UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad 1
Departamento de Técnicas de Programación

Clasificación automática de datos con enfoque multiobjetivo usando Optimización con Colonia de Hormigas

Trabajo final presentado en opción al título de
Máster en Informática Aplicada

Autora: Lic. Yeneit Delgado Kios
Tutor: Dr. Juan M. Otero Pereira

La Habana, marzo de 2012

Agradecimientos

A mi familia y Joviel, por la ayuda y comprensión.

A mi tutor, Otero, por su paciencia y guía.

A Luis, por la ayuda en los últimos momentos.

A la facultad, por el apoyo.

A mis amigos.

Dedicatoria

A mi mamá, mi abuela y mi hermana.

Declaración jurada de autoría

Declaro por este medio que yo, Yeneit Delgado Kios, con carné de identidad 81110904535, soy la autora principal del trabajo final de maestría “Clasificación automática de datos con enfoque multiobjetivo usando Optimización con Colonia de Hormigas”, desarrollado como parte de la Maestría en Informática Aplicada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría en La Habana a los __ días del mes de _____ del año ___.

Resumen

En el presente trabajo se propone un algoritmo basado en la metaheurística Optimización con Colonia de Hormigas para resolver el problema de clasificación supervisada y posteriormente, sobre la base del algoritmo propuesto y utilizando ideas de la optimización multiobjetivo se propone una estrategia para encontrar solución al problema de clasificación automática. El problema consiste en determinar agrupaciones naturales o clases presentes en un conjunto de elementos sobre los que se han medido ciertas características. En la estrategia propuesta se tienen en cuenta dos criterios opuestos para determinar las agrupaciones, los cuales miden la similitud intraclase y la disimilitud interclases, obteniéndose un conjunto de soluciones no-dominadas, dado que ninguna es mejor que otra con respecto a los dos criterios utilizados.
Se desarrolló una aplicación en el lenguaje C# que ofrece funcionalidades para la experimentación con los algoritmos implementados. La herramienta permite la carga de datos con formatos ajustables a los de bases de datos internacionales y la visualización de datos en $\Re^2$, así como de los resultados de la aplicación de los algoritmos para datos de este tipo. Además, es posible realizar de forma automática la configuración de parámetros para la metaheurística utilizada.

Palabras clave: Clustering, clasificación automática, Optimización con Colonia de Hormigas (ACO), optimización multiobjetivo.

Abstract

An algorithm based on Ant Colony Optimization to solve the problem of classification is proposed and afterwards, on basis of the proposed algorithm and making use of ideas from the multi-objective optimization field, a strategy is presented to solve the clustering problem. The problem consists in determining natural groups or classes in a set of elements on which some characteristics have been measured. By taking into account two opposed criteria to determine groups, measuring the intracluster similarity and the intercluster dissimilarity, the proposed strategy obtains a set of non-dominated solutions.

An application in C# language was developed, which offers options for the experimentation with the implemented algorithms. The tool allows loading data with formats adjustable to international data bases and permits also the visualization of data in $\Re^2$ as well as that of the result of using the algorithms for this kind of data. Besides, it is possible to automatically carry out the parameters tuning for the meta-heuristics.

Keywords: Clustering, classification, Ant Colony Optimization (ACO), multi-objective optimization.

Tabla de Contenidos

Introducción

La manipulación de datos en aras de obtener información es una actividad muy común en nuestros días. El desarrollo científico depende en gran medida de cuánto pueda hacerse para lograr mejoras en este proceso, que se ha complejizado debido a los grandes volúmenes de datos que se generan en diversas áreas como pueden ser la genética, el comercio o la web. Paralelamente, la disponibilidad de computadoras con cada vez más altas prestaciones ha permitido la creación de aplicaciones para el manejo eficiente de datos, favoreciendo el análisis de fenómenos que en otros tiempos no podrían haber sido estudiados. Aun así, subsisten problemas cuyas soluciones tardarían meses, e incluso años, en ser obtenidas utilizando las aplicaciones con las que se cuenta actualmente, por lo que se requieren herramientas más eficientes para obtener aquellas respuestas que puedan permanecer ocultas en esos datos.

A la exploración y análisis de grandes cantidades de datos con vistas a descubrir patrones y reglas significativos subyacentes en los mismos se le denomina minería de datos [1]. Esta, forma parte de un proceso más amplio conocido como Descubrimiento del Conocimiento en Bases de Datos (KDD, por sus siglas en inglés: Knowledge Discovery in Databases), del que constituye su fase principal [2], aunque en ocasiones suelen utilizarse ambos términos –minería de datos y KDD- indistintamente [2-4]. La minería de datos abarca diversas técnicas o actividades para lograr "la extracción, a partir de los datos, de información implícita, previamente desconocida y potencialmente útil" [4, 5]. Una de las tareas de la minería de datos consiste en la agrupación de elementos en clases o grupos "naturales" de acuerdo a sus características.

En la literatura aparecen diferentes términos relacionados con esta tarea de agrupación de datos, entre los que se encuentran: clasificación (o también clasificación supervisada) y clasificación automática (o clustering), cuya diferencia esencial radica en que el número de clases sea prefijado o no.

En el contexto de este trabajo se entenderá por *clasificación* o *clasificación supervisada* a la agrupación de los datos en un número predeterminado de clases (en este caso solamente se identificaría la composición de los grupos, puesto que la cantidad de estos se conoce a priori).

Por *clasificación automática* o *clustering* se entenderá la agrupación que se realiza tratando de identificar simultáneamente la cantidad de grupos "naturales" de los datos y su composición.

La clasificación es otra de las actividades típicas de la minería de datos y correspondería a la llamada clasificación supervisada, dado que en la misma se conocen previamente las clases en las que deben ser ubicados los elementos. Es conocida también como análisis discriminante. Véase, por ejemplo, la figura 1, donde pueden distinguirse tres grupos de elementos en $\mathcal{R}^2$, teniendo en cuenta la cercanía entre aquellos pertenecientes a un grupo y la lejanía de estos respecto a los miembros de otro. La tarea de clasificación consistiría en determinar para cada elemento a cuál de las tres clases predefinidas pertenece.

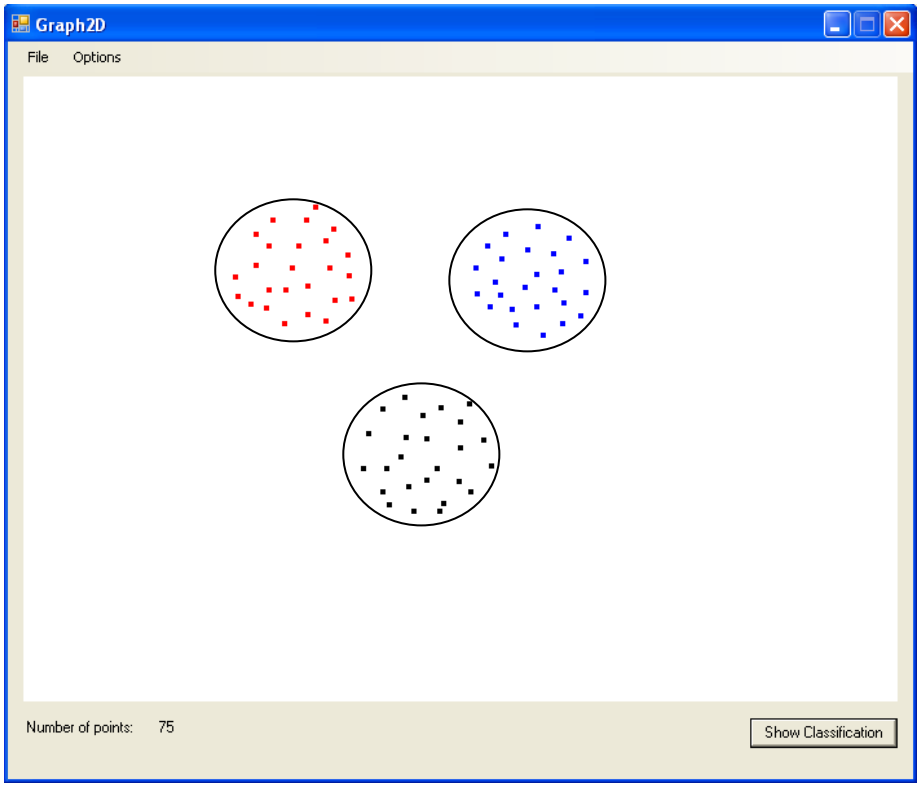


Figura 1: Elementos agrupados en tres clases

En la clasificación supervisada, e igualmente en la clasificación automática, se buscará que los grupos sean lo más homogéneos posible –que los elementos dentro de un grupo sean parecidos entre sí- y que a la vez se diferencien o distancien entre ellos. El parecido entre los elementos se determina haciendo uso de alguna medida de similitud o distancia.

Por otra parte, la clasificación automática también se conoce como análisis de conglomerados, análisis de grupos, análisis tipológico, análisis de agrupaciones, o clustering y clusters analysis, del inglés [6]. Si se toma como ejemplo un conjunto de datos como el que se muestra en la figura 2, la tarea de clasificación automática consistiría en determinar alguna estructura presente en los datos, esto es, cuáles son los grupos naturales (cantidad y composición de los mismos). En este caso existe más de una posibilidad, pues, dado que no está prefijado el número de clases, podrían obtenerse como resultado agrupaciones de dos, cuatro u ocho clases que pueden ser consideradas naturales. Decidir entre una u otra de estas agrupaciones, y con ello también lo que se entiende por "natural", dependerá del objetivo para el cual se realice la clasificación automática y del problema en particular.

