Jeux de poursuite-évasion en temps discret

FOURNIER Némo

15 juin 2017

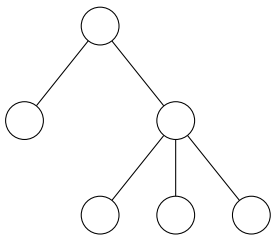
(Ce rapport est issu d'un travail à deux pendant l'année, les parties présentées sont celle qui ont constitué ma part plus personnelle de travail ; les calculs de convexités sont notamment présentés en dimension 2 seulement, car j'ai choisi d'étudier le jeu sur les graphes pendant que mon camarade traitait la dimension supérieure.)

1 Étude d'un jeu sur les graphes

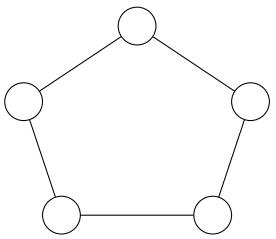
1.1 Règles et premières observations

On se donne un graphe fini non orienté G. Initialement un policier se place sur un des sommets, suivi par un voleur. Ils se déplacent ensuite tour à tour le long des arêtes du graphe, d'au plus une arête à chaque mouvement. On dira que le policier gagne le jeu s'il parvient à attraper le voleur en un nombre fini de tours (*i.e.* le policier parvient à se placer sur le même sommet que le voleur), et que le voleur gagne s'il parvient à s'échapper indéfiniment.

On a rapidement l'intuition que certains graphes sont particulièrement propices à une victoire du policier, d'autres à une victoire du voleur.



Un arbre semble condamner le voleur



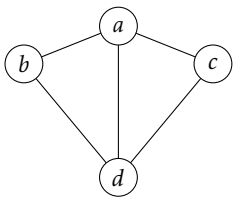
Un 5-cycle semble offrir une issue au voleur

On dira qu'un graphe est gardable si, quelle que soit la stratégie adoptée par le voleur, le policier peut gagner le jeu. On peut montrer de manière élémentaire qu'un arbre est gardable, et qu'un 5-cycle ne l'est pas, confirmant l'intuition précédente.

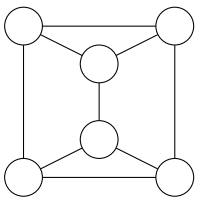
1.2 Caractérisation des graphes gardables

On va, dans cette section, préciser un critère qui permet de décider la question de la gardabilité d'un graphe de manière générale, dû à Aigner et Fromme [1]. Pour cela commençons par décrire une certaine configuration sur des graphes.

Définition 1 (Coin)**.** *Un sommet u d'un graphe G est un coin s'il existe un autre sommet v de G tel que les voisins de u et u sont des voisins de v. On dira que u est un coin relativement à v, ou que v contrôle u.*



a est un coin par rapport à d



Graphe sans coin

Cette structure de coin a une pertinence toute particulière vis à vis du problème étudié grâce aux propositions suivantes, évoquées par Aigner et Fromme [1].

Proposition 1. *Un graphe gardable admet un coin.*

En effet, pour un graphe gardable, quelle que soit la stratégie adoptée par le voleur, le policier peut parvenir à l'attraper en un nombre fini de tours. S'il n'existait pas de coin, alors à chaque tour le voleur aurait la possibilité de rejoindre un sommet qui n'est pas controlé par le policier, faisant durer la partie aussi longtemps que souhaité.

Proposition 2 (Ajout d'un coin)**.** *Ajouter un coin à un graphe ne change pas son caractère gardable ou non gardable.*

En effet, si le voleur parvenait à s'échapper indéfiniment sur le graphe privé du nouveau coin, il peut se déplacer sur le nouveau graphe comme il l'aurait fait sur l'ancien, en assimilant le coin à un sommet le dominant. Un policier pourrait étendre sa stratégie de la même manière.

Ces deux propositions justifient l'algorithme suivant décidant de la gardabilité d'un graphe :

Entrée(s) G un graphe
tant que G admet un coin u **faire**
$G \leftarrow G \setminus \{u\}$
renvoyer vrai si G est réduit à un sommet, faux sinon.

Une implémentation en utilisant des listes d'adjacence pour représenter un graphe à n sommets mène à une complexité en $O(n^4)$, même si cette borne est pessimiste et dépend fortement de la forme du graphe considéré.

