Algorithme AntTree : Classification non supervisée par des fourmis artificielles

Hanène Azzag*, Nicolas Monmarché*
Mohamed Slimane*, Christiane Guinot**
Gilles Venturini*

*École Polytechnique de l'Université de Tours, Laboratoire d'Informatique, 64, Avenue Jean Portalis, 37200 Tours, France
hanene.azzag@etu.univ-tours.fr, (monmarche,slimane,venturini)@univ-tours.fr,
http ://www.antsearch.univ-tours.fr/webrtic
**CE.R.I.E.S., 20, rue Victor Noir, 92521 Neuilly-sur-Seine Cédex, France
christiane.guinot@ceries-lab.com,
http ://www.ceries.com

Résumé. Nous présentons dans cet article un nouvel algorithme de classification non supervisée. Il utilise le principe d'auto-assemblage observé chez une colonie de fourmis où des fourmis se fixent progressivement à un support fixe puis successivement aux fourmis déjà fixées afin de construire des structures vivantes. Les fourmis artificielles que nous définissons vont de manière similaire construire un arbre. Chaque fourmi représente une donnée. Les déplacements et les assemblages des fourmis sur cet arbre dépendent de la similarité entre les données. Nous comparons cet algorithme aux k-means et à l'algorithme AntClass sur des bases de données numériques artificielles et réelles, et sur des données du CE.R.I.E.S. Nous montrons que AntTree apporte des améliorations significatives au problème posé.

1 Introduction

De nombreuses applications de fouille de données sollicitent l'utilisation d'un algorithme de classification, par exemple pour réduire le nombre de données à des classes plus facilement interprétables ou encore pour découvrir des structures dans des données non étiquetées par une classe. Les problèmes de classification possèdent de multiples facettes et c'est pour cela qu'un grand nombre d'algorithmes existent suivant les cas traités.

Nous nous intéressons dans cet article à une approche particulière, l'utilisation de méthodes probabilistes inspirées de certaines propriétés du monde du vivant. Le problème du regroupement d'objets ou d'individus est en effet très présent dans la nature et de nombreuses espèces ont du développer des comportements souvent sociaux pour le résoudre. Citons l'exemple du tri du couvain chez les fourmis [Franks et al. 1992] ou encore les déplacements collectifs chez de nombreuses espèces [Camazine et al. 2001]. Dans les travaux sur les fourmis artificielles, les chercheurs tentent de trouver de nouveaux modèles pour l'apprentissage et l'optimisation [Monmarché 2001 a] inspirés des multiples comportements des fourmis réelles [Hölldobler et al. 1990]).

L'intérêt de ces modèles pour l'extraction de connaissances et la fouille de données est de produire une optimisation globale de la classification évitant les minima locaux (construction probabiliste) et dans un temps de calcul « raisonnable » pour les applications réelles, de traiter les données de manière incrémentale ou encore de ne pas nécessiter d'informations préalables (nombre de classes, classification initiale).

Nous proposons dans cet article une modélisation nouvelle qui n'a jusqu'à présent encore jamais été appliquée à la résolution de problèmes en informatique. Il s'agit de modéliser la manière dont les fourmis forment des structures vivantes [Lioni 2001] et d'utiliser ce comportement pour organiser les données selon un arbre qui se construit de manière distribuée. Intuitivement, chaque fourmi/donnée est située au départ sur un support fixe (racine de l'arbre). Le comportement des fourmis consiste alors soit à se déplacer soit à s'accrocher à la structure pour la prolonger et permettre à d'autres fourmis de venir et de s'accrocher à leur tour. Ce comportement est déterminé notamment par la similarité entre les données et la structure locale de l'arbre. Il en résulte une organisation arborescente des données dont les propriétés vont permettre de nombreuses applications : détermination automatique d'une classification « plane » (que nous utilisons à titre de comparaison avec d'autres méthodes), aperçu visuel de l'arbre, et pour nos travaux futurs, la détermination automatique d'une classification hiérarchique, la construction automatique de sites portails.

La suite de cet article est organisée comme suit : dans la section 2 nous présentons les principes du modèle biologique que nous utilisons. Dans la section 3, nous détaillons l'algorithme de construction d'arbres et les règles locales de comportement des fourmis artificielles. Nous analysons les propriétés dans la section 4 et nous présentons des résultats comparatifs sur des bases de données numériques. Nous terminons par les nombreuses perspectives qui découlent de ce travail.

