SORBONNE UNIVERSITÉ

22 mai 2019

Carnet de bord projet P-ANDROIDE

Estimation de la contribution marginale dans un essaim de robots

Encadrants : Nicolas Bredèche
Nicolas Maudet

Clément Chrétien Kevin Jabbour Émile Savalle

Table des matières

1 Problématique

Dans le cadre de notre projet, nous nous intéressons à l'estimation de la contribution de chaque robot dans un essaim devant réaliser une tâche fixée au préalable par l'utilisateur. La performance de l'essaim de robots est considérée comme connue, cependant la contribution de chaque robot à cette performance globale est inconnue, et peut varier grandement en fonction du rôle pris par le robot. Par exemple, un robot qui ne fait rien bénéficiera d'une bonne note sans contribuer, au même titre qu'un robot qui réalise une action essentielle. Ou bien encore, s'il y a des parasites dans l'environnement, avoir des "défenseurs" qui bloquent ces derniers et permettent à des "ramasseurs" d'être plus efficaces est meilleur qu'une équipe uniquement composée de ramasseurs mais qui se ferait entraver (et ce, bien que les défenseurs ne ramassent rien).

Réussir à effectuer une bonne estimation de la contribution marginale peut avoir des utilités diverses :

- si les agents sont tous polyvalents, c'est à dire s'ils peuvent tous assumer les différents comportements, alors on peut trouver, pour tout groupe d'agents et pour toute situation, une répartition optimale des rôles dans une situation critique[1].
- dans des jeux coopératifs et adversairiels, tels que des matches de foot, cela pourrait servir à identifier les meilleurs joueurs, ceux qui ont les meilleures synergies etc[2]

De façon générale, l'estimation de la contribution marginale sert dans les cas de formation de groupes "ad hoc", c'est à dire des équipes cherchant à collaborer sans pré-coordination[3].

Nous souhaitons donc mettre en place une méthode pour identifier la contribution de chaque robot. On dispose alors d'un ensemble de robots, et d'un simulateur qui permet d'estimer la performance globale d'un groupe constitué de l'ensemble des robots, d'une partie, voire d'un seul individu.

La valeur de Shapley[4] est en théorie la meilleure façon de calculer la contribution d'un agent. En effet, elle correspond à la moyenne de ce qu'apporte l'agent à tout groupe. Nous reviendrons plus tard sur son calcul. Toutefois, le problème avec cette valeur est le nombre de groupes à considérer, qui est exponentiel en le nombre d'agents. Ce n'est donc pas compliqué à calculer sur des petites tailles d'instance mais cela prend très rapidement un temps considérable lorsque le nombre d'agents augmente.

2 Expérience

2.1 Article

Pour répondre à ce problème, nous nous sommes basés sur une expérience dont le but était de reproduire le comportement des fourmis coupe-feuilles[5]. Celles-ci, comme leur nom l'indique, coupent des feuilles servant de substrat au champignon avec lequel elles vivent en symbiose et dont elles se nourrissent. Leur grande particularité, et la raison de l'intérêt qu'elles suscitent dans l'article précédemment évoqué, est leur répartition des rôles. En effet, au lieu de toutes monter à l'arbre pour couper des feuilles et les ramener, ces fourmis forment deux groupes : un premier monte à l'arbre, coupe les feuilles et les laisse tomber tandis que le second reste au pied de l'arbre, ramasse les feuilles qui en tombent et les ramène à leur nid. Or les fourmis ne communiquent pas entre elles, mais elles parviennent à se mettre d'accord pour se répartir plutôt équitablement en ces deux groupes. C'est ce qui a été étudié dans la nature, reproduit dans l'article sus-cité, et c'est également ce que nous avons tenté d'observer.

2.2 Simulateur

Pour modéliser l'expérience, nous avons utilisé le simulateur *Roborobo !*[6] qui nous permet de modéliser un environnement et des agents pour ensuite réaliser diverses tâches.

Ce simulateur met à notre disposition 3 éléments modifiables :

- Des agents dotés de capteurs et de systèmes de déplacements
- Un terrain sur lequel les agents effectueront diverses tâches
- Un observateur qui permet d'analyser l'évolution de l'environnement et d'y appliquer des modifications.

2.2.1 Terrain

Pour simuler le terrain dans lequel les fourmis graviraient un arbre pour y récolter des feuilles, nous avons créé un terrain séparé en 3 sections distinctes :

- Une zone de collecte d'objets (zone gris foncé)
  Cette zone représente le feuillage de l'arbre. Nous y plaçons donc des objets, ramassables par nos agents.
  Pour les fourmis, la quantité de ressources qu'un arbre offre est vue comme illimitée. C'est pour cela que lorsqu'un agent récupérera un objet, un nouvel objet apparaîtra automatiquement dans cette zone pour simuler l'abondance du feuillage.
- Une pente (zones orange et rouge)
  Cette zone simule le tronc de l'arbre que les fourmis doivent gravir.
  Il est à noter que les fourmis ralentissent pour résister à la gravité et rester accrochées au tronc, et ce dans les deux sens de circulation. (Une fourmi voulant descendre ne va pas se jeter de l'arbre).
  Nous avons donc, dans notre simulation, ralenti nos agents dans les deux sens également. Tout robot passant par cette zone aura une vitesse égale à 1% de sa vitesse maximale.
  De plus, si un objet est déposé par un agent dans cette zone, celui-ci apparaîtra instantanément en bas de la pente, simulant ainsi la chute rapide de la feuille.
- Une zone de dépôt (zones gris clair et verte)
  En gris nous avons le sol, sur lequel les feuilles tombent et en vert la fourmilière.
  Ici, tout objet ramené et déposé au nid est supprimé et compté comme collecté par le simulateur.