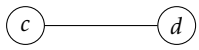
1.3 Génération d'une stratégie explicite pour les policiers

Une fois le caractère gardable d'un graphe décidé, je me suis intéressé à la possibilité de décrire explicitement une stratégie pour le policier. Il s'est avéré que ceci pouvait se faire de manière assez efficace en modifiant l'algorithme de Aigner et Fromme décrit en section 1.2.

Observons d'abord qu'il est possible de stocker au cours de l'exécution du premier algorithme les coins que l'on supprime, ainsi qu'un sommet du graphe qui les domine. On peut donc obtenir, si le graphe G est gardable, une liste $L = [(c_1, d_1); (c_2, d_2); \dots; (c_n, d_n)]$ telle que c_1 est un coin dans G par rapport à d_1, et plus généralement, pour $i \in [\![2, n]\!]$, c_i est un coin dans $G \setminus \{c_1, \dots, c_{i-1}\}$ par rapport à d_i.

On souhaite déterminer une stratégie pour le policier, que l'on formalisera comme une fonction s de V^2 dans V où V désigne l'ensemble des sommets de G, telle que si le policier est sur p et le voleur sur v, $s(p, v)$ retourne le sommet sur lequel doit se diriger le policier. On va construire récursivement une telle fonction à partir de L, et on note s_L la stratégie engendrée par L.

Si $L = [c, d]$, G est de la forme :



Si le policier est initialement en d, une stratégie gagnante est alors clairement décrite par

$$s_L : (p, v) \mapsto \begin{cases} c & \text{si } (p, v) = (d, c) \\ d & \text{sinon} \end{cases}$$

Si $L = [c, d] :: q$, on définit s_L comme suit :

$$s_L : (p, v) \mapsto \begin{cases} c & \text{si } (p, v) = (d, c) \\ s_q(p, d) & \text{si } v = c \\ s_q(p, v) & \text{sinon} \end{cases}$$

Proposition 3. *Si le policier est initialement sur d_n (le sommet restant après avoir retiré tous les coins), la stratégie s_L est valide (*i.e. *n'entraîne pas de mouvements illégaux).*

Proposition 4. *Suivre s_L permet au policier de gagner, et ce en au plus n mouvements.*

On peut remarquer que l'ordre dans lequel on retire les coins est sans importance pour décider de la gardabilité, mais va en revanche stratégie générée. En particulier, certains ordres de retrait des coins peuvent conduire à des stratégies plus efficaces que d'autres en terme de nombre de mouvement nécessaires au policier pour gagner. Il serait intéressant d'arriver à déterminer explicitement un ordre d'élminitation engendrant une stratégie «optimale». Je n'ai pas réussi à progresser sur cette question.

1.4 Vers une généralisation

On a évoqué le fait qu'un 5-cycle n'est pas gardable (ce qui est d'ailleurs une conséquence de la proposition 1). En revanche une variante du jeu impliquant non plus un, mais deux policiers, se déplaçant alors ensemble, en alternance avec le voleur semble intuitivement rendre le 5-cycle gardable. Cette remarque conduit à introduire la garde d'un graphe G (*cop-number* dans la littérature anglophone), notée $c(G)$ qui correspond au plus petit nombre de policier permettant d'assurer la capture sur le graphe.

1.4.1 Autour d'une caractérisation de la garde

J'ai d'abord essayé, à la manière de ce qui a été fait dans la section 1.2, de trouver un critère permettant de caractériser les graphes de garde k quelconque. Il m'est apparu que l'on pouvait essayer d'utiliser la caractérisation que l'on a déjà du cas $k = 1$.

Définition 2 (Graphe des paires et des combinaisons de k sommets)**.** *À un graphe $G(V,E)$, on associe $\widetilde{G}_2(\widetilde{V}_2,\widetilde{E}_2)$, tel que $\widetilde{V}_2 = \mathcal{P}_2(V)$ et $\widetilde{E}_2 = \left\{\{\{u_1,v_1\},\{u_2,v_2\}\} \mid (\{u_1,u_2\},\{v_1,v_2\}) \in E^2 \text{ ou } (\{u_1,v_2\},\{v_1,u_2\}) \in E^2\right\}$. On peut généraliser cette définition au graphe $\widetilde{G}_k$ dont les sommets sont les combinaisons de k sommets de G.*

Proposition 5 (Une condition suffisante)**.** *Si $c(\widetilde{G}_k) = 1$, alors $c(G) \leq k$.*

L'idée de cette proposition est d'assimiler le mouvement d'un policier sur $\widetilde{G}_k$ au mouvement de k policiers sur G.