2 Propriétés d'auto-assemblage chez les fourmis réelles

Dans la nature, les fourmis offrent un modèle stimulant pour le problème de la classification. L'exemple de la formation de gouttes de fourmis ou de la construction de chaînes de fourmis qui servent à relier des branches ou des feuilles entre elles [Sauwens 2000] montre que les fourmis sont capables de se connecter entre elles pour construire des structures mécaniques utiles à leur environnement. Ce phénomène d' auto-assemblage a été observé notamment chez les fourmis d'Argentine (*Linepithema humiles*) et les fourmis africaines du genre *Oecophylla longinoda*. En ce qui concerne les *L.humile* [Theraulaz et al. 2001], leur capacité à former une grappe d'ouvrières suspendues dans le vide semble être un comportement associé à une fonctionnalité méconnue jusqu'à ce jour. Cette capacité n'a été mise en évidence que récemment à partir de dispositifs expérimentaux [Sauwens 2000] : l'accrochage interindividuel y est réalisé au moyen des tarses. De plus des amas (gouttes) de fourmis peuvent se décrocher de cette grappe et chuter.

Quant aux *Oecophylla longinoda* [Lioni 2001], elles élaborent deux types de chaînes qui diffèrent selon la fonctionnalité et le mode d'accrochage. Il y a d'une part les chaînes

dites de passage permettant le franchissement d'un espace vide. L'accrochage est alors réalisé par leurs tarses. Il y a d'autre part les chaînes de construction du nid permettant le rapprochement de feuilles [Lioni 2000] : l'accrochage entre individus est ici réalisé en refermant les mandibules sur le pétiole de l'individu suivant. Pour les deux structures on observe après un certain temps la résorption de la chaîne. À partir de ce modèle de comportement d'auto-assemblage observé chez ce type de fourmis, on peut dégager un certain nombre de propriétés qui constituent le squelette de notre algorithme :

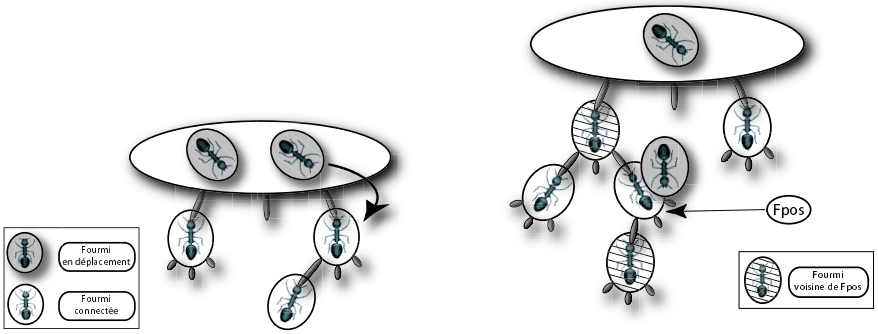
- les fourmis construisent ce type de structure vivante à partir d'un point de départ sur un support fixe (tige, feuille, ...) ;
- des fourmis peuvent se déplacer sur la structure en cours de construction, comme s'il s'agissait d'une structure fixe ;
- les fourmis peuvent se fixer a priori n'importe où sur la structure puisqu'elles peuvent atteindre tous les points qui la composent. Néanmoins, pour la formation de chaînes par exemple, les fourmis vont se fixer prioritairement sur l'extrémité de la chaîne, car elles peuvent être notamment contraintes par la pesanteur mais aussi attirées par l'objet à atteindre ;
- la majorité des fourmis qui composent la structure peuvent être bloquées sans aucune possibilité de déplacement. Par exemple dans le cas d'une chaîne de fourmis, cela correspond aux fourmis placées en milieu de chaîne ;
- un certain nombre de fourmis (généralement beaucoup plus réduit que celui du point précédent) sont reliées à la structure mais par un lien qu'elles maintiennent par elles-mêmes, et peuvent donc se détacher de la structure comme elles le veulent. Pour les chaînes de fourmis, cela correspond aux fourmis placées aux extrémités de la chaîne ;
- on observe des phénomènes de croissance mais aussi de décroissance de la structure (des fourmis qui se décrochent de la structure).

