SCIENCES SORBONNE UNIVERSITÉ

Recherche de consensus en robotique en essaim

présenté par LY Jean-Baptiste
M1 ANDROIDE

dans le cadre de l'UE P-ANROIDE encadrée par
BREDECHE Nicolas et MAUDET Nicolas

Juin 2020

Table des matières

Introduction

Le sujet de projet a pour l'objet l'étude du problème du *best-of-n* en robotique en essaim. Le problème du *best-of-n*, consiste à une prise de décision collective au sein d'un ensemble de robots aux capacités de communication et de calcul limitées. Parmi n options disponibles, l'essaim doit choisir l'option qui offre la meilleure solution possible afin de satisfaire leurs besoins actuels [1].

L'objectif de ce projet est d'étudier l'émergence de consensus au sein de l'essaim, c'est-à-dire une prise de décision collective résultant à un accord commun parmi tous les agents. Le sujet s'inspire du comportement collectif des insectes sociaux tels que les abeilles et les fourmis. Il permet alors de fusionner plusieurs domaines ensemble tels que les systèmes multi-agents, l'éthologie et la biologie. Afin de simuler ces situations, le projet devait initialement se mener sur des robots appelés "Kilobots", mais finalement il a été mené sur un de ses simulateurs, nommé "Kilombo".

Dans un premier temps, il s'agit d'implémenter un algorithme existant permettant d'atteindre de manière distribuée un consensus entre n ressources. Dans un second temps, il est question d'implémenter un algorithme d'apprentissage afin d'évaluer les difficultés que peuvent poser l'apprentissage de consensus.

L'espace de recherche est défini à partir des comportements de phototaxis (les robots se dirigent vers une source de lumière), anti-phototaxis (les robots fuient la lumière) et déambulation libre (les robots se dirigent de manière aléatoire). Il s'agira d'apprendre les conditions de transitions entre chaque comportement. On s'intéresse d'abord au cas ou l'essaim doit se regrouper autour de la ressource de plus grande valeur, puis le cas où l'essaim doit distribuer ses forces au pro-rata de la valeur de chaque ressource.

Ce projet inclut ces trois articles et se basera sur ces derniers :

* Valentini et al. (2016) Collective decision with 100 Kilobots : Speed versus accuracy in binary discrimination problems. AAMAS.

* Valentini et al (2017) The best-of-n problem in robot swarms : Formalization, state of the art, and novel perspectives. Frontiers in AI and Robotics.

* Bredeche et al. (2012) Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents. MCMDS.

Chapitre 1

Présentation

1.1 Le kilobot : un mini robot

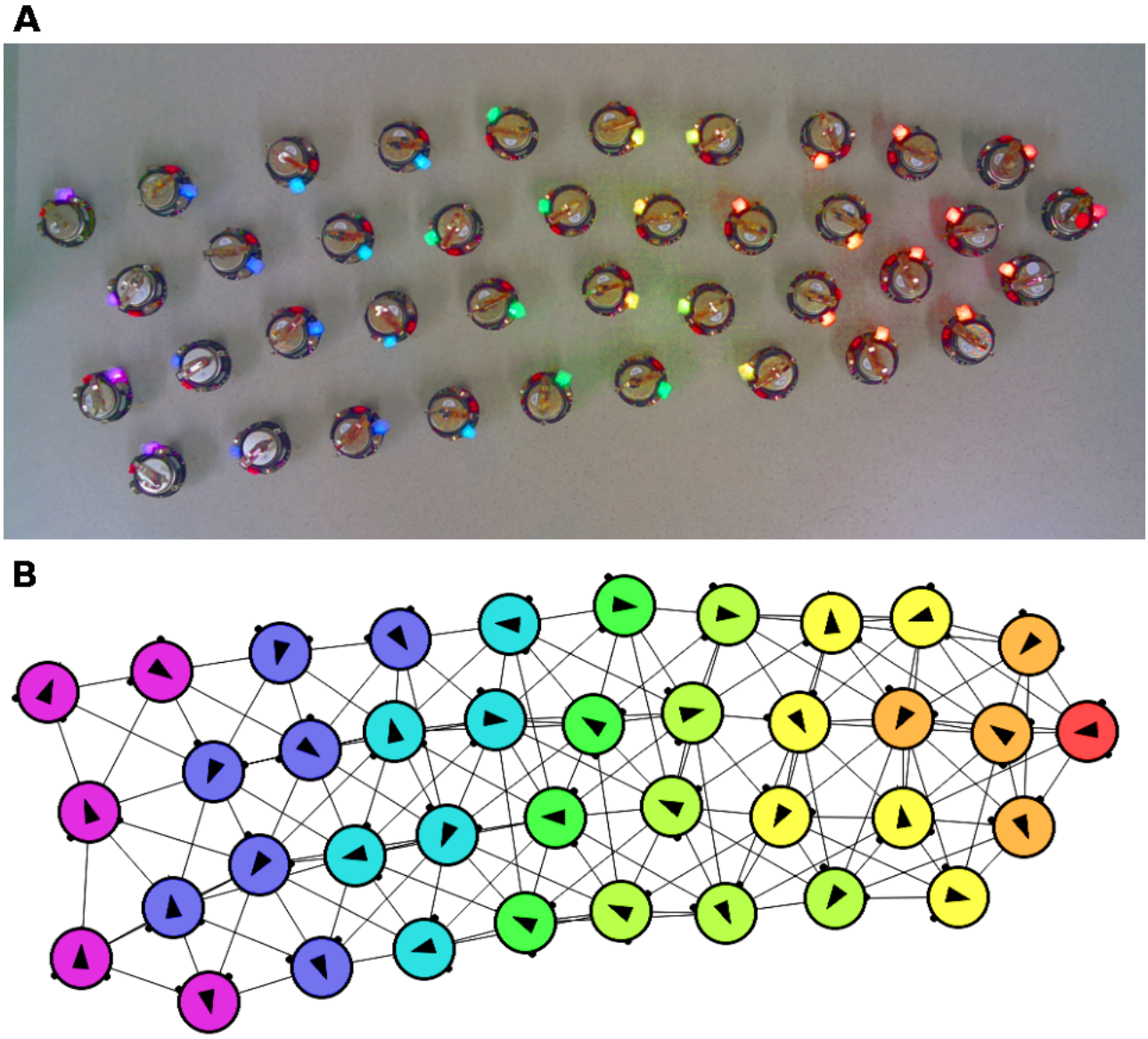
Les kilobots sont des mini robots possédant un diamètre de 33mm, destinés à la robotique en essaim. Ils ont été élaborés par l'Université d'Harvard dans le but de pouvoir appliquer des algorithmes utilisant des dizaines voire des centaines de robots, tout en réquérant un faible coût. Ils contiennent un microcontrôleur Atmel ATmega328P, qui est programmable en langage C. Les robots sont équipés de LED, de capteurs de lumière ambiante et de des installations de communication infrarouge à courte portée. Les kilobots peuvent communiquer avec leurs voisins jusqu'à une distance de 7 cm en réfléchissant la lumière infrarouge (IR) sur la surface du sol. Ils se déplacent sur des tiges métalliques raides à l'aide de deux moteurs à vibration.

FIGURE 1.1 – Un kilobot

1.2 Kilombo : un simulateur de kilobots

Le simulateur Kilombo permet de simuler efficacement les mouvements de kilobots, et permettent la possibilité d'accélérer leurs mouvements tout en veillant à une bonne simulation. Ainsi cela permet un gain de temps considérable par rapport aux vrais kilobots. De plus, le langage d'implémentation de Kilombo est le même que celui des kilobots, le langage C. Ce qui permet une bonne portabilité pour le code entre les deux plateformes : les vrais kilobots et leur simulateur. Les expérimentations du projet ont été faites exclusivement dessus.

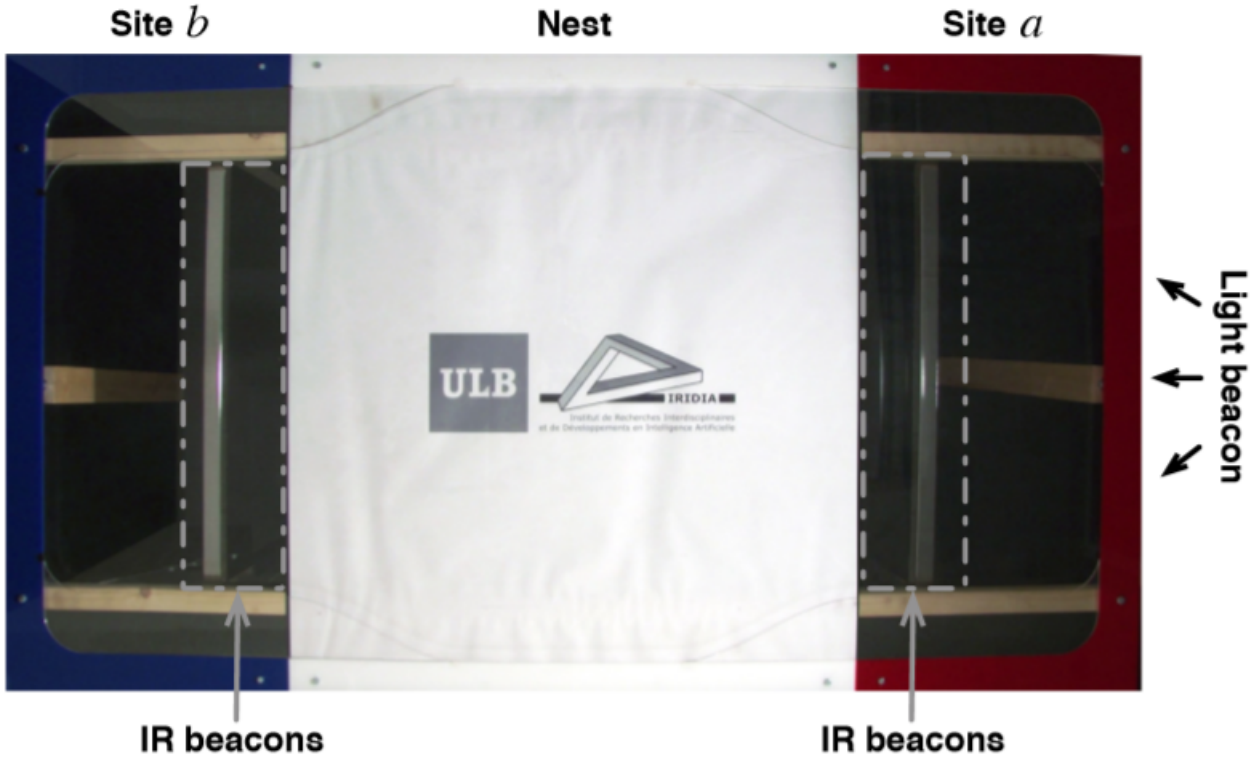
FIGURE 1.2 – Exemple d'un essaim de robots propageant un signal qui forme un gradient. (A) Le programme fonctionnant sur des Kilobots réels ; (B) Le même programme fonctionnant dans le simulateur. [4]



1.3 L'arène réelle

Tout au long du projet, les expériences ont été faites dans le cadre d'une reproduction de l'expérience de Gabriele Valentini, Eliseo Ferrante, Heiko Hamann et Marco Dorigo [2]. Il s'agit d'une arène rectangulaire de taille 100 x 190 cm^2 avec comme surface de déplacements une plaque de plexiglass. L'arène est composée d'une zone centrale appelée "le nid" (appelée aussi zone de négociation ou de dissémination) où est placée en-dessous du plexiglass une surface opaque, et n sites représentant les ressources à explorer, toutes à égales distances de la zone de négociation. Dans le cas de 2 sites à explorer par exemple, ils sont placés à gauche et à droite de la zone de dissémination, et mesurent chacun 80 $\times$ 45 cm^2. Le nid est la seule zone où les agents pourront prendre leur décision. Les kilobots sont initialement placés dans le nid avec une position et orientation aléatoires.
Chaque site possède des lumières infrarouges qui donnent la qualité du site, ces lumières infrarouges sont données par des kilobots retournés.