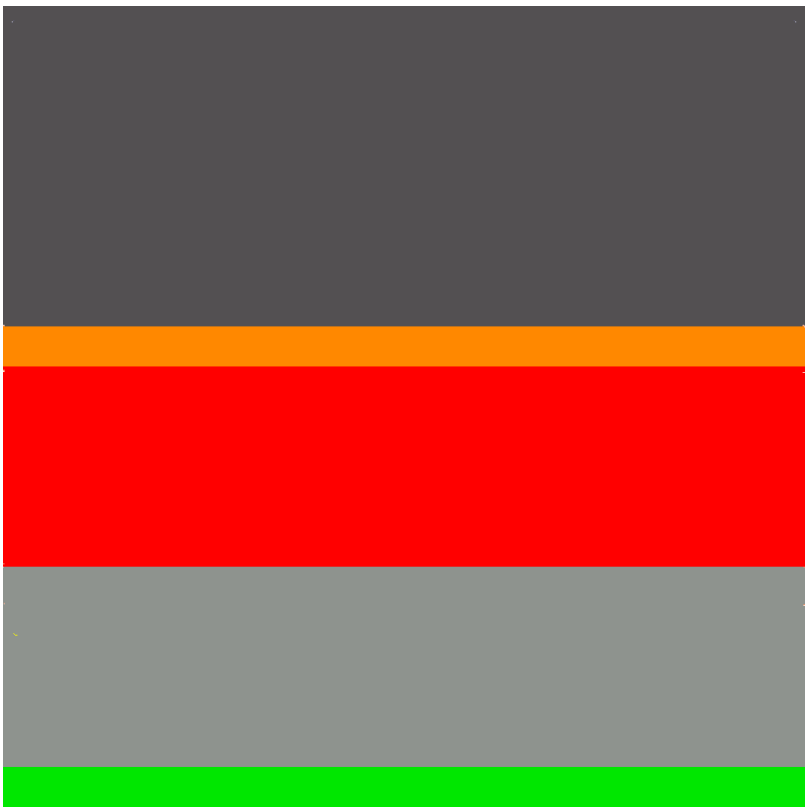


FIGURE 1 – Représentation du terrain

2.3 *Agents autonomes*

2.3.1 Les agents

FIGURE 2 – Représentation d'un agent

Sur l'image ci-dessus on voit donc un agent regardant en haut à droite, accompagné d'un affichage de ses capteurs (6 sur la partie avant, dont 3 à gauche et 3 à droite ; un à sa gauche et sur l'arrière gauche, de même pour la droite ; et le dernier directement derrière lui). La longueur des traits des capteurs représente leur portée. Ainsi les robots ne "voient" pas très loin autour d'eux.

Par ailleurs, sur les 11 capteurs présents, nous ne considérons que les 8 premiers (capteurs avant et latéraux) car nous avons programmé nos agents pour qu'ils ne fassent qu'avancer. S'ils veulent rebrousser chemin ils devront au préalable faire demi-tour. Cela peut conduire à des blocages contre les murs ou entre agents, mais de tels comportements ne seront pas conservés par l'évolution de notre population.

Notons également que tous nos robots ont la même apparence, indépendamment de leur type.

2.3.2 Algorithme génétique

Les algorithmes génétiques appartiennent à la famille des algorithmes évolutionnistes. Leur but est d'obtenir une solution approchée à un problème d'optimisation, lorsqu'il n'existe pas de méthode

exacte (ou que la solution est inconnue) pour le résoudre en un temps raisonnable. Les algorithmes génétiques utilisent la notion de sélection naturelle et l'appliquent à une population de solutions potentielles au problème donné. La solution est approchée par « bonds » successifs, comme dans une procédure de séparation et évaluation (branch & bound).

Les algorithmes génétiques sont toujours composés d'une population, elle même composée de spécimens, d'une méthode de sélection et éventuellement d'une méthode de mutation et/ou de croisement.

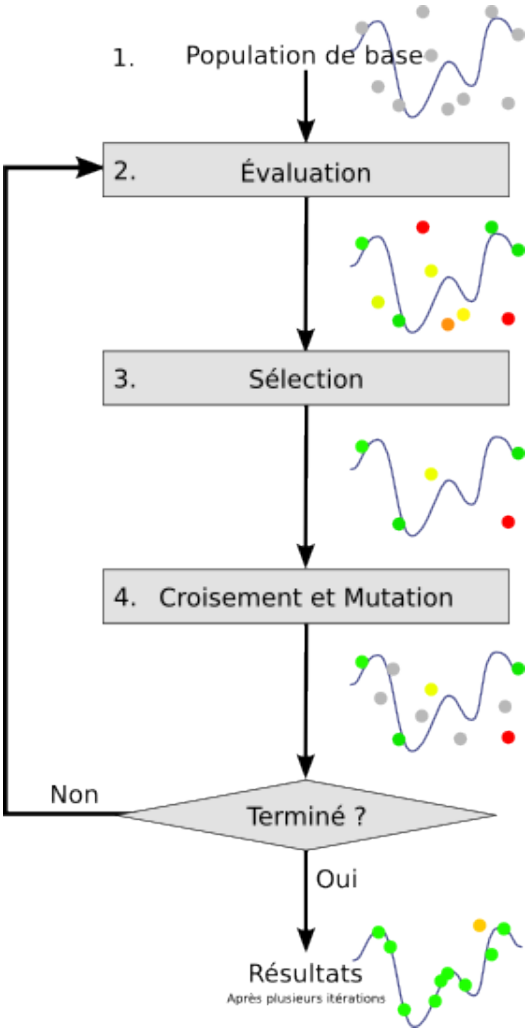


FIGURE 3 – Explication du fonctionnement des algorithmes génétiques

Voici à quoi correspondent ces différents termes :

