Modélisation de la fermeture coloniale chez les fourmis pour la classification non- supervisée

Nicolas Labroche, N. Monmarché, Gilles Venturini

Modélisation de la fermeture coloniale chez les fourmis pour la classification non-supervisée

N. Labroche, N. Monmarché et G. Venturini
Laboratoire d'Informatique de l'Université de TOURS
École d'Ingénieur en Informatique
64 avenue Jean Portalis 37000 TOURS
{labroche,monmarche,venturini}@univ-tours.fr

October 15, 2015

Abstract

Nous présentons dans cet article une nouvelle méthode de classification non-supervisée appelée AntClust, inspirée du système de reconnaissance chimique des fourmis. Celui-ci, connu sous le nom de fermeture coloniale, repose sur l'apprentissage et le partage d'une odeur coloniale commune à toutes les fourmis d'un même nid. Dans notre méthode, une fourmi artificielle est associée à un objet à classer. Chaque fourmi artificielle est capable d'apprendre une odeur coloniale grâce à un processus de rencontres aléatoires et un ensemble de règles comportementales locales. Les fourmis se regroupent ainsi en colonies qui partagent une odeur similaire, ce qui définit une partition des données. Nous comparons cet algorithme à la méthode des K-Means et montrons que les résultats que nous obtenons sont meilleurs sur des jeux de données réelles et artificielles. Cette méthode ne nécessite pas d'initialisation particulière des données (partition initiale, nombre de classes à obtenir) ou de format particulier (données numériques ou symboliques). Nous discutons également de la pertinence des choix relatifs à l'apprentissage de l'odeur coloniale pour la convergence de notre algorithme.

1 Introduction

Plusieurs méthodes algorithmiques se sont inspirées des fourmis pour définir de nouvelles heuristiques pour la résolution de problèmes complexes. Par exemple, les comportements collectifs des fourmis ont été modélisés dans l'approche algorithmique ACO ("Ant Colony Optimization") dans laquelle les pistes de phéromones sont utilisées [2]. Similairement, des algorithmes de classification ont déjà été proposés [5], [6]. Dans ces travaux, les chercheurs ont modélisé la capacité des fourmis à trier leur couvain. Une fourmi est alors capable de porter un ou plusieurs objets et de les déposer selon certaines probabilités. Le

seul moyen de communication dont dispose les fourmis est alors la configuration des objets déposés sur le sol. Après un certain temps, les fourmis établissent des groupes d'objets similaires et résolvent un problème connu sous le nom de classification non-supervisée.

Nous nous intéressons dans cet article au phénomène de fermeture coloniale qui passe par la construction et l'apprentissage d'une odeur propre à chaque nid et qui permet la détection des intrusions. La modélisation de ce mécanisme n'a pour l'instant pas été utilisée dans la résolution de problèmes d'informatique et nous montrons qu'elle peut être appliquée au problème de classification non-supervisée.

L'article s'articule comme suit : dans la section 2 sont introduits les fondements biologiques sur lesquels repose la modélisation pour le problème de classification qui est présenté dans la section 3. La 4^{ieme} section détaille les résultats encourageants obtenus avec notre méthode AntClust et discute des différentes options concernant l'apprentissage de l'odeur coloniale par les fourmis et leur impact sur le résultat de la classification. Enfin, la dernière section discute des perspectives de développement et des domaines d'application envisagés.

2 A propos des fourmis réelles

Comme tous les insectes sociaux, les fourmis ont développé un mécanisme de fermeture coloniale qui leur permet de privélégier les relations avec les membres de leur nid et de rejeter les intrus qui peuvent être de la même espèce.

