Calcul d'attracteurs dans un jeu d'accessibilité en temps linéaire

Thibaut Antoine (thibaut.antoine@ens-rennes.fr)

Phrase d'introduction

Dans un jeu d'accessibilité à deux joueurs (et alternance stricte) dans un graphe $G = (V, E)$, on peut se demander comment calculer l'ensemble des positions gagnantes pour chaque joueur. Un calcul de point fixe permet d'obtenir un algorithme en temps $\mathcal{O}(|V|^2)$; on présente ici un algorithme en temps linéaire (i.e. $\mathcal{O}(|V| + |E|)$).

1 Algorithme [COR]

On suppose déjà connues les notions de stratégie gagnante et de graphe d'état, elles seront présentées dans le plan. On commence donc par écrire directement l'algorithme. Attention, on suppose ici qu'un sommet sans successeurs est perdant pour le joueur qui le contrôle.

Il se décompose en deux parties : la détection des sommets gagnants pour un joueur, et leur propagation à travers le graphe. On présente la détection en premier pour un aspect plus pédagogique.

Entrée. Un graphe d'états d'un jeu à deux joueurs.
Sortie. Les sommets gagnants pour chaque joueur.

Winning$(G = (V, V_0, V_1, E))$

```
1 win := [v : ⊥, v ∈ V]                                  # Statut du sommet
2 P := [v : pred(v), v ∈ V]                              # Prédécesseurs
3 n := [v : |succ(v)|, v ∈ V]                            # Nombre de successeurs de statut indéterminé
4 pour chaque j ∈ {0,1}, v ∈ V_j faire
5 |   si n[v] = 0 alors
6 |   |   Propagate(v, 1 − j)
7 renvoyer win
```

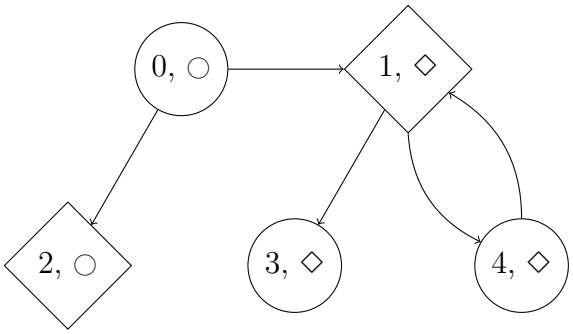
Propagate$(v,\, j)$

```
1 si win[v] = ⊥ alors
2 |   win[v] ← j
3 |   pour chaque u ∈ P[v] faire
4 |   |   n[u] ← n[u] − 1
5 |   |   si n[u] = 0 ou u ∈ V_j alors
6 |   |   |   Propagate(u, j)
```

On comprend intuitivement bien le fonctionnement de l'algorithme :

- La fonction **Propagate** marque chaque sommet gagnant pour un joueur par le numéro du joueur en question,
- Pour chaque sommet initialement sans successeur, on le marque gagnant pour le joueur opposé,
- Si un prédécesseur d'un sommet marqué gagnant pour j appartient à j, alors j a une stratégie gagnante à partir de ce prédécesseur,
- Si un sommet u prédécesseur de v venant d'être marqué pour j n'a aucun successeur non marqué et est encore non marqué, alors aucun de ses successeurs n'est gagnant pour le joueur opposé : on peut donc le marquer gagnant pour j.

Exemple.



1. $n[2] = 0 \to$ **Propagate**$(2, \bigcirc)$;
   $0 \in V_{\bigcirc} \to$ **Propagate**$(0, \bigcirc)$
2. $n[3] = 0 \to$ **Propagate**$(3, \diamond)$;
   $1 \in V_{\diamond} \to$ **Propagate**$(1, \diamond)$;
   $n[4] = 0 \to$ **Propagate**$(4, \diamond)$

2 Correction et complexité

2.1 Complexité

- Initialisation en $\mathcal{O}(|V| + |E|)$,
- On appelle le corps de **Propagate** au plus une fois par sommet : coût en $\sum_{v \in V} 1 + |P[v]| = \mathcal{O}(|V| + |E|)$.

