



Robotique en essaim

Julien ANDRES
Thomas TAYLOR

Nicolas BREDECHE



Sommaire

1	Introduction	2
1.1	Robotique en essaim	2
1.2	Couverture	2
1.3	Agrégation	2
1.4	Algorithmes génétiques en ligne	2
1.5	Problématique	2
2	Matériel utilisé	3
2.1	Kilobot	3
2.2	Bloc Actif	3
2.3	Arène	4
2.4	Simulateur	4
3	Couverture	5
3.1	Algorithme	5
3.2	Résultats	6
3.2.1	Commencement groupé	6
3.2.2	Commencement éparpillé	7
3.2.3	Comparaison	8
4	Agrégation	9
4.1	Agrégation naïve	9
4.2	Agrégation probabiliste	10
4.3	Résultats	11
4.3.1	Agrégation naïve	11
4.3.2	Agrégation probabiliste - Analyse Probabilité	11
4.3.3	Agrégation probabiliste - Analyse Paramètres	13
5	mEDEA	14
5.1	Algorithme	14
5.2	Résultats	15
6	Conclusion	16
7	Bibliographie	16

1 Introduction

1.1 Robotique en essaim

La robotique en essaim s'inspire d'observations d'insectes et d'animaux sociaux pour l'étude de comportements collectifs issus de simples tâches individuelles. Les principales propriétés de ce domaine sont la robustesse, grâce à un code décentralisé, et l'échelle proposée, le code pouvant être déployé sur différentes tailles d'essaim.

1.2 Couverture

La couverture est un problème fondamental dans la robotique en essaim. En effet, elle permet aux robots de se déployer dans un environnement inconnu ou dynamiques, tout en maintenant un contact entre eux. Ce problème se retrouve dans beaucoup de domaine, comme le sauvetage de personnes en milieu hostile ou inconnu.

1.3 Agrégation

L'agrégation est un comportement de base des essaims. Il permet aux organismes de faire face ensemble à un problème. Dans le domaine de la robotique en essaim, ce comportement permet à des robots dispersés dans un environnement de se regrouper en un ou plusieurs groupes.

1.4 Algorithmes génétiques en ligne

L'objectif de ces algorithmes est d'arriver, à partir d'un génome aléatoire définissant un comportement d'un robot, à un comportement permettant au robot de survivre dans son environnement. L'aspect *en-ligne* de l'algorithme représente la distribution du code sur l'ensemble des robots. ??? ajouter apprentissage ensemble ??

1.5 Problématique

2 Matériel utilisé

2.1 Kilobot

Un Kilobot [1] est un robot miniature, créé par l'université d'Harvard. Ces robots permettent l'étude de la robotique en essaim, leur faible cout étant un avantage pour la manipulation d'un grand nombre d'individus.

Communication Les robots peuvent communiquer entre eux grâce à des messages qu'ils envoient à leurs voisins à une distance d'environ 7cm. L'intensité du signal permet à un Kilobot d'évaluer la distance à laquelle se trouve la source du message. Chaque robot possède aussi un capteur de luminosité.

Mouvements Chaque robot possède deux moteurs, placé de chaque coté de son corps. Ils permettent au robot d'avancer en faisant vibrer ses pattes. Les déplacement d'un Kilobot se font donc en modulant la vitesse de chacun de ses moteurs selon la direction voulu.

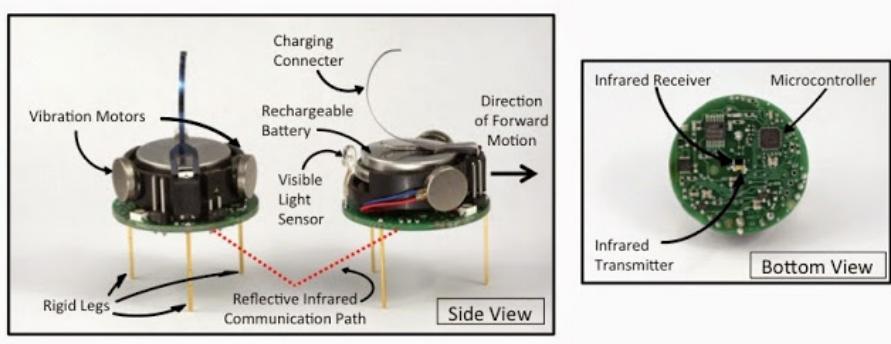


Figure 1: Kilobot

Les comportements des robots sont écrits en C, en utilisant l'API kilolib et peut être transmis à l'ensemble des Kilobots grâce à un contrôleur infrarouge. reçoit

2.2 Bloc Actif

Le bloc actif a été développé par l'ISIR dans le cadre de la recherche sur Kilobots. Elle permet d'interagir avec les robots en émettant un signal infrarouge. Le numéro du message et l'intensité du signal peuvent être modulé grâce à des boutons sur le bloc actif. Il fera office d'obstacle et de point de ralliement pour certains de nos algorithmes.