Pour un graphe tel que $c(G) \neq 1$ et $c(\widetilde{G}_2) = 1$, on peut donc affirmer que $c(G) = 2$. Pour mettre en œuvre ce critère, il suffit de pouvoir calculer le graphe des paires, ce qui peut se faire pour un graphe G à n sommets en $O(n^4)$, et l'application de l'algorithme de la section 1.2 sera en $O(n^8)$, le nouveau graphe ayant de l'ordre de n^2 sommets. Cette condition n'est cependant que suffisante (car trop forte : le mouvement d'un voleur sur $\widetilde{G}_2$ s'assimile également au mouvement de deux voleurs sur G), et en pratique, ne permet de caractériser la garde que de peu de graphes.

1.4.2 Une conjecture de Meyniel

En faisant des recherches sur la possibilité de caractériser plus précisément la garde de certains graphes, j'ai pris connaissance d'un problème ouvert relatif à la garde des graphes, dû à Meyniel. En notant, pour n entier naturel, $C(n) = max\left\{c(G) \mid G \text{ est connexe et d'ordre } n\right\}$, il est conjecturé que $C(n) = O(\sqrt{n})$.

Le problème est toujours ouvert à l'heure actuelle, mais on peut montrer que si la conjecture est vraie, alors elle est optimale, dans le sens $\sqrt{n} = O(C(n))$, résultat établi par Praãlat [4]. J'ai pu étudier sa preuve, et observer qu'on pouvait d'ailleurs limiter les résultats sur la garde utilisés à ceux proposés par Aigner et Fromme [1], en utilisant notamment le théorème suivant :

Proposition 6 (Théorème 3 dans [1])**.** *Un graphe G de maille soit plus grande que 5 vérifie $c(G) \geq \delta$ où δ est le degré minimum de G.*

La mise en place de cette preuve m'a conduit à m'intéresser aux structures géométriques que sont les plans projectif, à la garde de leur graphe d'incidence, ainsi qu'à la question de leur existence. On peut en effet prouver l'existence d'une famille infinie de plan projectif d'ordre strictement croissant grâce à l'existence de corps finis de cardinal premier et aux résultats précisés dans [3] ; l'étude de la garde des graphes d'incidence associés à cette famille permet de montrer la relation $\sqrt{n} = O(C(n))$.

2 Étude d'un jeu dans R^n

On s'intéresse ici à un jeu similaire dans le principe, mais dont le cadre est cette fois ci l'espace R^n.

2.1 Règles et observations

On se donne N policiers, et un voleur, repérés initialement par leur position dans l'espace notée P_k^0 pour le k-ième policier et V^0 pour le voleur. À chaque tour, le voleur commence par se déplacer d'une distance d'au plus 1, puis les policiers se déplacent, eux aussi d'une distance d'au plus 1. On dit que les policiers gagnent si l'un d'eux parvient, en un nombre fini de tours à occuper la même position que le voleur, le voleur gagne si il parvient à s'échapper indéfiniment.

On s'aperçoit là encore intuitivement que certaines situations semblent permettre à l'une où l'autre des parties de gagner à coup sûr en suivant une stratégie adaptée (*e.g.* le cas d'un seul policier face à un voleur dans R^n laisse la possibilité au voleur de s'échapper indéfiniment, et le cas dans R où initialement $V^0 \in [P_1^0, P_2^0]$ condamne le voleur).

2.2 Critère de capturabilité

De la même manière que pour le premier jeu, on veut essayer de trouver un critère permettant de déterminer si une des parties possède une stratégie gagnante.

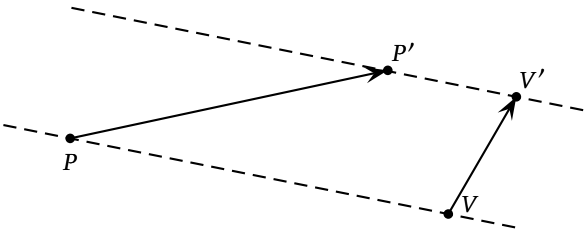
Le critère suivant est proposé par Kopparty et Ravishankar [2].

