UNIVERSIDAD DE GRANADA

E.T.S.I. INFORMÁTICA Y TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Metaheurísticas

Práctica 4.b:
Optimización basada en Colonias de Hormigas para el Problema del Clustering

Curso 2012-2013

Tercer Curso del Grado en Ingeniería Informática

Alumno: José Arcos Aneas
D.N.I: 74740565-H
Correo electrónico: joseaa@correo.ugr.es

ÍNDICE

0.Introducción.

En la introducción describo como se organiza esta practica.

El apartado 1 contiene la descripción del problema del clustering. Es una descripción general.
El siguiente apartado describe de forma general los algoritmos basados en hormigas y describe con un poco de mayor profundidad los algoritmos en los que se centra la práctica.
El apartado 3 informa sobre los métodos mas importantes y la estructuración de los componente necesarios para aplicar los algoritmos de hormigas al problema del clustering. Este apartado se describen los pseudocódigos de los métodos empleados para realizar la práctica.
El ultimo aparta es una breve comparación entre los métodos de hormigas y el k-medias de la practica P1.

1. Descripción del Problema Del Clustering.

El problema que abordamos es el del clustering. El clustering puede definirse como el proceso de agrupar un conjunto de objetos físicos o abstractos dentro de clases con objetos similares se denomina clustering. El clustering consiste en agrupar una colección dada de datos no etiquetados en un conjunto de grupos de tal manera que los objetos que pertenecen a un grupo sean homogéneos entre sí y buscando además, que la heterogeneidad entre los distintos grupos la sea lo más elevada posible . Podríamos ver un problema de clustering como un problema de optimizaron que localiza los centroides óptimos de los clusters en lugar de encontrar la partición del espacio, cayendo en óptimos locales.

El problema que abordamos es el del clustering. El clustering puede definirse como el proceso de agrupar un conjunto de objetos físicos o abstractos dentro de clases con objetos similares se denomina clustering. El clustering consiste en agrupar una colección dada de datos no etiquetados en un conjunto de grupos de tal manera que los objetos que pertenecen a un grupo sean homogéneos entre sí y buscando además, que la heterogeneidad entre los distintos grupos la sea lo más elevada posible . Podríamos ver un problema de clustering como un problema de optimizaron que localiza los centroides óptimos de los clusters en lugar de encontrar la partición del espacio, cayendo en óptimos locales.

En la configuración de los clúster cada objeto permanece a un único clúster, y su distancia al centroide de éste es siempre menor que a los restantes. Así, suponiendo un conjunto M formado por m objetos y configuración de k clústeres C_i, debe cumplirse:

a) $C_i \neq \varnothing$, es decir, $|C_i| > 0$.
b) $C_i \cap C_j = \varnothing$, para todo $i \neq j$.
c) $\sum_{i=1}^{k} C_i = M$, donde:
   - M = $\{X^1, \ldots, X^m\}$ es el conjunto de patrones
   - $X^i = (x_1^i, \ldots, x_n^i)$, con $x_j^i \in R$.

En el algoritmo que implementaré intentaré deducir J, según la siguiente formula:

$$Minimizar\ J = f(W, Z; X) = \sum_{i=1}^{m} \sum_{j=1}^{k} w_{ij} \| X^i - Z^j \|^2$$

Sujeto a que:

- $\sum_{j=1}^{k} w_{ij} = 1, 1 \leq i \leq m$
- $w_{ij} \in \{0,1\}, 1 \leq i \leq m, 1 \leq j \leq k$

Donde:

- $X^i = (x_1^i, \ldots, x_n^i)$, con $x_l^i \in R$ es el i-ésimo patrón de M.
- $Z^j = (z_1^j, \ldots, z_n^j)$, con $z_l^j \in R$ es el centroide del clúster C_j.
- W es una matriz de pertenencia con dimensión mxk en la que, un dato pertenece a un solo centroide, representando este pertenece con un 1, y un 0 en el caso contrario.
- K es el número de clústeres, m es el número de patrones en M, n es la dimensión del espacio y ||.|| es la distancia Euclídea al cuadrado.

