ et $l \in \{0, ..., r\}$, les destinations et coûts sont définis par :

$$d'(j,l) = \begin{cases} (d(j), l+1) & \text{si } j \in R \text{ et } l < r \\ \bot & \text{si } j \in R \text{ et } l = r \\ (d(j), l+1) & \text{dans les autres cas} \end{cases}$$

$$c'_{j,l} = \begin{cases} \infty & \text{si } j \in R \text{ et } l = r \\ c_j & \text{dans les autres cas} \end{cases}$$

Une exécution dans ce jeu est notée ρ' et les fonctions de valeur v'. Dans [10], il a été démontré que $v'_{k,0}(x) = v_k(x)\ \forall\ (k,x) \in S \times [0, M]$. On peut ensuite observer que les emplacements $(k,l) \in S \times \{0, ...r\}$ ne dépendent pas des emplacements (k,l') où $l' < l$. Le jeu peut donc être résolu par induction sur l en partant depuis la fin. En effet, quand $v'_{k,l+1}(x)$ est connu pour tout k et x, le sous-jeu contenant les emplacements (k,l) peut être vu comme un PTG indépendant dont les actions resets mènent à des emplacements dont les valeurs sont connues.

Nous allons maintenant voir comment réduire ces jeux en SPTG en trois étapes :

- réduire ces jeux en des jeux dont les intervalles d'existence I_j sont $\{(0,1), [1,1]\}$;
- réduire à nouveau ces jeux en de nouveaux jeux dont les intervalles d'existence sont $\{[0,1], [1,1]\}$;
- réduire ces derniers en SPTGs.

Soit $X = e(I) \cup \{0\}$, soit M_i le i-ème plus grand élément de X et $M_1 = M$.

Définissons, pour un PTG $\mathrm{G} = (S_1, S_2, A_k, d, r_k, I_j, R)$ où $R = \emptyset, x \in \mathbb{R}^+$ et un vecteur $v \in (\mathbb{R}^+ \cup \{\infty\})^n$, soit le jeu à coûts $\mathrm{G}^{v,x} = (S_1, S_2, (A'_k)_{k \in S}, c', d')$

définis par :

$$\forall k \in S:\ A'_k = \{j \in A_k |\ x \in I_j\} \cup \{\bot_k\}$$
$$\forall j \in A'_k:\ c_j = \begin{cases} v_k & \text{si } j = \bot_k \\ c_j & \text{dans les autres cas} \end{cases}$$
$$\forall j \in A'_k:\ d_j = \begin{cases} \bot & \text{si } j = \bot_k \\ d_j & \text{dans les autres cas} \end{cases}$$

Pour commencer la réduction, chaque $(n, m, 0, d)$-PTG peut être résolu en temps $O((n \log\, n + \min(m, n^2))d)$ en faisant, au plus, d appels à un oracle R résolvant des $\text{PTGs}^{n,m+n}_{(0,1)}$. En effet, il est facile de trouver la valeur de $k \in S$ à l'instant M_1 dans un PTG sans resets puisque ce jeu est équivalent à un jeu à coûts. Entre les instants M_2 et M_1, le jeu est presque un SPTG puisqu'on peut soustraire tous les instants x entre M_2 et M_1 par M_2 et les diviser par $M_1 - M_2$ pour avoir un jeu se déroulant entre 0 et 1. Une fois que les valeurs sont trouvées pour tous les x, on peut trouver la valeur en M_2. On peut ensuite trouver les valeurs entre M_3 et M_2 et ainsi de suite jusqu'à la résolution totale. Nous utilisons donc bien d appels à l'oracle pour résoudre $d + 1$ jeux à coûts.

Nous allons maintenant réduire un $\text{PTG}^{n,m}_{(0,1)}$ dont la fonction de valeur est v en un $\text{PTG}^{n,m}_{[0,1]}$ en $O((n \log\, n + \min(m, n^2))d)$. Notons qu'il est facile de trouver $v(k, 1)$ en utilisant un jeu à coûts. Il aussi est clair que $v(k, 0) = \lim_{x \to 0^+} v(k, x)$ car la seule option à l'instant 0 est d'attendre.

Soit un $\text{PTG}^{n,m}_{(0,1)}$ dont la fonction de valeur est v. Soit G' la version modifiée de G, dans laquelle tous les intervalles d'existence sont, à la place, de la forme [0,1] (le jeu G est donc un $\text{PTG}^{n,m}_{[0,1]}$) et dont la fonction de valeur est v'. On a alors : $\forall k \in S, x \in (0, 1):\ v(k, x) = v'(k, x)$. L'idée de la preuve est que l'extension vers 0 n'est pas compliquée puisqu'on ne peut aller à un instant avant celui-ci. L'extension vers 1 est plus compliquée mais pour tout $\varepsilon > 0$, une stratégie $\frac{\varepsilon}{2}$-optimale dans G peut être modifiée dans un petit intervalle proche de 1 pour mener à une stratégie ε-optimale dans G'. On peut alors poser que les valeurs sont dans un voisinage ε des autres. Pour plus de détails concernant la preuve, cf. [10].

Résoudre un $\text{PTG}^{n,m}_{[0,1]}$ G' peut être réduit en temps polynomial en la résolution d'un SPTG à $n + 1$ emplacements et $m + 1$ actions.