Proposition 7 (Condition nécessaire et suffisante de capturabilité)**.** *Les policiers peuvent gagner à coup sûr si et seulement si le voleur est initialement dans l'intérieur de l'enveloppe convexe formée par les policiers.*

La preuve repose sur une description d'une stratégie gagnante pour les policiers dans le cas où le voleur est dans l'enveloppe convexe au début du jeu, et d'une stratégie gagnante pour le voleur dans le cas complémentaire. On se propose d'implémenter ces stratégies.

2.2.1 Stratégie gagnante pour les policiers

Tant que le voleur n'est pas attrapé, on donne la stratégie suivante pour chaque policier : ayant connaissance de la position actuelle (V') et précédente (V) du voleur (qui a bougé avant P), le policier (P) se déplace, dans le plan contenant P, V et V', sur la droite parallèle à (PV) et passant par P', en cherchant à minimiser la distance entre lui et le policier (c'est à dire soit en l'attrapant directement, soit en se déplaçant à l'intersection de la boule de rayon 1 centrée en P et la droite passant par P').



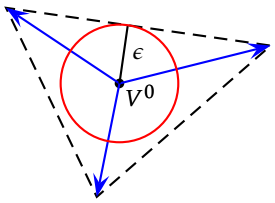
Proposition 8 (Rapprochement large)**.** *La distance individuelle de chaque policier au voleur diminue largement à chaque tour.*

Proposition 9 (Rapprochement strict)**.** *Il existe une constante C strictement positive telle que la somme des distances entre chaque policier et le voleur diminue d'au moins C à chaque tour.*

Cette seconde proposition prouve la terminaison de la stratégie des policiers, la somme des distances de chaque policier au voleur constituant un variant positif et strictement décroissant au cours du jeu.

En étudiant la preuve d'existence de cette constante, on observe qu'une constante qui convient est entièrement déterminée par les conditions initiales.

En effet, en considérant ce qu'on appelle les «vecteurs pointants normalisés», c'est à dire les vecteurs $\overrightarrow{p_k} = \frac{\overrightarrow{V^0 P_k^0}}{\|\overrightarrow{V^0 P_k^0}\|}$, comme V^0 est à l'intérieur de l'enveloppe convexe engendrée par les P_k^0, il sera à l'intérieur de l'enveloppe convexe engendrée par les extrémités des vecteurs pointants normalisés : on pourra donc trouver un ϵ strictement positif tel que la boule centrée en V^0 et de rayon ϵ soit incluse dans l'enveloppe convexe formée par les vecteurs pointants normalisés.



On montre que le constante $C = \frac{\epsilon}{n+1}$ convient.

L'implémentation de cette stratégie est directe, il suffit à chaque tour, de mettre à jour la position de chaque policier en suivant la règle ci-haut. En annexe on montre quelques poursuites où le voleur était astreint à se déplacer sur une courbe, et initialement dans l'enveloppe convexe.

Il est intéressant d'observer que la connaissance de la constante ci-haut associée à celle de la somme des distances initiales de chaque policier au voleur nous donne une majoration de la durée de la partie ; on peut déterminer cette constante en calculant l'enveloppe convexe évoquée. Je me suis restreint au calcul d'enveloppe convexe en dimension 2, l'algorithme de Jarvis étant suffisant pour les cas testés qui n'impliquaient pas suffisamment de points pour que la complexité en $O(nh)$ soit gênante, et donnant directement les segments composants l'enveloppe, permettant donc de déterminer le rayon de la boule.

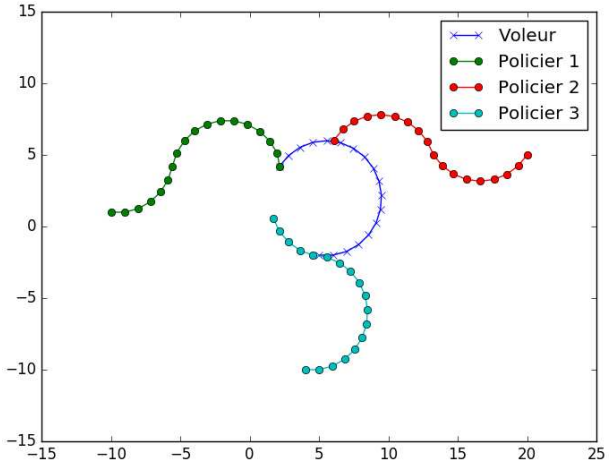
2.2.2 Stratégie gagnante pour le voleur

L'existence d'une stratégie gagnante pour le voleur si celui-ci est hors de l'enveloppe convexe formée par les policiers initialement vient de l'existence d'un hyperplan séparant un convexe ouvert et un point hors de ce convexe, conséquence du théorème de Hahn-Banach. À chaque tour le voleur peut s'éloigner des policiers en se dirigeant normalement à cet hyperplan, à «vitesse maximale».