- population : Groupe constitué de plusieurs spécimens.
- spécimen : Un spécimen est une réponse possible au problème que l'algorithme génétique essaye de résoudre.
- sélection : La sélection est une méthode permettant de choisir quels spécimens garder d'une génération à une autre. Il existe différentes méthodes de sélection, cependant elles se basent toutes sur la "fitness" des spécimens.
- fitness : Fonction permettant d'évaluer un spécimen (et donc une réponse potentielle au problème).
- mutation : Fonction permettant de modifier légèrement un spécimen.
- croisement : Fonction permettant, à partir de deux spécimens existants, de créer un nouveau spécimen ayant des caractéristiques de ses deux parents.

Dans notre cas :

- spécimen : Un robot fonctionnant avec un réseau de neurones.
- sélection : Nous utilisons une méthode de sélection appelée "sélection $\mu - \lambda$ (mu-lambda)". Le principe est de sélectionner aléatoirement les spécimens parmi les λ meilleurs d'entre eux. Ceci permet de s'assurer d'avoir de bons spécimens tout en n'étant pas trop élitiste avec le risque d'atteindre un maximum local.
- fitness : La fitness dépend du comportement que l'on souhaite obtenir. Généralement nous récompensons la prise d'objet et le dépôt de ceux-ci aux bons endroits ainsi que le déplacement des robots aux endroits voulus.
- mutation : Une mutation dans notre algorithme consiste en la modification de la valeur d'un neurone (valeur entre -1 et 1).
- croisement : Nous ne faisons aucun croisement, nous avions fait une première version avec des croisements mais ceux-ci ne nous aidaient pas à obtenir de bons résultats donc nous les avons enlevé (sous conseils de MM. Maudet et Bredèche, nos encadrants).

Comme évoqué ci-dessus notre algorithme génétique a pour but de trouver une solution, c'est à dire un comportement pour un robot. Cette solution prend la forme d'un réseau de neurones permettant de donner au robot sa prochaine vitesse de rotation (angle) et de translation (accélération).

Un réseau de neurones artificiels (ou *Neural Network* en anglais) est un système informatique s'inspirant du fonctionnement du cerveau humain pour apprendre puis donner une solution à un problème. En particulier un réseau de neurones est composé de plusieurs couches de neurones, une d'entrée, une de sortie et un certain nombre de couches cachées. Un tel réseau de neurones est appelé perceptron multicouches[7] (*Multi-Layered Perceptron* en anglais, souvent abrégé en MLP).

Tous les neurones d'une couche sont connectés à tous les neurones de la couche suivante et cela ainsi de suite jusqu'à la couche finale.

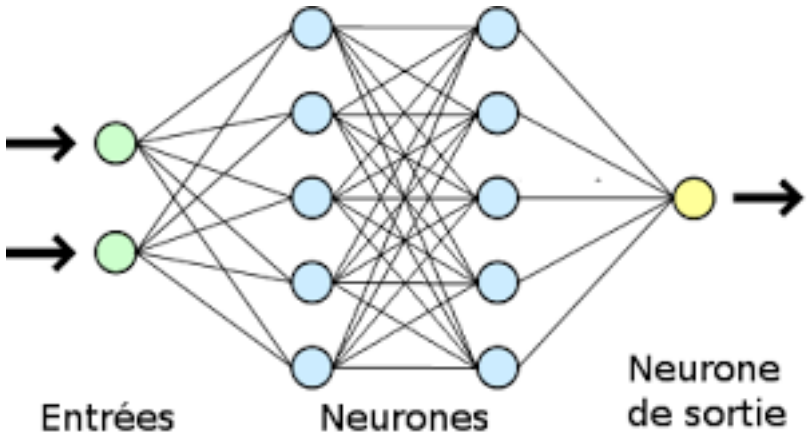


Figure 4 – Explication du fonctionnement des réseaux neuronaux

Dans notre cas nous utilisons un réseau de neurones avec 23 entrées, 2 fois 10 neurones en couches cachées et 3 neurones en couche de sortie.