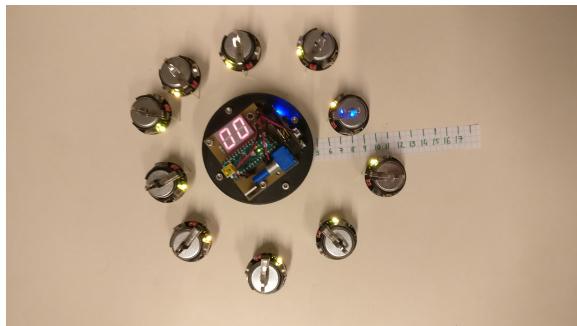


Figure 2: Distance max à intensité max

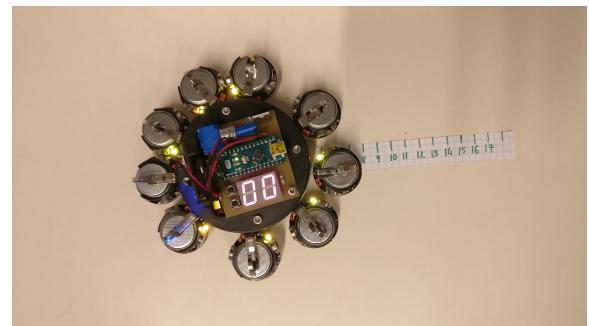


Figure 3: Distance max à intensité min

Les robots devant être vraiment trop collés au bloc pour recevoir le signal en intensité minimale, nous utiliserons dans toutes les expériences l'intensité maximale du bloc actif.

2.3 Arène

Les algorithmes ont été testé sur un essaim de 70 robots, fournit par le laboratoire de rechercher ISIR. L'arène utilisé pour l'étude des algorithme faisait 1x2 mètres.

PHOTO ARENE

2.4 Simulateur

Afin de faciliter le développement et l'étude des algorithmes, le simulateur Kilombo [2] a été utilisé. Il inclut l'API kilolib et permet ainsi une compatibilité direct du code entre le simulateur et l'essaim de robots.

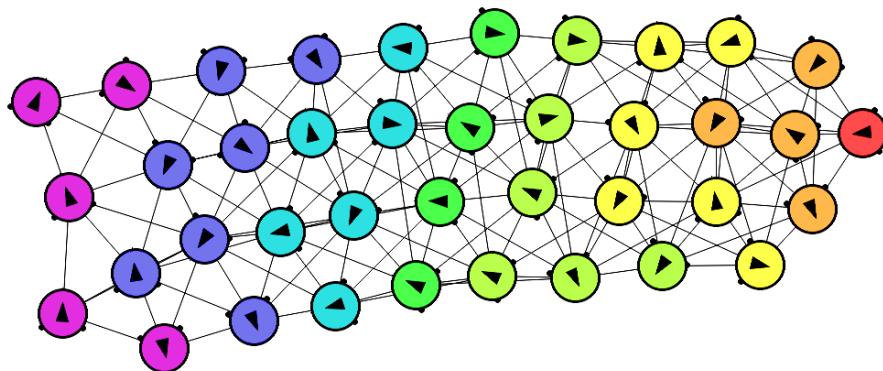


Figure 4: Modélisation d'un gradient sur simulateur

3 Couverture

3.1 Algorithm

Cet algorithme est utilisé afin de couvrir la plus grande surface possible, tout en gardant un lien de communication entre les Kilobots. Il est composé de 3 comportements basique : L'éloignement (*repel*), le déplacement aléatoire(*searching*) et l'attente(*wait*)

Le déplacement aléatoire Ce comportement s'effectue lorsque le Kilobot ne détecte aucun voisin. Il bouge alors aléatoirement dans son environnement jusqu'à la détection d'un autre robot. Il passe alors en mode d'attente ou d'éloignement selon la distance à laquelle il est de ses voisins.

L'éloignement Si le Kilobot se trouve trop proche de ses voisins, il essayera de s'en éloigner. Soit en s'éloignant de son voisin le plus proche, soit en bougeant aléatoirement jusqu'à être à une distance désirée de tous ses voisins.

L'attente Le Kilobot entre en attente quand il se trouve à une distance supérieure ou égale à celle demandé de tous ses voisins. Il sort de ce comportement s'il ne capte plus aucun voisins ou si un Kilobot arrive trop proche.