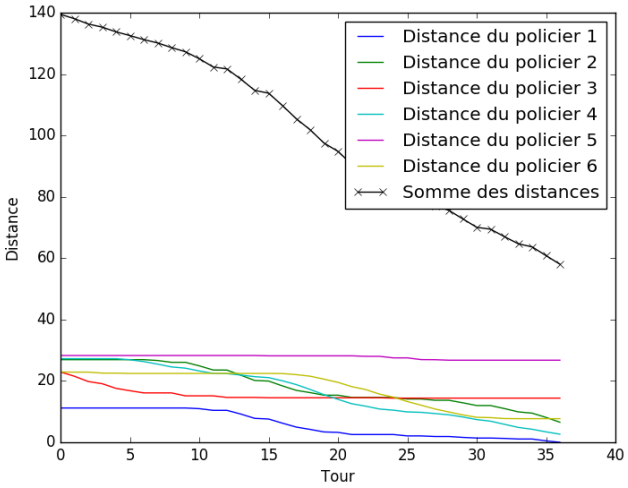
Implémenter cette stratégie en pratique nécessite de trouver un tel hyperplan. Dans le cas de la dimension 2, on observe que calculer l'enveloppe convexe engendrée par les policiers permet de trouver un tel hyperplan : il suffit de trouver une face «vue» par le voleur hors de l'enveloppe. Dans l'idée de diminuer le coût de cette implémentation (qui est principalement le coût du calcul de l'enveloppe convexe), j'ai essayé de tirer profit du fait que l'enveloppe convexe des policiers pouvait ne pas beaucoup changer d'un tour à l'autre, mais l'abscence de contrôle sur les mouvements des policiers m'a semblé limiter l'intérêt de cette approche par rapport à un calcul «classique», qui n'a pas abouti.

3 Annexes

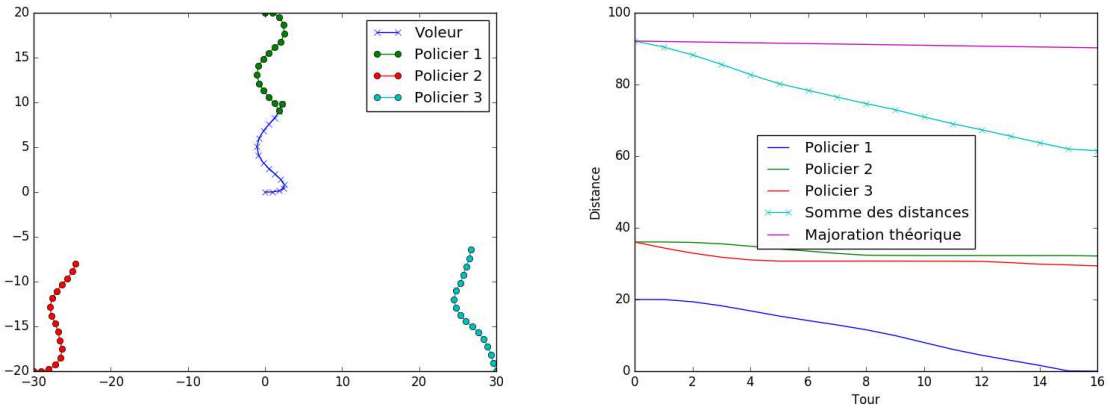
3.1 Quelques poursuites dans $\mathbb{R}^n$