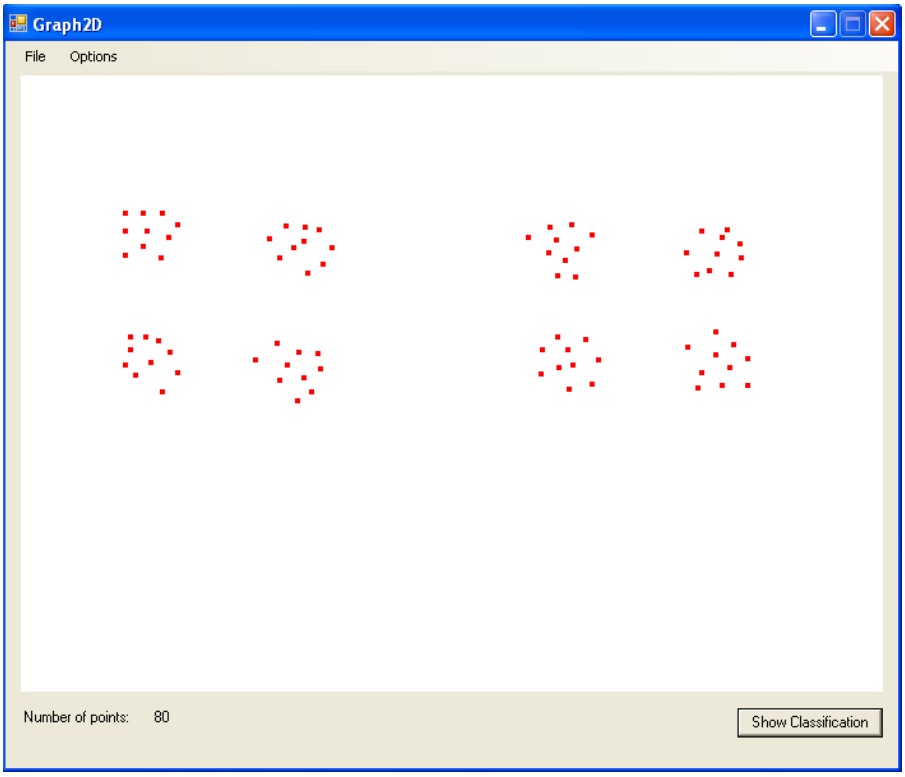
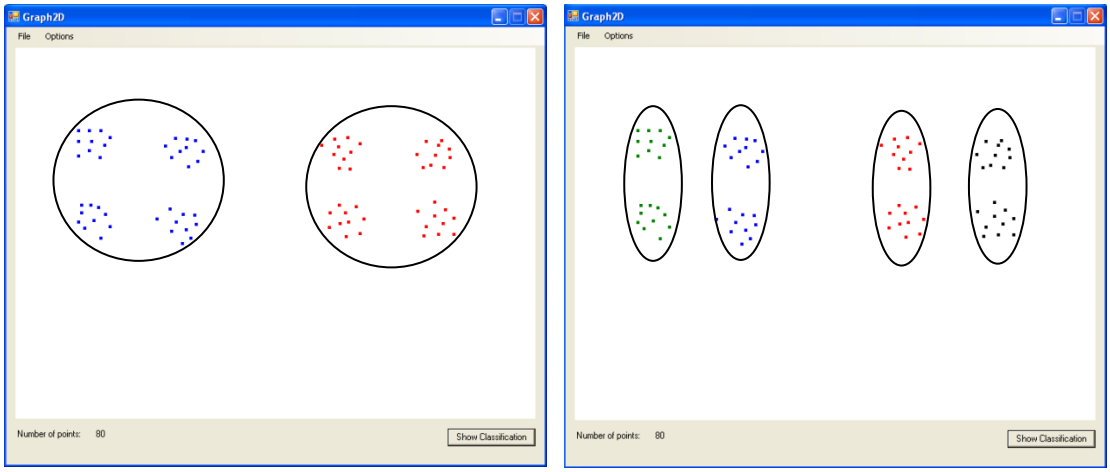
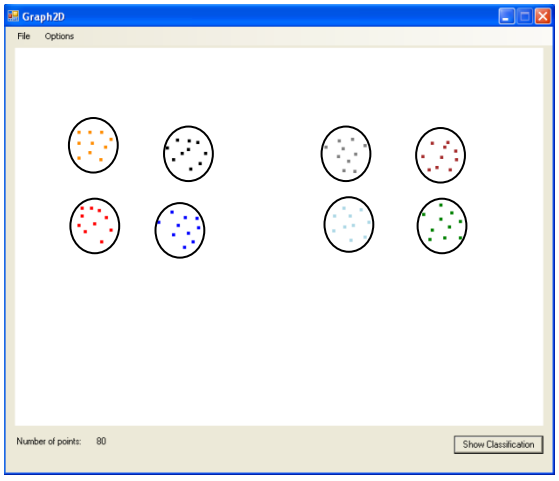


Figura 2: Conjunto de elementos



a) b)



c)

Figura 3: Tres clasificaciones diferentes para el mismo conjunto de datos: a) dos clases, b) cuatro clases, c) ocho clases

La clasificación automática de datos tiene en sus orígenes a la estadística y está relacionada con muchas otras ramas de la ciencia. En [7] se tiene un referente de otra área en la que puede enmarcarse, cuando se trata de documentos: la recuperación de información, en la cual se han desarrollado sistemas automatizados, originalmente para manejar la vasta literatura científica producida desde los años ’40, siendo este aún el uso más común de estos sistemas que se utilizan ampliamente en bibliotecas universitarias, corporativas y públicas.

Estos son solo algunos ejemplos de áreas del conocimiento que guardan relación con la clasificación automática de datos. Jim Gray[1] [8, 9], en el prefacio al libro "Data Mining. Practical Machine Learning Tools and Techniques" [5], plantea que "la síntesis de la estadística, el aprendizaje automático, la teoría de la información y la computación ha creado una ciencia sólida, con una base matemática consistente y con herramientas muy poderosas", refiriéndose a todo este proceso de extracción del conocimiento de grandes volúmenes de datos.

Entre las áreas en las que se ha aplicado la clasificación automática de datos están la bioinformática, minería de textos, análisis de mercados, minería de web, reconocimiento de formas, detección de fraudes (tarjetas de crédito, telecomunicaciones, etc.), psiquiatría, astronomía y arqueología. Ejemplos y referencias sobre algunas de estas –y otras- aplicaciones pueden encontrarse en [6] y [10]. Dada su extensa aplicabilidad es un problema de gran importancia y cuanto pueda hacerse en aras de lograr mejores soluciones será de utilidad.

El problema de clasificación automática de datos se ubica en la categoría de problemas NP-completos, lo que significa que no existe un algoritmo determinista que pueda solucionarlo en un tiempo polinomial [11]. Debido a su carácter combinatorio –a medida que crece la cantidad de datos de la entrada, la cantidad de posibles soluciones aumenta considerablemente-, se han empleado numerosas metaheurísticas[2] [12] para solucionarlo, entre ellas pueden mencionarse recocido simulado, búsqueda tabú, algoritmos genéticos y colonias de hormigas.

El presente trabajo aborda la clasificación automática a partir de la introducción de un enfoque multiobjetivo [13], siendo el problema a resolver *cómo identificar posibles agrupaciones "naturales" dentro de un conjunto de datos.* Como objeto de estudio se tendrá *la clasificación automática de datos* y como campo de acción *los algoritmos de*

[1] James Nicholas Gray (1944-2007), también conocido como Jim Garay, fue un científico de la computación estadounidense. Recibió el Premio Turing en 1998 "por contribuciones originales a la investigación en bases de datos y procesamiento de transacciones, y su liderazgo técnico en la implementación de sistemas". Trabajó como investigador y diseñador de software en varias compañías industriales, incluyendo IBM, Tandem Computers, DEC y Microsoft, de la cual era Asociado Técnico cuando desapareció en el mar en el año 2007 [8], [9].

[2] Las metaheurísticas son métodos de solución que instrumentan una interacción entre procedimientos de mejoras locales y estrategias de más alto nivel para crear un proceso capaz de escapar de óptimos locales y de realizar una exploración robusta de un espacio de solución [12]. Son un marco algorítmico general que puede ser aplicado a diferentes problemas de optimización con relativamente pocas modificaciones para adaptarlo a un problema específico [28].

la metaheurística Optimización con Colonia de Hormigas aplicados al problema de clasificación automática de datos.

Como objetivo general de la tesis se plantea entonces *diseñar e implementar una estrategia para identificar el número de clases existentes en un conjunto de elementos, utilizando ideas de la optimización multiobjetivo y la metaheurística Optimización con Colonia de Hormigas.*

Como objetivos específicos se tienen:

- Diseñar e implementar un algoritmo basado en Optimización con Colonia de Hormigas para resolver el problema de clasificación.
- Implementar una herramienta computacional que permita el ajuste automático de los parámetros que intervienen en el algoritmo propuesto.
- Proponer e implementar una estrategia para resolver el problema de clasificación automática basada en ideas de la optimización multiobjetivo y el algoritmo implementado para resolver el problema de clasificación supervisada.
- Evaluar los resultados de los algoritmos propuestos.

Para dar cumplimiento a estos objetivos se plantean las siguientes tareas de investigación:

- Caracterización de los algoritmos de Optimización con Colonia de Hormigas.
- Caracterización de los problemas de optimización multiobjetivo.
- Diseño e implementación de una herramienta que permita visualizar particiones de conjuntos de elementos de $\mathbb{R}^2$.
- Diseño e implementación de una herramienta para ejecutar el algoritmo propuesto que permita el ajuste automático de los parámetros que intervienen en el mismo.
- Evaluación de las particiones obtenidas y selección de una de ellas usando una medida de calidad de particiones.

Se establece como hipótesis de esta investigación: *La metaheurística Optimización con Colonia de Hormigas puede ser utilizada para resolver problemas de clasificación tanto supervisada como automática*.

En el desarrollo de la investigación fueron empleados métodos de trabajo científico entre los que se destacan:

Métodos teóricos

Hipotético-deductivo: para la formulación de la hipótesis de investigación y la definición de futuras líneas de trabajo.

Análisis-síntesis: para descomponer el problema de investigación en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.

Modelación: para el desarrollo de los algoritmos propuestos y las herramientas para visualizar particiones y ejecutarlos.

Métodos empíricos

Experimental: para probar los algoritmos desarrollados y obtener medidas de calidad de las particiones resultantes.

La principal contribución de este trabajo es la propuesta de algoritmos basados en Optimización con Colonia de Hormigas que permiten particionar un conjunto de datos en un número prefijado de clases o identificar la estructura inherente a un conjunto de datos para determinar el número de clases más conveniente en que pueden ser agrupados, todo ello integrado en una aplicación que permite además la configuración automática de parámetros de los algoritmos, la carga de datos con diferentes formatos y la visualización de las particiones obtenidas para el caso de datos en $\mathbb{R}^2$.

El resto del documento está estructurado en tres capítulos, conclusiones, recomendaciones y anexos.

El capítulo uno aborda los fundamentos teóricos de la investigación y está compuesto por tres secciones dedicadas a la clasificación automática de datos (1.1), la Optimización con Colonia de Hormigas (1.2) y la optimización multiobjetivo (1.3). En la sección 1.1 se describe formalmente el problema de clasificación automática de datos, incluyendo una descripción de los procedimientos utilizados para resolver este problema y una síntesis de otras técnicas que han sido utilizadas en el tratamiento del mismo. La sección 1.2 se dedica a describir los elementos básicos de la metaheurística Optimización con Colonia de Hormigas. En la sección 1.3 se introducen los conceptos fundamentales relacionados con la optimización multiobjetivo.

En el capítulo dos se muestra la aplicación de un algoritmo de Optimización con Colonia de Hormigas al problema de clasificación, describiéndose el proceso de construcción de una solución y una síntesis de los resultados del proceso experimental.

En el capítulo tres se describe una estrategia de solución para el problema de clasificación automática basada en el algoritmo propuesto para la clasificación supervisada e ideas básicas de la optimización multiobjetivo y se ilustra a través de algunos ejemplos el comportamiento del algoritmo implementado.

Finalmente, se exponen las conclusiones y algunas recomendaciones para trabajos futuros.

1 Marco teórico

1.1 Clasificación automática de datos

1.1.1 Definición del problema

Se utilizará la definición del problema abordada en [14]. Sea Γ un conjunto de n elementos denotados por x_i, $i = 1,\ldots, n$, ponderados con pesos $p_i > 0$, sobre los que se han medido m variables cuantitativas[3] $x_i(j)$, $j = 1,\ldots, m$.

Se quiere obtener una partición $P = (C_1, C_2,\ldots, C_k)$ de Γ en k clases bien separadas entre sí y lo más homogéneas posible, o sea, que los elementos dentro de la clase sean similares, de acuerdo a algún criterio de similitud.

Una partición P de Γ es un subconjunto de P(Γ) –el conjunto potencia de Γ, que comprende todas las posibles divisiones de dicho conjunto-, tal que:

i. $\forall C_i \in P: C_i \neq \emptyset$

ii. $\forall C_1, C_2 \in P: C_1 \cap C_2 = \emptyset$

iii. $\bigcup_{C_i \in P} C_i = \Gamma$

Se tiene entonces un conjunto finito que consiste en todas las particiones $P_k(\Gamma)$ del conjunto de individuos de Γ. Casi todos los métodos de clasificación por particionamiento[4] se basan en optimizar una función matemática F, que depende de la partición del conjunto en clases:

$$F: P_k^*(\Gamma) \rightarrow \Re$$

siendo $P_k^*(\Gamma)$ el conjunto de las particiones del conjunto Γ en k subconjuntos. A dicha función en lo adelante se le denominará *criterio de clasificación* [15].

Como se plantea en [15], hallar la estructura de clases subyacente al conjunto no es una tarea trivial y envuelve principalmente dos problemas:

- hallar el número de clases que existen en el conjunto

[3] O cualitativas, siempre que se haga una conversión, teniendo en cuenta que los algoritmos utilizados trabajan con valores numéricos.
[4] En la sección 1.1.2.1 se ahondará en este tipo de métodos.

- dado el número de clases, repartir los objetos del conjunto en cada clase de forma que se optimice la función *F*.