2. Información Algoritmos de Hormigas

En general, para aplicar la OCH a un problema, es necesario que pueda ser representado en forma de grafo con pesos . Cada arco ars información: del grafo contendrá dos tipos de información heurística.

Los algoritmos ACO son procesos iterativos. En cada iteración se "lanza" una colonia de m hormigas y cada una de las hormigas de la colonia construye una solución al problema. Las hormigas construyen las soluciones de manera probabilística, guiándose por un rastro de feromona artificial y por una información calculada a priori de manera heurística.
Cuando todas las hormigas han construido una solución debe actualizarse la feromona en cada arco con el valor de evaporación de la feromona.

Veremos a la hormiga artitfial como un agente que recuerda los nodos que ha recorrido, utilizando para ello una lista de nodos visitados (L). Al finalizar, esta lista contiene la solución construida por la hormiga . En cada paso, estando en la posicion r elige hacia qué posición s moverse de entre las vecinas de r que no hayan sido visitados aún , según una regla probabilística de transición .

Se usa una retroalimentación positiva para reforzar en el futuro los componentes de las buenas soluciones mediante un aporte adicional de feromona . Cuanto mejor sea la solución, más feromona se aporta . Se usa la evaporación de feromona para evitar un incremento ilimitado de los rastros de feromona y para permitir olvidar las malas decisiones tomadas La evaporación es la misma para todos los rastros, eliminándose un porcentaje de su valor actual: $0 \leq \rho \leq 1$ Es un mecanismo de evaporación más activo que el natural, lo que evita la perduración de los rastros de feromona y, por tanto, el estancamiento en óptimos locales .
Los arcos visitados por hormigas en la iteración actual (arcos prometedores) reciben un aporte extra de feromona y los no visitados por ninguna hormiga (poco prometedores) la pierden .

El Sistema de Colonia de Hormigas (Ant Colony System) (SCH) extiende a su predecesor, el SH, en tres aspectos:
La regla de transición establece un equilibrio entre la exploración de nuevos arcos y la explotación de la información acumulada . Para la actualización (global) de feromona sólo se considera la hormiga que generó la mejor solución hasta ahora. Sólo se evapora feromona en los arcos que componen ésta . Se añade una nueva actualización (local) de feromona basada en que cada hormiga modifica automáticamente la feromona de cada arco que visita para diversificar la búsqueda .
La regla de transición de SCH (regla proporcional pseudo-aleatoria)

La actualización global de feromona que sólo se aplica sobre los arcos que pertenezcan a la mejor solución. Sólo la hormiga que generó la mejor solución hasta el momento modifica los niveles de feromona. El aporte de feromona es función de la calidad de esta solución.
Cada vez que una hormiga recorre un arco, aplica la actualización local de feromona. Con esta operación, la feromona asociada a un arco disminuye cada vez que lo visita una hormiga. Los arcos ya visitados van siendo menos prometedores según los recorren más hormigas en la iteración actual, lo que favorece la exploración de arcos no visitados . Así, las hormigas tienden a no converger a soluciones parecidas en la iteración actual .

El Sistema de Hormigas Max-Min (Max-Min Ant System) (SHMM) es una nueva extensión del SH con una mayor explotación de las mejores soluciones y un mecanismo adicional para evitar el estancamiento de la búsqueda. Se evaporan todos los rastros de feromona . Se establecen unos límites en los valores posibles de feromona en los rastros.
Los límites se calculan de forma heurística . Todos los rastros de feromona se inicializan al máximo valor τmax, en lugar de a un valor pequeño $\tau 0$.
Al aplicar la regla de actualización los arcos de las buenas soluciones mantienen valores altos mientras que los de las malas reducen el valor de sus rastros .

Existe la posibilidad de hibridar los algoritmos de OCH con técnicas de búsqueda local para mejorar su eficacia . La hibridación consiste en aplicar una búsqueda local sobre las soluciones construidas por todas las hormigas en cada iteración antes de actualizar la feromona . El aumento en la eficacia provoca una disminución en la eficiencia, por lo que es habitual emplear la búsqueda local junto con las llamadas Listas de Candidatos, que consisten en estudiar sólo los valores candidatos más prometedoras en cada paso de la hormiga .