Une poursuite dans $\mathbb{R}^2$*, le voleur est capturé par le policier* 1



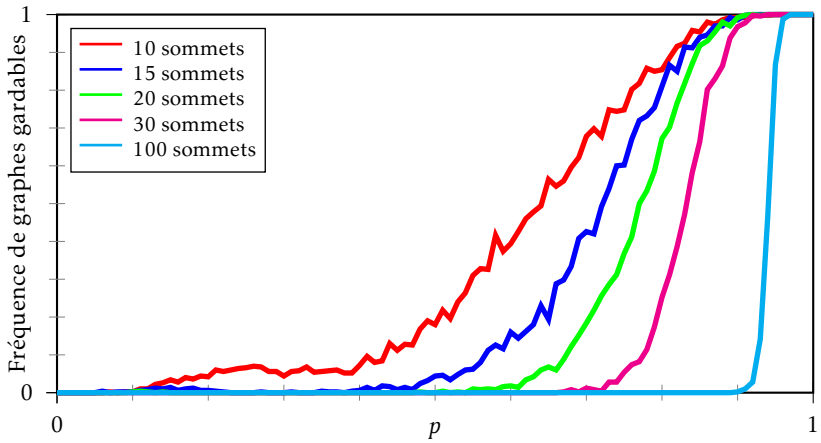
Pour les plus grandes dimensions (ici $\mathbb{R}^5$), on doit se contenter d'une représentation de l'évolution des distances au cours du temps. On constate bien, sur cet exemple, le caractère largement décroissant d'un point de vue individuel, et le caractère strictement décroissant de la somme des distances de chaque policier au voleur.



Un exemple de poursuite dans R^2, avec le graphe d'évolution des distances associé. On fait aussi apparaître la majoration donnée par l'évaluation de la constante associée aux conditions initiales. Cette majoration théorique est largement respectée.

3.2 Étude des graphes gardables

Je me suis aussi intéressé à la proportion de graphes gardables dans une population de graphes générés aléatoirement avec le modèle suivant : on fixe un nombre n de sommets, et on fixe une probabilité p d'existence telle que chaque arête va, au moment de la génération du graphe, avoir une probabilité p d'exister. On peut ainsi générer des graphes, et vérifier si ils sont gardables en appliquant le premier algorithme.



On observe qu'un phénomène de type «tout ou rien» se met en place, avec l'existence d'un seuil critique en deçà duquel les graphes semblent être majoritairement non gardables, et au-delà duquel ils sont majoritairement gardables. Je n'ai pas réussi à caractériser précisément ce phénomène, mais la forme du graphe est en tout cas cohérente avec ce que l'on peut prévoir : on peut montrer que tout graphe complet (*i.e.* situation rencontrée systématiquement pour $p = 1$) est gardable, et qu'un graphe complet à n sommets auquel on retire moins que $\lceil \frac{n}{2} \rceil - 1$ arêtes (soit avec une très forte probabilité dans un petit palier à droite de $p = 1$) est reste gardable. À l'inverse, pour p faible, la probabilité de trouver des graphes non connexes et donc non gardables augmente, expliquant les parties gauches des courbes.

3.3 Codes des programmes utilisés

3.3.1 Programmation sur les graphes

Caractérisation des graphes gagnables

```
(* Retourne la liste des entiers de 0 a n *)
let liste_0_a n =
   let rec liste a = match (a = n) with
      | true -> [n]
      | _ -> a :: liste (a + 1) in liste 0;;

(* Verifie l'inclusion de petite dans grande
  petite et grande doivent etre triees par ordre croissant *)
let rec inclus petite grande = match petite, grande with
   | [], _ -> true
   | _, [] -> false
   | a :: _, b :: _ when a < b -> false
   | a :: q1, b :: q2 when a = b -> inclus q1 q2
   | _, _ :: q2 -> inclus petite q2;;

(* Verifie si le sommet s est un coin dans le graphe g *)
let coin g s =
   let rec parcours_candidats candidats = match candidats with
      | [] -> false
      | d :: q when d <> s && inclus g.(s) g.(d) -> true
      | _ :: q -> parcours_candidats q
   in parcours_candidats g.(s) (*Seuls les voisins de s peuvent le dominer*);;

(* Supprime le sommet s du graphe g, en O((nb_voisins)^2) *)
let supprime_sommet g s =
   let rec supprimer_dans_liste liste acc = match liste with
      | [] -> rev acc
      | t :: q when t = s -> supprimer_dans_liste q acc
      | t :: q -> supprimer_dans_liste q (t :: acc)
   and parcours_voisins voisins = match voisins with
      | [] -> g.(s) <- []
      | v :: q -> g.(v) <- supprimer_dans_liste g.(v) [];
         parcours_voisins q;
   in parcours_voisins g.(s);;

(* Verifie si le graphe g est gardable *)
let gardable g =
   let n = vect_length g in
      let rec cherche_coin restants acc = match restants with
         | [] -> false; (* Il n'y a plus de coin dans le graphe *)
         | a :: q when coin g a -> supprime_sommet g a; parcours (acc @ q);
         | a :: q -> cherche_coin q (a :: acc);
      and parcours restants = match restants with
         | [a] -> true;
         | _ -> cherche_coin restants [];
      in parcours (liste_0_a (n - 1));;
```