Para resolver solamente el segundo problema, se podría pensar, una vez seleccionado un criterio de clasificación, en calcular su valor para cada posible partición del conjunto en *k* subconjuntos no vacíos –una cantidad finita dado que el número de elementos del conjunto es finito- y seleccionar la partición que optimice el criterio [10]. Sin embargo, la cantidad de posibles particiones de un conjunto de *n* elementos en *k* clases está dada por la expresión [10, 15]:

$$N(n,k) = \frac{1}{k!}\sum_{m=1}^{k}(-1)^{k-m}\binom{k}{m}m^n$$

conocida como número de Stirling de segundo orden, y existe un gran número de posibles particiones, incluso para *n* y *k* moderados:

$$N(2,5) = 15$$

$$N(10,3) = 9330$$

$$N(50,4) \approx 5.3 \times 10^{28}$$

$$N(100,5) \approx 6.6 \times 10^{67}$$

Esto hace que sea imposible, incluso con las computadoras actuales, determinar todas las posibles particiones [10] y es por ello que se emplea entonces otro tipo de técnicas para abordar el problema.

1.1.2 Métodos de clasificación automática

De acuerdo a [6] existe gran cantidad de métodos de clasificación automática, entre los que se pueden distinguir los siguientes:

- los métodos jerárquicos, que buscan una serie de particiones encajadas de tal manera que puedan representarse mediante un árbol;
- los métodos piramidales, que –como los jerárquicos- buscan particiones encajadas, pero que permiten a una clase de nivel inferior estar contenida en dos clases de nivel superior;
- los métodos de particionamiento, que buscan una sola partición del conjunto de individuos;

- los métodos de clasificación no exclusiva, que buscan grupos en los datos de tal manera que un individuo pueda pertenecer a varios grupos al mismo tiempo;
- los métodos de clasificación difusa, que buscan grupos homogéneos de individuos pero que dan el grado de pertenencia difusa (en el intervalo [0, 1]) de cada individuo a cada clase;
- los métodos de clasificación cruzada, que tratan de hacer la clasificación simultáneamente sobre dos conjuntos de individuos, o uno de individuos y uno de variables.

De acuerdo al tipo de clasificación que se desea obtener en el presente trabajo, el método que se usará seguirá los principios de los métodos de particionamiento, como se verá en la sección siguiente. En [16] puede verse el resto de los métodos mencionados y también pueden encontrarse otros tipos de métodos –o modos de agruparlos- en [15], [17] y [18]. Un conjunto de métodos jerárquicos y de particionamiento se describe en [19] y otra excelente y actual bibliografía sobre el tema es [10].

1.1.2.1 Métodos de particionamiento

Los métodos de particionamiento, también llamados no jerárquicos [7], construyen una partición –con las características descritas formalmente en la sección 1.1.1- del conjunto de elementos en *k* clases. Los elementos son agrupados de acuerdo a algún criterio de similitud, de modo que en cada clase se encuentran elementos similares, y que a su vez se diferencian de los elementos de las clases restantes, quedando las clases bien delimitadas entre sí. El objetivo es usualmente descubrir una estructura que está ya presente en los datos, pero en ocasiones el algoritmo se utiliza para forzar una nueva estructura, por ejemplo cuando se divide un país en áreas telefónicas [19].

Como criterio de similitud entre dos elementos es común usar una función de distancia o métrica, pudiendo ser esta alguna de las definidas por la expresión:

$$L_k(x_a, x_b) = \left(\sum_{j=1}^{m} |x_a(j) - x_b(j)|^k \right)^{\frac{1}{k}} \qquad (1.1)$$

donde x_a y x_b son elementos de dimensión m, con $x_a(j)$ y $x_b(j)$ sus respectivas componentes j-ésimas y k es un parámetro variable. Dicha expresión es conocida como métrica de Minkowski o norma L_k, de la que son casos particulares la distancia

de Manhattan o city blocks, cuando *k*=1, y la euclidiana, cuando *k*=2 [20]. Otras funciones de distancia válidas son la distancia de Mahalanobis [10, 20], y la distancia de Tanimoto [20], por ejemplo.

Además del criterio de similitud, es preciso contar con una función o criterio para valorar la calidad de las particiones –el criterio de clasificación-. Un criterio que suele utilizarse es el de la *inercia intraclase* o *varianza total intraclase* [6, 15]:

$$W(P) = \frac{1}{n}\sum_{i=1}^{k}\sum_{x_j \in C_i} d(x_j, g_i)^2 \qquad (1.2)$$

donde:

d es una medida de disimilitud o distancia,

$g_i = \frac{1}{|C_i|}\sum_{j|x_j \in C_i} x_j$ es el centro de gravedad de la clase C_i.

Este criterio, que corresponde al promedio de las distancias entre los elementos y el centro de gravedad de la clase a la que pertenecen, enfatiza la similitud de los elementos dentro de las clases.

La *inercia interclases*, en cambio, es un criterio que valora la separación existente entre las diferentes clases por medio de la distancia de los centros de gravedad de las mismas al centro de gravedad del conjunto de elementos:

$$B(P) = \sum_{i=1}^{k}\frac{|C_i|}{n} d(g_i, g)^2 \qquad (1.3)$$

donde:

$g = \frac{1}{n}\sum_{i=1}^{n} x_i$ es el centro de gravedad del conjunto de elementos Γ.

Un resultado importante que relaciona estos criterios es la propiedad de Fisher [6], que establece que dada una partición $P=(C_1, C_2, \ldots, C_k)$ de un conjunto Γ, se cumple:

$$B(P) + W(P) = T \qquad (1.4)$$

siendo *B* y *W* las funciones definidas anteriormente y *T* el promedio de las distancias entre cada objeto del conjunto de datos y el centro de gravedad del mismo:

$$T = \frac{1}{n}\sum_{i=1}^{n} d(x_i, g)^2 \qquad (1.5)$$

Por tanto, fijado el número de clases, minimizar la inercia intraclase W es equivalente a maximizar la inercia interclases B, pues ambos extremos se alcanzarán sobre la misma partición P, o sobre las mismas particiones en caso de que la solución óptima no sea única.

Cuando el número de clases no se fija de antemano, el mínimo de la inercia intraclase es cero y se alcanza en la partición que considera a cada elemento como una clase. En este caso la inercia interclases alcanza el mayor valor posible, esto es, T. Contrario a lo que pudiera pensarse, ambos criterios se mantienen, en este caso, como no antagónicos. Nótese que al comparar dos particiones P_1 y P_2 con cantidades de clases diferentes, a la partición que corresponda el mayor valor de W también corresponderá el menor de B, ya que de acuerdo a la expresión (1.4) la suma de los valores de W y B en una partición determinada es una constante (T) y es la misma para todas las posibles particiones, puesto que T es el centro de gravedad del conjunto de datos y su valor no depende de la partición:

$$B(P_1) + W(P_1) = B(P_2) + W(P_2) = T \qquad (1.6)$$

Luego, cuando se intenta identificar el número de clases, el óptimo se alcanzaría si se tuviera una clase por cada elemento, lo cual en general no sería la solución buscada. Es preciso entonces evitar que esto suceda y una idea podría ser trabajar con alguna otra función que lo evite. De ahí que se haya valorado tener en cuenta más de un criterio y resolver el problema de clasificación automática utilizando ideas de optimización multiobjetivo.

1.1.3 Otras técnicas utilizadas

Otras técnicas han sido utilizadas para dar solución al problema de clasificación automática de datos. En [15] y [6] puede verse la aplicación exitosa de algoritmos genéticos, y en [13] se ofrece una aplicación de los mismos haciendo uso del enfoque multiobjetivo que se ha tomado como base en el presente trabajo.

También Trejos junto a otros autores ha desarrollado una biblioteca de funciones con diferentes metaheurísticas aplicadas a dicho problema, entre las que pueden encontrarse, además de los algoritmos genéticos: recocido simulado, búsqueda tabú,

colonias de hormigas y sistemas de partículas; trabajo avalado por publicaciones relevantes en el área de optimización en revistas especializadas [21-23].

Pueden consultarse además [24], [25] y el libro "Cluster Analysis" [10] que constituye una bibliografía muy actualizada y completa acerca del tema, en la cual se puede profundizar sobre aspectos específicos del mismo, así como en técnicas que se han utilizado para abordar el problema durante los últimos 40 años. Podrían mencionarse los mapas autoorganizados de Kohonen, clases jerárquicas (HICLAS), dos pasos (SPSS), entre otros, que aparecen en la tabla "Resumen de tipos de datos y métodos aplicables de clasificación automática" (ver Anexo 1), proveniente de dicha bibliografía. Los autores destacan que no existe un único y mejor método para clasificación automática, sino que pueden seleccionarse algunos con buenos resultados según el tipo de dato con que se vaya a trabajar.

1.2 Optimización con Colonia de Hormigas

1.2.1 Introducción

La Optimización con Colonia de Hormigas, conocida como ACO [26-31] (por Ant Colony Optimization, en inglés), es una metaheurística surgida como generalización de una serie de algoritmos basados en el comportamiento de las colonias de hormigas reales. Estos insectos son capaces de encontrar el camino mínimo desde sus nidos a las fuentes de alimento siguiendo los trazos de feromona, una sustancia química que pueden oler y que depositan al realizar sus recorridos.

Los primeros trabajos en esta área, en los años '90, promovieron la idea de usar hormigas y pistas de feromona artificiales que se relacionaran con las soluciones obtenidas por estas. Desde el comienzo se han generado aplicaciones para afrontar problemas tan complejos como el problema del vendedor viajante, primero en ser abordado mediante esta técnica, que continúa siendo utilizado para estudiar nuevos algoritmos, como puede verse en el anexo 3. El algoritmo denominado Sistema de Hormigas, conocido como AS por sus siglas en inglés, de los autores Marco Dorigo, Vittorio Maniezzo y Alberto Colorni [32-34], dio paso a todo un conjunto de aplicaciones y extensiones exitosas que han sido unificadas en la metaheurística ACO. Un ejemplo de estas extensiones es el Sistema de Colonias de Hormigas [35] (Ant Colony System:

ACS), en el cual se basa el algoritmo de Optimización con Colonia de Hormigas que se presenta en el capítulo dos de esta tesis.

Actualmente existe un gran número de aplicaciones de ACO a problemas complejos de optimización entre los que pueden mencionarse la asignación cuadrática, el enrutamiento en redes y el plegado de proteínas (véase Anexo 2), siendo en algunos casos la mejor solución que se ha alcanzado para el problema en cuestión.

El desarrollo de investigaciones acerca de esta metaheurística y de algoritmos de hormigas en general, ha propiciado la celebración de congresos bienales bajo el nombre de ANTS [36], desde el año 1998 con "ANTS'98–From Ant Colonies to Artificial Ants", el primer taller internacional sobre este tema que tuvo lugar en Bruselas, Bélgica, con la participación de más de 50 investigadores de todo el mundo. La más reciente edición tuvo lugar en el año 2010, bajo el nombre de "ANTS 2010–International Conference on Swarm Intelligence"[5]. A partir del año 2003, la IEEE[6] ha organizado también el "IEEE Swarm Intelligence Symposium", en el que se presentan trabajos relacionados con las metaheurísticas agrupadas bajo el nombre de Inteligencia Colectiva (Swarm Intelligence), que incluyen entre otras la Optimización con Sistemas de Partículas (PSO: Particle Swarm Optimization), Optimización con Colonia de Abejas (Bee Colony Optimization) y ACO como la metaheurística más consolidada en este campo. Unido a esto, se ha incrementado el número de publicaciones que incluyen contribuciones relacionadas con ACO, como las memorias de estos eventos [37-43] que desde el año 2002 se publican como parte de la serie Lecture Notes in Computer Science (LNCS) de Springer[7] y la revista Swarm Intelligence[8], de más reciente creación.