FIGURE 1.3 – L'arène réelle [?]



1.4 L'arène simulée sur Kilombo

Dans le cadre du simulateur Kilombo, il est impossible de reproduire exactement les conditions réelles de l'arène. Ainsi il a fallu "tricher" pour parvenir à reproduire le principe de l'expérience réelle. Pour cela, on affecte une lumière différente à toutes les zones de l'arène (nid, sites...). Lorsqu'un kilobot veut rejoindre une telle zone, il sera attiré par phototaxis par la lumière de la dite-zone. Si cett dernière est un site, il connaîtra directement la valeur du site (il n'y aura pas de mesure concrète comme sur la vraie arène). Il n'y a donc pas de comportement d'antiphototaxis pour cette arène simulée.

1.5 Dissémination et exploration

L'expérience est composée de deux phases qui se suivent et recommencent. La première phase nommée "dissémination" ou "négociation" consiste au rassemblement des kilobots au nid. Lors de cette étape, le kilobot retourne au nid pour partager son avis aux autres agents en dissémination, et pour observer les avis de ses voisins afin de prendre une décision pour la seconde phase qu'est "l'exploration". Cette phase permet au kilobot, en fonction de sa prise de décision lors de la négociation, d'aller explorer le site voulu.

FIGURE 1.4 – L'arène simulée sur Kilombo (nid)

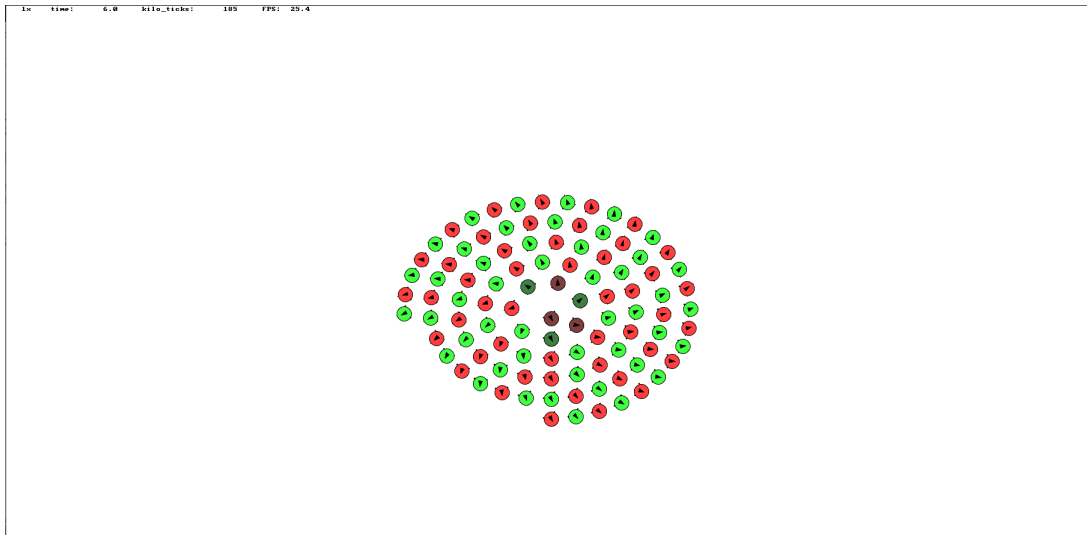
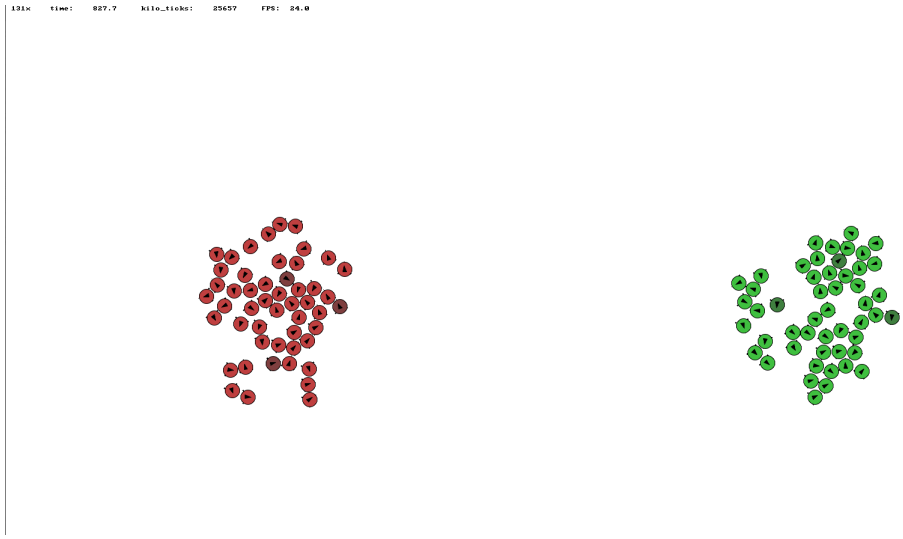


FIGURE 1.5 – L'arène simulée sur Kilombo (les deux ressources)



Chapitre 2

Les algorithmes dédiés

2.1 Descriptions et définitions

2.1.1 Majority rule

La stratégie de prise de décision collective nommée *Majority rule* consiste à prendre la décision majoritaire en fonction des voisins du kilobot rencontrés lors de la négociation dans le nid [2]. Lors de la négociation, l'agent observe l'avis de ses voisins et partage le sien. Par exemple pour un cas où il y a deux ressources a et b à explorer, s'il rencontre plus d'agents pour l'opinion a que pour l'opinion b, alors il ira au site a à la prochaine phase d'exploration, et inversement. En revanche s'il en rencontre autant d'agents pour la ressource a ou b, alors il gardera l'avis qu'il avait à la précédente exploration. Le temps de négociation est proportionnellement aussi élevé que la qualité du site qu'il l'a visité précédemment. Plus la qualité du site est élevée, plus il restera longtemps en phase de négociation, et meilleure sera sa propagation d'opinion.
Pour plus de deux ressources à explorer, en cas d'égalité entre plusieurs ressources, l'agent reprendra aussi tout simplement l'avis qu'il avait précédemment.

2.1.2 Voter model

La stratégie de prise de décision collective nommée *Voter model* consiste à prendre une opinion aléatoire parmi les agents rencontrés lors de la négociation. Ainsi dans le cas de deux sites a et b par exemple, plus il y a de voisins pour le site a en négociation, plus la probabilité de choisir le site a sera meilleure. Le temps de la phase de négociation se fait identiquement à la stratégie du *Majority rule*.

2.1.3 Les agents dits "têtus" ou "explorateurs"

Eliseo Ferrante et al. [6] mettent cette notion en pratique dans le cadre de sites dynamiques (décrit peu après). Les agents dits "têtus" ou "explorateurs" sont des agents qui ne changeront jamais d'opinion, mais pourront toujours propager la leur. Ils ont toutefois toujours les deux phases de dissémination et d'exploration.

2.2 Cas du regroupement autour de la ressource de plus grande valeur

Ce cas a déjà été expérimenté par Gabriele Valentini, Eliseo Ferrante, Heiko Hamann et Marco Dorigo [2] [5] avec les stratégies précédemment décrites.

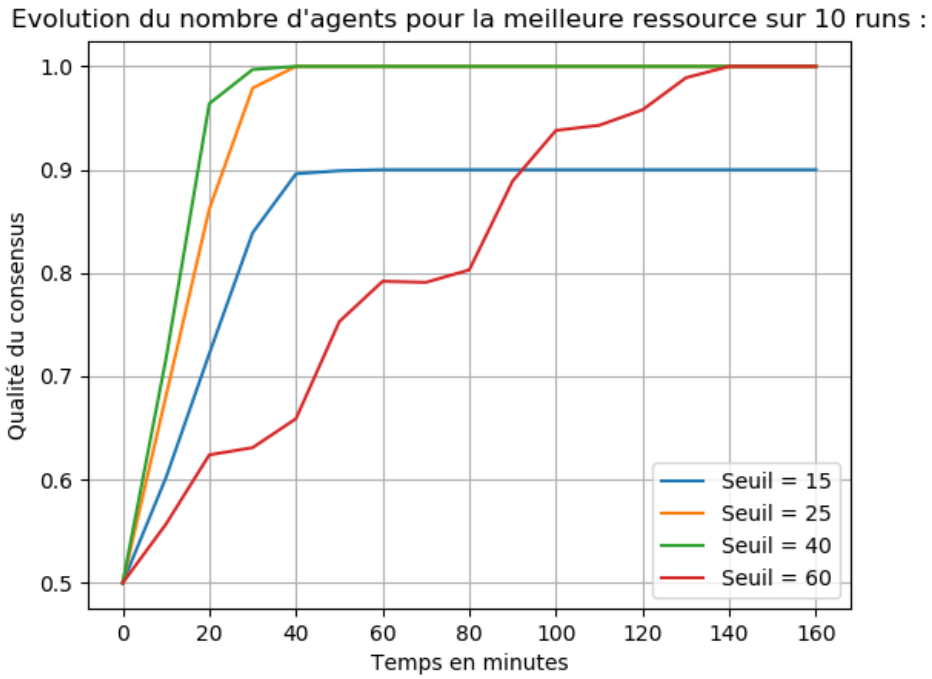
2.2.1 Sites statiques

Tout d'abord considérons des sites dont leur qualité ne change pas au cours du temps et restent la même pour chaque ressource.

Majority rule

L'objectif serait de réappliquer dans ce cas différents seuils décrits dans l'article [2] pour la stratégie du *Majority rule*, dans le cas où les qualites des ressources ne changent pas durant le temps. Les seuils utilisés sont de 15, 25, 40 et 60. Un seuil de k signifierait que dès qu'un agent rencontre k voisins différents, il prendra une nouvelle décision selon la stratégie du *Majority rule*. Si durant le temps de négociation, il n'atteint pas ces k voisins, alors il appliquera la règle à la fin du temps de négociation. Comme on peut constater sur la figure ci-dessous, un seuil de 60, qui équivaut à un seuil inexitant car rarement atteint lors de la négociation, met le plus de temps pour atteindre un consensus. Néanmoins si le seuil est trop faible, par exemple de 5, le consensus prendra du temps à être atteint, dû à la faible précision de prise de décision des agents. Soit $q_a = 1$ et $q_b = 2$ les qualités respectives de la ressource a et de la ressource b.