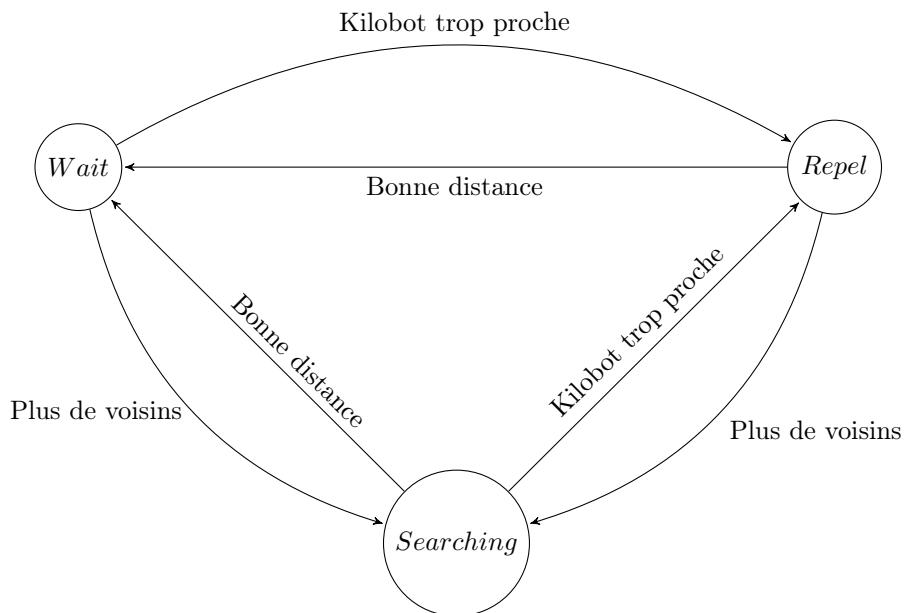


Figure 5: Couverture

3.2 Résultats

Deux positions de départ sont possibles pour l'essaim de Kilobot pour le problème de couverture. Nous pouvons partir d'un essaim regroupé en un groupe compact ou d'un essaim éparpillé sur l'ensemble de l'environnement. Nous étudions ici le comportement de l'essaim en fonction de la distance minimale demandé entre chaque robot.

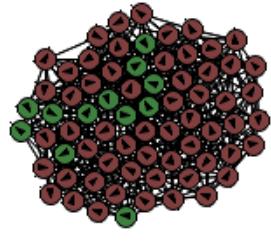


Figure 6: Commencement groupé

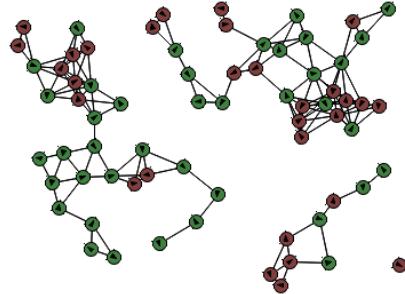


Figure 7: Commencement éparpillé

3.2.1 Commencement groupé

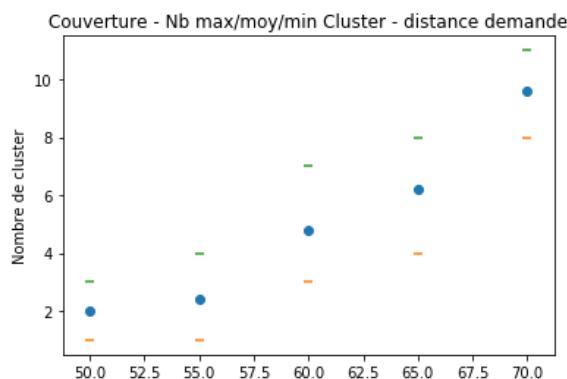


Figure 8: Nombre de cluster selon la distance

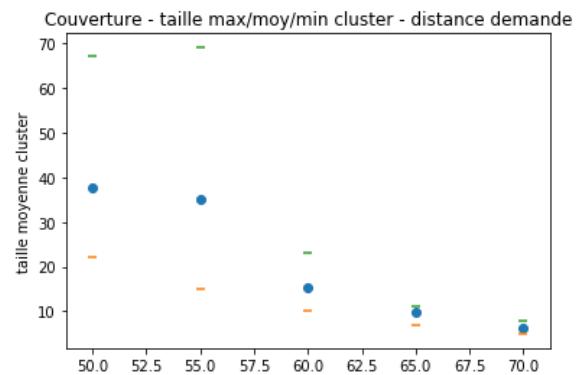


Figure 9: Taille moyenne d'un cluster selon la distance

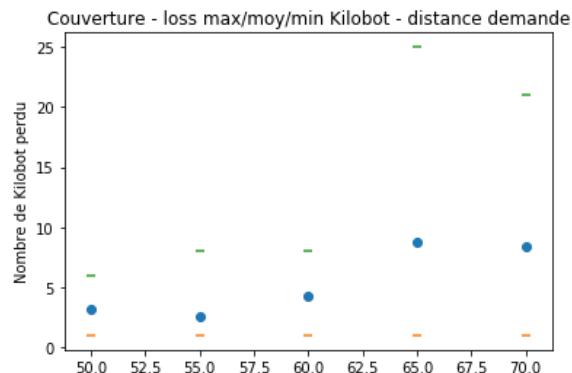


Figure 10: Nombre de Kilobot perdu selon la distance

On remarque ici un nombre très bas de robot *perdus* (c'est à dire qui n'appartiennent à aucun groupe).

Le nombre de robots dans un agrégat (c'est à dire un groupe de Kilobot en attente, communiquant tous ensemble) est assez grand pour des valeurs inférieures à 55mm, ce qui s'explique par la proximité des robots dès le départ. Les robots étant proches, il est normal qu'ils soient bien connectés entre eux.