Internacionalmente varias instituciones desarrollan investigaciones sobre ACO, destacándose el Institut de Recherches Interdisciplinaires et de Développements en

[5] En las últimas ediciones, a partir del incremento de trabajos que se enmarcan en el área más amplia conocida como Inteligencia Colectiva –Swarm intelligence-, ha cambiado el nombre de la conferencia, añadiéndose este término en la edición del año 2004: "ANTS–International Workshop on Ant Colony Optimization and Swarm Intelligence", hasta dedicarse en el año 2010 a este campo como un todo: "ANTS 2010–The International Conference on Swarm Intelligence".

[6] IEEE, acrónimo de Institute of Electric and Electronics Engineers, Inc., Instituto de Ingenieros Eléctricos y Electrónicos. Asociación de ámbito internacional radicada en Estados Unidos, que se encarga de definir estándares para las comunicaciones, la industria eléctrica, las aplicaciones biomédicas o la electrónica. ["IEEE." Microsoft® Student 2008 [DVD]. Microsoft Corporation, 2007].

[7] Las memorias de la primera y segunda ediciones aparecieron como tópicos especiales en las revistas Future Generation Computer Systems [37] y en IEEE Transactions on Evolutionary Computation [38] respectivamente.

[8] http://www.springer.com/11721.

Intelligence Artificielle (IRIDIA)[9], en la Universidad Libre de Bruselas, Bélgica, y los laboratorios de Soft Computing and Intelligent Information Systems (SCI2S)[10], en la Universidad de Granada, España. En Cuba, el Laboratorio de Inteligencia Artificial de la Universidad Central "Marta Abreu" de Las Villas (UCLV)[11], el Centro de Estudios de Ingeniería de Sistemas del Instituto Superior Politécnico "José Antonio Echeverría" (ISPJAE)[12] y el grupo de Optimización de la Facultad de Matemática y Computación de la Universidad de La Habana[13] trabajan también en esta área.

1.2.2 Estructura general de los algoritmos

Los algoritmos ACO se basan en la comunicación indirecta en una colonia de hormigas "artificiales" mediante rastros de feromona "artificial", que son usados para construir soluciones al problema dado y que las hormigas modifican durante la ejecución del algoritmo para compartir su experiencia en la búsqueda [44].

El proceso de construcción de soluciones en los algoritmos ACO, además de tener en cuenta la experiencia acumulada por las hormigas, considera también información disponible acerca del problema –información heurística-.
La figura 4 muestra el esquema general de funcionamiento de estos algoritmos[14].

```
Procedimiento metaheurísticaACO
    Mientras criterio de parada no satisfecho
      ProgramarActividades
            construirSolucionesDeHormigas()
            actualizarFeromona()
            decisionesGlobales()
      Fin ProgramarActividades
    Fin mientras
Fin metaheurísticaACO
```

Figura 4: Seudocódigo de la metaheurística ACO. Fuente [23]

A continuación se describen de manera general las partes del algoritmo [28, 44].

Inicialización: Primeramente se inicializan los parámetros del algoritmo, incluyendo un valor τ_0, que se asigna como valor inicial a todos los rastros de feromona.

[9] Institución que organiza los congresos ANTS. Accesible desde http://code.ulb.ac.be/iridia.home.php.
[10] Accesible desde http://sci2s.ugr.es/presentation/index.php.
[11] Accesible desde http://www.uclv.edu.cu/.
[12] Accesible desde http://www.cujae.edu.cu/centros/ceis/.
[13] Accesible desde http://www.matcom.uh.cu.
[14] Para el caso de problemas de optimización combinatoria estáticos, o sea, que no cambian su topología ni costos durante la ejecución del algoritmo.

Una vez concluida la inicialización, se repite una serie de acciones hasta que se cumpla una condición de parada predefinida en dependencia del problema que se esté resolviendo:

ConstruirSolucionesDeHormigas: *m* hormigas construyen soluciones al problema de manera asíncrona e incremental. Aunque es dependiente del algoritmo ACO en particular, este procedimiento consiste en la adición de una nueva componente a la solución en construcción. Las hormigas se mueven aplicando una regla de decisión probabilística que hace uso de la información heurística y los rastros de feromona disponibles.

ActualizarFeromona: se actualiza la feromona de modo que las componentes de buenas soluciones sean más deseables para las hormigas de las iteraciones posteriores. Los rastros de feromona pueden incrementarse al ser depositada por las hormigas o pueden disminuir mediante un proceso de evaporación. Esta evaporación constituye un mecanismo que evita una rápida convergencia a óptimos locales, favoreciendo la exploración de nuevas áreas del espacio de búsqueda.

DecisionesGlobales: Opcionalmente se llevan a cabo otras acciones centralizadas que no pueden ser ejecutadas por las hormigas individuales, que los autores denominan “acciones del demonio”. Estas pueden manejar información global y permiten la implementación de mejoras al algoritmo, como pueden ser búsquedas locales para refinar una solución obtenida.

El esquema descrito no especifica cómo se sincronizan y programan estas partes del algoritmo, lo que permite suficiente flexibilidad para el desarrollo de algoritmos ACO aplicables a una amplia gama de problemas.

1.3 Optimización multiobjetivo

En el área de la optimización aparecen muchos problemas que requieren soluciones en las que más de un objetivo sea tenido en cuenta. No se trata de encontrar una solución que optimice cada uno de los objetivos por separado, pues en general esto no ocurre en la inmensa mayoría de los problemas con estas características en los que generalmente los objetivos que se consideran son antagónicos, lo que significa que la mejoría de uno de los objetivos se alcanza sacrificando otro. Por esta razón, en el caso de la optimización multiobjetivo se obtiene como resultado un conjunto de soluciones

llamadas de intercambio, puntos eficientes o soluciones no-dominadas, entre otros términos. Estas soluciones de alguna manera proporcionan un balance entre todos los objetivos.

1.3.1 Conceptos básicos

Un problema de optimización multiobjetivo puede definirse como [45]:

$$minimizar/maximizar \;\; f_m(x), \quad m = 1,2,\dots,M$$

$$sujeto\ a \;\; x \in S$$

donde:

$x=(x_1, x_2,\dots, x_n)^t$ es un vector de n variables de decisión, también llamado *vector de decisión*,

$f_m: \mathbb{R}^n \rightarrow \mathbb{R}$, $m=1,2,\dots, M$, son las *funciones objetivo* conflictivas que se quieren minimizar (o maximizar) simultáneamente,

$S \subset \mathbb{R}^n$, $S \neq \phi$, es la *región factible*.

En esta formulación de problema general no se especifica el tipo de restricciones para la región factible S, también denominada *espacio de búsqueda* [46] y constituida por el conjunto de soluciones factibles.

Los *vectores objetivo* son imágenes de los vectores de decisión y consisten en *valores (de función) objetivo* $z = f(x) = (f_1(x), f_2(x),\dots, f_k(x))^t$.

A continuación se darán algunas definiciones para comprender los tipos de soluciones que interesa alcanzar en la optimización multiobjetivo.

1.3.1.1 Dominancia y optimalidad de Pareto

Para poder determinar cuándo una solución es mejor que otra en este tipo de problemas, se define el criterio de dominancia [47].

Dados dos vectores $x, y \in \mathbb{R}^n$, se dice que $x \leq y$, si $x_i \leq y_i$ para $i=1,\dots, n$, y x *domina* a y (denotado por $x \prec y$) si $x \leq y$ y $x \neq y$.

Un vector de variables de decisión $x \in X \subset \mathbb{R}^n$ es *no-dominado* con respecto a X, si no existe otro $x' \in X$ tal que $f(x') \prec f(x)$.

Se dice que un vector de variables de decisión $x^* \in S \subset \Re^n$ (con S la región factible) es *Pareto-optimal* si es no-dominado con respecto a S.

El *Conjunto Pareto Optimal* P^* se define como:

$$P^* = \{x \in S \mid x \text{ es Pareto-optimal}\}$$

El *frente de Pareto* FP^* está definido por:

$$FP^* = \{f(x) \in \Re^n \mid x \in P^*\}$$

Es importante destacar que en los problemas de optimización multiobjetivo la cardinalidad del Conjunto Pareto Optimal solamente es uno si las funciones objetivo no son antagónicas o conflictivas, lo cual no es común que ocurra. De no ser así, la solución óptima correspondiente a uno solo de los objetivos será también óptima respecto a los otros [46].

Luego, para el problema de clasificación automática, debe determinarse el Conjunto Pareto-Optimal –en lo adelante conjunto de soluciones no-dominadas-. Esto es, el conjunto de las mejores particiones, para a partir de estas aplicar algún criterio de selección y obtener una solución final.

2 Aplicación de Optimización con Colonia de Hormigas al problema de clasificación

Para resolver el problema de clasificación se desarrolló el algoritmo CSOCH (Clasificación Supervisada basada en Optimización con Colonia de Hormigas), inspirado en el Sistema de Colonias de Hormigas (ACS) [35] y con antecedentes en [14, 48-50], que permite clasificar un conjunto de elementos en *K* clases.

Este algoritmo será utilizado posteriormente como parte de la estrategia propuesta en el capítulo tres para la resolución del problema de clasificación automática.

2.1 Descripción del algoritmo

En el algoritmo que se propone se asocian las hormigas con los elementos del conjunto *Γ* a clasificar, por lo que en cada iteración se trabajará con una colonia de *n* hormigas.

En una fase de inicialización se garantiza que sean creadas *K* clases. Para ello se ubica un elemento en cada clase, pudiendo realizarse este proceso de manera aleatoria o de acuerdo a algún otro criterio como se expondrá en la sección 2.1.1. Nótese que la forma en que se realiza esta asignación puede influir en el comportamiento del algoritmo.

Durante el proceso de clasificación cada hormiga (elemento) selecciona a qué clase incorporarse utilizando una regla *seudoaleatoria proporcional.*

La probabilidad de que la hormiga *j* decida incorporarse a la clase C_k, que ya contiene al menos un elemento se define como:

$$p_{kj} = \frac{\left[\tau_j^k(t)\right]^{\alpha}\left[\eta_j^k(t)\right]^{\beta}}{\sum_{i|x_{i\in clasificados}}\left[\tau_j^k(t)\right]^{\alpha}\left[\eta_j^k(t)\right]^{\beta}} \qquad (2.1)$$

donde $\tau_j^k(t)$ y $\eta_j^k(t)$ representan la feromona y la información heurística, respectivamente, asociadas al arco virtual que conecta a la hormiga x_j con la clase C_k en el instante *t*[15] y se calculan de acuerdo a las expresiones:

[15] Entiéndase por instante *t* el momento en que el elemento x_j está decidiendo a qué clase incorporarse, puesto que cada vez que un elemento se incorpora a una clase se actualiza el centro de gravedad de la misma, lo cual influye en los valores de τ_j^k y η_j^k.

$$\tau_j^k(t) = \sum_{i|x_i \in C_k} \tau_{ij}(t) \tag{2.2}$$

$$\eta_j^k(t) = \frac{1}{d(x_j, g_k)} \tag{2.3}$$

En la ecuación (2.2) la expresión $\tau_{ij}(t)$ representa la feromona asociada al arco que conecta los elementos x_i y x_j. Es decir, se asocia al arco virtual que conecta a la hormiga x_j con la clase C_k, la suma de los niveles de feromona sobre los arcos que conectan a x_j con los elementos ya ubicados en la clase.

En (2.3) se define $\eta_j^k(t)$ que constituye una estimación del costo de la decisión de una hormiga x_j de incorporarse a la clase C_k, siendo g_k el centro de gravedad de la clase C_k en el momento t. Como puede apreciarse, mientras más lejos está el centro de gravedad de una clase del elemento a clasificar, más costosa será esa opción, o lo que es lo mismo, menos deseable será esta clase para dicho elemento.

Los valores α y β en (2.1) son parámetros que determinan la importancia de la feromona y la información heurística, respectivamente.

