Ricochet Robots
Parcours par atteignabilité

8 avril 2010

Abstract

L'algorithme qui sera décrit ici consiste à repérer le coup minimal d'atteignabilité de chacune des cases du plateau, à partir de la position initial d'un robot. Ainsi, si la cible est accessible, chaque robot saura en combien de coups il pourra directement y accéder. Le but est ensuite d'optimiser l'utilisation de ce nouveau plateau pondéré afin de faire intéragir les robots, et ainsi améliorer la solution.

1 Notations

Les notations utilisées dans ce document sont les suivantes :

M_n	$\leftrightarrow$	la matrice de dimension $n \times n$ représentant le plateau de jeu
R_c	$\leftrightarrow$	le robot de couleur c, avec $c \in red, blue, yellow, green$
$R_{c_{i,j}}$	$\leftrightarrow$	le robot de coordonnées i et j, avec $c \in red, blue, yellow, green$
R_{cible}	$\leftrightarrow$	le robot qui doit atteindre la cible ("robot ciblé")
$C_{i,j}$	$\leftrightarrow$	la cible, de coordonnées i et j
$d(M_{i,j}, R_c)$	$\leftrightarrow$	la distance minimal de R_c à $M_{i,j}$, c'est à dire le nombre de coups minimal pour atteindre cette case

2 Génération de la matrice pondérée

Le but de cette partie est d'affecter un poids à toutes les cases accessibles par les robots du plateau. L'idée est de produire des clones de ces robots, qui iront marquer le plateau. Ces clones sont virtuels, ils représentent la réflexion d'un robot, qui ne se déplace pas pendant cette phase. Voici maintenant une description d'un algorithme effectuant ce travail.

2.1 Initialisation

```
génération ← 0

Pour i de 1 à (nombre de robots) faire
    Pour chaque direction disponible faire
        Créer un clone
        Propager(clone, génération)
    Masquer robot // Pour que le robot réel ne soit pas considéré comme un obstacle pendant l'exploration
```

2.2 Propagation

Propager(clone, génération)
Tant que le clone n'est pas bloqué faire
| $case_{i,j} \leftarrow min(generation, case_{i,j})$
FinTq *// le robot est arrêté*

Si $case_{i,j} >$ génération
Alors
| $case_{i,j} \leftarrow$ génération
| Détruire le clone
| génération $\leftarrow$ génération + 1
|
| Pour chaque direction disponible faire
| | Créer clone
| | Propager(clone, génération)
// Sinon ne rien faire

2.3 Coût et limites

Dans le pire des cas, un robot peut initialement se déplacer dans toutes les directions. A partir de l'itération suivante, au maximum trois directions sont disponibles. Ainsi, dans le pire des cas, $4 * 3^n$ clones sont à créer pour chacun des robots (où n est le nombre total de générations produites).

Les interactions dynamiques entre robots ne sont pas prises en compte. Sinon à chaque itération il faudrait considérer toutes les directions comme possibles, et le coût de l'algorithme serait en $O(4^n)$.

Au niveau du stockage de l'information, il n'y a pas besoin de mémoriser les chemins, seuls les numéros de générations sont mémorisés sur chaque case du plateau. La mémoire occupée ici est donc en $O(n^2)$ où n est la taille d'un côté du plateau. Un chemin se retrouve alors en avancement suivant l'ordre croissant des numéros en partant du robot, ou décroissant en partant de la cible.

La mémoire utilisée pendant la cartographie du plateau est donc $O(4 * 3^n + n^2)$, soit $O(3^n)$.

Si la cible a été atteinte par le robot ciblé, alors on sait qu'une solution a été trouvée, et on connaît son coût. Toutefois, celle-ci n'est pas forcément optimale, et il n'y a pas encore de moyen pour l'estimer.

2.4 Autre idée de construction

En remarquant que de nombreux chemins sont communs à l'ensemble des robots, la question se pose de savoir s'il est nécessaire de parcourir tous les chemins possibles pour chacun d'entre eux. La figure 1 montre que si le robot rouge construit sa carte en premier, les autres robots n'ont plus qu'à ajouter des bouts de chemins pour construire la leur.