Les neurones en entrée correspondent à :

- 7 neurones qui correspondent aux 8 senseurs du robot quand il détecte un mur
- 7 neurones qui correspondent aux 8 senseurs du robot quand il détecte un objet
- 6 neurones qui correspondent aux 6 zones du terrains dans lequel évolue les robots
- 1 neurone correspondant au fait que le robot porte ou non un objet
- 1 neurone sur l'orientation actuelle
- 1 neurone de biais

Les neurones en sortie correspondent à :

- une valeur de vitesse de rotation
- une valeur de vitesse de translation
- le dépôt ou non d'un objet

2.3.3 Comportements

Pour la suite du projet nous avions besoin de créer quatre comportements distincts :

- haut : Ce sont les robots se trouvant en haut de la rampe, leur rôle est de prendre les objets, de les déposer sur la rampe afin que ceux-ci tombent et de recommencer.
- bas : Ce sont les robots se trouvant en bas de la rampe, leur rôle consiste à récupérer les objets lâchés par les types haut puis de les ramener dans le nid et de recommencer.
- complet : Ce sont les robots qui vont monter la rampe, prendre un objet, redescendre la rampe et amener l'objet jusqu'au nid puis recommencer.
- random : Ce sont des robots qui vont se déplacer et agir de manière aléatoire. Il s'agit de notre population de contrôle, donc pour s'assurer que la tâche demandée est réellement efficace et nécessaire. En effet, si le type random était aussi performant que les autres, nous n'aurions pas besoin d'implémenter ces comportements plus complexes.

Tous ces comportements consistent donc en un réseau de neurones avec des poids différents pour chacun. Ces réseaux de neurones ont été obtenus avec l'algorithme génétique décrit précédemment. Seule la fonction d'évaluation a changé (la fitness), suivant le comportement voulu la fitness consistait à récompenser ou punir la prise d'objet ainsi que le dépôt de ceux-ci dans les différentes zones.

Type désiré	Caractéristiques récompensées	Caractéristiques pénalisées
Haut	Ramassage d'un objet Dépôt de celui-ci dans la zone orange Vitesse	Être en bas de la rampe Dépôt d'un objet hors de la zone orange Aller vers le haut avec un objet
Bas	Ramassage d'un objet Dépôt de celui-ci dans la zone verte Vitesse	Monter sur la rampe Dépôt d'un objet autre part Aller vers le haut avec un objet
Complet	Ramassage d'un objet Descendre la rampe avec un objet Dépôt de celui-ci dans la zone verte Vitesse	 Descendre la rampe sans objet Dépôt d'un objet autre part Aller vers le haut avec un objet

Bien évidemment, pour pouvoir obtenir des types bas fonctionnels, lors de la création de leur comportement nous avons déplacé les objets en bas de la pente (sans cela ils auraient appris à monter pour gagner des points, ce qui n'est pas ce que nous attendions d'un type bas).

Nous avons également testé d'autres paramètres comme par exemple récompenser/punir l'orientation du robot ou bien sa vitesse de rotation. Cependant ces paramètres n'ont pas été gardés car ils amenaient les robots à avoir des comportements qui n'étaient pas satisfaisants. Par exemple récompenser la vitesse de rotation donnait naissance à des comportements où les agents tournaient en rond.

3 Évaluation des agents

Comme défini dans l'introduction, évaluer l'efficacité d'un agent dans un essaim est complexe. En particulier, calculer la valeur de Shapley est un problème NP-complet[8].

Pour cela, nous avons déterminé plusieurs algorithmes permettant de calculer cette efficacité. Un de ces algorithmes était celui des Marginal Contribution Nets[9], cependant nous n'avons pas retenu ce dernier, celui-ci posant des hypothèses trop fortes et ne s'appliquant donc pas à notre expérience.

Quant aux résultats attendus, au vu de l'expérience sur laquelle nous nous sommes basés, nous souhaiterions vérifier que la façon la plus optimale pour ramasser un grand nombre de feuilles serait d'avoir un groupe de types haut et un autre de types bas.

3.1 Analyse simple des résultats de différentes coalitions

La première technique utilisée était la plus intuitive : Nous avons fait tourner nos simulations sur un très grand nombre de coalitions et les avons analysées en les classant par score (nombre d'objets amenés à la zone objectif) et en observant leur composition.

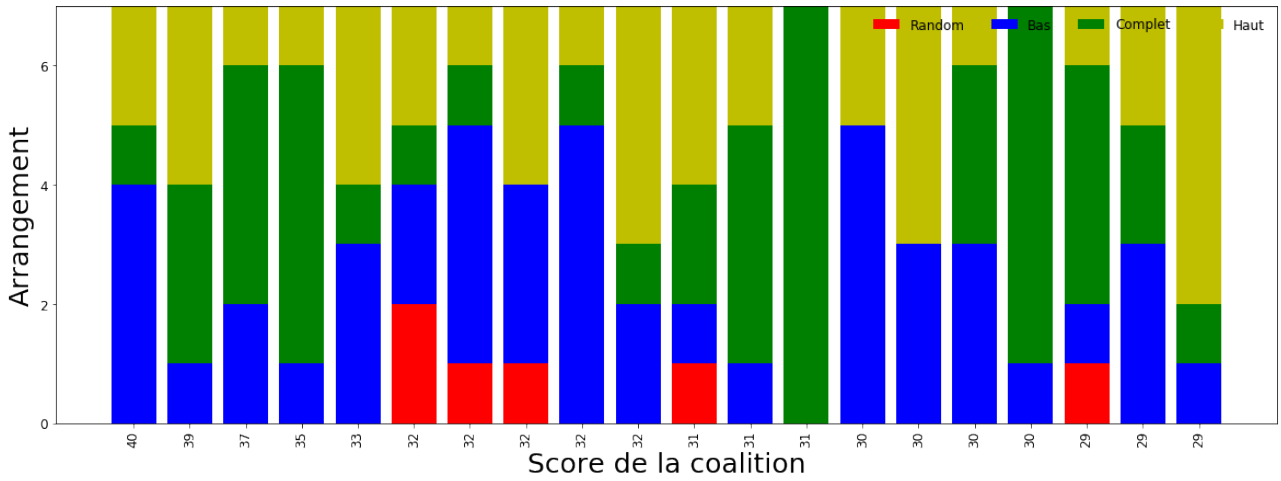


FIGURE 5 – Arrangement des agents en fonction du score de la coalition

Ces résultats sont une moyenne de 5 exécutions par coalitions pour des équipes de 7 agents.

