Un Algoritmo ACO para el Problema de Clustering

Mariela Flores y Patricia Soria

Facultad de Ciencias Exactas, Universidad Nacional de Salta
Salta, CP 4400, Argentina

Abstract

Clustering Problem consists of partition of a data set into groups so that they contain similar objects each of them and they are also different among them.
Ant Colony Optimization is based on behaviour of real ant colonies to solve optimization problems.
In this work we present an ACO algorithm that is applied to a numeric data set to obtain the best possible clustering. Our proposal which is based on AntTree algorithm it was tested with different test instances reaching efficient results.

Keywords: Ant Colony Optimization, Clustering, Metaheuristics

Resumen

El Problema de Clustering consiste en la división de datos en grupos de objetos similares entre sí y diferentes a los demás contenidos en otros grupos.
Ant Colony Optimization (ACO) se inspira en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización.
Presentamos un algoritmo ACO aplicado sobre un conjunto de datos numérico para encontrar el mejor agrupamiento posible. Nuestra propuesta se basa en el algoritmo AntTree aplicado para resolver problemas de Clustering. Las pruebas sobre colecciones de datos de la literatura, alcanzaron excelentes resultados

Palabras Claves: Colonia de Hormigas, Clustering, Metaheurísticas

1 Introducción

En términos generales, podemos definir Cluster como un conjunto de objetos similares (Vicente et al. 2005). La forma de medir la similitud entre objetos varía de acuerdo a la naturaleza de sus datos.
Existen muchos algoritmos para resolver problemas de Clustering tales como K-means, Average Linkage, Single Linkage (Rui Xu y Wunsch 2005).
En este trabajo desarrollamos un algoritmo heurístico para resolver el problema de Clustering Jerárquico basado en el algoritmo AntTree (Azzag et al. 2003). Dicho algoritmo agrupa en forma progresiva a las hormigas para formar un "árbol jerárquico", sobre el cual las hormigas se mueven hasta lograr su mejor posición.

XXXIV Conferencia Latinoamericana de Informática (CLEI 2008), pp. 122-131, 2008.

El presente trabajo tiene la siguiente estructura: en la Sección 2 se hace una breve revisión de los métodos de Clustering. En la Sección 3 se describe la metaheurística ACO. La Sección 4 describe el algoritmo AntTree (AT) y AntTree-Extended (ATE), en la Sección 5. Los resultados obtenidos se muestran en la Sección 6 y finalmente las conclusiones y el trabajo futuro, en la Sección 7.

2 Clustering

El Problema de Clustering tiene múltiples aplicaciones dentro de las Ciencias de la Computación, donde se buscan obtener grupos de datos con ciertas características de interés. Por ejemplo, en el descubrimiento de segmentos de clientes con el fin de mejorar los servicios que brinda una empresa (Fayyad et al. 1996), en la comprensión de imágenes (Scheunders 1996) a través de la división de la misma en regiones homogéneas, entre otras.
Dado un conjunto de objetos, el proceso de Clustering consiste en encontrar grupos cuyos objetos sean similares entre sí. Para medir la similaridad entre objetos, se suelen utilizar diferentes medidas de distancia: Euclideana, de Manhattan, de Mahalanobis, etc (Rui Xu y Wunsch 2005). La representación de los datos mediante una serie de clusters conlleva a la pérdida de detalle, pero consigue la simplificación de los mismos.
Los métodos de Clustering existentes difieren uno del otro en la forma de agrupar los datos. Aquellos que encuentran clusters que corresponden a una partición del conjunto de objetos se los conoce como métodos de Hard-Clustering, siendo el más conocido el algoritmo K-Means. Los métodos que asignan a cada objeto un valor de pertenencia con respecto a cada cluster se los conoce como métodos de Soft-Clustering y entre los más representativos de este tipo de Clustering se encuentran los algoritmos Fuzzy C-Means y Expectación Maximización.
El algoritmo propuesto se considera dentro de las técnicas de Hard-Clustering porque ubica los datos en una estructura jerárquica.