El criterio que se utiliza para escoger la clase m a la que se incorporará el elemento x_j es, de forma análoga a ACS, el siguiente (regla *seudoaleatoria proporcional* de transición de estado):

$$m = \begin{cases} argmax\left\{\left[\tau_j^k(t)\right]^\alpha \left[\eta_j^k(t)\right]^\beta\right\} & si\ q \le q_0 \\ aleatorio\ utilizando\ p_{kj} & si\ q > q_0 \end{cases} \tag{2.4}$$

donde q es un número aleatorio uniforme en el intervalo [0,1] y q_0 es un parámetro que toma valores en el mismo intervalo. Esta regla provee una forma directa de equilibrio entre la exploración de nuevos estados y la explotación del conocimiento a priori y acumulado [51].

Otro aspecto que debe ser adecuado al problema de clasificación es la actualización de los niveles de feromona sobre los arcos virtuales que conectan elementos. Este proceso se realiza de acuerdo a la siguiente expresión:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij} \tag{2.5}$$

donde ρ es un coeficiente de persistencia de la feromona –parámetro de entrada- y $\Delta\tau_{ij}$ es la cantidad de feromona que se deposita en la iteración correspondiente sobre los arcos que conectan observaciones clasificadas en la misma clase.

```
//Entrada de parámetros del algoritmo
Entrar α, β, q0, ρ, Q, subciclos, ciclos

clasif = ϕ  //variable donde se guardará la mejor clasificación
wMin= infinito    //variable donde se guardará el mejor valor de W

Para c = 1,…, ciclos
   wMinCol = infinito  //variable donde se guardará el mejor valor
                       de W en un grupo de colonias
   Para s = 1,…, subciclos
      Inicializar clases
      Para i = 1,…, cantidadDeElementos //ciclo de una colonia
         Seleccionar aleatoriamente un elemento xj no clasificado
         Para cada clase Ck
            Calcular τj^k según (2.2)
            Calcular ηj^k según (2.3)
            Calcular Pkj según (2.1)
         Fin para
         Generar q aleatoriamente
         Seleccionar clase m de acuerdo a (2.4)
         Adicionar elemento xj a la clase seleccionada
         Actualizar centro de gravedad de la clase seleccionada
      Fin para (i)//fin de una clasificación (colonia de hormigas)
      Calcular W
      Si W < wMinCol
         wMinCol = W
      Fin si
      Si W < wMin
         wMin = W
         clasif = clasifActual
      Fin si
   Fin para (s)//fin de un grupo de colonias de hormigas (subciclo)
   Si variante reubicación
      Ejecutar algoritmo Reubicación
   Fin si
   Actualizar matriz de feromona
Fin para (c) //fin de un ciclo
```

Figura 5: Seudocódigo del algoritmo de clasificación

La actualización puede realizarse al concluir cada iteración o al final de un ciclo con una cantidad de iteraciones prefijada, a la cual se identificará como *subciclos*. En este

caso se actualiza utilizando la mejor partición obtenida de entre todas las colonias del ciclo.

La cantidad $\Delta\tau_{ij}$ puede definirse atendiendo a diferentes criterios. En 2.1.2 se presentan algunos de ellos.

En la figura 5 puede verse el seudocódigo del algoritmo descrito en esta sección.

A continuación se exponen algunas variantes que se tuvieron en cuenta sobre ciertos aspectos del algoritmo descrito.

2.1.1 Asignación inicial

Teniendo en cuenta que la asignación inicial de forma aleatoria puede ubicar en clases diferentes a elementos que debieran encontrarse en la misma clase en una partición óptima, se consideran otras variantes para realizar la asignación inicial.

2.1.1.1 Centros de gravedad iniciales distantes

Primeramente se consideró que podrían obtenerse mejores resultados si en lugar de hacer la asignación inicial de manera totalmente aleatoria, se escogían para ser ubicados inicialmente elementos bien separados entre sí. Para ello, se asigna un primer elemento seleccionado aleatoriamente a una clase vacía. Luego se escoge entre los restantes el elemento más alejado de este y se le asigna a otra clase. A continuación, se calcula el centro de gravedad de los elementos seleccionados y se determina entre los restantes el más alejado de dicho centro para asignarlo a la clase siguiente. Y así sucesivamente, los elementos asignados a cada clase se escogen siempre como los más alejados del centro de gravedad de aquellos que ya han sido asignados –nótese que al inicio, cuando solamente se ha ubicado un elemento, el centro de gravedad corresponde a él mismo-.

Aquí es importante que el primer elemento que se ubique no sea el mismo en todas las clasificaciones, ya que de esta forma siempre se haría la misma asignación inicial, a menos que hayan elementos equidistantes a alguno(s) de los centros de gravedad en cuestión. La selección aleatoria del primer elemento es suficiente para evitar que esto suceda.

Aunque con esta variante se intenta no separar desde el principio observaciones que deben quedar clasificadas en la misma clase, no se puede asegurar que se resuelva

este problema definitivamente ya que no necesariamente el elemento que se escoge para ser asignado debe ir fuera de las clases que ya contienen algún elemento.

2.1.1.2 Con información previa

Por otra parte se consideró la posibilidad de que existiese un conocimiento previo del conjunto de datos con el que se trabaja. En ocasiones la clasificación no se realiza completamente a ciegas, sino que se tiene información sobre algunos elementos –por ejemplo, se puede saber de antemano que hay dos elementos específicos que deben ir en clases separadas, o juntos-. En este caso los elementos que se sabe pertenecen a la misma clase se colocan juntos, y de la misma manera los que deben ir separados se colocan en clases diferentes. De esta forma se minimiza el número de posibles particiones, y aumenta la probabilidad de que se obtenga la mejor.

2.1.2 Actualización de la feromona

Fueron implementados diferentes esquemas de actualización de la feromona, que se diferencian esencialmente por la magnitud de $\Delta\tau_{ij}(t, t+1)$.

- **Incremento constante**

En este caso se incrementa el valor de la feromona de cada arco conectando elementos de la misma clase en la partición construida, en una magnitud constante prefijada previamente, es decir:

$$\Delta\tau_{ij}(t, t+1) = Q \tag{2.6}$$

- **Incremento proporcional a la inercia de la clase**

El incremento de la feromona sobre los arcos conectando elementos x_i, x_j pertenecientes a una misma clase C_k en la partición construida en la iteración t+1, es proporcional a la inercia de dicha clase, esto es:

$$\Delta\tau_{ij}(t, t+1) = \mathrm{Q}/L \tag{2.7}$$

siendo *Q* una constante positiva y

$$L = \frac{1}{|C_k|} \sum_{i|x_i \in C_k} d(x_i, g_k)^2 \tag{2.8}$$

la inercia de la clase C_k.

- **Incremento proporcional a la inercia de la partición**

En esta variante se utiliza también la expresión (2.7) pero el valor de *L* será la inercia intraclase de la partición, es decir:

$$L = W = \frac{1}{n}\sum_{k=1}^{K}\sum_{i|i\in C_k} d(x_i, g_k)^2 \qquad (2.9)$$

De acuerdo a esta expresión, la cantidad de feromona que se deposita sobre cada arco conectando elementos ubicados en la misma clase es la misma, independientemente de la clase de que se trate.

2.1.3 Reubicación de elementos

En la implementación de metaheurísticas en la actualidad es usual encontrar algoritmos híbridos –incorporan elementos de otras metaheurísticas o métodos deterministas- que obtienen buenos resultados. En [52] pueden verse algunos ejemplos.

Para el algoritmo descrito en el presente capítulo se incluyó una fase adicional de reubicación de los elementos similar a la que se realiza en el método de transferencias, que realiza movimientos de elementos de una clase a otra si se cumple cierta condición.

Una vez obtenida la mejor clasificación de un grupo de colonias de hormigas, para cada clase se valora si pudiera ser conveniente mover un elemento –o varios- a alguna de las clases restantes. Se considera conveniente mover un elemento a otra clase si este se encuentra más próximo al centro de gravedad de la misma que al de la clase a la cual pertenece, en el momento de realizar dicho análisis. Además, la clase a la cual pertenece debe tener como mínimo un elemento más, para evitar que la misma desaparezca, pues la cantidad de clases se mantiene fija.

Cada vez que un elemento es movido se actualizan los centros de gravedad de las clases origen (g_i) y destino (g_j) de acuerdo a las ecuaciones 2.10 y 2.11. Cabe destacar que esta forma de actualización es eficiente, ya que no implica recorrer todos los elementos del conjunto, sino que se basa en el cambio que supone para un centro de gravedad la adición o sustracción de un elemento.

$$g_i = \frac{|C_i| * g_i + x}{|C_i| + 1} \qquad (2.10)$$

$$g_j = \frac{|C_j| * g_j - x}{|C_j| - 1} \tag{2.11}$$

En la figura 6 se muestra el seudocódigo del algoritmo para la reubicación de los elementos.

```
cantCambios = 0
Para cada clase Ci
   Si |Ci| > 1
       cambios = falso
       j = 0
       Mientras j < |Ci| y cambios == falso
          menorDist = Distancia(xj, gi) // xj elemento y gi centro de
                                        gravedad de la clase origen
          k = 0
          Mientras k < cantidadDeClases y cambios == falso
              Si k ≠ i
                   d = Distancia(xj, gk) // gk centro de gravedad de la
                                            clase destino
                   Si d < menorDist
                       cambios = verdadero
                       Mover elemento xj a clase Ck
                       Actualizar centro de la clase Ci
                       Actualizar centro de la clase Ck
                       Incrementar cantCambios
                   Fin si
              Fin si
              Incrementar k
          Fin mientras
          Incrementar j
       Fin mientras
   Fin si
Fin para
Si cantCambios ≠ 0
   Calcular W
   Si W < wMinCol  //si se mejora el mejor valor alcanzado en una
                    familia de hormigas
       wMinCol = W
   Fin si
   Si W < wMin   //si se mejora el mejor valor global
       wMin = W
       clasif = clasifActual
   Fin si
Fin si
```

Figura 6: Seudocódigo del algoritmo de reubicación

2.2 Configuración automática de parámetros

Un aspecto importante en el trabajo con metaheurísticas es la configuración de los parámetros, dado que una inadecuada selección de los valores que se asignan a los mismos puede conllevar a la obtención de resultados malos, con independencia de las dificultades propias del problema que se trata de resolver, lo cual podría propiciar la realización de valoraciones erróneas sobre el funcionamiento del algoritmo –o los algoritmos- en cuestión, y su pertinencia en la resolución del problema.

La configuración manual de los parámetros es, por lo general, un proceso engorroso, además de consumir mucho tiempo. En este trabajo se utilizó un algoritmo para llevar a cabo la configuración de algunos de los parámetros de manera automática, incluyendo en la aplicación desarrollada una funcionalidad para realizar esta tarea.

El algoritmo utilizado, F-Race [53], se basa en el test de Friedman, un método estadístico para pruebas de hipótesis. Es un algoritmo de los llamados "de carrera", que evalúa, en diferentes instancias de un problema[16], un conjunto de diferentes configuraciones –entiéndase como una configuración la asignación a cada uno de los parámetros de un valor perteneciente a un conjunto de valores posibles-. Durante el proceso se van eliminando aquellas configuraciones que resultan menos prometedoras de acuerdo a pruebas estadísticas realizadas. De este modo no se emplean demasiados recursos computacionales evaluando configuraciones que no resultan competitivas, mientras que las configuraciones que no son descartadas –las que permanecen en la carrera- son evaluadas en una mayor cantidad de instancias, con lo que se logra que sea más confiable el resultado.

En las secciones siguientes se describirá el problema de encontrar la configuración óptima de los parámetros –problema de configuración- y el algoritmo F-Race, de acuerdo a [53], como se expone en [14].

2.2.1 Definición del problema de configuración

Para definir formalmente el problema de configuración Birattari et al. consideran los siguientes elementos:

- Θ es el conjunto finito de configuraciones candidatas

[16] Llámese instancias de un problema a un conjunto de juegos de datos que tienen características similares entre ellos, como pueden ser la cantidad de elementos, el tamaño de los mismos y el número de clases en las que serán agrupados.