3 Algorithme AntTree

3.1 Principes généraux de AntTree

Pour obtenir une classification des données, nous allons construire un graphe (un arbre) dont les nœuds sont connus a priori et représentent les données, et dont les arcs restent à déterminer. Notons qu'il ne s'agit pas d'une classification hiérarchique au sens strict du terme puisque l'arbre construit contiendra des données à tous les nœuds, ce qui n'est pas le cas d'un dendogramme [Jain et al. 1988] où seulement les feuilles représentent les données. Néanmoins, comme nous le montrerons dans les résultats de la section 4, il est possible d'interpréter cet arbre de multiples façons, y compris de manière hiérarchique. Pour notre étude, chaque nœud représente une donnée à classer et nous considérons que nous disposons d'une mesure de similarité $Sim(i, j)$ qui prend comme paramètre un couple de nœuds i et j, et qui retourne une valeur comprise entre 0 (i et j sont totalement différents) et 1 (i et j sont identiques). Nous ne faisons pas d'autres hypothèses sur la représentation des données qui peuvent être de tous types de représentation pourvu que l'on puisse définir une mesure de similarité.

Le principe de l'algorithme AntTree est le suivant : chaque fourmi f_i, $i \in [1, n]$



(a) (b)

FIG. 1 – Construction du graphe de fourmis : principe général (a) et calcul du voisinage d'une fourmi (b)

représente un nœud du graphe à assembler, c'est-à-dire une donnée à classer. Partant d'un point de départ matérialisé par un nœud fictif f_0 qui représente le support sur lequel va être construit le graphe, les fourmis vont progressivement se fixer sur ce point initial, puis successivement sur les fourmis fixées à ce point initial, et ainsi de suite jusqu'à ce que toutes les fourmis soient rattachées à la structure. Tous ces déplacements et ces accrochages dépendent de la valeur retournée par la fonction de similarité $Sim(i,j)$ entre les données, et du voisinage local de la fourmi en déplacement (figure 1(a)). Pour chaque fourmi nous allons donc définir les notions suivantes :

- un lien sortant est un lien que f_i peut maintenir avec ses tarses, mandibules, etc, vers une autre fourmi ;
- les liens entrants de f_i sont les liens que les autres fourmis maintiennent vers f_i, ces liens peuvent être les pattes de la fourmi ;
- une information v_i portée par chaque fourmi et qui représente la donnée à classer (i-ème donnée de la base) ;
- un seuil de similarité $S_{Sim}(f_i)$ et un seuil de dissimilarité $S_{Dissim}(f_i)$ associés à chaque fourmi f_i (et mise à jour par f_i) permettant de définir si la donnée v_i est suffisamment similaire ou bien suffisamment dissimilaire à une autre donnée portée par une autre fourmi.

Donc lors de la construction de la structure, chacune des fourmis sera soit :

- **en déplacement** sur le graphe. Dans ce cas, nous considérons que chaque fourmi f_i est positionnée sur le support f_0 ou sur une autre fourmi f_{pos} mais qu'elle n'est pas assemblée/connectée à la structure. Elle est donc totalement libre de se déplacer sur le support (sa position initiale) ou bien vers une autre fourmi. Considérons également que le voisinage autour d'une fourmi f_k connectée est défini de la manière suivante : f_k et f_j sont voisines s'il existe un lien de f_k vers f_j ou un lien de f_j vers f_k. Les déplacements auront lieu aléatoirement suivant les voisins du nœud (f_{pos}) sur lequel la fourmi se trouve (voir figure 1(b)) ;
- **assemblée** à une extrémité ou au milieu du graphe. Elle ne pourra plus se dégager

de la structure. De plus nous allons considérer le fait que chaque fourmi a un seul lien sortant vers d'autres fourmis et ne peut avoir plus de $L_{\max}$ liens connectés à elle venant d'autres fourmis. Cela nous permet de construire un arbre ayant au plus $L_{\max}$ fils par nœud (voir figure 1(b)).