Génération d'une stratégie explicite pour le policier

```
(* Modifications pour pouvoir generer une strategie *)

(* Verifie si le sommet s est un coin dans le graphe g
  Si oui, renvoie un sommet dominant le sommet teste
  Si non, renvoie -1 *)
let coin_strategie g s =
   let rec parcours_candidats candidats = match candidats with
```

```
      | [] -> -1
      | d :: q when d <> s && inclus g.(s) g.(d) -> d
      | _ :: q -> parcours_candidats q
   in parcours_candidats g.(s) (*Seuls les voisins de s peuvent le dominer*);;

(* Verifie si le graphe g est gardable
  Si oui, renvoie une liste coin-dominant
  Si non, renvoie une liste vide *)
let gardable_strategie g =
   let n = vect_length g in
      let rec cherche_coin restants non_coins coin_dom = match restants with
         | [] -> [] (* Il n'y a plus de coin dans le graphe *)
         | a :: q -> let d = coin_strategie g a in
               if d <> - 1 then begin (* si a est un coin *)
                     supprime_sommet g a;
                     parcours (non_coins @ q) ((a, d) :: coin_dom)
                  end
               else cherche_coin q (a :: non_coins) coin_dom
      and parcours restants coin_dom = match restants with
         | [a] -> rev coin_dom
         | _ -> cherche_coin restants [] coin_dom
      in parcours (liste_0_a (n - 1)) [];;

(* Retourne une strategie pour le policier a partir du resultat de
  gardable_strategie *)
let rec strategie coin_dominant = match coin_dominant with
  | [] -> failwith "Graphe non gardable"
  | [(c,d)] -> (fun (p,v) -> if v=c then c else d)
  | (c,d)::q -> (fun (p,v) -> if v=c && p=d then c
                  else if v=c then (strategie q) (p,d)
                  else (strategie q) (p,v));;
```

Calcul du graphe des paires

```
(* Calcul du graphe des paires *)

(* Passage de la representation par liste a la
  representation par matrice d'adjacence *)
let liste_vers_mat g =
   let n = vect_length g in
      let m = make_matrix n n false in
         for i = 0 to (n - 1) do
            let rec ajoute_voisins voisins = match voisins with
               | [] -> ()
               | s :: q -> m.(i).(s) <- true; ajoute_voisins q
            in ajoute_voisins g.(i)
         done;
         m;;

let graphe_paire g =
  let m = liste_vers_mat g and n = vect_length g in
  let N = n * (n - 1) / 2 in let graphe_paire = make_vect N [] in

  (* On trouve la prochaine paire, selon l'ordre lexicographique *)
  let prochaine_paire (u, v) =
       if v < (n - 1) then (u, v + 1)
       else (u + 1, u + 2)
  in

    let paire_ligne = ref (0, 1) in
      for i = 0 to N - 1 do
```

```
        let (u1, v1) = !paire_ligne in

        (* On regarde toutes les autres paires pour trouver les
             voisins de (u1,v1) *)

        let rec parcours_paire paire indice = match paire with
          | (u, v) when u > (n - 2) -> [] (* Plus de paire valide *)
          | (u, v) when m.(u1).(u) && m.(v1).(v) ||
                m.(u1).(v) && m.(v1).(u) ->
            indice :: parcours_paire (prochaine_paire paire) (indice + 1)
          | _ -> parcours_paire (prochaine_paire paire) (indice + 1)

        in
          graphe_paire.(i) <- parcours_paire (0, 1) 0;
          paire_ligne := prochaine_paire !paire_ligne
      done;
      graphe_paire;;
```