3 Ant Colony Optimization

ACO es un framework general que puede aplicarse a diversos problemas de optimización con pocos cambios significativos si ya existe previamente algún método heurístico específico para el problema.
Se basa en una colonia de hormigas artificiales, esto es, agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales. Los algoritmos ACO son esencialmente constructivos (Dorigo et al. 1999). En cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de información que guían el movimiento de la hormiga:

- *Información heurística*, que mide la preferencia heurística de moverse desde un nodo a otro. Las hormigas no modifican esta información.

– *Información de los rastros de feromona*, que mide la "deseabilidad aprendida"del movimiento del nodo en que se encuentra hacia un próximo. Imita a la feromona real que depositan las hormigas. Se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas.

4 El Algoritmo AntTree

Está basado en el comportamiento de las hormigas reales (Azzag et al. 2003). Construye de forma incremental una estructura representando una organización jerárquica del conjunto de datos. Cada hormiga representa un dato y ésta se mueve en la estructura de acuerdo a una similitud con otra ya conectada. Para obtener una partición del conjunto de datos, se construye un árbol donde los nodos representan datos y donde cada hormiga inicialmente está ubicada en un lugar fijo llamado soporte.

AntTree utiliza una función de similitud Sim entre un par de elementos, es decir:
N: cardinal del conjunto de datos

(d_i,d_j): par arbitrario de datos (i,j) $\in$ [1,N]

Sim(i,j) $\in$ [0,1]

Si Sim(i,j) $\longrightarrow$ 0 $\Rightarrow$ d_i y d_j son completamente diferentes

Si Sim(i,j) $\longrightarrow$ 1 $\Rightarrow$ d_i y d_j son son bastantes similares

El algoritmo se describe de la siguiente manera: cada hormiga representa un nodo a ser conectado al árbol. Es decir, un dato a clasificar. Todas las hormigas inicialmente se encuentran en el soporte (raíz del árbol) el cual es un nodo artificial denotado por a_0. Las hormigas gradualmente se fijarán por sí mismas a este nodo, o a cualquier otra hormiga ya conectada. Este proceso continúa hasta que todas las hormigas son conectadas a la estructura, o sea, hasta que todos los datos estén ensamblados en el árbol. Todos los movimientos y la estructura final dependen de Sim(i,j). Cada hormiga a_i tendrá asociado los siguientes términos:

– Link saliente: indica la hormiga anterior inmediata a ella (nodo padre).
– Link entrante: representa el conjunto de enlaces que mantienen otras hormigas hacia a_i (nodos hijos).
– d_i: corresponde a un dato representado por la hormiga.
– TSim(a_i) y TDissim(a_i): umbrales de Similitud y Disimilitud. Estos son actualizados durante el proceso de construcción del árbol.

Mientras se construye la estructura, cada hormiga a_i se encontrará:

– Moviéndose en el árbol: una hormiga se moverá sobre el soporte a_0 ó sobre otra hormiga. En ambos casos no está conectada todavía en la estructura. Por lo tanto, es libre de ubicarse sobre el soporte o sobre otras. Si a_{pos} indica donde se encuentra la hormiga actualmente, entonces puede moverse hacia alguna vecina de a_{pos}.

– Conectada al árbol: en este caso, la hormiga a_i tiene asignado un valor para el nodo padre. Esto produce que no pueda moverse más. Por otra parte, una hormiga no puede tener más de L_{max} enlaces entrantes.

La subsección siguiente hace una descripción más detallada.

4.1 Descripción de AntTree

El primer paso coloca todas las hormigas sobre el soporte del árbol e inicializa sus umbrales. El algoritmo principal se muestra en la Fig 1. En esta etapa, la colonia de hormigas se representa mediante una lista L. Durante el proceso de construcción, cada hormiga a_i puede conectarse al soporte (o a otra hormiga) o moverse buscando un lugar correcto al cual conectarse. Este proceso continúa hasta que todas hayan encontrado el lugar más adecuado donde posicionarse.

```
Sea L la lista ordenada de hormigas esperando a ser conectadas
Sea Soporte la lista de hormigas que se encuentran en el soporte
TSim(a_i)=1, ∀ i=1...n /*umbral de similitud para cada hormiga a_i*/
TDissim(a_i)=0, ∀ i=1...n /*umbral de disimilitud para cada hormiga a_i*/
Mientras L≠[ ]: /*mientras existan hormigas en la lista L */
      Seleccionar a_i/a_i ∈ L
      Si a_i esta en el Soporte:
            a) Caso Soporte (Fig 2)
            b) Caso Hormiga (Fig 3)
```