FIG. 2 – Algorithme principal

1. initialement toutes les fourmis sont sur le support et leurs seuils de similarité et de dissimilarité sont respectivement initialisés à 1 et à 0
2. tant que il existe une fourmi f_i non connectée faire
3. si sa position est le support alors *Cas support*
4. sinon *Cas fourmi*
5. fin tant que

FIG. 3 – Cas support

1. si aucune fourmi n'est connectée au support f_0 alors connecter f_i à f_0 /* une nouvelle classe */
2. sinon /* f_+ la fourmi connectée au support la plus similaire à f_i */
3. si $Sim(f_i, f_+) \geq S_{Sim}(f_i)$ alors déplacer f_i vers f_+
/* f_i est suffisamment similaire à f_+ */
4. sinon
5. si $Sim(f_i, f_+) < S_{Dissim}(f_i)$ alors
/* f_i est suffisamment dissimilaire à f_+ */
6. si place libre sur le support alors connecter f_i au support f_0 /* une nouvelle classe */
7. sinon diminuer $S_{Sim}(f_i)$ et déplacer f_i vers f_+
8. fin si
9. sinon diminuer $S_{Sim}(f_i)$ et augmenter $S_{Dissim}(f_i)$
de f_i /* rendre f_i plus tolérante */
10. fin si
11. fin si
12. fin si

Le détail de l'algorithme AntTree est donné dans les figures 2, 3 et 4. À chaque itération, une fourmi f_i extraite de la liste des fourmis triées (on expliquera la notion de liste de fourmis triées dans la section suivante sur les résultats) va se connecter ou bien se déplacer au mieux suivant la similarité avec son voisinage. Tant qu'il existe une fourmi f_i en déplacement, on simule une action pour f_i en fonction de sa position (sur le support ou sur une autre fourmi). Initialement, toutes les fourmis sont en

FIG. 4 – Cas fourmi

/* f_{pos} la fourmi sur laquelle f_i est en déplacement */

1. si $Sim(f_i, f_{pos}) \geq S_{Sim}(f_i)$ alors
2. si $Sim(f_i, f_+)$ ¡ $S_{Dissim}(f_i)$ alors
   /* f_+ la fourmi voisine de f_{pos} la plus similaire à f_i */
3. si place libre sur f_{pos} alors connecter f_i à f_{pos}
4. sinon déplacer f_i aléatoirement vers f_k /* f_k k=1...n, tous les éléments (fourmis ou support) connectés à f_{pos} */
5. fin si
6. sinon diminuer $S_{Dissim}(f_i)$ et augmenter $S_{Sim}(f_i)$ et déplacer f_i aléatoirement vers f_k
7. fin si
8. sinon déplacer f_i aléatoirement vers f_k
9. fin si

déplacement sur le support (nœud fictif f_0). Leurs seuils de similarité et de dissimilarité sont initialisés respectivement à 1 et à 0. La première fourmi est connectée directement au support. Ensuite pour chaque fourmi f_i, on distingue 2 cas :

- f_i est sur le support : si f_i est suffisamment similaire suivant son seuil de similarité à f_+ où f_+ est la fourmi connectée au support la plus similaire à f_i, on déplace f_i vers celle-ci de manière à la placer dans la même classe que f_+. Sinon (f_i n'est pas assez similaire à f_+), si f_i est suffisamment dissimilaire à f_+ on la connecte au support, ce qui signifie que l'on vient de construire une nouvelle classe (Figure 5). Enfin, si f_i n'est ni suffisamment similaire, ni suffisamment dissimilaire, on met à jour ses seuils ($S_{Sim}(f_i) = S_{Sim}(f_i)*0.9$ et $S_{Dissim}(f_i) = S_{Dissim}(f_i)+0.01$) pour la rendre plus tolérante et augmenter ses chances de se connecter à la prochaine itération la concernant. Entre-temps, d'autres fourmis auront de plus modifié l'arbre.
- f_i est sur une autre fourmi (notée f_{pos}) : s'il existe un lien entrant libre sur f_{pos} et que f_i est suffisamment similaire à f_{pos} et suffisamment dissimilaire aux voisines de f_{pos}, on connecte f_i à f_{pos}. Dans ce cas, f_i appartient à la même classe que f_{pos} et sa dissimilarité avec les ¡¡filles¿¿ de f_{pos} lui permettra de représenter une sous-classe la plus dissimilaire possible des autres sous-classes de f_{pos}. Sinon, on positionne f_i aléatoirement sur une des voisines de f_{pos} et on met à jour les seuils de la même façon que précédemment. f_i dans ce cas se déplace dans l'arbre et pourra trouver un meilleur endroit pour se fixer.

L'algorithme se termine lorsque toutes les fourmis sont connectées. Les différentes méthodes d'interprétation des résultats sont présentées dans la section suivante.