3.3.2 Jeu sur R^n

Mouvement de chaque policier

```
def norme(vecteur):
    R = 0
    for x in vecteur: R += x**2
    return sqrt(R)

def produit_scalaire(v1, v2):
    p = 0
    for i in range(0, len(v1)):
        p += v1[i]*v2[i]
    return p

def poursuivant_etape(poursuivi_avant, poursuivi_apres, poursuivant):
    vecteur_pointant = (poursuivi_avant - poursuivant)
    vecteur_pointant_n = vecteur_pointant / norme(vecteur_pointant)

    deplacement = poursuivi_avant - poursuivi_apres
    distance = norme(deplacement)

    # espace des vecteurs appartenants a la droite // passant par poursuivi_apres:
    #       { poursuivi_apres + vecteur_pointant_n * t | t in |R}
    # on veut trouver un vecteur nouveau_poursuivant tel que :
    #       || nouveau_poursuivant - poursuivant || <= 1
    #       || nouveau_poursuivant - poursuivi_avant || soit minimale

    # Si on peut attraper
    if norme(poursuivi_apres - poursuivant) <= 1:
        return poursuivi_apres

    # Sinon, t_inf et t_max verifient t^2-2xdcos(a)t + d^2-1 = 0
    delta = 4 * ( produit_scalaire(deplacement, vecteur_pointant_n)**2 + 1 - distance**2 )

    t_inf = (2*produit_scalaire(deplacement, vecteur_pointant_n) - sqrt(delta))/2
    t_sup = (2*produit_scalaire(deplacement, vecteur_pointant_n) + sqrt(delta))/2

    return poursuivi_apres + vecteur_pointant_n * ((-norme(vecteur_pointant) + t_sup))
```

Calcul de la constante, et choix d'une face comme hyperplan

```
import numpy as np
```

```
# Les policiers seront representes par une tableau L
# contenant leurs coordonnees respectives.

def sgn(x):
    if x < 0 : return -1
    else : return 1

def orientation(a,b,c):
    ab = [b[0]-a[0],b[1]-a[1]]
    ac = [c[0]-a[0],c[1]-a[1]]

    composante = ab[0]*ac[1]-ab[1]*ac[0]
    return sgn(composante)

def jarvis(L):

    i_point_gauche = 0

    #On trouve le point le plus a gauche
    for i in range(len(L)):
        if L[i][0] < L[i_point_gauche][0]: i_point_gauche = i

    point_courant = i_point_gauche
    point_prochain = point_courant - 1

    liste_points = [L[i_point_gauche]]

    while point_prochain != i_point_gauche :

        point_prochain = 0

        for i in range(len(L)):
            if point_prochain == point_courant or orientation(L[point_courant],L[
point_prochain],L[i])  == -1 :
                    point_prochain = i

        liste_points.append(L[point_prochain])

        point_courant = point_prochain

    return liste_points

# Renvoie une liste de points correspondants aux extremites des
# vecteurs pointants normalises
def normalise(L,V):
    return [V + (p-V)/norme(p-V) for p in L]

# Renvoie la distance de V a la droite passant par A et B """
def distance(A,B,V):
    # (AB) : ax + by + c = 0
    a = B[1] - A[1]
    b = A[0] - B[0]
    c = B[0]*A[1] - B[1]*A[0]

    return np.abs(V[0]*a + V[1]*b + c)/np.sqrt(a**2+b**2)

# Trouve le rayon d'une boule contenue dans l'enveloppe convexe
# evoquee et centree en V
def boule(L,V):
```

```
    L_norm = normalise(L,V)
    enveloppe = jarvis(L_norm)
    dist = []
    for i in range(0,len(L)-1):
        dist.append(distance(enveloppe[i],enveloppe[i+1],V))
    return min(dist)

# L est la liste des policiers, V le voleur
def trouve_face(L,V):
    enveloppe = jarvis(L)

    for i in range(0,len(env)-1):
        if(orientation(V,env[i],env[i+1])== -1):
            return(env[i], env[i+1])
```

Références

[1] Martin Aigner and Michael Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1) :1–12, 1984.

[2] Swastik Kopparty and Chinya V Ravishankar. A framework for pursuit evasion games in R^n. *Information Processing Letters*, 96(3) :114–122, 2005.

[3] Gabriel Pallier. Une géométrie pour les graphes d'amitié. *Quadrature*, 99 :16–19, 2016.

[4] Paweãl Praãlat. When does a random graph have constant cop number. In *Australasian Journal of Combinatorics*. Citeseer, 2010.