Para esta practica se realizan los algoritmos con búsqueda local.

MÉTODO LOCAL SEARCH

Descripción en pseudocódigo de la búsqueda local empleada.

```
busquedaLocal(numero iteraciones, numero de evaluaciones){

        Solucion Inicial


        Aumentamos en 1 eval.
        for interacciones < numero_iteraciones
                limpiamos el vector de usados
                for i=0 i < numero_clusters
                        inicializamos un iterador para cada clusters con el primer elemento de cada clusters.
                        recorremos con el iterador los cluster
                                Buscamos la distancia de cada patrón al dendroide
                                        si no a sido usado buscamos en otro cluster distinto para ver si alguno
mejora
                                                si la mejora
                                                recalculamos centroides
                                                movemos patrón al nuevo centroide
                                                aplicamos el operador de vecino

                                                        recalcularCentroides(i,min,it);
                                                        moverPatron(i,min,it);
                                                        calculamos nueva solucion
                                                        eval++
                                                        si solucionInicial>nueva solucion
                                                        incluirUsados(usados,it);
                                                        mejorSolucion=solVecino;


Devolver mejor solución
```

Factorización: Recalcula los centroides eliminando la contribución del elemento cambiado y añadiendo esa contribución al nuevo cluster.

```
RecalcularCentroides(clusterOrigen, clusterDestino,  posPatron){

        calcular coste de ese patrón en el cluster Origen y restarlo
        calcular coste del patrón en el cluster destino y sumarlo.
FinRecalcularCentroides
```

3. Información de los algoritmos de hormigas asociado al problema del Clustering.

Esquema de representación: Las componentes de la solución L son k patrones escogidos de entre los l más prometedores (Lista de Candidatos). El grafo de construcción es un grafo totalmente conexo que incluye los l nodos .

Restricciones: No se pueden formar ciclos, es decir, no se puede visitar un nodo (no se puede escoger un patrón como medoide) más de una vez .

Función objetivo: El índice J de la partición resultante de considerar los k patrones de la solución construida en L como medoides y aplicar la regla de asignación del cluster más cercano para el resto de patrones .

Método de obtención de la lista de candidatos.

```
ObtenerListaCandidatos( tabla_patrones, candidatos, patrones,dimensiones, clusters, num_candidatos)
        Lista que almacena todos los patrones.
        Vector que almacena la ganancia de cada patrón.
        Vector que almacena las dimensiones (X0..Xn-1) de la media de todos los elementos del conjunto.
        Vector de pare auxiliar;
        La inicializamos con todos los patrones de auxiliar del 0 al n-1.
        Recorremos todos los patrones y vamos acumulando sus dimensiones en el vector centro.
        Buscamos el patrón con menor distancia a la media de todos los patrones (El más céntrico).
        Recorremos todos los patrones en busca del mas cercano a la media.
                Introducir el patrón más céntrico en la lista de candidatos.
        Ahora comienza el bucle greedy en el que, en cada paso, se escoge el patrón mas alejado de los centros ya
determinados y con mas ejemplos alrededor como nuevo centro
        Usando únicamente el medoide más céntrico del conjunto obtenemos las ganancias de los demás.
                Para el primer patrón i no seleccionado. Su valor en relaciona su posición en la lista de patrones es -1
                Para los demás
                                inicializamos ganancias a 0
                                para cada patrón distinto de i y no seleccionado.
                                        Para cada centroide ya calculado:
                                        Hallar el más cercano.
        Ordenar el vector con las ganancias obtenidas con respecto al más céntrico de los patrones y elegirlas para la
lista de candidatos.
FinObtenerListaCandidatos
```