4 Résultats

4.1 Méthodologie de test

Afin d'évaluer les résultats obtenus par AntTree nous avons utilisés des bases de données numériques artificielles que nous avons généré (Art1, ..., Art8, où Art7 et Art8 sont des données à 1 classe générées aléatoirement), des bases réelles issues du Machine Learning Repository [Blake et al. 1998] et les données du CE.R.I.E.S. dans le domaine de la peau humaine saine [Guinot at al. 2001]. Ces bases de données sont supervisées (on connaît la classe de chaque donnée) afin de pouvoir évaluer la qualité de la classification obtenue. Cette classe n'est bien entendu pas donnée à AntTree. Nous avons utilisé la mesure d'erreur suivante. Si, pour chaque donnée v_i on connait sa classe d'origine Cr_i et la classe obtenue par AntTree C_i, l'erreur de classification Ec est calculée comme indiqué dans les équations 1 et 2. Cette erreur considère tous les couples de fourmis possible et augmente à chaque fois que deux objets n'ont pas été classés ensemble par AntTree alors qu'ils faisaient partie de la même classe d'origine et réciproquement.

$$Ec = \frac{2}{N(N-1)} \sum_{(i,j)\in\{1,\dots N\}^2, i<j} \epsilon_{ij} \tag{1}$$

où :

$$\epsilon_{ij} = \begin{cases} 0 \ si \ (C_i = C_j \wedge Cr_i = Cr_j) \vee (C_i \neq C_j \wedge Cr_i \neq Cr_j) \\ 1 \ sinon \end{cases} \tag{2}$$

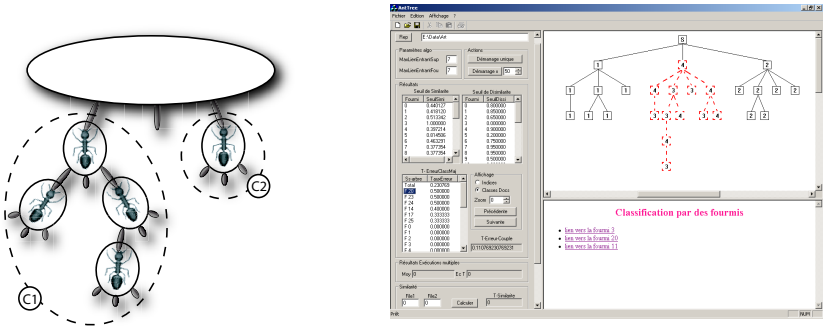
La mesure de similarité que nous utilisons pour ces données (qui sont toutes numériques afin de faciliter les comparaisons avec d'autre algorithmes) est définie comme suit avec une distance euclidienne :

$$Sim(i,j) = 1 - \sqrt{\frac{1}{M}\sum_{k=1}^{M}(v_{i_k} - v_{j_k})^2} \tag{3}$$

où v_{i_k} designe la k-ième composante normalisée de la donnée v_i. Par la suite nous avons paramétré AntTree avec $L_{Max} = 10$.

4.2 Interprétation des résultats

Les résultats de l'algorithme peuvent être interprétés de plusieurs manières différentes et complémentaires. Tout d'abord, comme cela est représenté sur la figure 5(a), les sous-arbres placés directement sous le support (f_0) constituent la classification « plane » trouvée par AntTree, chaque sous-arbre correspondant à une classe constituée de toutes les données présentes dans ce sous-arbre. Cette classification peut être comparée à d'autres obtenues par des algorithmes différents (voir section suivante). De plus, comme le montre la figure 5(b) où est représentée l'interface utilisée par AntTree, l'utilisateur



(a) (b)

FIG. 5 – Transformation de l'arbre en classe (a) et exploration de l'arbre (b)

a la possibilité de naviguer au sein de l'arbre et de l'explorer visuellement et interactivement afin de mieux comprendre l'organisation arborescente des données. Il peut cliquer sur un nœud et afficher la donnée associée. Il peut évaluer directement la taille d'une classe et la répartition des données en sous-arbres. On observe par exemple que pour une fourmi f_i donnée, les fourmis classées sous f_i lui sont effectivement similaires mais forment des sous-classes qui sont les plus dissimilaires possible entre elles. La mesure d'erreur définie précédemment peut être également indiquée pour chaque nœud de l'arbre avec des couleurs pour les sous-arbres commettant le plus d'erreur. On observe que plus on descend dans l'arbre plus cette erreur diminue ce qui confirme le fait que la précision de la classification augmente au fur et à mesure de la ramification d'une classe. L'arbre ainsi construit pourrait être également transformé en dendogramme : pour cela, il suffit de faire descendre dans l'arbre les données portées par les nœuds internes jusqu'aux feuilles de l'arbre, en les transformant en feuilles. C'est l'une des perspectives évoquées dans la conclusion.