- I es el conjunto de instancias, que puede ser infinito
- P_I es una medida de probabilidad sobre el conjunto de instancias I: con cierto abuso de notación se indicará con $P_I(i)$ la probabilidad de que la instancia i sea seleccionada para ser resuelta[17]
- $t: I \rightarrow \mathbb{R}$ es una función que asocia a cada instancia un tiempo de cómputo
- $c(\theta, i) = c(\theta, i, t(i))$ es una variable aleatoria que representa el costo de la mejor solución encontrada al ejecutar la configuración θ en la instancia i durante $t(i)$ segundos[18]
- $C \subset \mathbb{R}$ es el rango de c, esto es, los posibles valores del costo de la mejor solución encontrada en una corrida de la configuración $\theta \in \Theta$ en una instancia $i \in I$
- P_C es una medida de probabilidad sobre el conjunto C: con la notación[19] $P_C(c|\theta,i)$, se indica la probabilidad de que c sea el costo de la mejor solución encontrada al correr durante $t(i)$ segundos la configuración θ en la instancia i
- $C(\theta) = C(\theta|\Theta, I, P_I, P_C, t)$ es el criterio a ser optimizado con respecto a θ. En el caso más general es de cierta manera una medida de la deseabilidad de θ

A partir de estos conceptos, el problema de configuración de una metaheurística puede ser formalmente descrito por el séxtuplo $\langle \Theta, I, P_I, P_C, t, C \rangle$. La solución de este problema es la configuración θ^* tal que:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, C(\theta) \qquad (2.12)$$

Para la definición del algoritmo, los autores consideraron la optimización del valor esperado del costo $c(\theta, i)$, dado que este criterio es adoptado en muchas aplicaciones diferentes y, además de ser muy natural, suele ser muy conveniente desde los puntos de vista teórico y práctico. Formalmente:

$$\mathrm{C}(\theta) = \mathrm{E}_{\mathrm{I,C}}[\mathrm{c}(\theta, \mathrm{i})] = \int_I \int_C c(\theta, i) dP_C(c|\theta, i) dP_I(i) \qquad (2.13)$$

donde el valor esperado es considerado con respecto a P_I y a P_C, y las integrales son tomadas en el sentido de Lebesgue [54].

[17] Dado que una medida de probabilidad se asocia a (sub)conjuntos y no a elementos, la notación correcta debería ser $P_I(\{i\})$. El abuso de notación consiste entonces en usar el mismo símbolo i tanto para el elemento $i \in I$ como para el conjunto unitario $\{i\} \subset I$.
[18] En lo adelante, para una notación más clara, la dependencia de c con t estará implícita.
[19] Es válida aquí la aclaración de la nota 17.

Usualmente no se dispone de las medidas P_I y P_C explícitamente, y la solución analítica de las integrales en la ecuación 2.13, una para cada configuración θ, no es posible. Para superar tal limitante dichas integrales fueron estimadas sobre la base de un conjunto de instancias, como se explica más adelante.

2.2.2 Algoritmos de carrera

Supóngase que se tiene una secuencia aleatoria de instancias de entrenamiento *i*, donde el término genérico k-ésimo i_k es seleccionado en *I* de acuerdo a P_I, independientemente para cada *k*. Se asume que *i* puede ser extendida a voluntad a un costo insignificante, a partir de las instancias de *I*.

Se denotará por $c^k(\theta, i)$ un arreglo de *k* términos, donde el m-ésimo corresponde al costo $c(\theta, i_m)$ de la mejor solución encontrada por la configuración θ en la instancia i_m en una corrida de $t(i_m)$ segundos. De ahí que, para una θ dada, el arreglo c^k puede obtenerse a partir de c^{k-1} adjuntando a este último el costo correspondiente a la k-ésima instancia en *i*.

Un algoritmo de carrera afronta el problema de optimización definido por la ecuación 2.12 generando una secuencia de conjuntos anidados de configuraciones candidatas: $\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq ...$, comenzando por $\Theta_0 = \Theta$. Al pasar de un conjunto Θ_{k-1} a Θ_k se pueden eliminar algunas configuraciones, al parecer suboptimales de acuerdo a la información disponible en el paso *k*.

En el paso *k*, cuando el conjunto de candidatas que permanecen aún en competencia es Θ_{k-1}, es considerada una nueva instancia i_k. Cada configuración candidata $\theta \in \Theta_{k-1}$ es ejecutada en i_k y cada costo observado $c(\theta, i_k)$ es adjuntado al correspondiente c^{k-1} para formar los diferentes arreglos $c^k(\theta, i)$, uno para cada θ. El paso *k* termina definiendo el conjunto Θ_k eliminando de Θ_{k-1} las configuraciones suboptimales a la luz de alguna prueba estadística que compare los arreglos $c^k(\theta, i)$ para toda $\theta \in \Theta$. Cabe destacar que, para cualquier θ, cada componente del arreglo $c^k(\theta, i)$, o sea, cualquier costo $c(\theta, i)$ de la mejor solución encontrada por una corrida de θ sobre una instancia *i* genérica extraída de acuerdo a P_I, es un estimado de $C(\theta)$, tal como se define en la ecuación 2.13. La muestra promedio de $c^k(\theta, i)$ es entonces, en sí misma un estimado de $C(\theta)$ y puede ser usada para comparar el rendimiento alcanzado por las diferentes configuraciones.

Este proceso se lleva a cabo iterativamente y se detiene cuando todas las configuraciones excepto una han sido eliminadas, o cuando se alcanza un tiempo total de cómputo predefinido t_{max}. O sea, el proceso podría detenerse antes de considerar la instancia (k + 1)-ésima si $\sum_{m=1}^{k} t(i_{m+1})|\Theta_m| > t_{max}$

2.2.3 Prueba estadística en el algoritmo

Como se vio en los apartados anteriores, los algoritmos de carrera valoran la posibilidad de descartar configuraciones cuando a la luz de alguna prueba estadística se logra evidenciar que estas clasifican como suboptimales. En el caso de F-Race, los autores utilizan la prueba de Friedman [55]. Esta prueba se usa con frecuencia en la realización de estudios estadísticos, lo cual se debe en primer lugar a su carácter no paramétrico. Los métodos de este tipo no hacen suposiciones acerca de la distribución de los datos –por ejemplo, normalidad-, lo cual permite su aplicabilidad en múltiples situaciones.

A continuación se describe cómo interviene la prueba de Friedman en F-Race. Supóngase que el algoritmo ha alcanzado el paso k, y quedan aún $n=|\Theta_{k-1}|$ configuraciones en competencia. La prueba de Friedman asume que los costos observados hasta este paso son k variables aleatorias n-variadas mutuamente independientes $(c^k(\theta_1,i_m), c^k(\theta_2,i_m),\ldots, c^k(\theta_n,i_m))$ llamadas bloques [56] correspondiendo al bloque k-ésimo los costos obtenidos al evaluar en la instancia i_m cada una de las configuraciones θ_i que permanecen en la carrera en el paso k. Dentro de cada bloque las cantidades $c^k(\theta,i_m)$ son ordenadas de menor a mayor, obteniéndose los llamados rangos, usándose valores promedio en caso de que coincidan algunos de ellos. Para cada configuración $\theta_j \in \Theta_{k-1}$ sea R_{mj} el rango de θ_j dentro del bloque m y $R_j = \sum_{m=1}^{k} R_{mj}$ la suma de los rangos sobre todas las instancias i_m con $1 \leqslant m \leqslant k$. La prueba de Friedman considera el siguiente estadístico [55]:

$$T = \frac{(n-1)\sum_{j=1}^{n}\left(R_j - \frac{k(n+1)}{2}\right)^2}{\sum_{m=1}^{k}\sum_{j=1}^{n} R_{mj}^2 - \frac{kn(n+1)^2}{4}} \qquad (2.14)$$

Bajo la hipótesis nula de que todos los rangos de los candidatos dentro de cada bloque son igualmente seleccionables –esto es, todas las configuraciones tienen el mismo efecto sobre el costo que se estudia-, T distribuye aproximadamente X^2 con n−1 grados de libertad. Si el valor de T observado sobrepasa el cuartil 1-α de esta distribución, se

rechaza la hipótesis nula al nivel α aproximadamente, en favor de la hipótesis de que al menos un candidato tiende a alcanzar mejores resultados.

Si la hipótesis nula es rechazada, se justifica la realización de comparaciones entre candidatos. Las candidatas θ_j y θ_h se consideran diferentes si:

$$\frac{\left|R_j - R_h\right|}{\sqrt{\frac{2k\left(1-\frac{T}{k(n-1)}\right)\left(\sum_{m=1}^{k}\sum_{j=1}^{n}R_{mj}^2 - \frac{kn(n+1)^2}{4}\right)}{(k-1)(n-1)}}} > t_{1-\alpha/2} \qquad (2.15)$$

donde $t_{1-\alpha/2}$ es el $1-\alpha/2$ cuartil de la distribución t de Student [55].

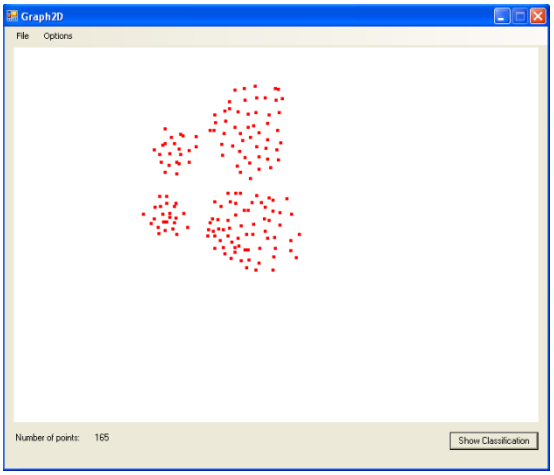
Luego, en F-Race, si en el paso k la hipótesis nula no es rechazada, todas las configuraciones candidatas en Θ_{k-1} pasan a Θ_k. Por otra parte, si es rechazada se hacen comparaciones entre las configuraciones candidatas que quedan para detectar aquellas que presentan una diferencia significativa con respecto a la mejor, o a las mejores, en caso de que haya más de una. Todas aquellas que resulten significativamente peores que la mejor se descartan y no pasan a formar parte de Θ_k.

Para más detalles respecto a dicho algoritmo pueden consultarse [14, 50, 53, 57-59].

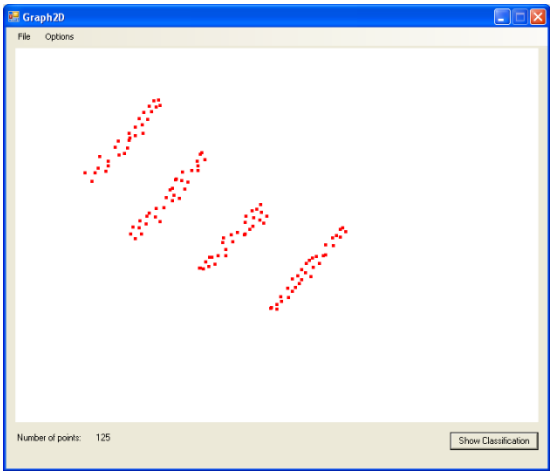
2.3 Resultados experimentales

Para el estudio del comportamiento del algoritmo de clasificación implementado, se utilizaron problemas de $\mathbb{R}^2$ generados con el apoyo de una herramienta gráfica incluida en la aplicación desarrollada, y tres problemas clásicos de la literatura sobre clasificación: los peces de Amiard, la sociomatriz de Thomas y los irises de Fisher.