Crea los individuos de la colonia y elige, para cada uno de ellos, el nodo de partida.

```
configurarColonia(colonia, candidatos){


                Creamos nueva hormiga homiga=Auxiliar
                inicializarHormiga(auxiliar, candidatos);
                primerPaso(auxiliar);
                colonia[i]=auxiliar;
FinConfiguracion
```

```
CrearSolucionesAsociadas(soluciones,  numero_soluciones, n_patrones,  numero_dimensiones, n_clusters,  patrones,
matrices_pertenencia){
        Para el numero de soluciones
                Creamos una estructura clustering auxiliar
                realizamos ConfuguracionInicial sobre ella y la introducimos en soluciones.
FinCrearSolucionesAsociadas
```

Rastros de feromona: Se asocia un rastro a cada nodo del grafo r, τr. Indica la preferencia memorística de escoger el patrón del nodo r como medoide .

Para el algoritmo ACS Se actualizan los valores de feromona asociados al arco (nodo solución) que escoge la hormiga h en la iteración i. T0 es el valor inicial de feromona. Deltha es el porcentaje que se evapora.

```
ActualizacionLocalFeromona(feromonas,  t0,  deltha, posicion){

        feromonas[posicion].second = ((1-deltha)*feromonas[posicion].second) + (deltha * t0);

FinActualizacionLocalFeromona
```

Se actualiza todos los componentes de feromona de los nodos solución de una hormiga con 1/coste de la mejor solución.

```
Actual_Global_Feromona(hormiga , feromonas, costeMejor, cons_evap){

        Buscamos la hormiga en el vector del rastro de feromonas (en relación al recorrido de su nodo solución)
        actualizamos el vector feromonas con el valor de la constante.

        aporte = 1/costeMejor;
        Mientras(nodo_solución < Numero de nodos)
                Para(i=0; i < TamañoFermonas)
                        Si(Recorrido[nodo_solucion] pertenece al vector Feromonas)
                                Valor_Nuevo_Feromona =
(1-cons_evap)*Valor_actual_Feromona+cons_evap*aporte;
                                nodo_solución += 1;
                        FinSi
                FinPara
        FinMientras

FinActual_Global_Feromona
```

En el ACS se evapora feromona de todos los nodos, para el MMAS se evapora solo los nodos pertenecientes a la mejor solución.

```
QuitarFeromona(vector de feromonas, cons_evap, modo, nodos_solución){

        Si(modo == ACS){
                Quitar un x% a todos los rastros.
        Si(modo==MMAS)
                Localizamos la mejor solución y aplicamos contante de evaporación sobre su valor en el vector
feromona.
```

Información heurística: Se define como la ganancia de escoger el patrón asociado al nodo r como medoide, $\eta r=\Sigma j$ Cji, calculada con las distancias a los patrones no seleccionados aún . Es una heurística dinámica, hay que recalcularla en cada paso del algoritmo al depender de la solución parcial L construida hasta el momento (de los patrones ya escogidos como medoides, que no influyen en el cálculo de distancias) .

REGLAS DE TRANSICION Y METODOS ASOCIADOS

Depende del valor de la probabilidad p.

```
reglaTransicionACS(hormiga, tabla_patrones, p, feromonas, alpha,  betha){

        val = valoraciones(h, tabla_patrones);
        x = heuristica(val, feromonas, alpha, betha);
        candidato_elegido = escogerCandidatoACS(val, p);
        añadirNodo(h, candidato_elegido);
        return candidato_elegido;

FinReglaTransicionACS
```

Depende de la probabilidad si. Calculamos un numero a al asar si la probabilidad es menor que ese numero entonces escogemos el mejor
en caso contrario aplicamos ruleta.

```
EscogerCandidatoACS(vector<double> valoraciones, int q){

    probabilidad=aleatorio
    Si(probabilidad <= q)
        candidato_escogido = escogerMejor(valoraciones)

    SiNo
        candidato_escogido = escogerPorRuleta(valoraciones)
    Devolver candidato_escogido

Fin EscogerCandidatoACS


reglaTransicionMMAS(hormiga ,tabla_patrones, feromonas,  alpha, betha){

        val = valoraciones(hormiga, tabla_patrones);
        x = heuristica(val, feromonas, alpha, betha);
        candidato_elegido = escogerPorRuleta(val);
        añadirNodo(h, candidato_elegido);

        Devolver candidato_elegido;

FinReglaTreansicionMMAS
```

Elección por ruleta se escoge un candidato aleatoriamente. Para esto se le asigna una probabilidad según su valoración, se suman las valoraciones de los elementos no escogidos anteriormente menos las negativas y se divide la valoración de cada uno por el total sumado lo de que nos da su probabilidad. Después creamos los intervalos para cada candidato. Posteriormente se crea un numero aleatorio entre 0 y 1 que determina el intervalo en que se encuentra el candidato a elegir.