4.3 Etude comparative

Pour notre algorithme, nous avons d'abord cherché à trouver la meilleure stratégie de tri des données de départ afin d'obtenir les meilleurs résultats possibles pour la classification. La conception de l'algorithme fait que les résultats sont influencés par la manière dont les données sont triées au départ et notamment par les premières fourmis qui vont se connecter au support.

Dans un premier temps, nous avons trié nos données de façon aléatoire. Dans un second temps, nous les avons triées par ordre croissant de la moyenne des similarités de chaque fourmi avec toutes les autres. En effectuant un tri croissant, les premières fourmis à se connecter sont celles qui sont les moins similaires à toutes les autres et donc proches de leur classe et éloignées des autres. Par contre avec un tri décroissant les premières fourmis à se connecter sont les plus similaires à toutes les autres ; ainsi une fourmi de classe différente aura plus de chance de se connecter à l'une d'elles que

Base	AntTree tri aléatoire		Nombre de
	Ec [σ_{Ec}]	C [σ_c]	Classes réelles
Art1	0.44 [0.01]	2.36 [0.08]	4
Art2	0.27 [0.01]	1.94 [0.03]	2
Art3	0.37 [0.01]	2.26 [0.08]	4
Art4	0.27 [0.00]	3.80 [0.05]	2
Art5	0.36 [0.02]	3.36 [0.09]	9
Art6	0.53 [0.02]	1.68 [0.06]	4
Art7	0.66 [0.00]	3.68 [0.06]	1
Art8	0.66 [0.01]	3.74 [0.06]	1
Iris	0.27 [0.01]	2.36 [0.08]	3
Wine	0.64 [0.00]	2.04 [0.04]	3
Glass	0.67 [0.01]	2.98 [0.07]	7
Pima	0.45 [0.00]	1.92 [0.11]	2
Soybean	0.10 [0.00]	3.90 [0.04]	4
Thyroid	0.36 [0.00]	2.76 [0.06]	3
CERIES	0.58 [0.02]	2.30 [0.11]	6

TAB. 1 – Résultats obtenus par AntTree avec un tri initial aléatoire en moyenne sur 50 essais, Ec représente l'erreur moyenne de classification et C le nombre moyen de classes trouvées, σ_{Ec} et σ_c représentent les écarts types respectifs

dans le cas croissant. Les résultats obtenus sont indiqués dans les tableaux 1 et 2. On constate de manière évidente que l'erreur de classification est moins importante dans la majorité des cas pour un tri croissant. Nous avons par conséquent choisi d'adopter la stratégie du tri croissant dans la suite de nos tests.

Nous avons ensuite comparé AntTree avec d'autres algorithmes de classification : AntClass [Monmarché 2000], un algorithme de classification non supervisée par une colonie de fourmis artificielles et l'algorithme K-means initialisé à 10 classes de départ générées aléatoirement (les données utilisées pour l'expérimentation ne contiennent pas plus de 10 classes). La table 3 montre les résultats obtenus pour les 10-means, AntClass et AntTree sur les données artificielles et les bases réelles. Sur 50 essais, C représente le nombre moyen de classes obtenues, Ec l'erreur de classification moyenne, σ_c, $\sigma_E c$ représentent les écart-types respectifs et Cr le nombre de classes réelles. On constate que l'algorithme AntTree donne une erreur moyenne plus basse que l'algorithme AntClass [Monmarché 2000] pour Art2, Art3, Art4, Art8, pima et soybean, et une erreur moyenne quasiment similaire pour Art1, glass, thyroïd. De plus, pour la majorité des bases, le nombre de classe trouvée par AntTree est plus proche du nombre de classes réelles que celui trouvé par l'algorithme AntClass [Monmarché 2000] (10 bases sur 15).

AntTree est également meilleur que l'algorithme 10-means pour Art2, Art3, Art4, Art7, Art8, pima, soybean et thyroid. De plus, le nombre de classes trouvées par AntTree est également meilleur (14 bases sur 15) pour ces seconds résultat. On remarque aussi que AntTree est plus précis que l'algorithme AntClass [Monmarché 2000] et la méthode 10-means : la majorité des écarts types de la moyenne des classes trouvées et de l'erreur moyenne sont tous à 0.