La discrimination repose sur la comparaison de l'odeur émise par chaque fourmi, le "label", et d'un modèle de référence nommé "template" : on parle alors de "*phenotype matching*" [3]. Chaque fourmi apprend les labels propres à sa colonie à sa naissance, en s'imprégnant physiquement des odeurs des ouvrières de son nid, lorsqu'elles la nourrissent. Par la suite, la fourmi remet à jour continuellement son template en intégrant les labels des autres fourmis et en diffusant le sien pour être reconnue à son tour. Ces échanges répétés conduisent à la mise en place d'une odeur coloniale commune à tous les membres du nid.

Les labels sont principalement constitués d'hydrocarbures cuticulaires et de substances chimiques extraites de la nourriture ou bien issues de matériaux constitutifs du nid. Selon les espèces certains facteurs peuvent influencer la reconnaissance entre fourmis. On note le rôle particulier que peut alors jouer la reine [1], outre celui de donner naissance aux nouvelles fourmis. Soit la reine ne participe qu'à la diffusion de l'odeur coloniale au plus grand nombre, soit elle intervient directement dans sa composition chimique.

D'un point de vue individuel, les hydrocarbures sont générés par les cellules œnocytes selon le génome de chaque fourmi et sont ensuite distribués par des circuits internes vers la glande post-pharyngienne (GPP) ou vers la cuticule des fourmis. Lors d'auto-toilettages les fourmis sont capables de renforcer leur propre label en déversant une partie du contenu de leur GPP sur leur cuticule et assurent ainsi leur reconnaissance au sein de la colonie.

L'odeur coloniale est un mélange de tous les labels des fourmis du nid

échangés par le biais de toilettages sociaux (une fourmi déverse un peu du contenu de sa GPP sur la cuticule d'une autre), par trophallaxie (une fourmi transfère le contenu de sa GPP dans la GPP d'une autre), ou plus simplement par contacts cuticulaires dans le nid. Ce modèle de répartition des labels à tous les membres d'une même colonie de façon homogène est appelé modèle Gestalt et conduit à l'établissement de l'odeur coloniale.

3 AntClust : classification et fourmis artificielles

Le problème de classification non-supervisée Dans ce type de problème, le but est de trouver des groupes d'objets similaires qui soient le plus proche possible de la partition naturelle de l'espace de départ. Aucune hypothèse n'est faite concernant le type des données manipulées : elles peuvent être numériques, symboliques ou encore du premier ordre. Il suffit, pour pouvoir utiliser notre heuristique, de définir une mesure de similarité qui prend comme paramètre d'entrée un couple d'objets i et j et qui retourne une valeur $Sim(i, j)$ comprise entre 0 et 1. Une valeur de 0 indique que les objets sont totalement différents et 1 qu'ils sont rigoureusement identiques.

Les fourmis artificielles Dans notre méthode, chaque donnée de l'espace de départ va être représentée par une fourmi artificielle et plus précisément par son génome. Tout au long des rencontres qu'elle va effectuer, la fourmi va tenter d'accorder son label et son template à son génome pour trouver la colonie qui lui ressemble le plus. Nous définissons donc les paramètres suivants pour une fourmi i :

- Le label $Label_i$ indique le nid d'appartenance de la fourmi i et est modélisé par une variable représentant l'indice du nid. Au départ, les fourmis n'appartiennent à aucun nid et donc $Label_i = 0$. Cette valeur évolue jusqu'à ce que la fourmi trouve le nid qui lui convient le plus.

- Le template est à la fois défini par le génome $Genome_i$ de la fourmi (i.e. une donnée de l'espace de départ) et par un seuil d'acceptation noté $Template_i$. Celui-ci fait l'objet d'un apprentissage à l'initialisation des fourmis artificielles et d'une mise à jour continue pendant la classification. Le calcul de $Template_i$ s'appuie sur l'estimation par la fourmi i des similarités maximales et moyennes observées lors de rencontres avec d'autres fourmis et notées respectivement $\max(Sim(i, \cdot))$ et $\overline{Sim}(i, \cdot)$ (voir section 3).