FIGURE 2.1 – Courbes des seuils du Majority rule avec deux ressources



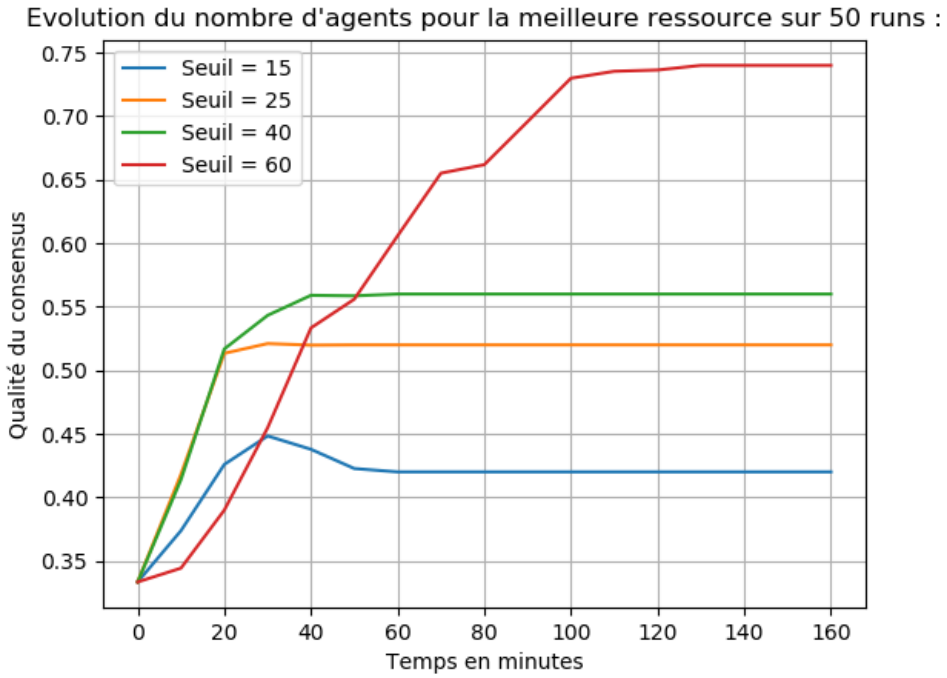
Un consensus est constaté lorsque la courbe devient stable (droite).

On constate que le seuil de 40 est meilleur alors que les expérimentations de Gabriele Valentini et al. [2] montrent que le seuil de 25 est le meilleur alors qu'on peut constater que le seuil de 40 est meilleur, l'obtention du consensus est plus rapide. Cela peut-être dû aux modifications de l'arène sur le simulateur. Un seuil de 15 nous montre bien en effet qu'un seuil trop bas, provoque un allongement du temps d'obtention de consensus. Sur le graphe, on constate que la qualité du consensus pour ce seuil de 15 n'est pas maximale car cela a pris trop de temps pour que le simulateur en prenne compte. Dû aux faiblesses du simulateur, ce dernier n'a pas pu terminé la simulation et a dû s'interrompre prématurément.

Retestons avec plus de deux ressources sur l'arène. On utilise toujours 100 agents (99 agents pour 3 ressources afin de faciliter les calculs) répartis équitablement pour chaque ressource lors de l'initialisation d'une simulation.

Pour trois ressources avec $q_a = 1$, $q_b = 2$, $q_c = 3$, on obtient ces courbes :

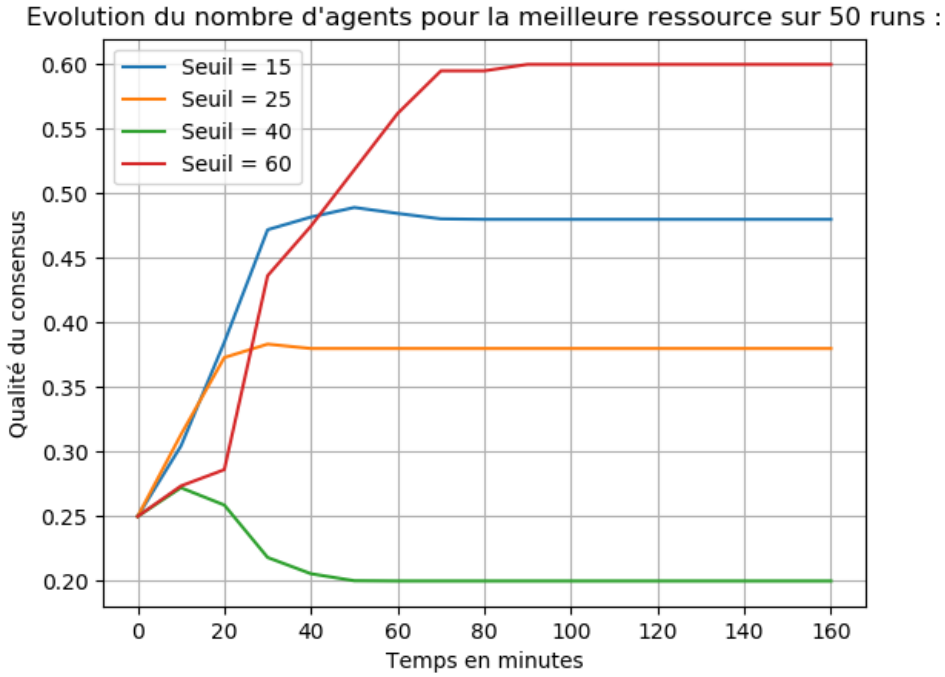
FIGURE 2.2 – Courbes des seuils du Majority rule avec trois ressources



On constate que sur 50 simulations, tous ne résultent pas à un consensus vers la meilleure ressource. Sur ces 50 simulations avec trois ressources, les résultats nous montrent que 37 simulations sur les 50 avec un seuil de 60 aboutissent à un consensus pour la meilleure ressource. Plus le seuil est élevé, plus le consensus prendra du temps à s'établir mais meilleur il sera. À l'inverse, lorsqu'un seuil est bas, on peut constater que le consensus est obtenu plus rapidement, néanmoins les agents auront plus de risques d'obtenir un consensus sur une autre ressource que la meilleure.

Pour quatre ressources avec $q_a = 1$, $q_b = 2$, $q_c = 3$, $q_d = 4$, on obtient ces courbes :

FIGURE 2.3 – Courbes des seuils du Majority rule avec quatre ressources



On constate qu'avec quatre ressources, le consensus vers une autre ressource autre que celle qui possède la meilleure qualité est encore plus souvent aperçu. De plus, on constate qu'avec un seuil de 40, seulement 10 simulations sur les 50 aboutissent au meilleur consensus.

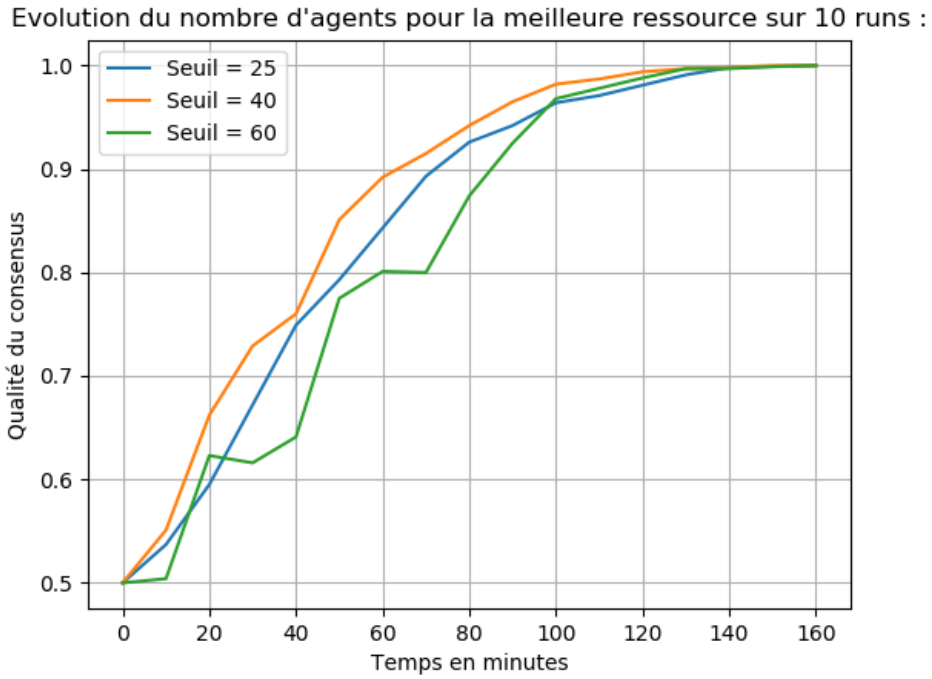
Interprétations

Ainsi plus le nombre de ressources augmente, plus les agents obtiendront un consensus vers une autre ressource que celle qui possède la meilleure qualité. Cette observation peut s'expliquer par les moyens limités du simulateur. Lors des simulations, beaucoup d'agents se retrouvaient bloqués par d'autres dans le nid. Ce qui les empêchaient d'explorer et d'être déjà en contact avec d'autres agents ayant une autre opinion trop tôt. Le fait aussi de reprendre l'opinion précédente de la stratégie du *Majority rule* en cas d'égalité peut aussi nuire à cela.

Voter model

Il s'agit de reproduire l'expérience précédente mais avec la règle du *Voter model* [5]. Les seuils expérimentés dans le graphe ci-dessous, sont 25, 40 et 60. Les seuils qui leur sont inférieurs ne sont pas représentés dans le graphe car du fait qu'ils prennent beaucoup plus de temps, le simulateur ne permet pas de finir au moment du consensus atteint, il se termine bien avant. Cette prise de temps considérable s'explique par le fait que la probabilité de prendre la ressource la plus faible est plus grande. Ainsi la convergence du consensus pour la plus grande ressource prendra beaucoup plus de temps. Prenons $q_a = 1$ et $q_b = 2$.

FIGURE 2.4 – Courbes des seuils du Voter model



Pour n'importe quel seuil, le bon consensus est atteint, contrairement à la stratégie du *Majority rule*. On constate qu'un seuil de 40 est correct. Un moindre seuil ou un seuil supérieur aboutirait à un consensus plus tardif.

Pour des cas possédant plus de deux ressources dans l'arène, les courbes ne sont pas malheureusement traçables dû aux limites du simulateur. En revanche en évitant l'implémentation des courbes, les limites sont bien plus souples et permettent de constater des simulations qui peuvent durer entre 3 heures et 4 heures (temps réel) pour trois ressources, et plus de 4 heures pour quatre ressources, avec $q_a = 1$, $q_b = 2$, $q_c = 3$ et $q_d = 4$ (si on prend quatre ressources).
Plus le nombre de ressources augmente, plus le consensus prendra du temps à s'obtenir.

2.2.2 Sites dynamiques

Considérons désormais des sites dont leur qualité change au cours du temps. Judhi Prasetyo et al. [6] décrit une arène possédant deux ressources a et b avec des valeurs de qualités différentes pour chacune. À un moment t, la valeur de chacune des qualités de ces ressources change en s'inversant par exemple. L'objectif est de changer le consensus une fois qu'un est atteint. Pour cela, on admettra la présence des agents dits "têtus" ou "explorateurs" comme l'a fait Judhi Prasetyo et al. [6]. On en initialise un pour chaque site dès le début d'une simulation.

Majority rule

Avec cette stratégie de décision, les agents ne pourront changer d'avis et resteront sur leur premier consensus. Cela s'explique par le fait qu'il est impossible par définition du *Majority rule*, qu'un agent change d'opinion avec un seul autre agent d'opinion opposé.