EscogerPorRuleta(valoraciones)

Sumar las valoraciones heurísticas de todos los elementos menos los que sean con valoración negativa.

para cada valoración positiva
guardar en un par en el primer elemento su posición
en el segundo guardar el valor de su valoración heurística/la suma total.

Creación del vector de intervalos::
Para cada elemento asignar una probabilidad definida entre dos valores hasta el tamaño de las probabilidades*
El primer intervalo sera [0,el valor de la primera componente del vector de probabilidades creado
Después repetir
primer elemento del intervalo = Segundo elemento intervalo anterior
segundo intervalo = (Segundo elemento intervalo anterior+ probabilidad de ese elemento.)
creamos un numero aleatori amente
buscamos el elemento al que pertenezca ese elemento
(que sea mayor o igual que el primer elemento del intervalo pero menor que el segundo elemento.)

Devolvemos el valor de la posición del elemento al que le pertenezca.

FinEscogerPorRuleta()

Se obtiene las valoraciones asociadas a los n-k nodos candidatos con respecto a los k ya seleccionados usando Kaufman

Valoraciones(hormiga , tabla_patrones, iteración){

Ganancias vector que almacena la ganancia de cada patrón.

Ahora comienza el bucle Greedy en el que, en cada paso, se escoge el patrón mas alejado de los centros ya determinados y con mas ejemplos alrededor como nuevo centro

Para i hasta total de candidatos
si es el primero ganancia[i]
sino ganancia[i]= 0
para cada patrón distinto de i y no seleccionado.
Para cada centroide ya calculado:
Hallar el más cercano.
Si (Valor >=0) ===> (ganancias[i] += valor);

Devolver ganancias;

FinValoraciones

Devuelve el vector con el coste (Dado por Kaufman) multiplicado por la feromona asociada a cada nodo.

Heurística(Ganancias,feromona, alpha, betha){

Para cada valoración
Actualizar a ((valor actual elevado a alpha)*(valor ganancia elevado a betha))
Devolver Valoraciones

FinHeuristica
Escoger el patrón de mejor ganancia del total. (Greedy)

EscogerMejor(valoraciones){

Para cada valoración i

Comparar guardar posición de la mejor i

Devolver mejor solución
FinEscogerMejor

Proceso constructivo:

Antes de iniciar el proceso constructivo, se escoge el patrón más céntrico del conjunto y se calcula la ganancia de los m-1 patrones restantes. Se seleccionan los l-1 patrones con mayor ganancia como candidatos a ser medoides, junto con el más céntrico (LC) .

Para inicializar, o bien se escoge directamente escoge el patrón más céntrico del conjunto, i, como primer medoide, o bien se escoge un nodo aleatorio i. $L[1] \leftarrow i$.

En cada uno de los n-1 pasos necesarios (k=2 hasta n):

Se construye la lista de candidatos factibles con los nodos asociados a los patrones no seleccionados: $J(k) = \{u \in \{1, ..., l\} \mid u \notin L\}$.

Se calculan las ganancias ηu de los nodos de la lista en función de la solución parcial L generada hasta el momento .

Se escoge un nodo r de la lista con la regla de transición (basada en la feromona τu y la heurística ηu). Se almacena en L, $L[k] \leftarrow r$.

Algoritmo ACS

Constantes del algoritmo

n_patrones, n_dim_patron, n_clusters,semilla, n_candidatos, int n_ants, cons_evap, alpha, betha, deltha, probalididad_eleccion_ACS, T0, n_LS, max_eval.