- L'estimateur M_i reflète la réussite des rencontres de la fourmi i. Au départ, M_i vaut 0 puisque la fourmi i n'a pas encore réalisée de rencontres. M_i estime la taille du nid de la fourmi i, c'est-à-dire le nombre de fourmis ayant le même label que la fourmi i. M_i est augmenté quand la fourmi i rencontre des individus de son nid et est diminué dans le cas contraire.

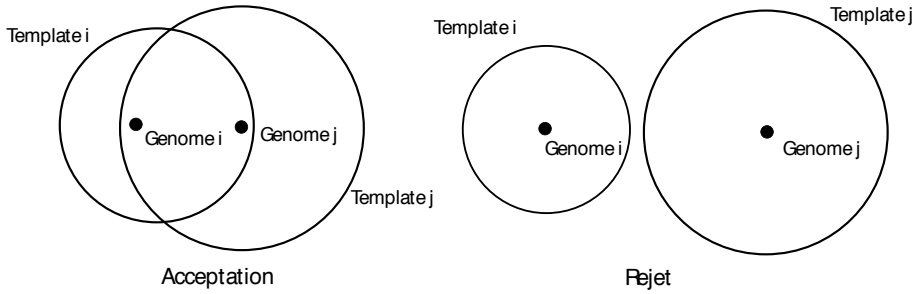

Figure 1: Principes de l'acceptation et du rejet entre 2 fourmis i et j

- L'estimateur M_i^+ mesure l'intégration de la fourmi i dans son nid. Il est augmenté si la fourmi i et une autre de son nid se rencontrent et s'acceptent et est diminué sinon.
- L'âge A_i, qui au départ vaut 0, est utilisé dans les calculs de mise à jour du seuil d'aceptation $Template_i$.

Apprentissage du seuil d'acceptation $Template_i$: Les fourmis artificielles estiment au cours d'un nombre fixé de rencontres aléatoires la similarité maximale et moyenne qu'il peut y avoir entre leur génome et celui des fourmis rencontrées. Une fourmi i définit alors son seuil initial d'acceptation $Template_i$ de la manière suivante :

$$Template_i \leftarrow \frac{\overline{Sim}(i,\cdot) + \max(Sim(i,\cdot))}{2} \tag{1}$$

Les valeurs des similarités maximale et moyenne sont aussi remises à jour après chaque rencontre en fonction de l'âge de la fourmi. Ainsi, la valeur du template est réapprise continuellement par chaque fourmi.

Acceptation entre fourmis La résolution des rencontres est conditionnée par l'acceptation ou le rejet préalable des fourmis. Une fourmi accepte toutes les fourmis dont le génome est proche du sien relativement à son seuil d'acceptation comme le montrent l'équation 2 et la figure 1.

$$\begin{aligned} Acceptation(i,j) \Leftrightarrow & (Sim(i,j) > Template_i) \\ & \wedge (Sim(i,j) > Template_j) \end{aligned} \tag{2}$$

Principe général d'AntClust L'algorithme AntClust repose sur l'association d'un objet à classer à une fourmi artificielle. Celui-ci simule, pendant un nombre fixé d'itérations et pour chaque fourmi, une rencontre avec une autre fourmi choisie aléatoirement. L'issue de ces rencontres est déterminée par un ensemble de règles comportementales qui font évoluer les labels et les

templates. A l'issue de l'algorithme, les fourmis les plus similaires entre elles possèdent le même label, ce qui définit une partition de l'espace de départ.

Algorithme 1: Algorithme principal d'AntClust

AntClust()

(1) Initialiser les fourmis artificielles :
(2) $Genome_i \leftarrow i^{eme}$ objet des données à classer
(3) $Label_i \leftarrow 0$
(4) $Template_i$ est initialisé selon l'équation 1
(5) $M_i \leftarrow 0$, $M_i^+ \leftarrow 0$, $A_i \leftarrow 0$
(6) Simuler Nb_{ITER} itérations durant lesquelles chaque fourmi en rencontre une autre choisie aléatoirement
(7) Supprimer les nids avec moins de $P \times n$ ($P \ll 1$) fourmis
(8) Ré-affecter chaque fourmi sans nid, au nid de la fourmi dont elle est la plus similaire.