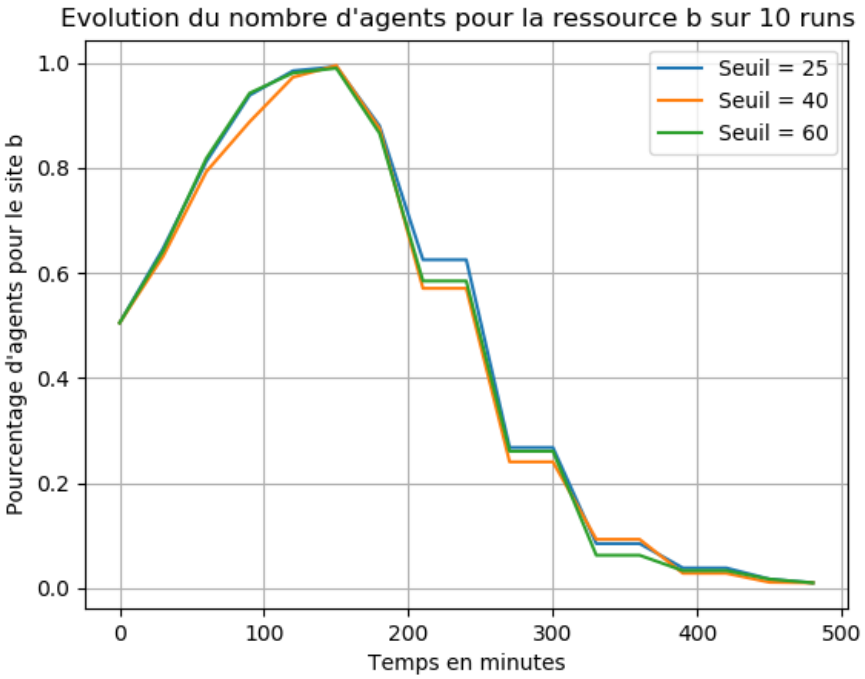
Voter model

Grâce à la notion d'aléatoire du *Voter model*, le changement de consensus au cours du temps est possible. En effet, une fois le premier consensus atteint, dès que les qualités des ressources s'inversent, le seul agent du site a (qui est un agent explorateur dans ce cas), restera plus longtemps dans le nid lors de la phase de dissémination. De ce fait, les agents pour le site b du premier consensus, seront de plus en plus nombreux à avoir le site a de disponible pour leur prochaine opinion à prendre lors de la dissémination. Ainsi, la probabilité d'obtenir plus d'agents pour le site a augmentera, et ainsi, le consensus changera.

Résultats

Posons les qualités des ressources a et b, $q_a = 1$, $q_b = 2$ jusqu'à 140 minutes (temps afin d'obtenir le premier consensus). Ensuite les qualités s'inversent avec $q_a = 2$ et $q_b = 1$. On mesure le nombre d'agents pour la ressource b.

FIGURE 2.5 – Courbes des seuils du Voter model



On constate que le consensus peut bien changer au cours du temps. À environ 10 minutes après le premier consensus pour le regroupement du site b, celui-ci s'inverse pour le regroupement du site a qui possède désormais la meilleure qualité.

2.3 Cas du pro-rata

2.3.1 Description

Pour l'instant seul le cas classique d'un regroupement général des agents autour de la meilleure ressource a été étudié. Il serait intéressant de s'occuper du cas où l'essaim doit distribuer ses forces au pro-rata de la valeur de chaque ressource, c'est-à-dire la distribution spatiale entre les deux ressources devra être à l'image de la valeur de chacune. Par exemple avec 100 agents, si deux sites a et b ont leur qualité respective 1 et 2, alors cela signifie qu'il y aurait théoriquement 33 agents pour le site a, et 67 agents pour le site b. Soit $theorique_s$ le nombre théorique d'agents qui devrait y être pour atteindre le meilleur consensus au pro-rata du site s, q_s la qualité de la ressource s, qs_{total} la somme de toutes des valeurs de qualité de chacune des ressources disponibles, et $nombre\_agents$ le nombre d'agents. D'une manière générale :

$$theorique_s = \frac{q_s}{qs_{total}} \times nombre\_agents$$

Ainsi pour la suite avec 100 agents (99 agents pour faciliter l'implémentation avec 3 ressources) voici des tableaux récapitulant le snombre d'agents théoriques en fonction des qualités des ressources.

2 ressources	$theorique_a$	$theorique_b$
$q_a = 1\,;\, q_b = 2$	33	67
$q_a = 1\,;\, q_b = 3$	25	75
$q_a = 1\,;\, q_b = 4$	20	80

3 ressources	$theorique_a$	$theorique_b$	$theorique_c$
$q_a = 1\,;\, q_b = 2\,;\, q_c = 3$	17	33	50
$q_a = 1\,;\, q_b = 3\,;\, q_c = 6$	10	30	60

4 ressources	$theorique_a$	$theorique_b$	$theorique_c$	$theorique_d$
$q_a = 1\,;\, q_b = 2\,;\, q_c = 3\,;\, q_d = 4$	10	20	30	40
$q_a = 1\,;\, q_b = 3\,;\, q_c = 5\,;\, q_d = 7$	6	19	31	44

Posons une manière de calculer la qualité d'un consensus au cours du temps. Ce n'est pas à proprement dit une qualité d'un consensus puisque nous n'avons pas encore atteint un consensus, mais plutôt si tous les agents deviennent à ce moment t des agents explorateurs et qui prennent leur opinion courante à ce moment t, quelle serait alors la qualité de ce consensus ?
Chaque site s possède inviduellement une qualité de consensus qui se calcule ainsi : c'est-à-dire que si le nombre d'agents théoriques pour tel site est égal au nombre d'agents expérimental, alors la qualité du consensus de ce site est maximale, égale à 1. Soit $qualite\_consensus_s$ la qualité du consensus du site s, $theorique_s$ le nombre théorique d'agents qui devrait y être pour atteindre le meilleur consensus au pro-rata du site s, $experimental_s$ le nombre réel ou expérimental d'agents ayant l'opinion s.
Si $experimental_s/2 \geq theorique_s$, alors $qualite\_consensus_s = 0$. Sinon :

$$qualite\_consensus_s = \frac{theorique_s - |theorique_s - experimental_s|}{theorique_s}$$

2.3.2 Premier algorithme

Afin de parvenir à un consensus au pro-rata, il faut une certaine flexibilité parmi les opinions des agents. L'idée serait d'utiliser le fait que les agents ne peuvent plus changer d'opinion afin de converger vers un consensus au pro-rata. Dans le cas précdent du regroupement autour de la plus grande ressource, il était inutile de se préoccuper de ça car cela se faisait automatiquement. Dans le cas du pro-rata, il faut indiquer à chaque agent quand s'arrêter de changer d'opinion. Pour cela, il faut établir un seuil $smax$ de changements d'opinions à affecter à chaque agent. Pour chaque agent, dès qu'il a changé son opinion $smax$ fois, il prendra aléatoirement une de ses opinions qu'il a eues parmi les $smax$ opinions et il deviendra un agent têtu (ou explorateur) : il ne pourra plus changer d'opinion mais pourra toujours transmettre le sien. On définit par ailleurs pour la suite, qu'une itération pour un agent est un aller-retour nid-ressource, en d'autres termes un changement d'opinion.

Supposons $smax = 5$, et les changements d'opinions suivantes :

itération	1	2	3	4	5
Opinion courante	a	b	a	b	b

Dès qu'il aura fait 5 itérations alors il prendra une opinion parmi ce qu'il avait décidé auparavant, c'est-à-dire qu'il aura 40% de chance de prendre l'opinion a et 60% de chance de prendre l'opinion b jusqu'à que le consensus soit atteint.

Dès que tous les agents deviennent des agents explorateurs, on considère que le consensus est atteint.

Variables	Signification
n	nombre initial d'agents explorateurs (têtus)
$nbreAgents$	nombre total d'agents
$smax$	seuil maximal de changements d'opinions
$nbreRessources$	nombre de ressources
$avisCourant[nbreRessources]$	liste des nombres d'opinions choisies
$indiceC$	indice de la ressource courante choisie

Algorithm 1: Premier algorithme

```
while n ≠ nbreAgents do
    forall agent disséminateur do
        while changementsOpinions ≠ smax do
            Dissémination
            Voter model
            indiceC = indice de la ressource choisie par le Voter model
            changementsOpinions = changementsOpinions + 1
            avisCourant[indiceC] = avisCourant[indiceC] + 1
            Exploration
        L'agent devient un agent explorateur en faveur de la ressource prise aléatoirement
         parmi ses anciennes opinions.
        n = n + 1
    return consensus
```

2.3.3 Résultats du premier algorithme

On va mesurer l'évolution du nombre d'agents, et donc la qualité du consensus, au cours du temps. Les expérimentations se sont donc faites sur 50 simulations, avec 100 agents pour 2 ressources et 4 ressources, 99 agents pour 3 ressources avec différents rapports de valeurs de qualités pour chaque site. Puis sur ces 50 simulations, la moyenne pour chaque ressource a été faite. On rappelle que q_s est la qualité de la ressource s.

Pour deux ressources avec $smax = 8$, on a :

FIGURE 2.6 – $q_a = 1$, $q_b = 2$

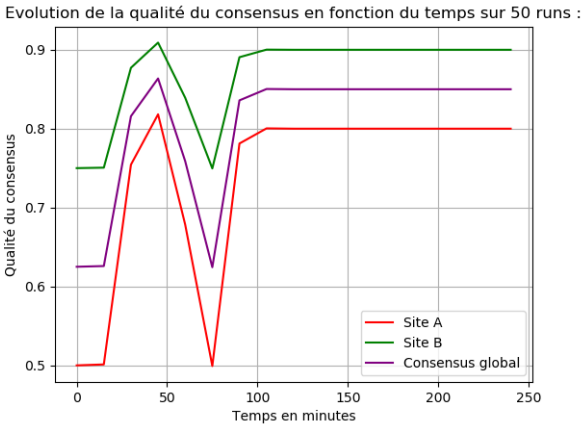


FIGURE 2.7 – $q_a = 1$, $q_b = 3$

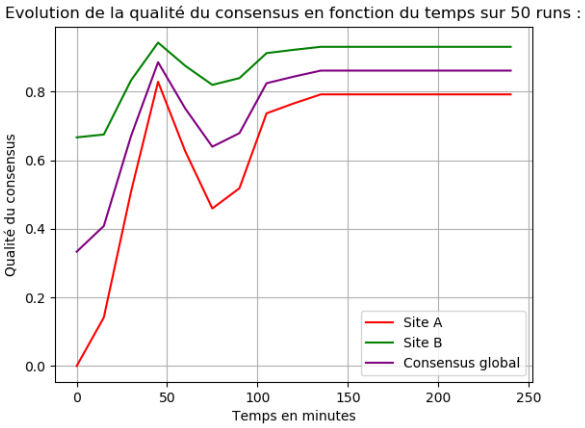
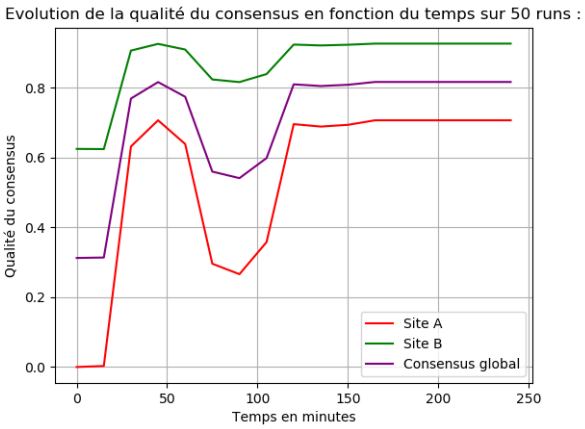


FIGURE 2.8 – $q_a = 1$, $q_b = 4$



Lorsqu'il y a deux ressources, on constate que plus l'écart des qualités entre les deux ressources s'accentue, plus la qualité diminue.