On voit donc, assez intuitivement, que le type random est rarement performant. C'est bien cohérent avec ce que nous attendions, pour rappel il s'agit de notre population de contrôle. Par ailleurs, on remarque que les types haut et bas ont souvent une bonne synergie. Le type complet peut être efficace par lui-même mais la pente est trop forte pour qu'il soit meilleur que les autres, ce qui est cohérent avec l'expérience que nous avons simulé.

3.2 *Valeur de Shapley et contribution marginale*

La valeur de Shapley est un algorithme datant de 1953 et a été créé pour résoudre les jeux coopératifs.

Pour cela, cet algorithme analyse chaque agent et somme pour toutes les coalitions réalisables la différence de score entre une coalition et cette même coalition auquel on aurait ajouté l'agent étudié. On obtient alors la contribution marginale de chaque agent.

$$sh_i = \sum_{\substack{C \subseteq N \\ i \in C}} \frac{(n - |C|)!(|C| - 1)!}{n!} [v(C) - v(C \setminus \{i\})]$$

avec sh_i la valeur de Shapley pour l'agent i, n le nombre d'agents, N l'ensemble des coalitions, $|C|$ le nombre d'agents dans la coalition C et $v(C)$ la valeur de la coalition C.

Pour appliquer cet algorithme nous effectuons le pseudo-code suivant

Nous créons un ensemble de coalitions de la façon suivante :

-Soit nous générons toutes les coalitions pour des équipes de 1 à 12 agents pour la valeur de Shapley totale

-Soit , pour la valeur de Shapley à cardinalité fixe, nous devons générer toutes les coalitions pour deux tailles (n et n-1).

Nous effectuons ensuite les simulations pour toutes ces coalitions et en extrayons les scores.

Enfin, nous appliquons l'algorithme de Shapley sur les valeurs obtenues.

3.2.1 *Valeur de Shapley totale*

Pour le calcul de la valeur totale de Shapley, nous avons créé des coalitions allant jusqu'à une taille de 12 agents. Nous avons effectué les simulations sur ces coalitions et extrait leur score (correspondant au nombre d'objets amenés à la zone de dépôt).

Une fois ces scores assemblés, les valeurs de contribution marginale sont calculées.

Nous obtenons les scores suivants :

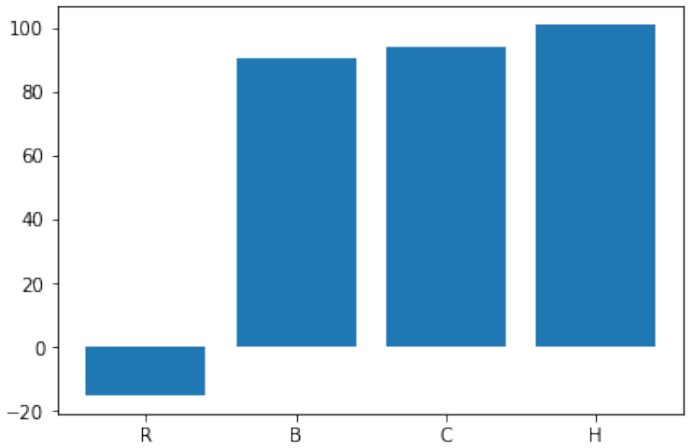


FIGURE 6 – Valeur de Shapley moyenne pour les différents types d'agents

3.2.2 Mention spéciale : Surprise lors du calcul de Shapley

Dans le graphique précédent, nous voyons que la valeur de contribution marginale de l'agent Random est négative, alors que d'après le calcul de la valeur de Shapley, tous les résultats sont censés être positif.

Cela nous a posé problème, nous pensions avoir fait une erreur de calcul. Mais après avoir vérifié le calcul plusieurs fois, nous avons déterminé que le problème ne venait pas de notre implémentation mais du calcul en lui-même.

Le calcul réalisant une somme de différence entre une équipe et cette même équipe auquel on a enlevé un agent impose une hypothèse forte :

Un agent à forcément un impact positif (ou du moins nul) sur une coalition.

Cependant il arrive que lors d'un ajout d'agent a une coalition, celui-ci peut nuire à l'efficacité de cette dernière. Dans notre cas il s'agit de l'agent aléatoire qui occupe de l'espace et ainsi gêne les autres agents. Il peut même bloquer un agent pendant toute une simulation.

On peux aussi prendre l'exemple d'un match de football où 2 équipes de 11 joueurs sont sur le terrain. Si on ajoute un joueur a une équipe, cela augmentera leur chance d'avoir un meilleur score, mais si nous ajoutons 100 joueurs ou des ramasseurs de balles, il est fort probable que les capacités de but de cette équipe chute drastiquement.

Ainsi, dans ce problème la valeur de Shapley n'assure pas la monotonie.

3.2.3 Valeur de Shapley pour des coalitions à cardinalité fixe

Le calcul de la valeur totale de Shapley est conséquente (pour n agents, il y a 2^n coalitions à considérer) et reflète les contributions dans le cas général.