Base	AntTree tri décroissant		AntTree tri croissant		Nombre de
	Ec [σ_{Ec}]	C [σ_c]	Ec [σ_{Ec}]	C [σ_c]	classes réelles
Art1	0.75 [0.00]	1 [0.00]	0.17 [0.00]	4 [0.00]	4
Art2	0.50 [0.00]	1 [0.00]	0.18 [0.00]	3 [0.00]	2
Art3	0.58 [0.00]	1 [0.00]	0.21 [0.00]	3 [0.00]	4
Art4	0.43 [0.00]	3 [0.00]	0.25 [0.00]	4 [0.00]	2
Art5	0.36 [0.00]	2 [0.00]	0.20 [0.00]	4 [0.00]	9
Art6	0.53 [0.00]	1 [0.00]	0.34 [0.00]	2 [0.00]	4
Art7	0.54 [0.00]	4 [0.00]	0.72 [0.00]	4 [0.00]	1
Art8	0.61 [0.00]	5 [0.00]	0.76 [0.00]	5 [0.00]	1
Iris	0.67 [0.00]	1 [0.00]	0.24 [0.00]	4 [0.00]	3
Wine	0.65 [0.00]	2 [0.00]	0.64 [0.00]	2 [0.00]	3
Glass	0.71 [0.00]	3 [0.00]	0.42 [0.00]	5 [0.00]	7
Pima	0.45 [0.00]	1 [0.00]	0.42 [0.00]	3 [0.00]	2
Soybean	0.15 [0.00]	3 [0.00]	0.08 [0.00]	4 [0.00]	4
Thyroid	0.38 [0.00]	3 [0.00]	0.24 [0.00]	3 [0.00]	3
CERIES	0.76 [0.00]	2 [0.00]	0.33 [0.00]	4 [0.00]	6

TAB. 2 – Résultats obtenus par AntTree avec un tri initial décroissant et un tri initial croissant

Base	10-Means		AntClass		Nombre de
	Ec [σ_{Ec}]	C [σ_c]	Ec [σ_{Ec}]	C [σ_c]	classes réelles
Art1	0.18 [0.01]	8.58 [0.98]	0.15 [0.05]	4.22 [1.15]	4
Art2	0.38 [0.01]	8.52 [0.96]	0.41 [0.01]	12.32 [2.01]	2
Art3	0.31 [0.01]	8.28 [0.96]	0.35 [0.01]	14.66 [2.68]	4
Art4	0.32 [0.02]	6.38 [0.75]	0.29 [0.23]	1.68 [0.84]	2
Art5	0.08 [0.01]	8.82 [0.91]	0.08 [0.01]	11.36 [1.94]	9
Art6	0.10 [0.02]	8.46 [1.08]	0.11 [0.13]	3.74 [1.38]	4
Art7	0.87 [0.02]	7.76 [1.03]	0.17 [0.24]	1.38 [0.60]	1
Art8	0.88 [0.01]	8.78 [0.83]	0.92 [0.01]	13.06 [2.18]	1
Iris	0.18 [0.03]	7.12 [1.11]	0.19 [0.08]	3.52 [1.39]	3
wine	0.27 [0.01]	9.64 [0.52]	0.51 [0.11]	6.46 [2.10]	3
Glass	0.29 [0.02]	9.44 [0.70]	0.40 [0.06]	5.60 [2.01]	7
Pima	0.50 [0.01]	9.90 [0.36]	0.47 [0.02]	6.10 [1.84]	2
Soybean	0.13 [0.02]	8.82 [0.97]	0.54 [0.17]	1.60 [0.49]	4
Thyroid	0.42 [0.02]	9.56 [0.57]	0.22 [0.09]	5.84 [1.33]	3
CERIES	0.11 [0.01]	9.38 [0.63]	0.27 [0.15]	3.40 [1.06]	6

TAB. 3 – Résultats obtenus par les 10-means et AntClass

Les temps d'exécution pour une classification vont en moyenne d'1ms (base thyroid) à 0,5s (base Art3 composée de 1100 données). 10-means et AntClass ont été programmés en *C*, AntTree en C++ et les tests ont été réalisés sur un PC (PIII 700 MHz). Les durées d'exécution moyennes sont relativement basses par rapport aux deux méthodes car dès qu'une fourmi est connectée, on ne cherche plus à la déplacer.