Pour trois ressources avec $smax = 8$, on a :

FIGURE 2.9 – $q_a = 1$, $q_b = 2$, $q_c = 3$

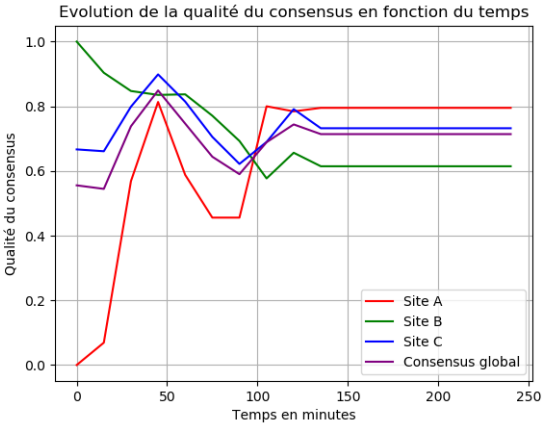
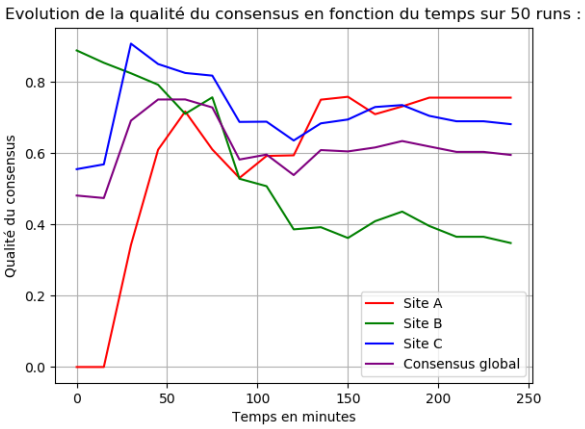


FIGURE 2.10 – $q_a = 1$, $q_b = 3$, $q_c = 6$



On constate le même phénomène avec 3 ressources, la qualité du consensus global diminue avec l'augmentation du rapport entre les différentes qualités.

Pour quatre ressources avec $smax = 12$, on a :

FIGURE 2.11 – $q_a = 1$, $q_b = 2$, $q_c = 3, q_d = 4$

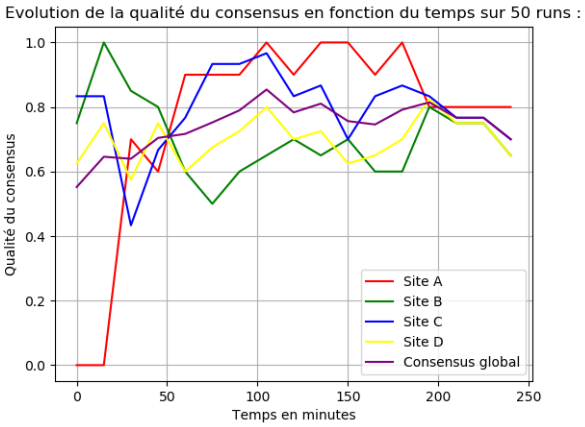
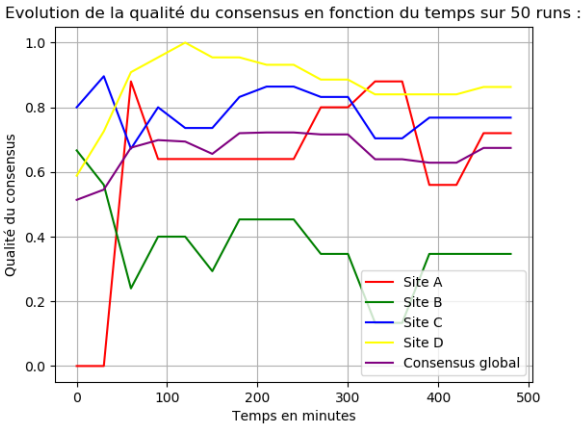


FIGURE 2.12 – $q_a = 1$, $q_b = 3$, $q_c = 5, q_c = 7$



La qualité globale reste très médiocre, de l'ordre de 0,7, lorsqu'on possède autant de ressources.

Essayons d'améliorer en modifiant l'algorithme. D'une manière générale, on constate qu'à certains moments des runs, en analysant l'évolution de la qualité du consensus au cours du temps dans les graphes précédents, le nombre d'agents ayant l'opinion optimale est meilleur. Par exemple pour deux ressources, à 40 minutes sur leurs graphes, le nombre d'agents possédant l'opinion optimale permettant le pro-rata a une qualité meilleure que celle du consensus atteint final. Il existerait donc une itération où si les agents garderaient leur avis courant, alors le consensus serait meilleur.

Il serait aussi intéressant de modifier le seuil de base $smax$, le modifier de sorte qu'il devienne un seuil dynamique qui pourrait se modifier au cours de la simulation.

Autre problème aperçu lors des simulations individuelles, il était possible d'avoir un énorme déséquilibre entre les ressources de meilleure et moindre qualité. Par exemple pour le cas de $q_a = 1$ et $q_b = 2$, il pouvait avoir seulement 10 au lieu de 34 agents pour le site a. Cela est dû à l'aspect aléatoire du *Voter model*. Il faut donc ajouter plus d'influence pour les sites possédant une qualité faible.
Pour cela, on peut mettre n agents têtus dès le début. Il y aurait alors n divisé par le nombre de ressources, d'agents têtus pour chaque ressource dès le commencement.

2.3.4 Second algorithme

Il s'agit d'essayer d'améliorer le processus précédent en ajoutant deux nouvelles fonctions à l'algorithme : rajouter un poids à l'itération où le nombre d'agents expérimental est le plus proche du nombre d'agents théorique, et modifier le seuil au cours du temps pour essayer de mieux l'adapter.

Dans ce cas l'objectif serait de trouver les meilleures itérations pour ça.

Ajouter un poids à une itération signifie qu'à cette itération critique, l'opinion compte une fois (comme dans l'algorithme précédent) mais en plus le poids.

Par exemple si on ajoute un poids de 1 à l'itération 2, on aurait :

itération	1	2	3	4	5
Opinion courante	a	b	b	b	b

On ajouterait la même opinion dans notre liste en plus, d'où le fait qu'on compte l'opinion b pour l'itération 3 (c'est l'effet du poids).

Modifier le seuil au cours du temps consiste à faire devenir un agent disséminateur un agent explorateur plus tôt que prévu en fonction d'une opinion à une itération précise, qui serait critique aussi. Cela permet d'ajouter une influence en faveur de la ressource choisie à cette itération. En effet, cela signifierait que parmi les $smax$ ressources (ou opinions) que l'agent a visitées, il y aurait au moins une opinion en faveur de cette ressource.
Soit $smin$ un autre seuil, et q_s la valeur de la qualité de la ressource s choisie à l'itération critique :

$$smax = \frac{smin}{q_s}$$

Choisissons arbitrairement $smin = smax$. Cette formule donnerait alors cet aspect : plus la qualité de la ressource choisie à cette itération critique, est faible, plus l'agent deviendra un agent explorateur rapidement. Et inversement plus la qualité de cette ressource est élevée, plus l'agent prendra des itérations avant d'arrêter de changer d'avis.
Cela permet d'ajouter davantage d'influence pour les agents pouvant choisir les ressources ayant

une faible qualité, afin de pallier au problème du déséquilibre du nombre d'agents entre la meilleure ressource et la moindre aperçu précédemment.

Variables	Signification
n	nombre initial d'agents explorateurs (têtus)
$nbreAgents$	nombre total d'agents
$smax$	seuil maximal de changements d'opinions
$smin$	seuil mimimal de changements d'opinions
$itCS$	itération critique pour le seuil dynamique
$poids$	poids à ajouter
$itCP$	itération critique pour le poids à ajouter
$nbreRessources$	nombre de ressources
$avisCourant[nbreRessources]$	liste des nombres d'opinions choisies
$indiceC$	indice de la ressource courante choisie
$qualiteC$	qualité de la ressource courante choisie

Algorithm 2: Algorithme "amélioré"

```
while n ≠ nbreAgents do
    forall agent disséminateur do
        while changementsOpinions ≠ smax do
            changementsOpinions = changementsOpinions + 1
            avisCourant[indiceC] = avisCourant[indiceC] + 1
            if changementsOpinions = itCP then
                avisCourant[indiceC] = avisCourant[indiceC] + poids
            if changementsOpinions = itCS then
                smax = smin/qualiteC
                if smax < changementsOpinions then
                    changementsOpinions = smax
        n+ = 1
```

2.3.5 Résultats du second algorithme

Tout d'abord prenons les cas avec deux ressources.

Le premier graphe correspond à ces valeurs :

Variables	Valeur
n	6
$nbreAgents$	100
$smax$	8
$smin$	0
$itCS$	0
$poids$	1
$itCP$	3
$nbreRessources$	2

Le second graphe correspond à ces valeurs :

Variables	Valeur
n	6
$nbreAgents$	100
$smax$	8
$smin$	8
$itCS$	2
$poids$	0
$itCP$	0
$nbreRessources$	2

Le troisième graphe correspond à ces valeurs :

Variables	Valeur
n	6
$nbreAgents$	100
$smax$	8
$smin$	8
$itCS$	2
$poids$	1
$itCP$	3
$nbreRessources$	2

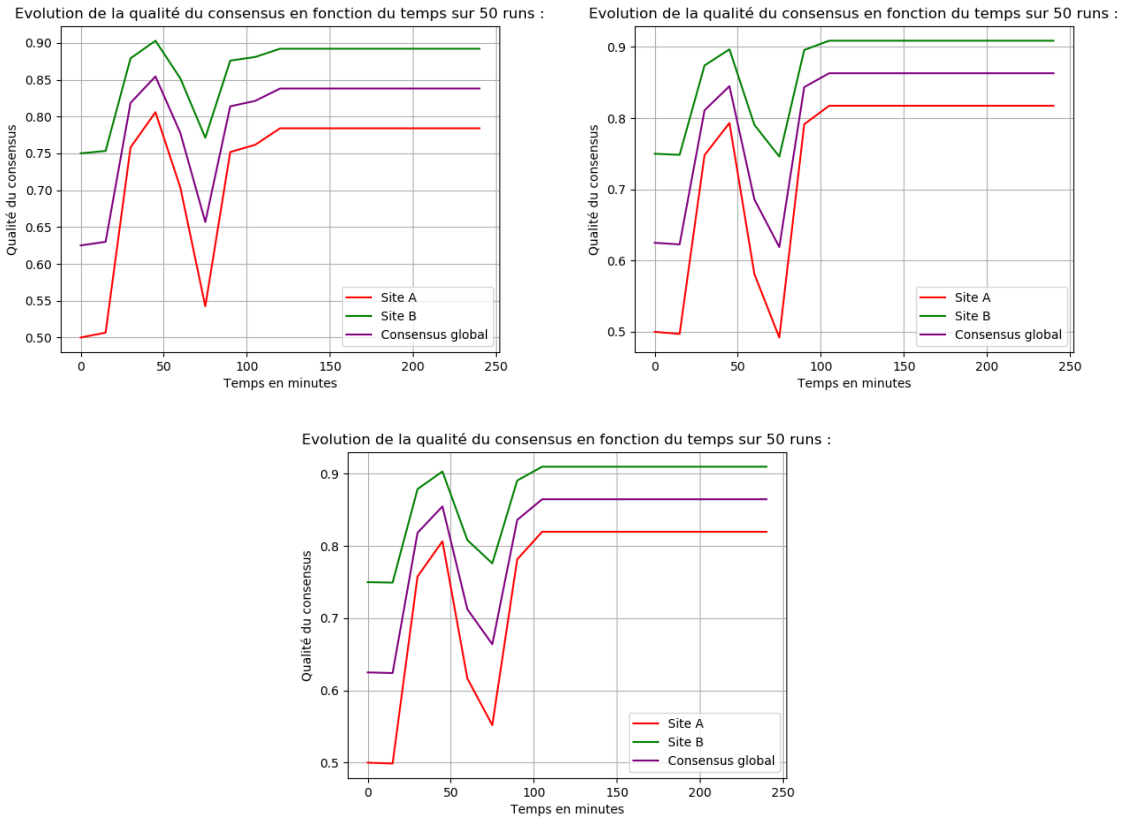


FIGURE 2.13 – $q_a = 1$, $q_b = 2$

On constate alors une légère amélioration par rapport au premier algorithme lorsqu'on active au moins le paramètre du poids à ajouter (deuxième graphe et troisième graphe).