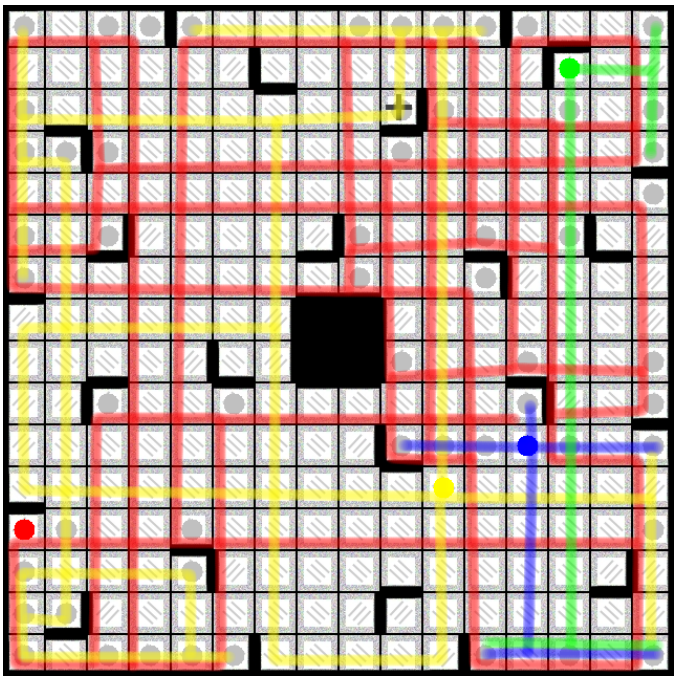

Figure 1: Cartographie simplifiée

On a le lemme suivant:

Lemme : Si en construisant sa carte, un robot s'arrête sur une case atteignable [1] par un robot dont la carte est déjà construite, alors il peut parcourir toute la carte de ce dernier.

3 Recherche d'une solution

Deux cas de figure peuvent se présenter :

- Soit le robot ciblé peut atteindre la cible; dans ce cas, il faut évaluer cette solution par rapport à un chemin utilisant d'autres robots;

[1] case où un robot peut s'arrêter

- Soit il ne l'atteint pas, auquel cas il faut obligatoirement utiliser un ou plusieurs autres robots pour trouver une solution.

Nous allons commencer par examiner la deuxième situation. Lors de la génération de la première carte, le robot ciblé ne peut pas atteindre la cible. Il est donc nécessaire de manipuler un ou plusieurs autres robots. Pour commencer, nous allons tenter de trouver une case atteignable par le robot ciblé et un autre robot en partant de la cible. Si de telles cases sont découvertes, celle qui est accessible le plus rapidement sera choisie.

3.1 Initialisation de la recherche à partir de la cible

génération $\leftarrow 0$

Pour chaque direction disponible faire
| Créer un clone
| Explore(clone, génération)
Masquer robot

3.2 Exploration

mem : tableau d'entiers
$choix \in M_{i,j} \times \mathbb{N}$
$i = 0$

Tant que le clone n'est pas bloqué faire
| Si ($M_{i,j}$ atteignable par R_{cible}) et ($\exists R_c | M_{i,j}$ atteignable par R_c)
| Alors $mem[i] \leftarrow (M_{i,j}, d(M_{i,j}, R_c))$
| Avancer

Trier mem par $d(M_{i,j}, R_c)$ croissant.
$tmp \leftarrow M_{i,j} + max\,(d(M_{i,j}, R_c))$

Pour i de 1 à $|mem|$ faire
| Si $(M_{i,j} + d(M_{i,j}, R_c)) < tmp$
| Alors
| | $tmp \leftarrow min(tmp, M_{i,j} + d(M_{i,j}, R_c) + d(M_{i,j}, C_{i,j}))$
// algorithme incomplet, il faut encore recommencer avec de nouvelles générations et chercher la condition d'arrêt notamment