Fig. 1. Algoritmo AntTree

En la Fig. 1 de acuerdo a la posición de a_i, se pueden realizar dos acciones. La situación de la primera hormiga de L es la más sencilla: siempre es directamente conectada al soporte.
Para las demás, hay dos casos a considerar. El primero es cuando a_i está en el soporte; se compara con a^+ que es la hormiga más similar a a_i de entre las que están directamente conectadas al soporte. Si a^+ es bastante similar a a_i (de acuerdo a lo que permite el umbral) entonces a_i será colocada sobre ella. En caso de que sean bastantes disimilares (de acuerdo al umbral de disimilitud) a_i es conectada directamente al soporte, generando un nuevo subárbol o sea un nuevo cluster debido a que la nueva hormiga conectada es bastante disimilar a las que estaban anteriormente conectadas. Finalmente, si a_i no es ni lo suficientemente similar ni disimilar, sus umbrales son actualizados de la siguiente manera:

$$TSim(a_i) = TSim(a_i) * 0.9 \tag{1}$$

$$TDissim(a_i) = TDissim(a_i) + 0.01 \tag{2}$$

```
Si no hay hormigas conectadas al Soporte:
    conectar a_i a a_0
Sino
    Tomar a^+ que sea la más similar a a_i de las conectadas a a_0
    a) Si Sim(a_i,a^+)≥TSim(a_i):
        Mover a_i hacia a^+
    b) Sino:
        i) Si Sim(a_i,a^+)<TDissim(a_i):/* a_i es bastante disimilar a a^+ */
            Si Links Entrantes de a_0<L_max:
                conectar a_i a a_0
            Sino
                mover a_i hacia a^+; TSim(a_i)= TSim(a_i)*0.9
        ii) Sino
            TSim(a_i)=TSim(a_i)*0.9;TDissim(a_i)=TDissim(a_i)+0.01
            /* a_i es mas tolerante */
```

Fig. 2. Caso Soporte

La actualización de los umbrales permite que a_i sea más tolerante. Es decir, se incrementa la posibilidad que a_i sea conectada en las próximas iteraciones.
El segundo caso a considerar es cuando a_i está sobre la hormiga a_{pos} (Fig. 3). Si a_i es bastante similar a a_{pos}, bastante disimilar a las otras conectadas a a_{pos} y además existen enlaces entrantes disponibles, entonces a_i se conecta a a_{pos}. Es decir, a_i será raíz de un subárbol conectado a a_{pos}. La principal diferencia entre a_i con las demás hormigas conectadas a a_{pos} es que representará un nuevo subgrupo similar al representado por a_{pos} y disimilar a los otros que cuelgan de a_{pos}. Si no existe un enlace disponible para que a_i sea conectada a a_{pos}, se moverá hacia una hormiga vecina elegida al azar, a_k. Si a_i es bastante similar a a_{pos} pero no es tan disimilar a las conectadas a ella, los umbrales son modificados dando más libertad a a_i para ubicarse sobre cualquier a_k. En caso que a_i y a_{pos} no sean similares, se trasladará hacia a_k.
El algoritmo finaliza cuando todas las hormigas están conectadas a la estructura. Finalmente, debemos mencionar que L_{max}=10 y que el orden en que se toman las hormigas desde la lista L afectará a la calidad de los resultados.