5 Conclusion

Dans cet article nous avons décrit une nouvelle approche fondée sur les colonies de fourmis et son application au problème de classification non supervisée. Nous avons comparé notre algorithme avec la méthode des k-means (initialisé à 10 partitions) et l'algorithme AntClass [Monmarché 2000]. Les résultats obtenus sont très satisfaisants et font de notre algorithme un outil intéressant capable de résoudre des problèmes de classification. De plus notre approche ne dépend pas du type de données à classer : il suffit uniquement de définir une fonction de similarité pour ces nouvelles données. Comme perspective nous voulons implémenter le décrochage des fourmis (gouttes de fourmis pour les *L. humiles* et la résorption de chaînes pour les *Oecophylla longinoda*. Chaque fourmi aura donc la possibilité de se déconnecter de sa position et de se déplacer vers d'autres fourmis peut-être plus similaires que celle sur laquelle elle se trouve. On s'intéresse aussi à une approche probabilisée de l'algorithme : à chaque fourmi on associe une probabilité de se connecter à sa position qui dépend de sa similarité avec les fourmis qui l'entourent. Nous comptons aussi appliquer cet algorithme à la construction automatique d'une hiérarchie de pages web pour la construction automatique d'un site portail. Il reste beaucoup de problèmes à résoudre pour ce type d'approche et les résultats obtenus jusqu'ici sont plus qu'encourageants.

Références

[Blake et al. 1998] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[Camazine et al. 2001] Scott Camazine, Jean-Louis Deneubourg, Nigel R Franks, James Sneyd, Guy Theraulaz, and Eric Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[Franks et al. 1992] N-R Franks and A Sendova-Franks. Brood sorting by ants : distributing the workload over the work surface. *Behav. Ecol. Sociobiol*, 30 :109–123, 1992.

[Guinot at al. 2001] C Guinot, D Malvy, J Latreille, M Tenenhaus, F Morizot, S Lopez, I Le Fur, L Dubertret, and E Tschachler. Recherche d'une classification de la peau du visage humaine saine à l'aide de la méthode de classification hiérarchique ascendante. In *VIIèmes Rencontres de la Société Francophone de Classification (SFC)*, 2001.

[Hölldobler et al. 1990] B Hölldobler and E-O Wilson. *The Ants*. Springer Verlag, Berlin, 1990.

[Jain et al. 1988] A-K Jain and R-C Dubes. *Algorithms for Clustering Data.* Prentice Hall Advanced Reference Series, 1988.

[Lioni 2000] Arnaud Lioni. *Auto-assemblage et transport collectif chez oecophylla.* Thèse de doctorat, Université libre de bruxelles, Université Paul Sabatier, 2000.

[Lioni 2001] Arnaud Lioni, Christian Sauwens, Guy Theraulaz, and J-L Deneubourg. The dynamics of chain formation in oecophylla longinoda. *Journal of Insect Behavior*, 14 :679–696, 2001.

[Monmarché 2000] Nicolas Monmarché. *Algorithme de fourmis artificielles : applications à la classification et à l'optimisation.* Thèse de doctorat, Université de Tours, 2000.

[Monmarché 2001 a] Nicolas Monmarché, Mohamed Slimane, and Gilles 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.

[Sauwens 2000] Christian Sauwens. *Étude de la dynamique d'auto-assemblage chez plusieurs espèces de fourmis.* Thèse de doctorat, Université libre de bruxelles, 2000.

[Theraulaz et al. 2001] Guy Theraulaz, E Bonabeau, Christian Sauwens, J-L Deneubourg, Arnaud Lioni, F Libert, L Passera, and R-V Solé. Model of droplet formation and dynamics in the argentine ant (linepithema humile mayr). *Bulletin of Mathematical Biology*, 63 :1079–1093, 2001.

Summary

We present in this paper a new clustering algorithm for unsupervised learning. It is inspired from the self-assembling behavior observed in real ants where ants progressively get attached to an existing support and successively to other attached ants. The artificial ants that we have defined will similarly build a tree. Each ant represents one data. The way ants move and build this tree depends on the similarity between the data. We have compared our results to those obtained by the k-means algorithm and by AntClass on numerical databases (either artificial, real, and from the CE.R.I.E.S.). We show that AntTree significatively improves the clustering process.