3.2.2 Commencement éparpillé

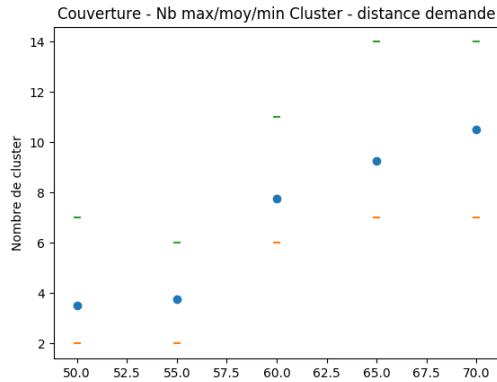


Figure 11: Nombre de cluster selon la distance

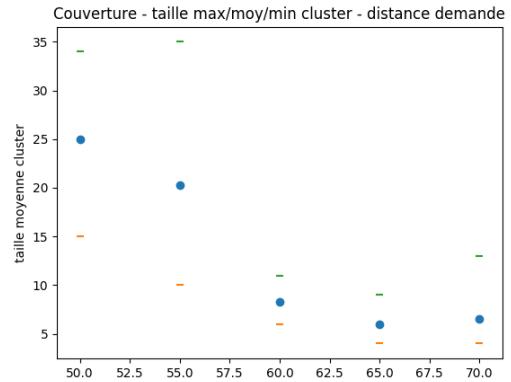


Figure 12: Taille moyenne d'un cluster selon la distance

On remarque immédiatement que plus la distance entre deux robot est grande (elle ne dépasse jamais la distance maximale de communication), moins la couverture est efficace. On remarque que plus la distance demandé est grande, plus le nombre total de groupe est grand et plus leur taille est petite. En effet, le groupe de robot, cherchant à atteindre cette distance minimale, va se fractionner et de nombreux petits groupes se formeront.

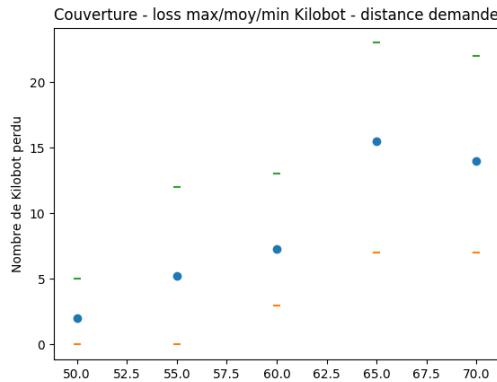


Figure 13: Nombre de Kilobot perdu selon la distance

On remarque aussi, que plus la distance est grande, plus le nombre de Kilobot *perdus* est grand. On observe un grand changement entre une distance de 60 et de 65. Ce grand écart est aussi observable sur les graphiques précédents.

3.2.3 Comparaison

On remarque que le départ depuis un essaim groupé amène à une configuration des Kilobots plus connectés contenant moins de robots perdus qu'un départ éparpillé. En effet, la proximité des Kilobots dès le début de l'expérience les empêche de rester seul.

Cependant, cette connexité se fait au détriment de la durée de convergence de l'algorithme, qui est plus rapide depuis un essaim éparpillé (l'aleatoire ayant déjà fait une partie du travail).

4 Agrégation

4.1 Agrégation naïve

Le but de cet algorithme est de faire s'agréger les Kilobot vers un objet pouvant représenter une source de nourriture par exemple. Cet objet sera représenté ici par le bloc actif. Le principal but de cet algorithme étant de former un groupe compact de Kilobot, autour du bloc actif.

Dans cet algorithme, les robots ne communiquent que pour indiquer l'emplacement du bloc actif. Ils n'ont donc aucune autre information sur leur environnement. Cet algorithme est composé de 3 comportements : la recherche, l'approche et l'émission.

Recherche Le robot bouge aléatoirement dans son environnement afin de trouver une source d'émission. Ici, le Kilobot ne fait aucune hypothèse sur son environnement. Il n'émet rien et ne reçoit donc aucune information provenant des autres robots dans le même état que lui. Ce comportement cesse lorsqu'il reçoit un message, le prévenant de la présence du bloc actif ou d'un robot proche de ce bloc actif. Il passe alors en comportement d'approche.

L'approche Lors de cette phase, le robot va essayer de s'approcher du signal le plus proche. La méthode d'approche utilisé (qui est la même dans tous les algorithmes qui suivent) consiste à approcher en faisant des rotations. A la réception du message, le robot choisit aléatoirement une direction entre droite et gauche. A chaque nouveau message, si le Kilobot s'est rapproché de la source, il continue dans la même direction, sinon il prend la direction opposée.

Il sort de ce comportement s'il perd le signal de la source (retourne à la recherche) ou s'il est assez proche de tous ses voisins émetteurs. Il passe alors dans l'état d'émission.

Émission Le robot se trouve à une distance inférieure ou égale à celle demandé par l'algorithme. Il se met donc à émettre à son tour afin de transmettre l'information aux robots qui lui sont proches.