On fait en réalité un parcours en profondeur du graphe d'états, d'où la complexité linéaire en sa taille.

2.2 Correction

Théorème. *À la fin de l'algorithme, on a* $\text{win}[v] = j$ *si et seulement si le joueur* j *a une stratégie gagnante à partir de* v*.*

Preuve. ($\Rightarrow$) Supposons qu'à la fin de l'algorithme, il existe un sommet w tel que $\text{win}[w] = j$ mais j n'a pas de stratégie gagnante à partir de w. Soit w_0 le tel sommet sur lequel on a appelé **Propagate**(w_0, j) en premier. Cet appel a pu être causé de deux manières :

- On a exécuté le corps de **Propagate**(v, j) (i.e. on avait $\text{win}[v] = \bot$) pour v un successeur de w_0 et $w_0 \in V_j$. Alors à la fin de l'algorithme, $\text{win}[v] = j$ donc par hypothèse, j a une stratégie gagnante à partir de v, donc aussi à partir de w_0 (j se déplace en v car il contrôle w_0). C'est absurde.
- On a exécuté le corps de **Propagate**(v, j) (i.e. on avait $\text{win}[v] = \bot$) pour v un successeur de w_0 et w_0 n'avait plus aucun successeur non marqué. Remarquons que le premier appel à **Propagate**(w_0, j) est le premier appel à **Propagate** sur w_0 tout court car à la fin de l'algorithme $\text{win}[w_0] = j$. Alors si w_0 avait un successeur v marqué par $1 - j$, on aurait fait l'appel **Propagate**$(w_0, 1 - j)$ avant **Propagate**(w_0, j), ce qui est absurde. Ainsi, tous les successeurs de w_0 sont marqués par j et j a une stratégie gagnante à partir de ces sommets par hypothèse sur w_0, et donc à partir de w_0 aussi : c'est absurde.

On conclut dans les deux cas à une absurdité, ce qui prouve la première implication.

($\Leftarrow$) On prouve facilement par récurrence la propriété $\mathcal{P}(k)$: « Pour tout v, si j a une stratégie gagnante en k coups depuis v alors on a appelé **Propagate** sur v pour la première fois avec j. »

□

Organisation du tableau

Titre 1. Algorithme		3. Correction a. Complexité	
	2. Exemple	b. Correction	

Remarques

— À l'heure où j'écris ce document (22 juin 2024), le développement n'a jamais été essayé en vrai. Il est peut-être trop long. S'il est trop court, il est toujours possible de présenter le problème initial comme un problème de décision, passer plus de temps sur l'exemple, expliquer plus en détails la preuve par récurrence...

— Le principal intérêt de cet algorithme est sa complexité linéaire *en la taille du graphe d'états*. Il est en fait de coût exponentiel sur beaucoup de jeux dont on a l'habitude de mesurer la taille différemment (échecs ou dames $n \times n$, morpion généralisé...).

— L'algorithme qu'on a montré ici ne dépend pas de si le jeu est à alternance stricte (i.e. le graphe d'états est biparti entre les sommets contrôlés par chaque joueur), donc il fonctionne évidemment dans ce cas aussi.

— Dans un jeu à deux joueurs, certains sommets peuvent n'être gagnants pour aucun joueur (on peut penser au cas d'un cycle entre un sommet d'un joueur et de l'autre). L'énoncé du théorème de correction contient ce cas-ci, en prenant sa contraposée.

— L'algorithme vient de **[GRÄ 3.1.3]** et la preuve aussi, mais l'auteur passe très rapidement dessus, cela rend le développement assez difficile je pense. Il faut être à l'aise avec les notions.

— Il faut mentionner à l'oral dans la preuve que la valeur de win$[u]$ pour un sommet u ne change plus jamais une fois attribuée.

Références

[GRÄ] *Finite Model Theory and Its Applications*, Erich Grädel