Sea a_{pos} la hormiga en la cual a_i está posicionada
Sea a_k una hormiga elegida al azar de las vecinas de a_{pos}
1) Si Sim(a_i,a_{pos})≥TSim(a_i):
    Sea a^+ la hormiga mas similar de las conectadas a a_{pos}
    i) Si Sim(a_i,a^+)<TDissim(a_i):
        Si Links Entrantes de a_{pos}<L_{max}:
            conectar a_i a a_{pos}
        Sino
            mover aleatoriamente hacia a_k
    ii)Sino:
        TSim(a_i)= TSim(a_i)*0.9; TDissim(a_i)= TDissim(a_i)+ 0.01
        mover a_i hacia a_k /* a_i es mas tolerante */
2) Sino
    mover a_i hacia a_k

Fig. 3. Caso Hormiga

5 AntTreeExtended

Es una mejora de AntTree que permite a las hormigas moverse de un grupo a otro. Este comportamiento (no incluído en AT), mejoró los resultados obtenidos según pruebas realizadas. ATE busca grupos que cuenten con dos características:

1. Ejecución de AT.
2. Búsqueda de hormigas ubicadas en grupos inapropiados.
   (a) Búsqueda del grupo en el cual la hormiga será reubicada.
   (b) Ordenar la lista de dichas hormigas.
3. Reubicación de las hormigas.

Fig. 4. Esquema del algoritmo propuesto (ATE)

- Grupos con elementos con alta cohesión[1].
- Grupos "bien definidos"o disímiles.

Esto es logrado ejecutando AntTree, el cual posee una modificación en la actualización del umbral de disimilitud[2]:

$$TDissim(a_i) = TDissim(a_i) + \alpha^3 \quad (3)$$

[1] cohesión: medida de la intensidad con la que los miembros de un grupo están unidos.
[2] El umbral de similitud TSim no ha sido modificado.
[3] Ver Sección 5.1.

Además el factor de bifurcación es modificado a L_{max}=15, permitiendo que el número de grupos generados sea mayor que el obtenido con AT, lo cual resultó más conveniente según pruebas efectuadas.
La última modificación en AT es en el Caso Hormiga (ver Sección 4), cuando se evalúa si a_i es bastante similar a a_{pos} (más de lo que pide su umbral de similitud) y no es tan disímil a las demás conectadas a a_{pos}, entonces esta hormiga no es movida hacia alguna a_k elegida al azar tal como propone AT (Azzag et al. 2003), sino que es ubicada sobre a^+ (de las conectadas a a_{pos}, la más idéntica a a_i) por lo tanto a_i en las futuras iteraciones tendrá más posibilidad de encontrar un grupo correcto, eliminando así la aleatoriedad originalmente propuesta.
En el *paso 2* de Fig. 4, se examina cada hormiga de las conectadas al árbol buscando para cada una, un grupo más adecuado para su reubicación. Esto termina cuando todas las hormigas han sido analizadas. En el *paso 2.a* cada hormiga en la estructura (menos las que están directamente conectadas al soporte), es evaluada para decidir si debe ser desconectada o no. Tal decisión se basa en la función Silhouette (Bolshakova y Azuaje 2003) que estima la cohesión que tiene un dato respecto al grupo al cual pertenece y se define:

$$s(i) = (b(i) - a(i))/max\{a(i), b(i)\} \quad siendo \ -1 \leq s(i) \leq 1 \tag{4}$$

Donde:
a(i): es la distancia entre el dato a_i y la media aritmética de su grupo.
b(i): es la distancia mínima entre a_i y la media de los grupos restantes.
Una vez calculado s(i) para cada hormiga y dado un umbral t=t_0 [4] se determina si la asignación a su grupo actual es correcta. Consideramos que está bien asignada si s(i)$\leq$ t. Si esto no ocurre, significa que existe un "mejor grupo"[5] para la hormiga a_i por lo que debería ser desconectada del árbol. Una vez extraída del árbol, es agregada a una lista L para su posterior re-asignación. Para el *paso 2.b*, el criterio de orden de la lista L es decreciente según el promedio de similitud de una hormiga a_i con las restantes. En este caso, el criterio de orden de la lista es totalmente opuesto al usado por AT. Por lo tanto el primer lugar en la lista L estará ocupado por la hormiga más similar a las restantes. Si usamos este orden, la hormiga más similar de la lista a las demás tendrá una importante influencia en el valor medio de los grupos a los cuales han sido asignadas y aquellas hormigas las que son más similares a ella serán asignadas al mismo grupo.
La reubicación de las hormigas en el *paso 3* es realizado con el movimiento de la hormiga hacia el nuevo grupo encontrado en el paso anterior. Por lo tanto a_i se mueve hacia la hormiga conectada al soporte que representa dicho grupo.