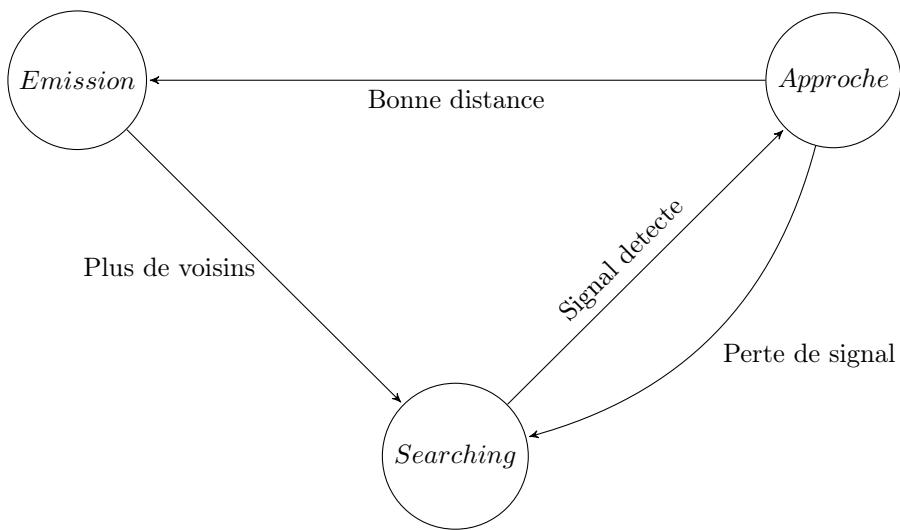


Figure 14: Agrégation Naïve

4.2 Agrégation probabiliste

Cet algorithme découle de celui présenté par O.Soysal et E.Sahin. [3] Il permet aux robots de s'agglomérer basé sur une probabilité calculé indépendamment par chaque robot.

Il est composé de 4 comportement basiques : l'approche (*approach*), l'attente (*wait*), l'éloignement (*repel*) et un évitemenent d'obstacle. Un Kilobot n'ayant pas de capteur de proximité, il lui est impossible de détecter un obstacle tel qu'un mur. L'évitement d'obstacle a donc été remplacé par un déplacement aléatoire dans l'environnement (*searching*).

Le déplacement aléatoire Quand le robot est seul dans l'environnement (aucun voisins), il bouge de façon aléatoire. Toutes les secondes, il choisit une direction parmi les 3 disponibles (gauche, droite, tout droit) et la garde jusqu'à la prochaine actualisation. Dès qu'il détecte un voisin, il passe en comportement d'approche.

L'approche Dans ce comportement, le robot décide de s'approcher du voisin le plus intéressant. Le voisin le plus intéressant est défini comme le celui ayant le plus de voisins à coté de lui. Si le Kilobot arrive à rejoindre le groupe, ce qui n'est pas obligatoire, à cause de l'absence de capteurs de proximité, il s'arrête et passe dans l'état d'attente.

L'attente Quand le robot est dans cet état, il fait parti d'un groupe composé d'au moins 2 individus. Il a cependant une probabilité de quitter ce groupe et d'entrer dans l'état d'éloignement à chaque seconde.

L'éloignement Ce comportement fait s'éloigner le robot du groupe auquel il était agrégé précédemment. Pendant toute la durée de cet éloignement, à chaque seconde, le Kilobot à une probabilité de revenir en mode d'approche et de rejoindre le groupe qu'il vient de quitter. Ce comportement s'arrête quand le robot est seul ou que 10 secondes se sont écoulées. Il revient en déplacement aléatoire.