C'est pour cela que nous avons décidé de calculer la valeur de Shapley pour des coalitions à cardinalité fixe pour voir si les coalitions ayant un rapport d'agents se comportent aussi bien dans des simulations à petit nombre d'agents et à grand nombre (ci-dessous des exemples de différences des meilleures coalitions lorsque le nombre d'agents change).

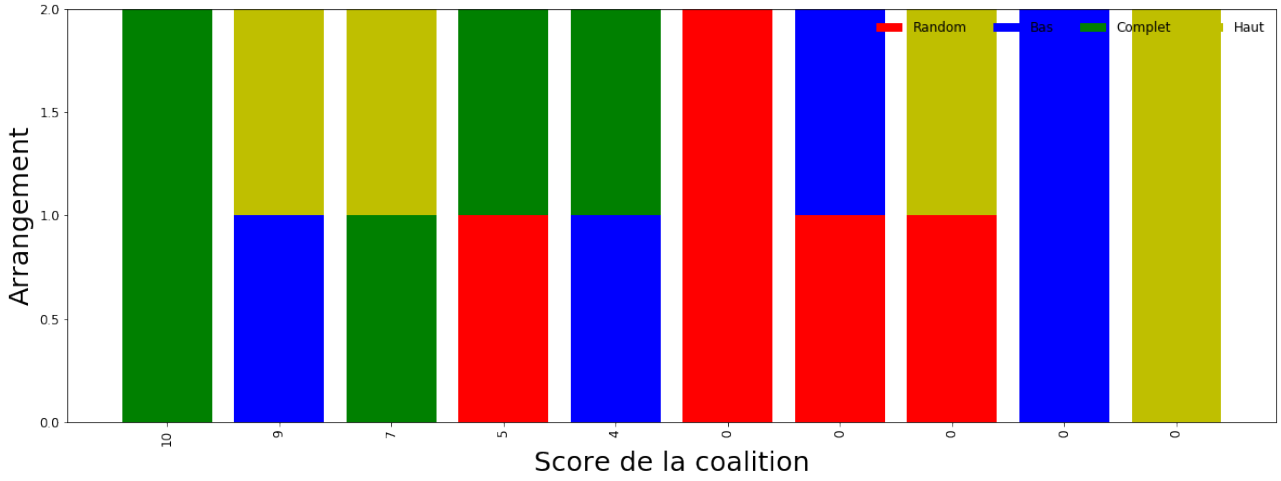


FIGURE 7 – Coalitions à 2 agents

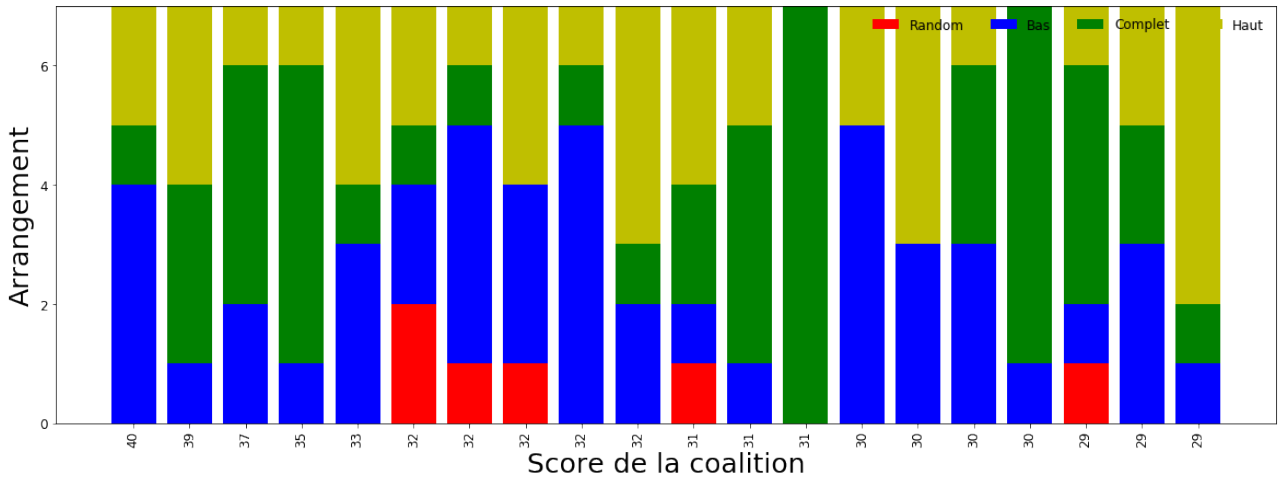
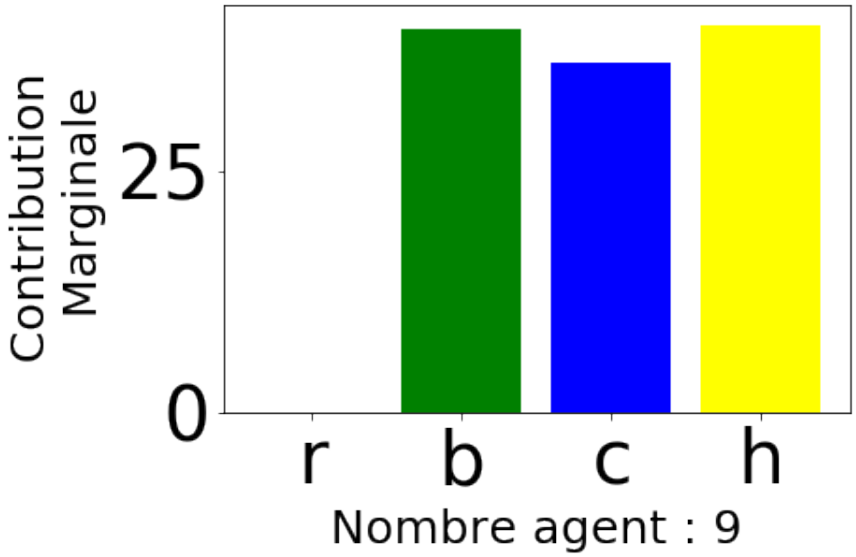