6 Evaluación de ATE

A continuación, se definen las instancias usadas, las funciones necesarias para la ejecución y evaluación de resultados. Y por último los resultados de las pruebas.

[4] t=0, si s(i)< t_0 entonces a_i debe ser reubicada.
[5] Es aquel grupo al cual la distancia entre a_i y su valor medio es menor que la distancia entre la media de su actual grupo y ella misma.

6.1 Colecciones de datos

Las instancias de prueba [6] que se usaron son las siguientes:

- Iris: contiene información sobre plantas de lirio.
- Glass: conformada por instancias de diferentes tipos de vidrios.
- Wine: registros de vinos estacionados de tres cosechas diferentes.
- Thryroid: información de pacientes con eurotiroidismo, hipertiroide o hipotiroide.
- Pima: datos de pacientes femeninas con diabetes de la comunidad Pima.
- Wisconsin: información de pacientes para detección de cáncer de mama.
- Bupa: registros de pacientes masculinos con trastornos de hígado.
- Heart: datos de pacientes con problemas cardíacos.

Table 1. Colecciones de datos

Nombre	NI	NA	K	DI
Iris	150	4	3	(50,50,50)
Glass	214	9	7	(70,76,17,13,9,29)
Wine	178	13	3	(59,71,48)
Thyroid	215	5	3	(150,35,30)
Pima	768	8	2	(500,268)
Wisconsin	569	30	2	(357,212)
Bupa	345	6	2	(145,200)
Heart	270	13	2	(150,120)

Table 1 muestra para cada una, el número de instancias (NI), atributos (NA), la cantidad de clases (K) y las distribuciones de las instancias en cada clase (DI).

6.2 Función de Error

Una manera de evaluar la clasificación de los datos es mediante la minimización de la función Ec, la cual se define como:

$$Ec = \frac{2}{N(N-1)} \sum_{i=1}^{n} \sum_{j=1}^{n} \epsilon_{ij} \tag{5}$$

$$\epsilon_{ij} = \begin{cases} 0 \text{ si } (k_i{=}k_j \text{ y } k_i^{'}{=}k_j^{'}) \text{ o}(k_i \neq k_j \text{ y } k_i^{'} \neq k_j^{'}) \\ 1 \text{ c.o.c} \end{cases} \tag{6}$$

[6] disponible en http://archive.ics.uci.edu/ml/datasets.html

k_i: representa la clase real a la cual pertenece el objeto d_i.
$k_i^{'}$: es la clase encontrada por el algoritmo a la cual pertenece d_i.
K: corresponde al número real de clases.
$K^{'}$: es el número de clases encontrado por el método.

6.3 Función de Similitud

Para determinar la distancia entre los datos, se usó la función de similitud:

$$Sim(i,j) = 1 - \sqrt{\frac{1}{M}\sum_{k=1}^{M}(v_{ik} - v_{jk})^2} \tag{7}$$

M: representa la cantidad de atributos usados para describir a cada dato.
v_{ik}: es el valor del k-ésimo atributo para el i-ésimo dato.

6.4 Calibración del parámetro α

Según análisis de resultados realizados sobre pruebas, se pudo determinar que en instancias con baja dispersión de datos, el valor propuesto en (Azzag et al. 2003) es inadecuado debido al lento crecimiento de α ocasionando un alto tiempo de ejecución para que las hormigas puedan conectarse a algún grupo.
Por esta razón y según corridas previas de ATE, el rango de α que brindó mejores resultados varía entre 0.015 y 0.25, siendo conveniente valores altos para instancias con baja dispersión (Iris, Glass) y valores pequeños de α para instancias con mayor dispersión (Pima).