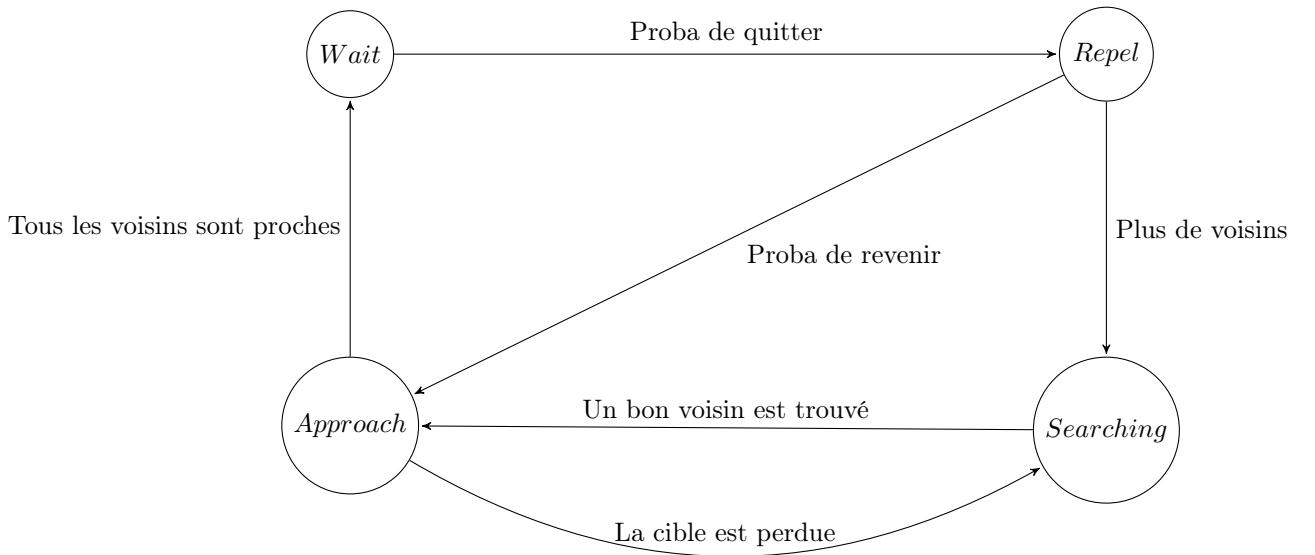


Figure 15: Agrégation probabiliste

4.3 Résultats

4.3.1 Agrégation naïve

4.3.2 Agrégation probabiliste - Analyse Probabilité

Cet algorithme se base sur la probabilité d'un robot de quitter son groupe afin de trouver potentiellement un nouveau groupe composé de plus d'individus (et donc plus intéressant). Le paramètre important de cet algorithme est donc le calcul de cette probabilité. Nous avons ici testé trois différentes façon de calculer cette probabilité.

Probabilité constante Nous utilisons ici la probabilité constante proposée par O.Soyal et E.Sahin comme étant celle pour laquelle ils ont obtenu les meilleurs résultats.

$$P(x) = 0.0001$$

Probabilité suivant taille groupe Afin de permettre aux gros agglomérats de robots de rester ensemble, nous avons défini une probabilité pour un Kilobot de quitter un groupe en fonction du nombre de voisins de celui ci par rapport à une constante définie comme la taille d'un groupe que nous considérons *stable*. Or après expérimentation, nous avons remarqué qu'un robot arrivant au bord d'un agglomérat ne reste pas forcément car connecté que à quelques individus de ce groupe. Afin de traiter les deux cas d'appartenance à un groupe (au centre ou sur un bord), nous prenons donc le maximum entre le nombre de voisin du Kilobot et le nombre maximum de voisin du meilleur de ses voisins. Le meilleur de ses voisins étant défini comme le voisin ayant le plus de voisins.

$$P(x) = \min(1, \frac{v(x)}{\text{TAILLE_GRAND_GROUPE}})$$

avec

$$v(x) = \max(\text{nombre_de_voisin}(x), \text{nombre_de_voisin}(\text{meilleur_voisin}(x)))$$

Probabilité mixte Cette probabilité a été trouvé après avoir vu les résultats expérimentaux des deux probabilités précédentes. Elle a comme objectif de créer des agglomérats stables en ne permettant pas à un robot de partir s'il appartient à un suffisamment grand groupe, et utilisant la probabilité trouvée par O.Soyal et E.Sahin. Elle est définie comme :

$$P(x) = \begin{cases} 0 & \text{si } v(x) \geq \text{TAILLE_GRAND_GROUPE} \\ 0.0001 & \text{sinon} \end{cases}$$

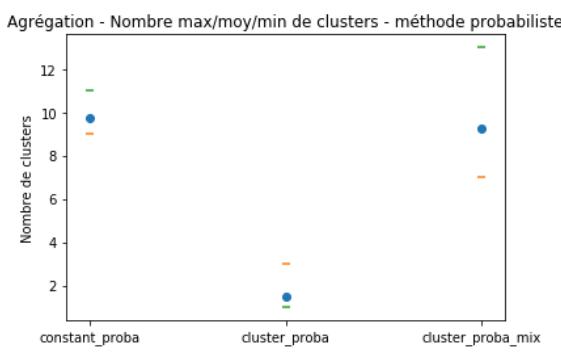


Figure 16: Nombre de cluster selon probabilité

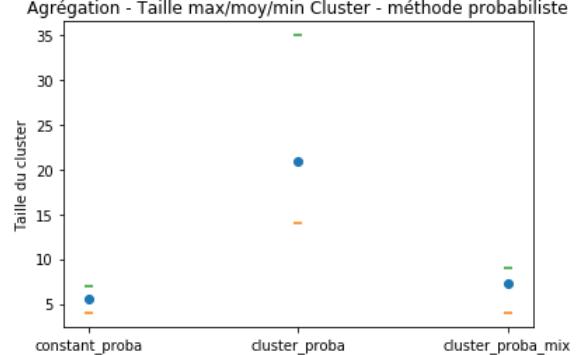


Figure 17: Taille moyenne d'un cluster selon la probabilité

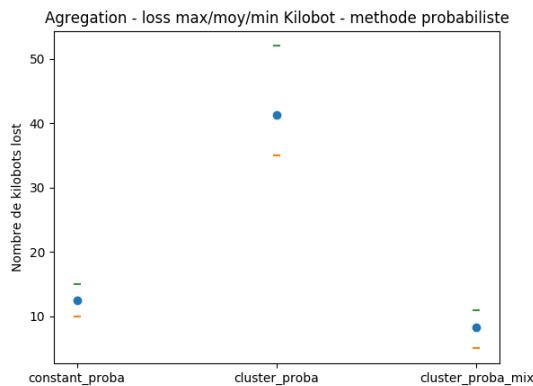


Figure 18: Nombre de robots perdu selon probabilité

On observe que la probabilité constante, proposée par O.Soysal et E.Sahin, donne un grand nombre de petits groupes. En effet, les agrégats sont formés rapidement et la faible probabilité de partir ne permet pas de la mise en place d'un grand groupe.