Asignar memoria estructuras

tabla de patrones patrones X dimensiones del patron.

CargarEstructura

leerpatrones

ObtenerListaCandidatos

LeerCandidatos

Vector de feromononas tamaño igual al numero de candidatos del problema inicialisados a T0.

Creamos la colonia de hormigas con el numero de hormigas.

Creamos la S soluciones asiciadas a cada hormiga.

cramos un vector de matrices de pertenencia de cada configuración de hormigas.Con los k medoide en sus respectivos clusters. Y los n-k patrones asiganados al clusters mas cercano.

mientras no se realicen todas las evaluaciones

configurColonia

Bucle para regla de trancisión. En la que se aplica la regla de transición ACS con la probabilidad asignada, alpha y betha. Ademas de aplicar una actualisacionLocalFeromona.

Bucle para establecerConfiguración para cada hormiga de la colonia

Inicializamos las soluciones asociadas a cada hormiga

Configurar inicialmente los clusters

Buscar mejor hormiga en base a la solucion correspondiente.

Evaporamos la feromona para ACS

Quitar al mejro y dar al mejor

Fin mientras
Volvemos a leerCluster() para obtener el calculo del proceso de las evaluaciones guardamos mejor solucion.
mostramos tiempo y coste mejor solución.

Algoritmo MMAS

Constantes del algoritmo

n_patrones, n_dim_patron, n_clusters,semilla, n_candidatos, int n_ants, cons_evap, alpha, betha, n_LS, max_eval, Gmax,Gmin.

Asignar memoria estructuras

tabla de patrones patrones X dimensiones del patron.

CargarEstructura

leerpatrones

ObtenerListaCandidatos

LeerCandidatos

Crear vector feromonas con tantas posciones como candidatos se consideren en el problema. Todos ellos inicializados a Gmax.

Creamos la colonia

Creamos un vector para poder copiar el recorrido aleatorio de la primera solucion en la mejor hormiga global.

creamos un vector de estructuras clustering para almacenar las s soluciones asociadas a cada hormiga.

La solución inicial es aleatoria.

con ellas se obtienen los valores Gmax y Gmin, como , Gmax -> 1/ρ·Coste(Saleatoria) y Gmin -> Gmax/500

Al principio la mejor solución es la que se crea al comienzo de la ejecución

La mejorhormiga global también al que tiene el recorrido los k primeros elementos de la solución.

Creamos un Vector que contenga la matrices de pertenencia de cada configuración de hormigas.

Los k medoides es sus receptivos clusters. Los n-k patrones asociados al cluster mas cercano.

Mientras no se complete el máximo de evaluaciones

Inicializamos la colonia, elegimos primer nodo al asar.

Aplicamos regla transición:

para todos los clusters recorrer los elementos de la colonia

Aplicar ReglaTransicionMMAS

Para cada hormiga establecemos una configuración

Inicializamos las soluciones asociadas a cada hormiga. Configuración inicial de los clusters.

CrearsolucionesAsociadas a casa hormiga

para cada hormiga aplicar búsqueda local

Buscamos mejor hormiga(en base a la solución correspondiente)

Comprobamos si mejora a la inicial

Si la mejora actualizamos Gmax=1/(cons_evap * mejor_solucion) y Gmin=Gmax/500

Aplicamos regla de transición para MMAS quitamosFeromona según método de la ruleta.

Truncamos los valores que excedan a Gmax a Gmax y los que queden por debajo de Gmin a Gmin.

FinMientras

Devolver coste de la mejor configuración , mejor configuración , hormiga correspondiente a la mejor configuración.

4. Valoración de los algoritmos.

Los parámetros de ejecución para los algoritmos son los detallados en la practica correspondiente a clustering, y se introducen de forma manual en la ejecución del algoritmo.
El numero veces que se repite la búsqueda local es 10, ya que dispongo de una BL factorizada.

El numero de evaluaciones totales son 20000, con una colonia de 10 hormigas y una lista de candidatos de 100 elementos.

Los valora de las semillas se introducen manualmente. Para las 5 ejecuciones de los diversos algoritmos han sido 12345678, 23456789, 34567891, 45678912, 56789123 respectivamente.