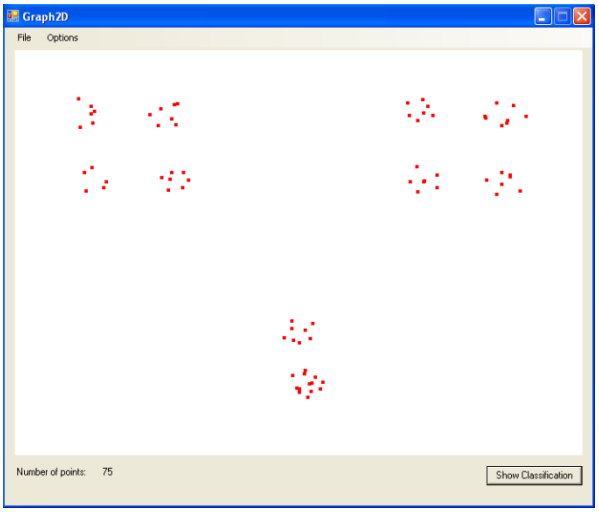
El problema de los peces de Amiard consiste en la clasificación de un conjunto de 23 peces sobre los que se han medido nueve variables acerca de la radioactividad en ciertos órganos y siete características físicas, conformando una matriz de 23×16. La sociomatriz de Thomas, de 24×24, contiene las notas de afinidad que se asignan 24 estudiantes de una clase, en una escala de 0 a 20. Iris de Fisher es quizás el juego de datos más conocido en la literatura sobre reconocimiento de patrones, son las mediciones de cuatro características de 150 ejemplares de estas plantas, y contiene tres clases –correspondientes a las variedades Iris setosa, Iris versicolor e Iris virginica-, de 50 elementos cada una. En lo adelante se hará referencia a estos problemas como Amiard, Thomas e Iris, respectivamente.



a) Gráfico 1



b) Gráfico 2



c) Gráfico 3

Figura 7: Problemas de $\Re^2$

Los problemas de $\Re^2$ –denominados Gráfico 1, Gráfico 2 y Gráfico 3 en lo adelante- pueden verse en la figura 7. Fueron diseñados con el objetivo de observar el comportamiento del algoritmo para conjuntos de puntos con diferentes distribuciones espaciales.

La tabla 2.3.1 muestra un resumen de las características de los juegos de datos mencionados. Como puede apreciarse, la cantidad de clases en la mayoría de los problemas estudiados es variable, ya que pueden resultar interesantes diferentes particiones. Por ejemplo, para el problema de Amiard se reportan en la literatura resultados para dos, tres, cuatro y cinco clases. En el caso de los problemas de $\Re^2$, también es de interés estudiar más de una clasificación de acuerdo a la distribución

espacial de los datos. Se especifican entre paréntesis las cantidades de clases estudiadas, y la cantidad de elementos por clase que aparece en la quinta columna corresponde a clasificaciones con el número de clases indicado fuera de los paréntesis en la columna anterior.

Problema	Cantidad de elementos	Cantidad de atributos	Cantidad de clases	Cantidad de elementos por clase
Gráfico 1	165	2	4 (2 o 4)	22, 24, 52, 67
Gráfico 2	125	2	4	29, 31, 32, 33
Gráfico 3	75	2	3 (3, 5 o 10)	21, 25, 29
Amiard	23	16	3 (2-5)	2, 4, 17
Thomas	24	24	3 (3-5)	6, 8, 10
Iris	150	4	3 (2 o 3)	50×3

Tabla 2.3.1: Resumen de las características de los juegos de datos utilizados

Con respecto a los parámetros del algoritmo de clasificación –α, β, ρ, q_0, Q, cantidad de *ciclos* y de *subciclos*-, se decidió mantener constantes los valores de ρ, q_0 y cantidad de *ciclos*, variar Q en dependencia del problema a resolver y configurar automáticamente α, para valores en el conjunto {0,1; 0,5; 1}, β en {1; 2; 4} y cantidad de *subciclos* en {1; 5; 10}, teniendo en cuenta resultados previos sobre la influencia de dichos parámetros en el comportamiento de algoritmos de este tipo. De este modo, se obtuvieron 27 configuraciones, para seleccionar de forma automática las más convenientes, a partir de los resultados del algoritmo F-Race, descrito en la sección 2.2.

Con el objetivo de obtener una buena configuración para los parámetros que serían utilizados en la resolución de cada uno de los problemas, se generaron 20 instancias de problemas con características similares a las de los que se estudiarían (cantidad de clases, cantidad de elementos y rangos de valores de los atributos). En la tabla 2.3.2 se indican las configuraciones resultantes y en el anexo (Anexo 4) se muestran todas las configuraciones utilizadas en el proceso.

La tabla 2.3.3 muestra el porciento de éxito del algoritmo de clasificación, en 20 corridas, utilizando las configuraciones expuestas en la tabla anterior y con algunas de las restantes en dos de los problemas estudiados.

Las columnas sombreadas en la tabla 2.3.2 corresponden a los parámetros que se configuraron de manera automática. Como puede apreciarse, en el caso de la cantidad de subciclos se obtuvo siempre el valor mayor en el rango que se dio para dicho parámetro, de ahí que se pensara que aumentando los valores de la cantidad de ciclos

se podían alcanzar mejores resultados con respecto al porciento de éxito, aunque con un costo adicional en el tiempo de ejecución. Otro parámetro que se modificó fue *Q*, obteniéndose también buenos resultados.

Problema	*Configuración*	α	β	ρ	q_0	*Q*	*ciclos*	*subciclos*
Gráfico 1	c9	0,1	4	0,9	0,9	1 000	10	10
Gráfico 2	c6	0,1	2	0,9	0,9	1 000	10	10
	c9	0,1	4	0,9	0,9	1 000	10	10
	c15	0,5	2	0,9	0,9	1 000	10	10
	c18	0,5	4	0,9	0,9	1 000	10	10
Gráfico 3	c18	0,5	4	0,9	0,9	1 000	10	10
	c27	1	4	0,9	0,9	1 000	10	10
Amiard	c6	0,1	2	0,9	0,9	1 000	10	10
	c9	0,1	4	0,9	0,9	1 000	10	10
Thomas	c9	0,1	4	0,9	0,9	1 000	10	10
Iris	c9	0,1	4	0,9	0,9	1 000	10	10

Tabla 2.3.2: Configuraciones de los parámetros utilizadas en cada uno de los problemas

Problema	*Configuración*	% de éxito
Gráfico 3	**c18**	100
	c1	60
	c8	75
	c21	40
Iris	**c9**	100
	c1	55
	c15	80
	c20	10

Tabla 2.3.3: Ejemplos del desempeño del algoritmo de clasificación con las configuraciones obtenidas utilizando el algoritmo F-Race y con algunas de las restantes

Las tablas 2.3.4 y 2.3.5 muestran los resultados en 20 corridas del algoritmo de clasificación. Los valores de la fila denotada como “W mínima” corresponden al mínimo alcanzado para la función objetivo W –inercia intraclase-. Las filas restantes contienen el promedio de los valores de W en las 20 corridas, la desviación estándar, el porciento de éxito alcanzado en las mismas y el tiempo promedio de ejecución en segundos. En el caso de los problemas de la tabla 2.3.5 los valores del mínimo de W coinciden con el reportado en la literatura y para los problemas de la tabla 2.3.4, corresponden a los que se alcanzan cuando las clases obtenidas son las que resultan naturales a la vista del observador, como puede apreciarse en la figura 8. Las columnas de la derecha contienen los resultados con los valores de los parámetros que se señalan en la última fila (no se incluyen para los problemas en los que se logró 100% de éxito con la configuración inicial, en este caso Gráfico 3 e Iris), el resto de los parámetros se

mantiene igual. Como se puede apreciar en tales casos los porcientos de éxito fueron mejores, sin embargo los tiempos aumentaron.

En el anexo (Anexo 5) se pueden ver los resultados, con un formato similar, para estos problemas con las restantes cantidades de clases mostradas previamente en la cuarta columna de la tabla 2.3.1. Se pudo observar que el algoritmo tuvo más dificultades con el problema Thomas y con Gráfico 3 para 10 clases, mientras con el resto de los problemas los resultados fueron buenos.

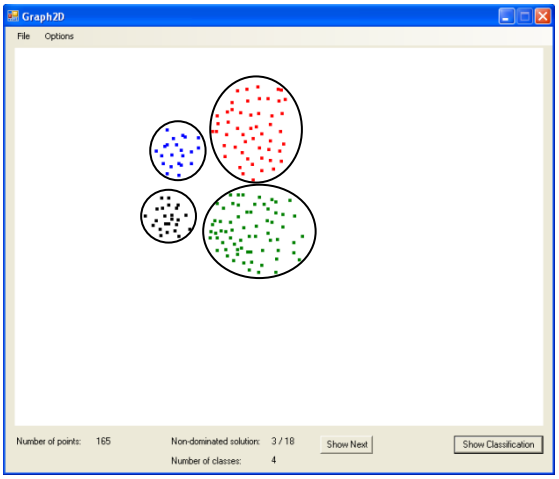
El proceso de experimentación se realizó en una computadora personal con procesador Intel (R) Pentium (R) D, 2,80 GHz, 512 MB RAM.

	Gráfico 1 (4 clases)	**Gráfico 2 (4 clases)**	**Gráfico 3 (3 clases)**		**Gráfico 1 (4 clases)**	**Gráfico 2 (4 clases)**
W mínima	1 361,18	1 700,17	3 280,33		1 361,18	1 700,17
W promedio	1 365,86	1 703,66	3 280,33		1 361,18	1 700,17
Desv. estándar	47,33	68,02	0		0	0
% de éxito	65	95	100		100	100
T. promedio (s)	0,42	0,25	0,10		0,95	0,50
					Q=1 300 ciclos=25	Q=1 700 ciclos=20

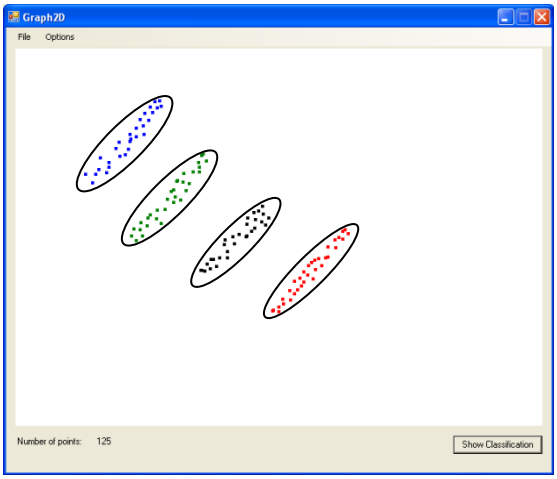
Tabla 2.3.4: Resultados en 20 corridas del algoritmo de clasificación para los problemas Gráfico 1, Gráfico 2 y Gráfico 3

	Amiard (3 clases)	**Thomas (3 clases)**	**Iris (3 clases)**		**Amiard (3 clases)**	**Thomas (3 clases)**
W mínima	32 213,38	271,83	0,521		32 213,38	271,83
W promedio	33 301,59	275,97	0,521		32 213,38	271,90
Desv. estándar	7 884,15	19,12	0		0	0,95
% de éxito	60	20	100		100	90
T. promedio (s)	0,09	0,07	0,35		0,36	0,67
					Q=32 200 ciclos=40	Q=200 ciclos=100

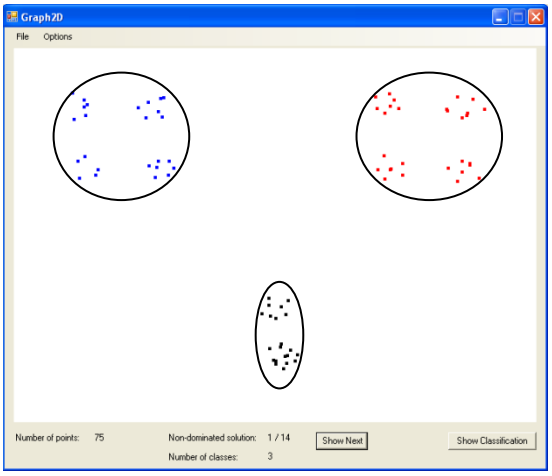
Tabla 2.3.5: Resultados en 20 corridas del algoritmo de clasificación para los problemas Amiard, Thomas e Iris



a) Gráfico 1 resuelto (cuatro clases)



b) Gráfico 2 resuelto (cuatro clases)



b) Gráfico 3 resuelto (tres clases)

Figura 8: Soluciones a los problemas de $\Re^2$

3 Clasificación automática como problema multiobjetivo

Para resolver el problema de clasificación de datos fijando previamente la cantidad de clases se vio que es posible utilizar de forma equivalente la inercia intraclase W o la inercia interclases B. Como se mencionó en el epígrafe 1.1.2.1 para el caso en que el número de clases no se fije de antemano la partición que minimiza la inercia intraclase maximiza también la inercia interclases. Se vio que esta partición es precisamente aquella que considera tantas clases como elementos a clasificar y clases formadas por un único elemento.