Règles comportementales des fourmis artificielles Ces règles s'appliquent lors de chaque rencontre entre fourmis et entraînent l'évolution des paramètres (*label*, *template*, M_i, M_i^+) jusqu'à ce que des nids stables soient formés. Nous détaillons ci-après les règles lorsque deux fourmis i et j se rencontrent :

R_1 Règle de création d'un nouveau nid :
Si ($Label_i = Label_j = 0$) et $Acceptation(i, j)$ Alors Créer un nouveau label $Label_{NEW}$ et $Label_i \leftarrow Label_{NEW}, Label_j \leftarrow Label_{NEW}$. Si *Acceptation* est faux alors la règle R_6 s'applique.

R_2 Règle d'ajout d'une fourmi sans label à un nid existant :
Si ($Label_i = 0 \wedge Label_j \neq 0$) et $Acceptation(i, j)$ Alors $Label_i \leftarrow Label_j$. Le cas où ($Label_j = 0 \wedge Label_i \neq 0$) est traité de manière similaire.

R_3 Règle de rencontre "Positive" entre 2 fourmis du même nid :
Si ($Label_i = Label_j$) $\wedge$ ($Label_i \neq 0$) $\wedge$ ($Label_j \neq 0$) et $Acceptation(i, j)$ Alors Augmenter M_i, M_j, M_i^+ et M_j^+. On entend par Augmenter (3) ou par Diminuer (4) une variable x :

$$x \leftarrow (1 - \alpha) \times x + \alpha \tag{3}$$

$$x \leftarrow (1 - \alpha) \times x \tag{4}$$

(Ici, nous choisissons $\alpha = 0.2$)

R_4 Règle de rencontre "Négative" entre 2 fourmis du même nid :
Si ($Label_i = Label_j$) $\wedge$ ($Label_i \neq 0$) $\wedge$ ($Label_j \neq 0$) et $Acceptation(i, j) = Faux$ Alors Augmenter M_i, M_j et Diminuer M_i^+ et M_j^+. La fourmi x (x=i, x=j) qui est la moins intégrée dans son nid ($x | M_x^+ = \min(M_i^+, M_j^+)$), perd son label et n'a donc plus de nid ($Label_x \leftarrow 0$, $M_x \leftarrow 0$ et $M_x^+ \leftarrow 0$).

R_5 Règle de rencontre entre 2 fourmis d'un nid différent :
Si ($Label_i \neq Label_j$) et $Acceptation(i, j)$ Alors Diminuer M_i et M_j. La fourmi x qui a le plus petit M_x (i.e. la fourmi appartenant au nid le plus petit) change son label pour appartenir au nid de la fourmi rencontrée.

R_6 Règle par défaut : Si aucune autre règle ne s'applique, alors rien ne se passe!

Analyse des règles comportementales La règle R_1 a un rôle fondamental car c'est la seule règle qui peut créer un nouveau label et donc un nouveau nid. Elle entraîne le rassemblement des fourmis les plus similaires dans les premiers clusters, qui servent de "graines" pour générer les clusters définitifs.

La règle R_2 agrandit les graines de clusters issues de la règle R_1 en y ajoutant des fourmis ayant un génome compatible.

La règle R_3 augmente simplement les estimateurs M et M^+ en cas d'acceptation entre les deux fourmis qui se rencontrent.

La règle R_4 permet de réparer les mauvaises affectations de fourmis dans des nids. Elles peuvent survenir au départ lorsque les profils de ces derniers ne sont pas clairement établis. Cette règle autorise le rejet des fourmis les moins intégrées et permet de les réinitialiser afin qu'elles trouvent un nouveau nid plus adéquat. L'appartenance d'un objet, qui n'était pas classé optimalement, à un groupe peut ainsi être modifiée et peut améliorer la réussite globale de l'algorithme.