Dans le cas avec $q_a = 1$ et $q_b = 3$ ou $q_b = 4$, on constate qu'il est préférable d'utiliser le premier algorithme et non le second.

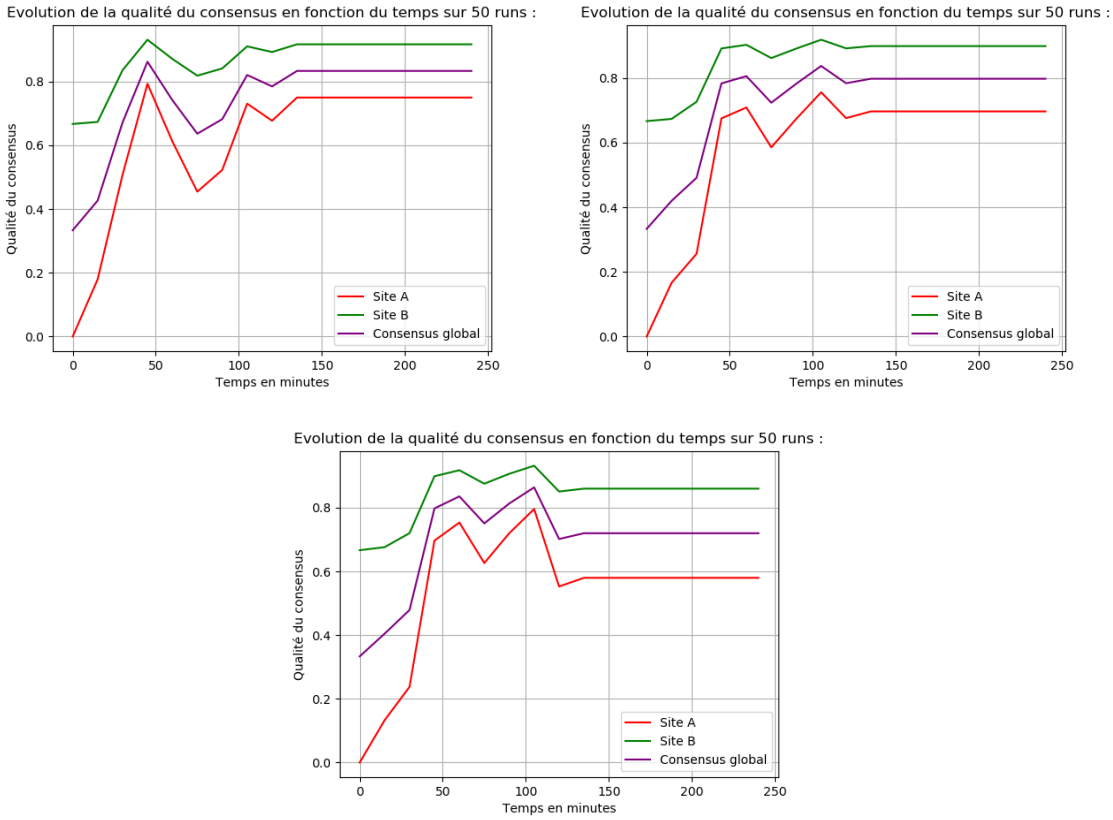


FIGURE 2.14 – $q_a = 1$, $q_b = 3$

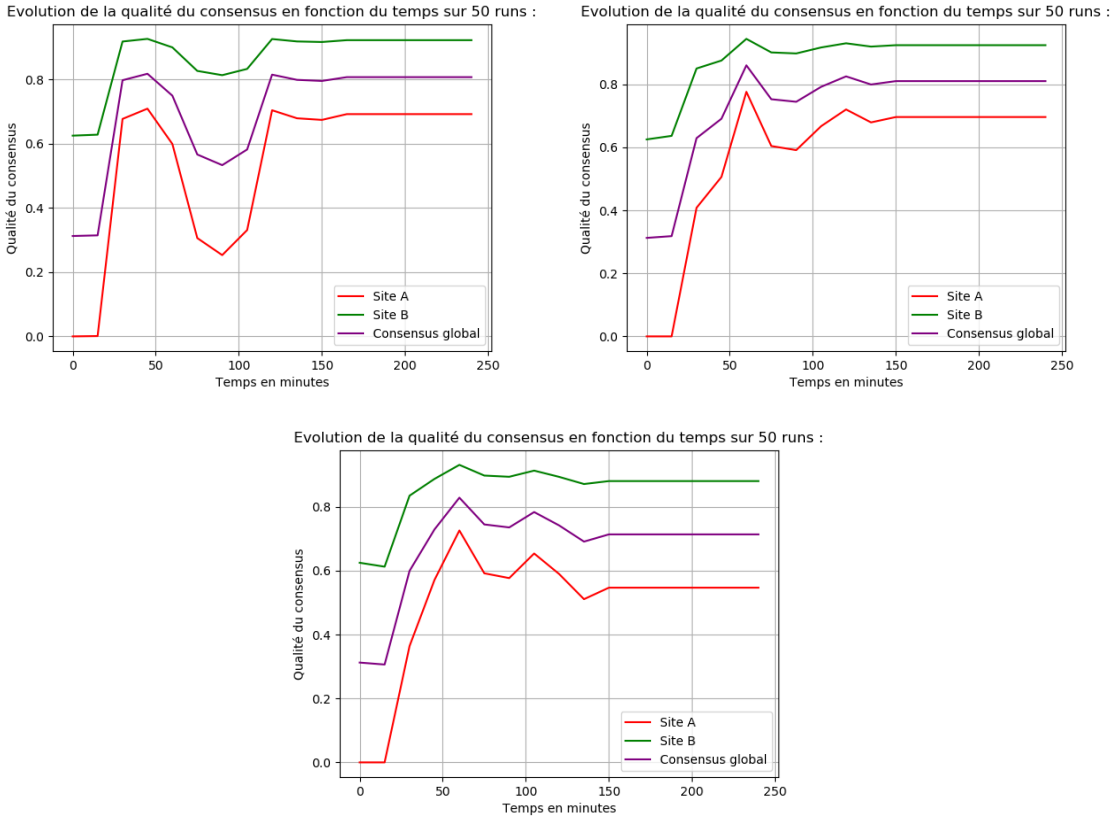


FIGURE 2.15 – $q_a = 1$, $q_b = 4$

Prenons les cas avec trois ressources.

Le premier graphe correspond à ces valeurs :

Variables	Valeur
n	12
$nbreAgents$	99
$smax$	8
$smin$	0
$itCS$	0
$poids$	1
$itCP$	3
$nbreRessources$	3

Le second graphe correspond à ces valeurs :

Variables	Valeur
n	12
$nbreAgents$	99
$smax$	8
$smin$	8
$itCS$	2
$poids$	0
$itCP$	0
$nbreRessources$	3

Le troisième graphe correspond à ces valeurs :

Variables	Valeur
n	12
$nbreAgents$	99
$smax$	8
$smin$	8
$itCS$	2
$poids$	1
$itCP$	3
$nbreRessources$	3