En revanche, la probabilité suivant la taille du groupe permet de remédier à ce problème. On observe en effet un petit nombre de groupe composée d'un grand nombre d'individu. Un individu dans un petit groupe ayant en effet une forte probabilité d'en sortir. Cependant, cette mesure de probabilité ne domine pas la première, le nombre de robot perdu étant largement supérieur à celle ci.

4.3.3 Agrégation probabiliste - Analyse Paramètres

Afin de remédier aux défauts de la probabilité linéaire suivant la taille d'un grand agrégat, nous avons essayé d'étudier les effets de cette taille TAILLE_GRAND_GROUPE, afin de faciliter la création d'un groupe robuste et de minimiser le nombre de robots perdus.

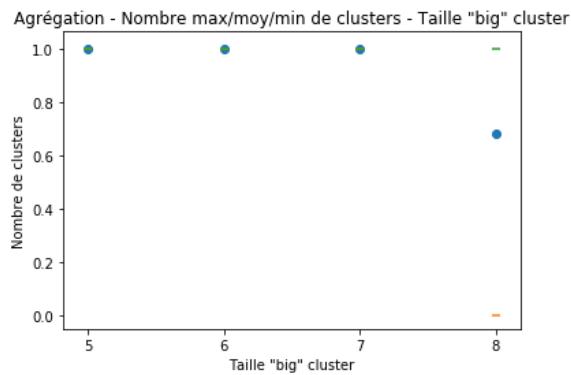


Figure 19: Nombre de cluster selon TAILLE_GRAND_GROUPE

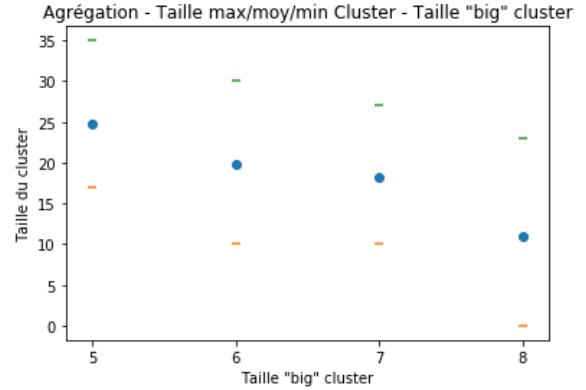


Figure 20: Taille moyenne d'un cluster selon TAILLE_GRAND_GROUPE

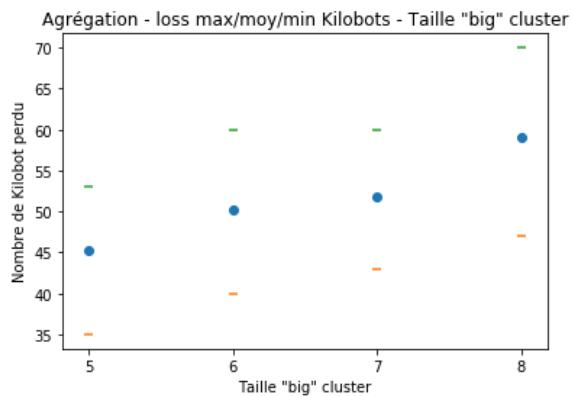


Figure 21: Nombre de robots perdu selon TAILLE_GRAND_GROUPE

On remarque ici que plus la valeur de TAILLE_GRAND_GROUPE est faible, moins les robots se perdent. Cependant, une petite valeur ne donne pas forcément plus de groupes.

Une valeur trop haute, en revanche, induit un grand nombre de Kilobot perdu et une taille de groupe plus petite. En effet, un Kilobot a rarement plus de 6/7 voisins autour de lui. La probabilité ne permet donc pas la création d'agrégats stables.

5 mEDEA

Nous implémentons ici la version Vanilla-EE de l'algorithme mEDEA présenté par J.Montanier, S.Carrignon et N.Bredeche. [5] Il permet à un essaim de robot de s'adapter à leur environnement en fonction des contraintes de celui ci.

5.1 Algorithme

```
1      Initialisation aléatoire du génome;
2      while True do
3          for i=0 to lifetime do
4              if self.genome!=NULL then
5                  move() //selon le genome
6                  fitness=calculerFitness()
7                  transmettre(génome,fitness) //aux voisins accessibles
8
9              end if
10             genomeQueue=receptionGénome() //traitement des messages
11             if genomeQueue!=NULL then
12                 genomeList.add(genomeQueue)
13             end if
14         end for
15         self.genome=NULL
16         if genomeList.size > 0 then
17             newGenome=select(genomeList) //selection selon meilleure fitness
18             newGenome=mutation(newGenome) //proba de muter
19             self.genome=newGenome
20         end if
21         genomeList.empty()
22     end while
23
```

Figure 22: Vanilla-EE

Dans notre cas, l'algorithme est exécuté sur chaque robot et évalue le comportement du robot pendant ses déplacements. Un robot correspond donc aux paramètres du comportement courant. Un Kilobot ne dispose que de 3 capteurs d'entrée :

- La luminosité
- Les messages reçus
- La distance du message à l'envoyeur(calculé par le Kilobot)

Il nous a donc fallu créer des capteurs *artificiels* via le transfert d'information entre les Kilobots par message. Notre génome est donc constitué des poids d'un perceptron sans couches cachées. Les entrées sont donc :

- Une entrée active en permanence comme neurone de biais
- La moyenne des distances à tous les voisins connus
- Le nombre de voisin actuel
- Une entrée binaire associé à la présence ou non du bloc actif

Il y a deux sorties, correspondantes aux gauche et droit de chaque Kilobot. Cela nous donne donc un génome de 8 gènes correspondant aux 8 paramètres du perceptron. Chaque paramètre peut prendre 3 valeurs : {-1,0,1}.