La règle R_5 autorise le regroupement entre clusters similaires, les plus petits étant intégrés au plus grands. Au début de la classification, il y a un très grand nombre de clusters. Cette règle permet une décroissance significative du nombre des clusters en regroupant plusieurs sous-clusters en un seul plus grand.

La règle R_6 se produit quand aucune autre ne s'applique.

4 Tests et résultats

Nous comparons dans ce paragraphe, notre algorithme AntClust à la méthode des K-Means [4]. Celle-ci repose sur une partition initiale des données (et donc sur un nombre maximum de classes). A chaque itération, chaque donnée est associée au groupe dont le centre est le plus proche, ce qui permet d'affiner graduellement la partition jusqu'à un état stable. Cette stabilité est mesurée par l'inertie intraclasse. Nous utilisons dans notre cas une partition initiale formée de 10 clusters générés aléatoirement. La méthode est nommée en conséquence 10-Means par la suite.

Jeux de données et paramètrages initiaux Dans le but de comparer les 2 approches, nous utilisons des jeux de données générées aléatoirement ainsi que des données réelles qui utilisent une représentation par attributs. Les données artificielles sont $\text{Art}_{i,i\in[1,8]}$ et les données réelles sont : Iris, Glass, Pima, Soybean et Thyroid. Concernant les données artificielles, Art1, Art2, Art3,

Jeux de données	#Objets	#Attributs	#Clusters
Art1	400	2	4
Art2	1000	2	2
Art3	1100	2	4
Art4	200	2	2
Art5	900	2	9
Art6	400	8	4
Art7	100	2	1
Art8	1000	2	1
Iris	150	4	3
Glass	214	9	7
Pima	798	8	2
Soybean	47	35	4
Thyroid	215	5	3

Table 1: Caractéristiques principales des jeux de données

ART5 et ART6 sont générées selon des lois gaussiennes ayant chaucne des difficultés propres (attributs non pertinents, recouvrement de clusters), ART4 est générée selon une loi uniforme et ART7 ainsi que ART8 correspondent à du bruit blanc. Les caractéristiques principales des jeux de données sont résumées dans la table 1.

Toutes les évaluations ont été menées sur 50 tests pour chaque jeu de données et chaque méthode. Dans le cas d'ANTCLUST, chaque test correspond à 300000 itérations durant lesquelles chaque fourmi en rencontre une autre choisie aléatoirement. Les résultats sont exposés dans la table 2.

Les champs suivants apparaissent dans la table 1 pour chaque jeu de données : le nombre d'objets à classer (*"#Objets"*) et le nombre d'attributs qui les décrivent (*"#Attributs"*) ainsi que le nombre de clusters théorique (*"#Clusters"*).

Mesure de similarité et erreur de classification Nous présentons ici, le fonctionnement de la mesure de similarité que nous utilisons pour traiter les bases de la table 1 et nous l'illustrons sur 2 exemples. Nous précisons le calcul de l'erreur de classification utilisée afin de donner un sens aux résultats présentés par la suite.

Notre mesure de similarité impose que chaque objet soit décrit par un ensemble d'attributs, chacun ayant un type θ_k parmi les Nb_{Types} types existants (i.e. numériques, symboliques, ...). La similarité entre 2 objets o_i et o_j peut alors s'écrire :

$$Sim(o_i, o_j) = \frac{1}{Nb_{Types}} \times \sum_{k=1}^{Nb_{Types}} Sim_{\theta_k}(o_i, o_j) \tag{5}$$

$$Sim_{\theta_k}(o_i, o_j) = 1 - \left(\frac{1}{Occ\,(\theta_k)} \times \sum_{k=1}^{Occ(\theta_k)} \Delta_{\theta_k}(o_i, o_j) \right) \tag{6}$$