En effet, puisque le Joueur 2 ne va jamais utiliser une transition j menant à l'instant 1 dans un SPTG, on peut changer toutes les transitions venant de sommets du Joueur 2 dont l'intervalle était [1,1] en [0,1] sans changer

les fonctions de valeur. Nous allons ensuite créer un nouvel emplacement appartenant au Joueur 2, *max*, ayant le taux maximum et une transition vers $\perp$ de coût nul et d'intervalle [0,1]. Toutes les transitions venant d'emplacements du Joueur 1 dont la destination était $\perp$ vont être redirigées vers *max* et changer leur intervalle en [0,1]. Le Joueur 1 va donc utiliser les transitions vers *max* uniquement en temps 1. Maintenant que tous les intervalles sont [0,1], on peut ne garder qu'une transition $j \in A_k$ dont $d(j) = l$ pour n'importe quelle paire de $k, l \in S$ puisque le joueur auquel appartient l'emplacement k va toujours jouer de manière optimale pour lui. On obtient donc un SPTG équivalent.

Pour un PTG G $= (S_1, S_2, A_k, d, r_k, I_j, R)$,$x \in \mathbb{R}^+$ et un vecteur $v \in (\mathbb{R}^+ \cup \{\infty\})^n$, soit le SPTG $\mathrm{G}^{v,x,d} = (S_1, S_2', (A_k')_{k \in S_1 \cup S_2'}, c', d', (r_k)_{k \in S_1 \cup S_2'})$ défini par :

$$
\begin{aligned}
&S_2' = S_2 \cup \{max\} \\
&\forall k \in S : \ A_k' = j \in A_k |\ x \in I_j \cup \{\perp_k\} \\
&A_{max} = \{\perp_{max}\} \\
&\forall j \in A_k' : \ c_j = \begin{cases} 0 & \text{si } k = max \\ v_k & \text{si } k \neq max \text{ et } j = \perp_k \\ c_j & \text{dans les autres cas} \end{cases} \\
&\forall j \in A_k' : \ d_j = \begin{cases} max & \text{si } k \in S_1 \text{ et soit } j = \perp_k \text{ soit } d_j = \perp \\ \perp & \text{si } k \in S_2 \text{ et soit } j = \perp_k \text{ soit } d_j = \perp \\ \perp & \text{si } k = max \\ d_j & \text{dans les autres cas} \end{cases} \\
&\forall k \in S : \ r_k' = r_k \cdot d \\
&r_{max}' = \max_{k \in S} \{r_k\}
\end{aligned}
$$

L'algorithme résolvant des SPTGs sans resets peut être trouvé à la page suivante.

N'importe quel (n, m, r, d)-PTG G peut être résolu en temps $O((r+1)d(n \log n + \min(m, n^2))d)$ en utilisant, au plus, $(r+1)d$ appels à un oracle R qui résout des SPTGs à $n+1$ emplacements et au plus $m+n+1$ transitions. La complexité totale de la résolution d'un $(n, m, r, d) - PTG$ est donc :

$$O((r+1)d(\min(m, n^2) + n \cdot \min\{12^n, \textstyle\prod_{k \in S}(|A_k| + 1)\}))$$

Algorithme 4 : `SolvePTG`

$v(M_1) \leftarrow$ `ExtendedDijkstra`$((S_1, S_2, (\{j \in A_k |\ M_1 \in I_j\})_{k \in S}, c, d))$
$i \leftarrow 1$
tant que $M_i > 0$ **faire**
  $i \leftarrow i + 1$
  $x \leftarrow \frac{M_{i-1}+M_i}{2}$
  $v' \leftarrow$ `ExtendedDijkstra`$(G^{v(M_{i-1},x)})$
  $v*(x) \leftarrow$ `SolveSPTG`$(G^{v',x,M_{i-1}-M_i})$
  pour tous les $x \in (M_i, M_{i-1})$ **faire**
    $v(x) \leftarrow v*(\frac{x-M_i}{M_{i-1}-M_i)}$
  fin
  $v(M_i) \leftarrow$ `ExtendedDijkstra`$(G^{v*(0),M_i})$
fin

Chapitre 4

Conclusion

Maintenant que ces résultats théoriques ont été étudiés et compris, la prochaine étape sera l'implémentation de cet algorithme. Une fois cela fait, il sera possible de lancer la résolution d'une variété de jeux temporisés à coûts différents afin de pouvoir en extraire des résultats concernant entre autre la complexité réelle de l'algorithme (et non plus une borne supérieure). Il peut ensuite être intéressant d'essayer d'étendre l'algorithme à des jeux à coûts plus généraux comme par exemple des jeux temporisés à coûts à la fois positifs et négatifs.

Bibliographie

[1] Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235, 1994.

[3] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. *Theoretical Computer Science*, 318(3) :297–322, 2004.

[4] Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem of weighted timed automata. *Formal Methods in System Design*, 31(2) :135–175, 2007.

[5] Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5) :188–194, 2006.

[6] Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, volume 3328 of *Lecture Notes in Computer Science*, pages 148–160. Springer, 2004.

[7] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in one-clock priced timed games. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[8] Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *Proceedings of the Third international conference on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2005.

[9] Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In

Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13), volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013.

[10] Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In *Proceedings of the Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, volume 57 of *Electronic Proceedings in Theoretical Computer Science*, pages 31–46, 2011.