Valores de K.Medias

Archivo Aggregation.txt
Max=29467,98
Min=9432.28
Media=
Archivo yeast.txt
Max=41,57
Min=38,15
Media=40,15

Archivo de prueba yeast.txt

Para valores de semilla mas bajos las diferencias de tiempo entre ACO y MMAS son mas significativas llegando a una diferencia de casi 10 seg.

Respecto a los valores (J) para la primera semilla (12345678) el coste de ACO es menor que el de MMAS. Pero conforme aumentamos el valor de la semilla MMAS mejora aunque no significavamente los valores de ACO.

El algoritmo k medias tiene un tiempo de ejecución de 1 seg mucho menor que el de cualquiera de y el mayor coste para k medias es menor que cualquiera de los obtenidos por los algoritmos de hormigas.

Archivo de prueba Aggreagation.txt

Respecto a los algoritmos de hormigas el ACO es siempre mas lento que MMAS.
En cuestión al coste el ACO para valores de semilla (12345678,23456789) da resultados que mejoran bastante los de MMAS pero a partir de la semilla 34567891 los resultados de MMAS dan un mejor coste y unos tiempos de ejecución algo mejores.

Al comparar con K-medias aquí ya si podemos ver que que el mayor valor de K-medias es mucho peor (mas del doble que cualquiera de MMAS y ACO), en el tiempo no hay comparación el k-medias es mucho mejor.
La media de los valores de K-medias es mayor que la media de ACO y MMAS.

En conclusión, el MMAS es mas agresivo que ACO ya que evapora todos los rastros y aporta solo a los de la mejor solución. Además define los topes para los rastros de feromona.. Solo se actualiza la feromona en la mejor solución, ya sea de la interacciones global o actual. Los rastros de feromona son inicializados a un valor alto para que la regla de actualización de mejores resultados,

porque los arcos de las buenas soluciones mantiene valores altos mientas que los de las malas reducen el valor de sus rastros.

También hace posible una mayor exploración al comienzo de la ejecución del algoritmo.
Todo esto hace que tengas un buen balance entre exploración y explotación, y reduce la posibilidad de estancamiento.

	Wdbc		Aggregation		Yeast	
	J	Tiempo	*J*	Tiempo	*J*	Tiempo
ACO(media)			11356,22	38,96	45,78	68,14
ACO(desv)			227,07	2,33	0,54	4,26
MAAS(media)			11493,40	36,47	45,63	63,14
MAAS(desv)			5,23	36,47	0,21	12,13

ACO	Wdbc		Aggregation		Yeast	
	J	Tiempo	*J*	Tiempo	*J*	Tiempo
Ejecución 1			11111,50	42,95	45,28	63,92
Ejecución 2			11111,50	37,88	45,92	63,40
Ejecución 3			11485,00	37,82	45,53	71,23
Ejecución 4			11585,80	38,99	45,53	69,40
Ejecución 5			11487,30	37,14	46,66	72,73
Mejor		---		---	46,66	---
Peor	0,00	0,00	11585,80	42,95	45,28	---
Media	#DIV/0!	#DIV/0!	11356,22	38,96	45,78	68,14
Desv. típica	#DIV/0!	#DIV/0!	227,07	2,33	0,54	4,26

MMAS	Wdbc		Aggregation		Yeast	
	J	**Tiempo**	***Tiempo***	**J**	***Tiempo***	**J**
Ejecución 1	###		36,73	11489,70	56,30	45,62
Ejecución 2	###		36,66	11482.6	45,44	45,44
Ejecución 3	###		36,23	11402.6	75,53	45,97
Ejecución 4	###		35,93	11497,10	68,40	45,48
Ejecución 5	###		36,82	11483.3	70,04	45,66
Mejor	###	---	36,82	---	75,53	---
Peor	###	---	35,93	---	45,44	---
Media	###	#DIV/0!	36,47	11493,40	63,14	45,63
Desv. típica	0,00	#DIV/0!	0,38	5,23	12,13	0,21