où Sim_{θ_k} est la similarité calculée entre tous les attributs de type θ_k pour les objets o_i et o_j, $Occ(\theta_k)$ le nombre d'occurrence du type θ_k dans la description d'un objet o et enfin Δ_{θ_k} une fonction qui renvoit la dissimilarité entre 2 attributs de type θ_k des objets o_i et o_j. Nous présentons 2 exemples de fonctions : Δ_{Num} (7) et Δ_{Symb} (8), qui s'appliquent sur des couples de valeurs (i, j) respectivement de types numériques et symboliques.

$$\Delta_{Num}(i,j) = \begin{cases} 0 \text{ if } \max_\theta = \min_\theta \\ \frac{|i-j|}{|\max_\theta - \min_\theta|} \text{ sinon} \end{cases} \tag{7}$$

$$\Delta_{Symb}(i,j) = \begin{cases} 0 \text{ if } i = j \\ 1 \text{ sinon} \end{cases} \tag{8}$$

L'erreur de classification E_c mesure la différence entre la partition théorique et celle obtenue. Elle peut être formalisée comme suit en considérant toutes les paires d'objets :

$$E_c = \frac{2}{N(N-1)} \times \sum_{(i,j)\in\{1,\ldots,N\}^2, i<j} \epsilon_{ij} \tag{9}$$

où:

$$\epsilon_{ij} = \begin{cases} 0 \text{ if } (c(o_i) = c(o_j) \wedge c'(o_i) = c'(o_j)) \vee \\ \quad (c(o_i) \neq c(o_j) \wedge c'(o_i) \neq c'(o_j)) \\ 1 \text{ sinon} \end{cases} \tag{10}$$

avec $c(o)$ l'identifiant de cluster théorique pour l'objet o et $c'(o)$ l'identifiant trouvé par l'algorithme testé.

Résultats La table 2 présente les nombres de clusters effectivement trouvés par les deux méthodes (*"#Clusters trouvés"*) avec leur écart-type (*"*σ_{cf}*"*) et aussi l'erreur commise par chacun des algorithmes (*"%Erreur Clust."*) associée à son écart-type(*"*σ_e*"*).

Notre algorithme AntClust obtient de meilleurs résultats que 10-Means. Cela est du au fait que, de manière générale, AntClust parvient à obtenir une meilleure appréciation du nombre de clusters à trouver. 10-Means part de 10 clusters générés aléatoirement et ne parvient pas à les regrouper du fait des légères différences apparaissant dans les jeux de données. En fait, 10-Means obtient de meilleurs résultats qu'AntClust uniquement 2 fois : pour Art5 et pour Glass car le nombre de clusters théoriques est proche de 10. Ces tests démontrent qu'AntClust est capable de traiter des jeux de données de taille variée avec le même succès (voir Soybean, Art1, Art2 et Art6) mais qu'il lui est difficile d'estimer le nombre de clusters quand celui-ci devient trop important (voir Art5 par exemple). Cela est peut-être du au fait qu'il n'existe qu'une seule règle de création de nouveau nid. Cette règle n'est pas appliquée après le départ de l'algorithme car une fourmi rejetée d'un nid a plus de chance d'être réintégrée dans un autre que d'en créer un nouveau.