FIGURE 8 – Coalitions à 7 agents

Pour cela, nous avons défini une taille de coalitions à étudier. Nous avons effectué les simulations sur toutes les coalitions de cette taille pour obtenir les contributions marginales suivantes :



On y voit donc clairement que le random ne contribue en rien, comme attendu. Quant aux autres types, les haut et bas sont tous deux légèrement meilleurs que le complet, ce qui s'explique par leur synergie.

3.3 *Approximation via Monte-Carlo Sampling*

Les calculs de Shapley sont très efficaces dans leur résultat mais le temps de calcul est exponentiel en fonction du nombre d'agents étudiés.

C'est pourquoi nous avons cherché à approximer ces résultats en réduisant le temps de calcul. Pour cela nous avons implémenté un algorithme de type Monte-Carlo[10], c'est-à-dire un algorithme induisant de l'aléatoire et dont on s'assure de la terminaison mais dont le résultat peut être erroné (avec une probabilité très faible). Par opposition à ceux-ci il existe les algorithmes de Las-Vegas, également aléatoires, qui retournent forcément la bonne solution mais qui peuvent tourner pendant un temps très long.

Application :

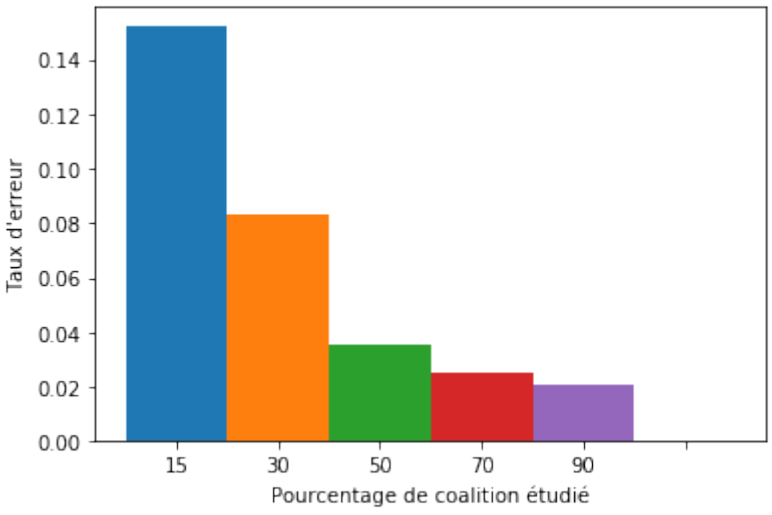
Pour approximer la valeur de Shapley, nous avons décidé de réduire le nombre de coalitions étudiées par l'algorithme. Pour ce faire nous générons un ensemble de coalitions : - Soit toutes les coalitions allant d'une taille de 1 a 12 agents pour la valeur totale - Soit toutes les coalitions d'une taille fixe Ensuite nous réduisons cet ensemble de coalition, en effectuant un tirage aléatoire sans reprise pour une taille correspondant a un certain pourcentage de l'ensemble initial.

Pour le calcul des coalitions a taille fixe, l'algorithme de Shapley demande pour chaque coalition, la coalition correspondant a la précédente auquel on aurait ajouté un agent.

Nous ajoutons alors a l'ensemble de coalitions toutes les altérations de coalitions (pour chaque coalition C et agent i nous ajoutons la coalition c∪i)

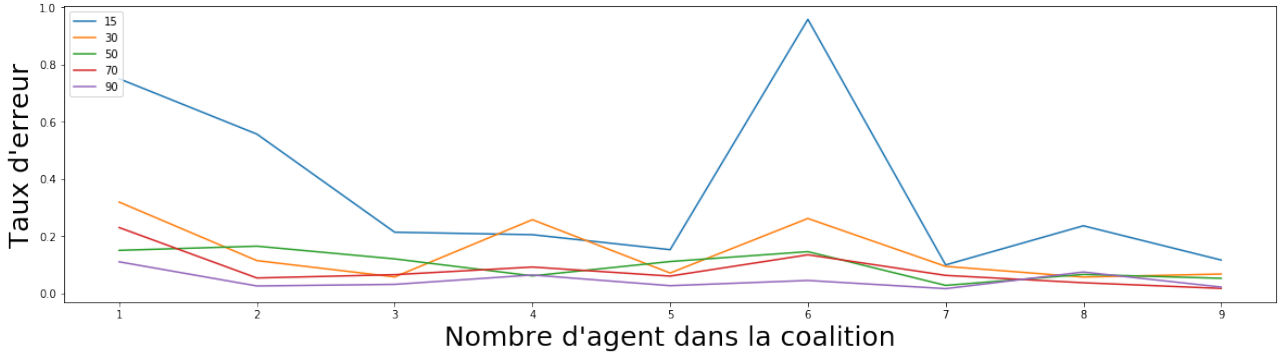
Nous effectuons ensuite les simulations sur toutes les coalitions et en récupérons les scores pour enfin calculer la valeur de Shapley

3.3.1 *Appliqué à la valeur de Shapley totale*



On observe que le taux d'erreur est décroissant en fonction de la proportion de coalitions que l'on considère, descendant jusqu'à seulement 2% d'erreur lorsque l'on étudie plus de 70% des coalitions.