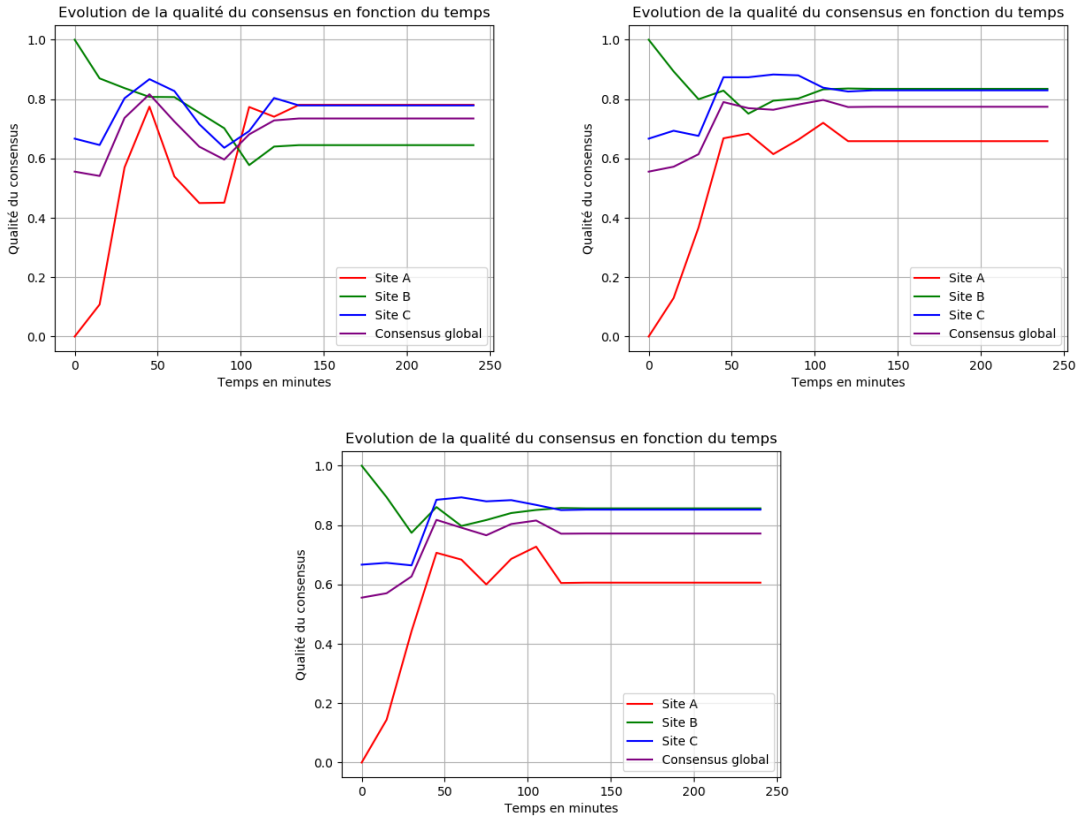


FIGURE 2.16 – $q_a = 1$, $q_b = 2$, $q_c = 3$

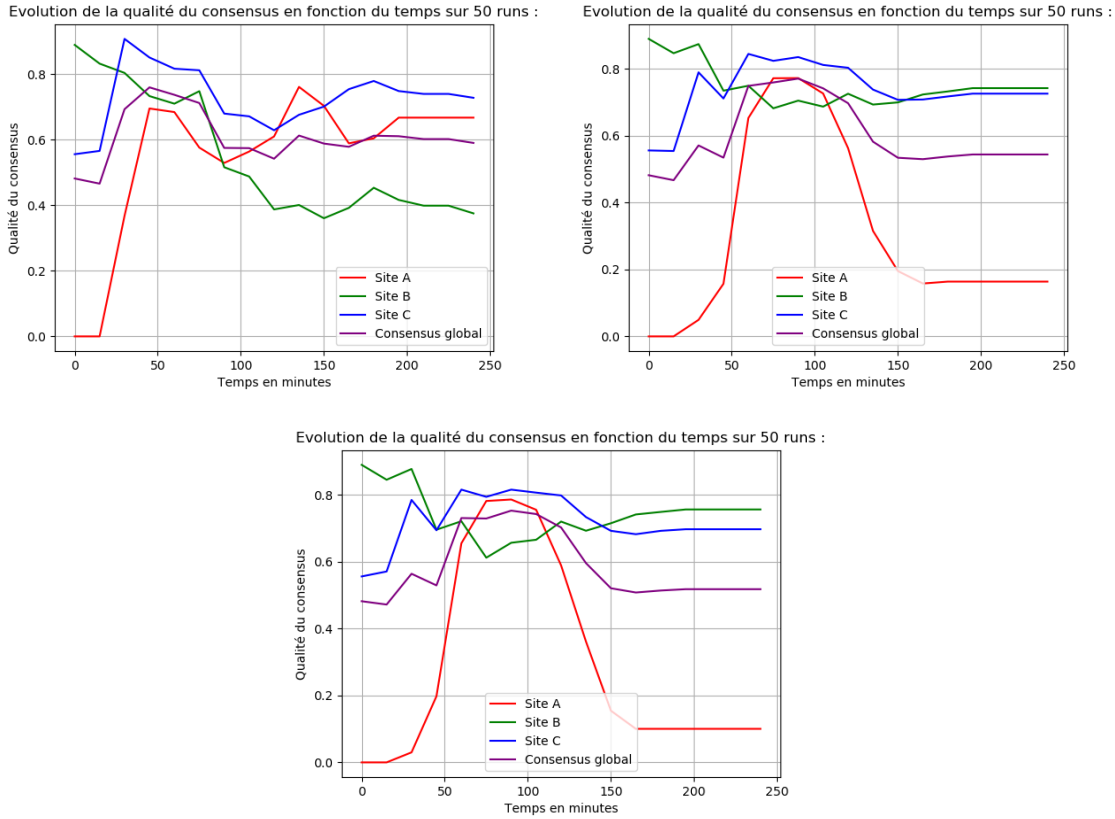


FIGURE 2.17 – $q_a = 1$, $q_b = 3$, $q_c = 6$

On constate que dans un premier temps, avec un rapport de qualité faible entre les qualités des ressources, le second algorithme est meilleur que le premier algorithme. En revanche, lorsque l'écart entre ces différentes valeurs est prononcé, le second algorithme est moins efficace que l'algorithme de base.

Prenons le cas avec 4 ressources.

Le premier graphe correspond à ces valeurs :

Variables	Valeur
n	16
$nbreAgents$	100
$smax$	12
$smin$	0
$itCS$	0
$poids$	2
$itCP$	1
$nbreRessources$	4

Le second graphe correspond à ces valeurs :

Variables	Valeur
n	16
$nbreAgents$	100
$smax$	12
$smin$	12
$itCS$	2
$poids$	0
$itCP$	0
$nbreRessources$	4

Le troisième graphe correspond à ces valeurs :

Variables	Valeur
n	16
$nbreAgents$	100
$smax$	12
$smin$	12
$itCS$	2
$poids$	2
$itCP$	3
$nbreRessources$	4

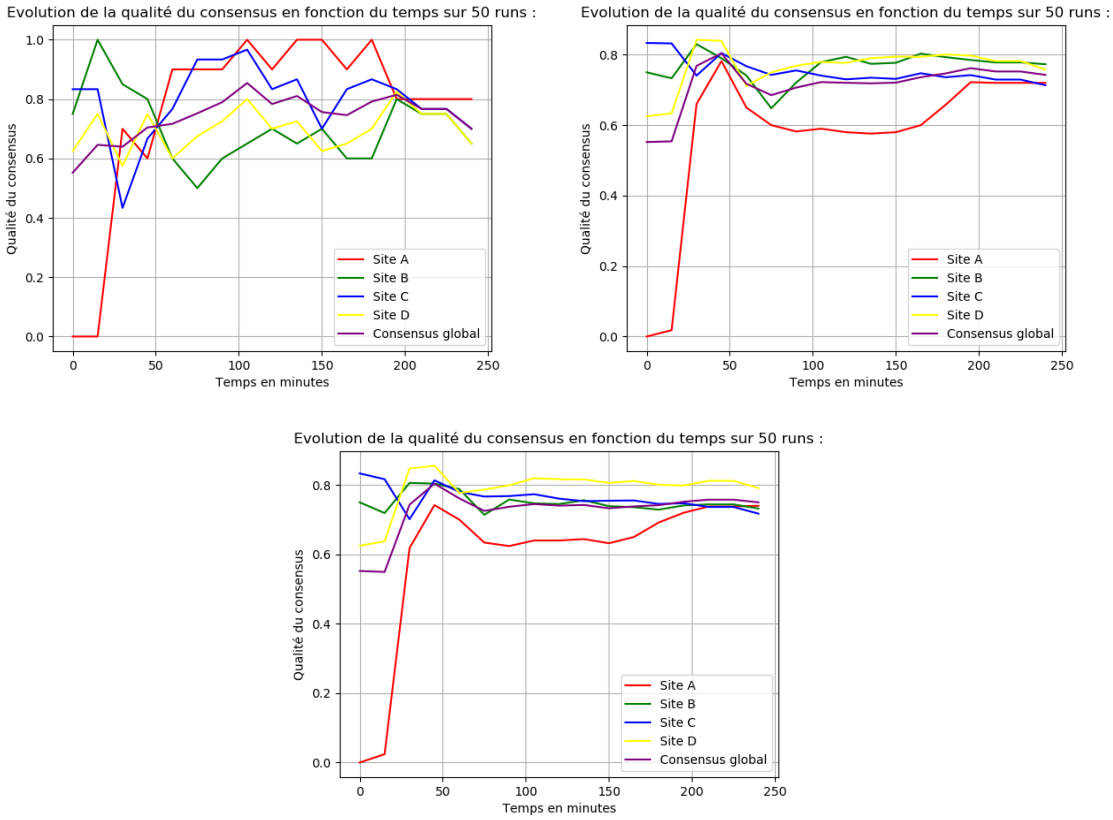


FIGURE 2.18 – $q_a = 1$, $q_b = 2$, $q_c = 3$, $q_d = 4$

Le consensus global est bien meilleur avec le second algorithme.

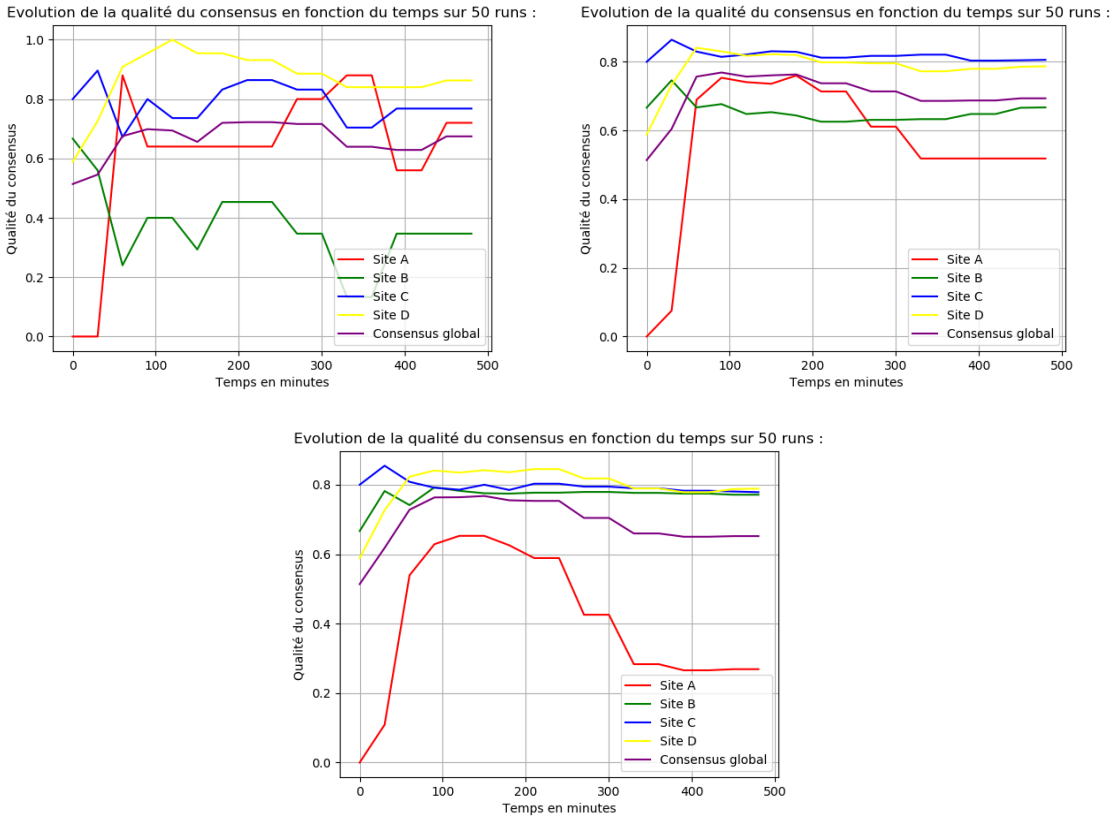


FIGURE 2.19 – $q_a = 1$, $q_b = 3$, $q_c = 5$, $q_d = 7$

La meilleure méthode dans ce cas serait d'utiliser seulement le poids à ajouter, et de ne pas utiliser la modification du seuil. Malgré cela, il reste assez équivalent au premier algorithme et ne l'améliore pas.

Chapitre 3

L'algorithme d'apprentissage

3.1 L'algorithme *mEDEA*

Par manque de temps, je n'ai malheureusement pas pu implémenter de façon correcte cette partie. Néanmoins, l'algorithme d'apprentissage consisterait à utiliser une variante de l'algorithme *mEDEA* (minimal Environment-driven Distributed Evolutionary Adaptation) décrit par Nicolas Bredeche et al. [3].
Tout d'abord chaque agent possèderait un génome représenté par une liste de préférences des ressources, initialisée aléatoirement. Une préférence représente une probabilité de prendre une direction donnée vers une ressource. Lors de la dissémination, les agents mesurent dans un premier temps la quantité d'énergie ramenée au centre de l'arène, qui dépendrait proportionnellement par rapport à la sous ou sur-consommation des ressources. Pour cela lors de la dissémination, chaque agent indique pendant un certain temps, aux autres agents quelle ressource il a précédemment exploré tout en récupérant l'information de ses voisins. Puis après cette récolte d'informations, chaque agent calculerait la qualité globale des ressources récoltées, au pro-rata ou non suivant notre objectif. En cas de répartition parfaite, le score est maximal et l'agent gagnerait de l'énergie. Dans le cas contraire, il en perderait.

FIGURE 3.1 – Algorithme mEDEA [3]

Algorithm 1 The mEDEA algorithm

genome.randomInitialize()
while forever **do**
  if *genome*.notEmpty() **then**
    agent.load(*genome*)
  end if
  for $iteration = 0$ to $lifetime$ **do**
    if agent.energy > 0 and *genome*.notEmpty() **then**
      agent.move()
      broadcast(*genome*)
    end if
  end for
  genome.empty()
  if *genomeList*.size > 0 **then**
    $genome = \text{applyVariation}(select_{random}(genomeList))$
  end if
  genomeList.empty()
end while

Chapitre 4

Conclusion et remarques

Durant tout le projet, j'ai été amené à lire et trier plusieurs articles scientifiques sur le domaine de la robotique en essaim. Tous étaient très intéressants mais seulement quelques uns étaient vraiment utiles pour ce projet. Les articles les plus utiles reposaient surtout sur les travaux de Gabriele Valentini, Heiko Hamann, et Marco Dorigo. Ils reposaient sur le cas du consensus autour de la plus grande ressource avec les stratégies du *Majority rule* et du *Voter model*. De plus, les travaux de Judhi Prasetyo et al. m'ont conduit à découvrir le cas des sites dynamiques, chose que je n'y avais pas pensée au début du projet, et surtout la notion des agents têtus qui m'ont permis d'avoir des idées, malgré les résultats discutables, sur la partie où l'essaim doit distribuer ses forces au pro-rata. Cette seconde partie m'a été la plus compliquée à prendre en main. En effet, je n'ai trouvé aucun article discutant de ce cas où le consensus ne serait pas total sur une seule ressource. Ainsi cela m'a contraint à essayer d'élaborer soi-même un algorithme pour ce problème.