Jeux de données	# Clusters trouvés		%Erreur Clust.	
	10_M $[\sigma_{cf}]$	A_C $[\sigma_{cf}]$	10_M $[\sigma_e]$	A_C $[\sigma_e]$
Art1	8.58 [0.98]	4.00 [0.00]	0.18 [0.01]	0.18 [0.02]
Art2	8.52 [0.96]	2.00 [0.00]	0.38 [0.01]	0.06 [0.02]
Art3	8.28 [0.96]	2.00 [0.00]	0.31 [0.01]	0.15 [0.02]
Art4	6.38 [0.75]	3.46 [0.50]	0.32 [0.02]	0.24 [0.05]
Art5	8.82 [0.91]	3.28 [0.45]	0.08 [0.01]	0.28 [0.03]
Art6	8.46 [1.08]	4.00 [0.00]	0.10 [0.02]	0.04 [0.01]
Art7	7.76 [1.03]	3.28 [0.45]	0.87 [0.02]	0.66 [0.02]
Art8	8.78 [0.83]	3.78 [0.42]	0.88 [0.01]	0.72 [0.04]
Iris	7.12 [1.11]	2.16 [0.37]	0.18 [0.03]	0.22 [0.01]
Glass	9.44 [0.70]	3.62 [0.64]	0.29 [0.02]	0.39 [0.03]
Pima	9.90 [0.36]	2.66 [0.56]	0.50 [0.01]	0.45 [0.01]
Soybean	8.82 [0.97]	4.42 [0.57]	0.13 [0.02]	0.07 [0.04]
Thyroid	9.56 [0.57]	2.88 [0.33]	0.42 [0.02]	0.18 [0.06]

Table 2: Resultats obtenus sur 50 tests pour chaque méthode et chaque jeu de données. 10_M et A_C désignent respectivement 10-MEANS et ANTCLUST

Pertinence des paramètres Notre algorithme dépend de deux paramètres : le nombre d'itérations Nb_{ITER} (fixé à 300000 jusqu'à présent) et le Template initialisé selon l'équation 1. Des tests nous ont permis de vérifier que le nombre d'itérations peut se limiter à 100000 et garantir la convergence souhaitée (voir figure 2). D'autres expériences ont montré que le plus important dans la phase d'apprentissage initiale du template n'est pas l'estimation du seuil d'acceptation mais l'évaluation des similarités maximale et moyenne rencontrées. En effet, si on remplace l'estimation de ces similarités par le tirage de 2 valeurs aléatoires, ANTCLUST ne parvient pas à générer plus de 3 classes. Cette imprécision n'arrive pas à être corrigée par la mise à jour ultérieure des templates car elle en-

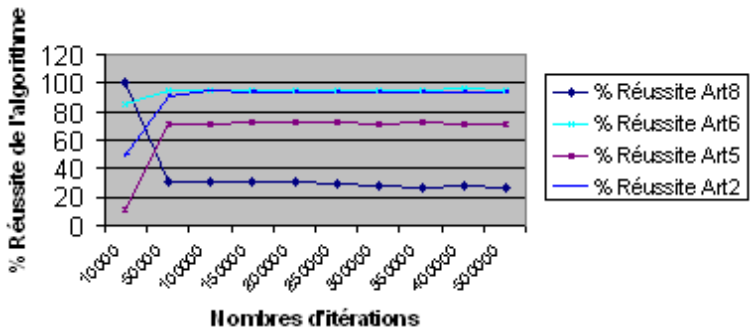

Figure 2: Convergence du nombre de classes trouvée en fonction du nombre d'itération de l'algorithme

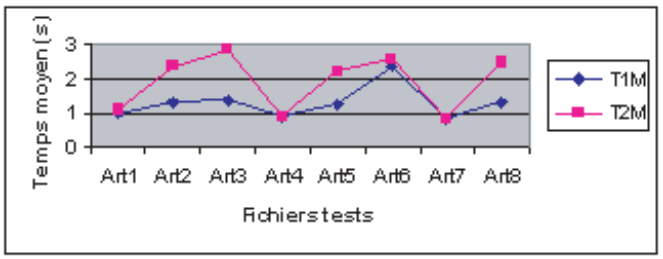

Figure 3: Valeurs moyennes de T_1("T1M") et T_2("T2M") en fonction des données de test (en secondes)

traîne des valeurs de seuils d'acceptation trop basses. Les fourmis se regroupent alors en trop peu de nids. Nous avons aussi démontré que l'on pouvait obtenir des résultats aussi bons que ceux présentés dans la table 2 si l'on supprimait la mise à jour des templates après les rencontres et si l'on conservait la valeur du seuil d'acceptation des fourmis exclues de leur nid au lieu de le mettre à 0 (voir règle comportementale R_4).