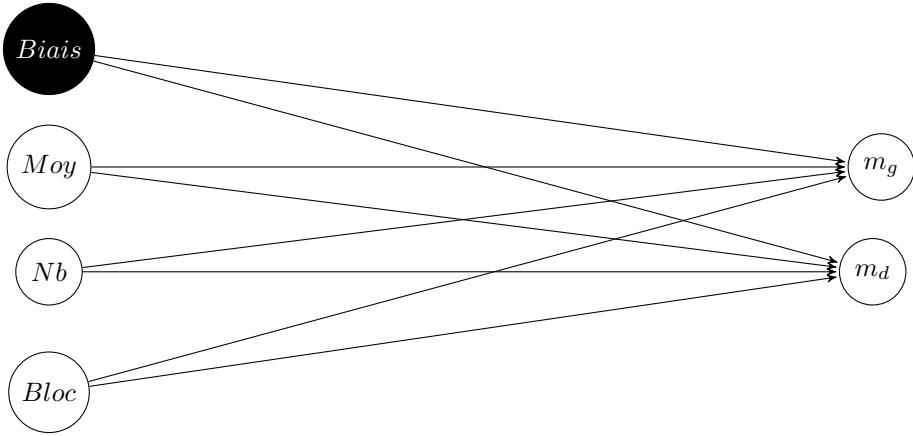


Figure 23: Contrôleur mEDEA

Fonction d'activation Notre fonction d'activation permet d'obtenir une sortie dans l'intervalle de vitesse des moteurs :

$$A(x) = \frac{1}{1+e^{-x}} * (Max_moteur - Min_moteur) + Min_moteur$$

Fitness

$$F(x) = \sum_i f(x) \quad \forall i \in 0, 1, \dots, evalTime$$

avec

$$f(x) = \begin{cases} 1 & \text{si Block actif detecté} \\ 0 & \text{sinon} \end{cases}$$

Notre fitness est ici calculé sur les 40 dernières demi-secondes. Elle est mise à jour à chaque envoi de message du Kilobot vers ses voisins.

Mutation La probabilité de mutation est un facteur important de l'algorithme. Un probabilité trop grande peut mener à une population incapable de se stabiliser alors qu'un taux trop bas peut amener à une absence de génome intéressant.

```

1   if random() < P_mutation then
2       for (i=0;i<len(genome);i++) do
3           if random() < 1/len(genome) then
4               modif=randint(1,2)
5               genome[i]=(genome[i]+modif+1) mod 3 -1
6           end if
7       end for
8   end if
9

```

Figure 24: Mutation

Après la décision d'une mutation sur le génome, nous avons décidé de mettre une probabilité fixe de $\frac{1}{\text{nombre de gènes}}$ par gène. Cela permettant d'avoir en moyenne un seul gène changé par mutation afin de ne pas avoir un génome changeant drastiquement d'une génération à une autre.

5.2 Résultats

6 Conclusion

7 Bibliographie

References

- [1] M. Rubenstein, C. Ahler, and R. Nagpal, “Kilobot: A low cost scalable robot system for collective behaviors”, in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3293–3298.
- [2] F. Jansson, M. Hartley, M. Hinsch, I. Slavkov, N. Carranza, T. S. G. Olsson, R. M. Dries, J. H. Grönqvist, A. F. M. Marée, J. Sharpe, J. A. Kaandorp, and V. A. Grieneisen, “Kilombo: A Kilobot simulator to enable effective research in swarm robotics”, Nov. 2015.
- [3] O. Soysal and E. Sahin, “Probabilistic aggregation strategies in swarm robotic systems”, in *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pp. 325–332.
- [4] A. Howard, M. J. Matarić, and G. S. Sukhatme, “Mobile Sensor Network Deployment using Potential Fields: A Distributed, Scalable Solution to the Area Coverage Problem”, in *Distributed Autonomous Robotic Systems 5*, 2002, pp. 299–308.
- [5] J.-M. Montanier, S. Carrignon, and N. Bredeche, “Behavioral Specialization in Embodied Evolutionary Robotics: Why So Difficult?”, *Frontiers in Robotics and AI*, vol. 3, 2016.
- [6] N. Bredeche and J.-M. Montanier, “Environment-Driven Embodied Evolution in a Population of Autonomous Agents”, in *Parallel Problem Solving from Nature, PPSN XI*, 2010, pp. 290–299.