3.3.2 *Appliqué à la valeur de Shapley pour des coalitions à cardinalité fixe*



On voit qu'on a une valeur approchée avec un taux d'erreur avoisinant les 10% pour des calculs de 70 à 90% du nombre de coalitions total.

3.4 *Méthode de Peter Stone*

Dans son article[3], Peter Stone détaille une méthode pour déterminer si un agent est plus utile qu'un autre pour une tâche donnée.

Pour faire ceci il décrit l'algorithme comme suit :

- Choisir deux agents a_1 et a_2 à comparer et une coalition initiale A
- Associer un score r_1 (resp. r_2) à a_1 (resp. a_2) et les initialiser à 0
- Répéter le processus suivant :
  - Choisir une tâche (dans notre cas la tâche est unique)
  - Choisir une sous-coalition B à partir de la coalition initiale A
  - On sélectionne aléatoirement un agent b à enlever pour obtenir une coalition B
  - ajouter le score de la coalition $B - \cup\{a_1\}$ à r_1
  - ajouter le score de la coalition $B - \cup\{a_2\}$ à r_2
- Si $r_1 > r_2$ l'agent 1 est plus utile que l'agent 2 et inversement

Comparons alors tous nos types d'agents. En appliquant cet algorithme, on obtient :

a_1	a_2	$r_1 - r_2$	$a_1 \,?\, a_2$
R	B	-3276	$R \prec B$
B	C	-1448	$B \prec C$
B	H	-2	$B \sim H$
C	H	1207	$C \succ H$
R	H	-3413	$R \prec H$
R	C	-4549	$R \prec C$

D'où l'ordre d'utilité des agents : $C \succ B \sim H \succ R$

4 Bibliographie

Norme utilisée : IEEE

[1] S. Liemhetcharat et M. Veloso, « Weighted synergy graphs for role assignment in ad hoc heterogeneous robot teams », oct. 2012, p. 5247–5254, isbn : 978-1-4673-1737-5. doi : 10.1109/IROS.2012.6386027.

[2] P. MacAlpine et P. Stone, « Evaluating Ad Hoc Teamwork Performance in Drop-In Player Challenges », in *AAMAS Multiagent Interaction without Prior Coordination (MIPC) Workshop*, Sao Paulo, Brazil, mai 2017. adresse : http://www.cs.utexas.edu/users/ai-lab/?macalpine:mipc17.

[3] P. Stone, G. A. Kaminka, S. Kraus et J. S. Rosenschein, « Ad Hoc Autonomous Agent Teams : Collaboration without Pre-Coordination », in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, juil. 2010.

[4] L. S. Shapley, « A Value for n-Person Games », in *Contributions to the Theory of Games (AM-28)*, H. W. Kuhn et A. W. Tucker, éds. Princeton University Press, 1953, t. 2, chap. 17, p. 307–318, isbn : 9781400881970. doi : https://doi.org/10.1515/9781400881970-018.

[5] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo et T. Wenseleers, « Evolution of Self-Organized Task Specialization in Robot Swarms », *PLOS Computational Biology*, t. 11, n° 8, p. 1–21, août 2015. doi : 10.1371/journal.pcbi.1004273. adresse : https://doi.org/10.1371/journal.pcbi.1004273.

[6] N. Bredèche, J. Montanier, B. Weel et E. Haasdijk, « Roborobo ! a Fast Robot Simulator for Swarm and Collective Robotics », *CoRR*, t. abs/1304.2888, 2013. arXiv : 1304.2888. adresse : http://arxiv.org/abs/1304.2888.

[7] F. Rosenblatt, « The Perceptron : A Probabilistic Model for Information Storage and Organization in The Brain », *Psychological Review*, p. 65–386, 1958.

[8] X. Deng et C. H. Papadimitriou, « On the Complexity of Cooperative Solution Concepts », *Mathematics of Operations Research*, t. 19, n° 2, p. 257–266, 1994. doi : 10.1287/moor.19.2.257. eprint : https://doi.org/10.1287/moor.19.2.257. adresse : https://doi.org/10.1287/moor.19.2.257.

[9] S. Ieong et Y. Shoham, « Marginal Contribution Nets : A Compact Representation Scheme for Coalitional Games », in *Proceedings of the 6th ACM Conference on Electronic Commerce*, sér. EC '05, Vancouver, BC, Canada : ACM, 2005, p. 193–202, isbn : 1-59593-049-3. doi : 10.1145/1064009.1064030. adresse : http://doi.acm.org/10.1145/1064009.1064030.

[10] T. Pawel Michalak, K. V. Aadithya, P. L. Szczepanski, B. Ravindran et N. R. Jennings, « Efficient Computation of the Shapley Value for Game-Theoretic Network Centrality », *arXiv e-prints*, fév. 2014. arXiv : 1402.0567 [cs.GT].