Enfin, nous avons cherché à évaluer les temps de calcul de notre méthode pour estimer son comportement lorsque le nombre d'attributs et le nombre d'objets varient. Nous avons définis pour cela les temps T_1 et T_2 qui correspondent respectivement à la durée de la phase de rencontres entre fourmis et à la durée totale d'exécution de l'algorithme comprenant la phase de réaffectation des fourmis seules. La figure 3 résume les résultats obtenus. On remarque que l'augmentation du nombre d'attributs est plus préjudiciable aux performances qu'une augmentation du nombre des exemples : entre ART1 et ART6 il y a une différence de 6 attributs et un ralentissement de 1.38 secondes en moyenne, alors qu'entre ART1 et ART3 il y a une augmentation de 700 objets et un ralentissement de seulement 0.4 (en considérant les périodes T_1). On constate par ailleurs que les fichiers ayant peu de données à traiter introduisent des durées T_1 et T_2 quasiment similaires (ART1, ART4, ART6 et ART7), alors que les autres possèdent jusqu'à parfois 1.92 secondes d'écart. Cela est dû au fait qu'il y a plus de fourmis à réaffecter et que pour chacune d'elles, une recherche sur l'ensemble des autres fourmis est réalisée pour trouver la plus similaire. Cette augmentation du temps de calcul ne semble pas liée à la qualité de la partition proposée par notre algorithme car pour ART7 tout comme pour ART8, la méthode ne fonctionne pas (car les bases correspondent à du bruit uniquement) et les temps ne sont pas comparables. Un dernier résultat a montré que les durées T_1 et T_2 augmentent linéairement en fonction du nombre d'itérations.

5 Conclusion

Nous décrivons dans cet article un nouvel algorithme de clustering ANTCLUST, inspiré du système de reconnaissance des fourmis. Les résultats sont bons en

regard de ceux obtenus avec les K-MEANS. Notre approche ne fait aucune hypothèse quant à la nature des données à classer et ne nécessite pas la connaissance du nombre de clusters à trouver. Par ailleurs, notre travail démontre l'importance du processus d'apprentissage dans notre méthode de classification. Nous planifions d'utiliser cet algorithme dans le domaine du Web Mining pour aider à déterminer les profils types de navigation sur les sites Web lorsqu'ANTCLUST aura été validé sur de plus amples jeux de données (plus de 10000 objets) et avec un plus grand nombre de types d'attributs manipulés (séquences de pages, ...).

References

[1] N.F. Carlin and B. Hölldobler. The kin recognition system of carpenter ants(camponotus spp.). i. hierarchical cues in small colonies. *Behav Ecol Sociobiol*, 19:123–134, 1986.

[2] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In F. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, pages 134–142. MIT Press, Cambridge, Massachusetts, 1991.

[3] B. Hölldobler and E.O. Wilson. *The Ants*, chapter Colony odor and kin recognition, pages 197–208. Springer Verlag, Berlin, Germany, 1990.

[4] A. K. Jain and Dubes R.C. *Algorithms for clustering Data*, chapter Square-Error Clustering Method, pages 96–101. Prentice Hall Advanced Reference series, 1988.

[5] E.D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In D. Cliff, P. Husbands, J.A. Meyer, and Stewart W., editors, *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 501–508. MIT Press, Cambridge, Massachusetts, 1994.

[6] N. Monmarché, M. Slimane, and G. Venturini. L'algorithme antclass : classification non supervisée par une colonie de fourmis artificielles. *Extraction des Connaissances et Apprentissage : Apprentissage et évolution*, 1(3):131–166, 2001.