Pour cela j'ai essayé de m'inspirer des précédentes stratégies de choix et des travaux de Judhi Prasetyo et al. sur les agents têtus. Ce qui m'a amené à élaborer le premier algorithme pour le pro-rata. J'ai alors constaté que les résultats pouvaient être très aléatoires, mais donnaient néanmoins un consensus dans tous les cas, malgré la distribution spatiale non parfaite. Cela m'a poussé à essayer d'améliorer cet algorithme. Pour cela, ayant manqué d'idées, il a fallu analyser les résultats du premier algorithme de manière plus fine pour constater des faits très intéressants. À partir de ces résultats, j'ai essayé de mettre en place une notion de poids à une certaine itération d'une simulation, et d'utiliser un seuil dynamique pour chaque agent. Malgré cela, les résultats restaient médiocres à cause de plusieurs raisons. La première serait tout simplement le fait que l'algorithme ne fonctionne pas du tout. La seconde raison serait les paramètres erronés. La troisième serait le simulateur. Ce dernier ne pouvait malheureusement pas reproduire exactement les conditions réelles de l'arène réelle. Ce qui m'a amené à tricher plusieurs fois pour essayer de simuler les bons comportements. Il n'est alors pas possible de porter mon code sur les réels kilobots. C'était paradoxalement l'un des principaux avantages de ce simulateur : la portabilité du code en C simulateur-kilobot. De plus, les résultats graphiques étaient compliqués à être obtenus, du fait que le simulateur ne le permet pas. J'ai donc dû faire générer ds fichiers .txt à partir de l'exécution du simulateur, et ensuite les traiter avec un script en Python. De plus, le simulateur était limité par sa taille en mémoire, ce qui avait pour conséquence de n'enregistrer l'état courant du run toutes les 15 voire 30 minutes en fonction de la valeur des qualités et du nombre de ressources.
Néanmoins il possède de nombreux avantages comme le fait de simuler très rapidement, chose impossible dans la réalité où les kilobots mettraient des heures à obtenir un simple consensus. Cela pourrait être très intéressant de coder ces stratégies sur les vrais kilobots, peut-être pourrions-nous constater d'autres résultats intéressants.

Je n'ai malheureusement pas eu le temps d'implémenter de manière correcte l'algorithme *mEDEA*, j'ai une ébauche de code sur cette partie mais mes résultats ne permettent pas du tout ni l'obtention d'un consensus ni une bonne distribution spatiale.
J'ai eu aussi l'idée d'utiliser mes connaissances acquises de l'UE "Fondements des Systèmes Multia-

gents" avec les notions de *busy buddy* ou encore celle du *four people* mais étant été concentré sur ma tentative de résoudre le problème du pro-rata en utilisant mes algorithmes, je n'ai malheureusement pas eu le temps d'y réfléchir davantage, il aurait été très intéressant de mettre ça en place sur les kilobots.

Pour conclure, la première partie reposant sur des algortihmes déjà élaborés par des chercheurs m'a permis d'avoir une approche professionnelle car je devais principalement m'occuper de l'implémentation. En revanche la seconde partie sur le pro-rata m'a offert une toute nouvelle approche pour moi, celle de la recherche où en plus de l'implémentation, j'ai été amené à établir un nouveau raisonnement pour essayer d'élaborer des nouveaux algorithmes, malgré des résultats non concluants.

Annexe A

Cahier des charges

Recherche de consensus en robotique en essaim

Cahier des charges

LY Jean-Baptiste
M1 ANDROIDE

encadré par
BREDECHE Nicolas et MAUDET Nicolas

Février 2020

Table des matières

Chapitre 1

Introduction

1.1 Présentation personnelle

Je m'appelle LY Jean-Baptiste, j'ai 24 ans. Actuellement je poursuis la première année de Master d'informatique ANDROIDE (AgeNts Distribues, Robotique, Recherche Opérationnelle, Interaction, DEcision).

Ce projet s'inscrit dans le cadre de l'UE P-ANDROIDE, qui consiste à s'intéresser à un sujet de projet qui dure tout le long du second semestre.

1.2 Présentation du projet

On s'intéresse dans ce projet au problème du *best-of-n* en robotique essaim. Le problème du *best-of-n*, consiste à une prise de décision collective au sein d'un ensemble de robots aux capacités de communication et de calcul limitées. Parmi n options disponibles, l'essaim doit choisir l'option qui offre la meilleure solution possible afin de satisfaire leurs besoins actuels. La problématique de ce projet est de déterminer comment des robots possédant des capacités limitées individuellement, peuvent nous fournir une bonne solution, voire la meilleure.

L'objectif de ce projet est d'étudier l'émergence de consensus. Le projet sera mené sur robots réels.

Chapitre 2

Organisation

Dans un premier temps, il s'agit d'implémenter un algorithme existant permettant d'atteindre de manière distribuée un consensus entre deux ressources. Ces ressources sont représentés par des objets émettant un signal infra-rouge donnant leur qualité, et l'objectif de l'essaim est de choisir la ressource de plus grande qualité. Ces deux ressources sont placées aux deux extrémités d'une arène rectangulaire, et une source de lumière permet en fuyant ou poursuivant la lumière de facilement se diriger vers l'une ou l'autre des deux ressources.

Dans un second temps, on implémentera un algorithme d'apprentissage en ligne et distribué afin d'évaluer les difficultés que peuvent poser l'apprentissage de consensus (par opposition à l'utilisation d'un algorithme dédié).

L'espace de recherche sera défini à partir des comportements de phototaxis, anti-phototaxis et déambulation libre – il s'agira d'apprendre les conditions de transitions entre chaque comportement. On s'intéressera d'abord au cas ou l'essaim doit se regrouper autour de la ressource de plus grande valeur, puis le cas ou l'essaim doit distribuer ses forces au pro-rata de la valeur de chaque ressource (i.e. la distribution spatiale entre les deux ressources devra être à l'image de la valeur de chacune).

Selon le temps, l'amélioration du système de tracking visuel de robots pourra être repris et amélioré afin de permettre un suivi temps réel de l'essaim.

Chapitre 3

Outils

Afin d'avoir les conditions nécessaires, le projet s'inspirera de l'expérience décrite dans l'article "Valentini et al. (2016) Collective decision with 100 Kilobots : Speed versus accuracy in binary discrimination problems. AAMAS.". Ainsi les expériences se feront sur une vingtaine de Kilobots disponibles à l'ISIR, qui sont des robots utilisés dans la robotique en essaim.

Ils sont peu coûteux (environ 120 l'unité) et de petite taille (3,3 cm de diamètre), équipés de deux moteurs indépendants, et de capteurs et d'émetteurs d'infra-rouge, ainsi qu'un capteur de lumière ambiante.

L'arène est rectangulaire et est constitué d'une plaque de plexiglas de taille 100 x 190 cm, consituée de trois zones : les sites a et b qui ont pour dimensions chacun 80 x 45 cm émettent des infra-rouges en-dessous du plexisglass. Ces derniers donneront la qualité du site aux Kilobots. Une lumière ambiante sera placée dans le site a, afin de permettre les comportements phototaxis et antiphototaxis. Enfin le nid situé entre ces deux sites, est de dimensions 100 x 100 cm, contrairement aux deux autres sites, il n'y aura pas d'infra-rouge dans cette zone, mais plutôt une surface opaque.

Pour des raisons pratiques, j'utilise aussi le simulateur Kilombo, qui possède de nombreux avantages. En effet, cela me permet de travailler chez moi sans les Kilobots, d'accélérer certaines expériences puisqu'il permet d'accélérer leurs déplacements, tout en gardant une certaine robustesse avce le bruit. De plus, le simulateur utilise aussi le langage C, comme les Kilobots. Cela me donne la possibilité de transférer les codes avec peu de modification entre les deux environnements. Néanmoins, le simulateur présente quelques limites, comme l'impossibilité de reproduire les conditions de l'expérience avec la présence du plexiglass.

Chapitre 4

Délais

Les résultats du projet sont à rendre le 9 juin 2020.

Annexe B

Quelques instructions pour le simulateur Kilombo

B.0.1 Installation du simulateur Kilombo

Les instructions sont disponibles sur ce lien : https ://github.com/JIC-CSB/kilombo Il n'est disponible que sur Linux. La ligne de commande permattant d'installer le simulateur est : "sudo apt-get install build-essential git gcc-avr gdb-avr binutils-avr avr-libc avrdude libsdl1.2-dev libjansson-dev libsubunit-dev cmake check".

B.0.2 Exécution du simulateur

Aller dans un répertoire d'exécution, et lancer avec la ligne de commande "make && dissemination_exploration" pour un simple run. Pour afficher ou non l'interface, modifier le booleen dans le fichier kilombo.json : mettre 1 à GUI pour afficher l'interface graphique. Le fichier script.py permet de tracer les courbes, il est à utiliser lorsque l'interface n'est pas affiché.

Bibliographie

[1] Gabriele Valentini, Eliseo Ferrante, and Marco Dorigo. *The Best-of-n Problem in Robot Swarms : Formalization, State of the Art, and Novel Perspectives*. Front. Robot. AI 4, (2017). DOI :https ://doi.org/10.3389/frobt.2017.00009

[2] Gabriele Valentini, Eliseo Ferrante, Heiko Hamann, and Marco Dorigo. *Collective decision with 100 Kilobots : speed versus accuracy in binary discrimination problems*. Autonomous Agents and Multi-Agent Systems 30, 3 (2016), 553–580. DOI :https ://doi.org/10.1007/s10458-015-9323-3

[3] Nicolas Bredeche, Jean-Marc Montanier, Wenguo Liu, and Alan F.T. Winfield. *Environment-driven distributed evolutionary adaptation in a population of autonomous robotic agents*. Mathematical and Computer Modelling of Dynamical Systems 18, 1 (February 2012), 101–129. DOI :https ://doi.org/10.1080/13873954.2011.601425

[4] Fredrik Jansson, Matthew Hartley, Martin Hinsch, Ivica Slavkov, Noemí Carranza, Tjelvar S. G. Olsson, Roland M. Dries, Johanna H. Grönqvist, Athanasius F. M. Marée, James Sharpe, Jaap A. Kaandorp, and Verônica A. Grieneisen. *Kilombo : a Kilobot simulator to enable effective research in swarm robotics.*. arXiv :1511.04285 [cs] (May 2016). http ://arxiv.org/abs/1511.04285

[5] Gabriele Valentini, Heiko Hamann, and Marco Dorigo. *Self-organized Collective Decision Making : The Weighted Voter Model.*. 2014.

[6] Judhi Prasetyo, Giulia De Masi, Pallavi Ranjan, and Eliseo Ferrante. *The Best-of-n Problem with Dynamic Site Qualities : Achieving Adaptability with Stubborn Individuals*. Swarm Intelligence 5, 3 (2011), 305–327. 11th International Conference, ANTS 2018, Rome, Italy, October 29–31, 2018, Proceedings. . 239–251.