6.5 Resultados Obtenidos

A continuación, se muestran los resultados alcanzados por ATE, el cual fue programado en Python, los obtenidos por AT en (Azzag et al. 2003), y los devueltos por K-Means (no jerárquico), Average Linkage, Complete Linkage y Single Linkage (jerárquicos)[7]. Las pruebas de ATE se realizaron en una PC Pentium III y los resultados corresponden a valores promedio de 10 corridas por instancia.
En Table 2 se muestran para cada instancia el valor de la función de error (Ec) y la cantidad de clusters obtenidos ($K^{'}$) por cada uno de los algoritmos. El valor de α para la instancia Pima, Bupa y Heart es de 0.015 y para las restantes es de 0.2.
De las pruebas se observa que los resultados obtenidos por ATE fueron mejores en la mayoría de las instancias. Los valores de Ec obtenidos por ATE fueron menores que los de AT en 4 de las 5 instancias utilizadas en (Azzag et al. 2003). Para instancias con poca dispersión, ATE igualmente logra el número real de grupos para las 5 instancias, lo cual no ocurrió con AT.
Es importante destacar que al igual que AT, ATE no necesita información previa de los grupos a obtener para proceder con la clasificación de datos. Las pruebas muestran que los resultados son similares o mejores que los de los otros algoritmos para la métrica aplicada en este trabajo.
Finalmente, el tiempo computacional promedio de ATE varió entre 0.1 y 2 segs.

[7] Algoritmos ejecutados con el software SPSS 11.5

Table 2. Resultados por Instancias

Intancias	AT		ATE		K-Means		Avg.Linkage		S.Linkage		C.Linkage	
	Ec	$K^{'}$	Ec	$K^{'}$	Ec	$K^{'}$	Ec	$K^{'}$	Ec	$K^{'}$	Ec	$K^{'}$
Iris	0.24	3	0.11	3	0.13	3	0.14	3	0.24	3	0.15	3
Glass	0.42	5	0.34	7	0.40	7	0.44	7	0.69	7	0.37	7
Wine	0.64	2	0.43	3	0.26	3	0.61	3	0.61	3	0.31	3
Thyroid	0.24	3	0.15	3	0.33	3	0.41	3	0.43	3	0.23	3
Pima	0.42	3	0.46	2	0.46	2	0.45	2	0.45	2	0.45	2
Wisconsin	-	-	0.44	3	0.27	2	0.46	2	0.46	2	0.32	2
Bupa	-	-	0.49	2	0.49	2	0.49	2	0.49	2	0.49	2
Heart	-	-	0.47	2	0.48	2	0.33	2	0.49	2	0.49	2

7 Conclusión y Trabajo futuro

De acuerdo a las pruebas, AntTreeExtended obtuvo un menor error en la clasificación de datos sobre las instancias probadas que el algoritmo original. Además, obtuvo el número correcto de clusters para cada instancia.
Actualmente, se está aplicando ATE para clasificar páginas de Internet devueltas por algún buscador como resultado de una búsqueda solicitada, permitiendo así un acceso a la información más rápido y preciso.

References

Azzag, H., Monmarche, N., Slimane, M., Venturini, G.: AntTree: A New Model for Clustering with Artificial Ants. Proceedings of the CEC2003. 4, 2642-2647 (2003).

Bolshakova, N., Azuaje, F.: Improving expression data mining through cluster validation. 4th IEEE EMBS Special Topic Conference on Information Technology Applications in Biomedicine. 19-22 (2003).

Dorigo, M., Di Caro, G.: Ant Algorithms for Discrete Optimization. Artificial Life. 5 (2), 137-172 (1999).

Fayyad, U., Piatetsky-Shapiro, G., Padhraic S.: From data Mining to Knowledge Discovery in Databases. AI Magazine. 17 (3), 37-54 (1996).

Rui Xu, Wunsch, D.: Survey of Clustering Algorithms. IEEE Transactions On Neural Networks. 16 (3), 645-678 (2005).

Scheunders, P.: A genetic Lloyd-Max image quantization algorithm. Pattern Recognition Letters. 17 (5), 547-556 (1996).

Vicente, E., Rivera, L., Mauricio, D.: Grasp en la Resolución del Problema de Clustering. RISI. 2 (2), 16-25 (2005).