Sin embargo, resulta obvio que esta no es la partición deseada. Resulta de interés obtener una partición que identifique las características inherentes a los datos y presente un número de clases apropiado, que se corresponda con dicha estructura.

Pudiera pensarse en minimizar la inercia intraclase y para contrarrestar la consideración de tantas clases como observaciones, plantear la minimización también de la inercia interclases. Sin embargo, la condición (1.4) provoca que todas las particiones que se generen sean puntos eficientes, ya que la que tenga menor inercia intraclase tendrá mayor inercia interclases y por tanto serán puntos no comparables (no puede decirse que uno de los dos domina al otro).

Por las razones antes mencionadas, será necesario introducir otro criterio que permita resolver el problema de la identificación de un número apropiado de clases mediante un enfoque multiobjetivo.

3.1 Formulación del problema

Para enfrentar el problema de clasificación automática como un problema multiobjetivo se utiliza el criterio de minimizar la inercia intraclase (1.1.2.1), esto es:

$$minimizar\ W(P) = \frac{1}{n}\sum_{i=1}^{k}\sum_{x_j \in C_i} d(x_j, g_i)^2 \qquad (3.1)$$

Para contrarrestar el efecto indeseado que provocaría el objetivo anterior de definir clases formadas por uno o muy pocos elementos, se introduce una función que proporciona también disimilitud entre clases diferentes:

$$F(P) = \frac{1}{k}\sum_{i=1}^{k} \min_{\substack{1\le j\le k, \\ j\neq i}} \{d(g_i, g_j)\} \tag{3.2}$$

donde g_i y g_j corresponden a los centros de gravedad de las clases C_i y C_j respectivamente.

La función d puede ser alguna de las descritas en la sección 1.1.2.1. En el caso particular de la aplicación implementada se utilizó la distancia euclidiana:

$$d(x_a, x_b) = \sqrt{\sum_{j=1}^{m} |x_a(j) - x_b(j)|^2} \tag{3.3}$$

La función descrita en la ecuación (3.2) determina para cada clase la distancia a su clase más cercana, a través de las distancias entre sus centros de gravedad, para luego efectuar la suma entre todas estas distancias, de modo que si se quiere maximizar la disimilitud interclases este valor debe maximizarse.

Una vez definidas estas funciones puede formularse el problema como:

$$\begin{array}{l} \textit{minimizar } W \\ \textit{maximizar } F \end{array} \tag{3.4}$$

3.2 Medida de calidad para una partición

Con el objetivo de evaluar la calidad de las particiones obtenidas –el conjunto de soluciones no-dominadas del problema-, se usará el índice de Silhouette [60, 61].

Para determinar dicho índice en una partición, es preciso determinar para cada elemento el valor $s(x_i)$, denominado anchura de Silhouette, que mide el grado de pertenencia del elemento x_i a la clase C_j a la cual fue asignado:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}} \tag{3.5}$$

donde:

$a(x_i)$ es la distancia promedio entre el elemento x_i y todos los demás elementos de la clase a la cual este pertenece –en la figura 9, esta sería la longitud promedio de todas las líneas dentro de C_1-,

$\mathrm{b}(x_i) = \min_{r\neq j}\{d(x_i, C_r)\}$,

siendo $d(x_i, C_r)$ la distancia promedio entre el elemento x_i y los elementos de la clase C_r.

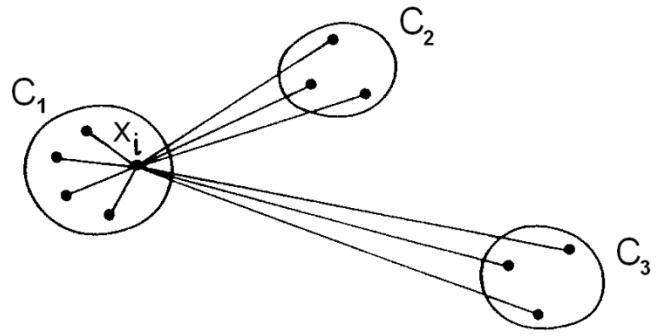


Figura 9: Ilustración de los elementos involucrados en el cálculo de s(i), donde el objeto x_i pertenece a la clase C_1. Fuente [41]

O sea, se determinan las distancias promedio entre x_i y todas las clases excepto la que lo contiene, y $b(x_i)$ correspondería al menor de estos valores. En la figura, $b(x_i)$ sería la longitud promedio de todas las líneas que van de x_i a C_2.

De acuerdo a la expresión (3.5): $-1 \leq s(x_i) \leq 1$. Cuando el valor es cercano a 1, puede decirse que el elemento x_i ha sido bien clasificado. Si es cercano a 0, el elemento podría asignarse a la clase más cercana. En caso de estar próximo a -1, entonces puede afirmarse que el elemento ha sido mal clasificado [61].

El valor $s(x_i)$ se utiliza para determinar el índice de Silhouette de cada clase:

$$S_j = \frac{1}{|C_j|}\sum_{i=1}^{|C_j|} s(x_i) \tag{3.6}$$

Luego, se determina el índice de Silhouette global como:

$$GS = \frac{1}{k}\sum_{j=1}^{k} S_j \tag{3.7}$$

siendo k la cantidad de clases de la partición.

En los resultados del proceso de clasificación automática se incluyen los valores de este índice para cada una de las particiones obtenidas, destacándose aquella con mayor valor del mismo.

3.3 Estrategia propuesta para resolver el problema de clasificación automática

La estrategia que se propone consiste en resolver en cada iteración un problema de clasificación supervisada. Estos problemas a su vez son resueltos utilizando el algoritmo de Optimización con Colonia de Hormigas (CSOCH) propuesto en el capítulo dos.

Como parámetros de entrada se tienen *kMin* y *kMax*, cota inferior y superior respectivamente de la cantidad de clases a identificar en el conjunto de observaciones. Otro parámetro de entrada del algoritmo es *iter*, la cantidad de iteraciones a realizar. Este parámetro determina la cantidad de veces que se aplicará el algoritmo para clasificación supervisada.

A partir de los parámetros de entrada se puede calcular M=*iter/(3*(kMax-kMin+1))*. Para cada valor entero entre *kMin* y *kMax* se aplica *M* veces el algoritmo CSOCH. Es decir, se generan *M* particiones con *kMin* clases, *M* particiones con *kMin+1* clases y así sucesivamente hasta *kMax*.

De las particiones generadas se almacena en el arreglo listaDeNoDominadas aquellas que son no-dominadas. Para ello, cada partición generada debe ser comparada con todas las que se encuentran en la lista, teniendo en cuenta las funciones W y F definidas en la sección 3.1 –ecuaciones 3.1 y 3.2-, bajo el criterio de dominancia dado en la sección 1.3.1.1. Si la partición generada en una iteración es dominada por alguna de las particiones que se encuentran en listaDeNoDominadas se desecha y se pasa a la siguiente iteración. Si por el contrario la partición generada domina a alguna partición de la lista, entonces la partición dominada se elimina de la lista y la nueva partición generada se añade a la misma. Si la partición generada en determinada iteración no es comparable con ninguna de las soluciones de la lista entonces se añade a la misma.

Al concluir un tercio del total de iteraciones a realizar, se revisa la lista de soluciones no-dominadas, con el objetivo de saber la cantidad de particiones que posee para cada número de clases posible.

Además de la lista mencionada, se registra también la cantidad de soluciones no-dominadas que se van obteniendo (arreglo cantXclase) por cada cantidad de clases en el rango dado, para posteriormente calcular una distribución de probabilidades que

será utilizada en la determinación de la cantidad de clases que tendrán las particiones que se generen en las iteraciones de la segunda fase.

Una vez obtenido un primer conjunto de soluciones no-dominadas, se calculan los valores de p_k, la distribución de probabilidad mencionada, de la siguiente forma:

$$p_k = \frac{cantidadNoDominadasConKclases}{cantidadNoDominadas}$$

donde *cantidadNoDominadasConKClases* corresponde a la cantidad de soluciones no-dominadas que se obtuvieron con una cantidad *K* de clases, y *cantidadNoDominadas* corresponde a la cantidad total de soluciones no-dominadas, en ambos casos durante la primera fase.

Ya en la segunda fase, en cada iteración se genera entonces un número aleatorio y se escoge, con probabilidad p_k –esto puede hacerse aplicando el método de la ruleta-, la cantidad de clases que tendrá la solución que se generará en la iteración usando el algoritmo CSOCH, del mismo modo que en la fase anterior.

Nótese que aunque en la primera fase se genera el mismo número de soluciones para cada cantidad *K* de clases, el número de soluciones no-dominadas puede ser –se espera que sea- diferente para cada valor de *K*. De ser iguales, entonces los valores de las probabilidades p_k también lo serán, y en la segunda fase existirá la misma probabilidad para cada cantidad de clases de ser escogida para que la solución que se genere tenga ese valor de *K*.

Finalmente se calculan los índices globales de Silhouette para cada una de las particiones que resultaron soluciones no-dominadas para el problema, valores que se ofrecerán como criterio adicional junto a cada solución.

Debe tenerse en cuenta que las soluciones obtenidas finalmente, son todas no-dominadas y por tanto pueden ser consideradas buenas soluciones –de acuerdo a las funciones objetivo que se utilizaron ninguna de estas soluciones es mejor que otra-. En el caso de conjuntos de elementos en $\mathbb{R}^2$, estas soluciones pueden visualizarse para una mejor apreciación.

```
//Entrada de parámetros
Entrar iter, kMin, kMax

cant = kMax-kMin + 1

cantXclase: int[cant] //para guardar la cantidad de no-dominadas
                                  por índice de clase
```

```
P: int[cant] //para guardar las probabilidades (selección de la
                  cantidad de clases de las soluciones a generar)
K = kMin
soluciónActual = ϕ
listaDeNoDominadas = ϕ
cantNoDominadas = 0

//Construcción de primeras soluciones
Mientras K ≤ kMax
   Para i=1,…, 1/3.iter.1/cant
(1)    Encontrar una clasificación de K clases con CSOCH[20]
(2)    Guardar solución obtenida en soluciónActual
(3)    Calcular W según (3.1)
(4)    Calcular F según (3.2)
(5)    Para cada solución en listaDeNoDominadas
(6)              Comparar solución con soluciónActual
(7)              Si solución domina a soluciónActual
(8)                    Salir del ciclo
(9)              Si no
(10)                   Si soluciónActual domina a solución
(11)                   Eliminar solución de listaDeNoDominadas
(12)                   Decrementar cantNoDominadas
(13)                   Fin si
(14)             Fin si
(15)   Fin para
(16)   Si soluciónActual es no-dominada
(17)             Agregar soluciónActual a listaDeNoDominadas
(18)             Incrementar cantNoDominadas
(19)             Incrementar cantXclase en la posición K
(20)   Fin si
   Fin para
   Incrementar K
Fin mientras

//Cálculo de probabilidades asociadas al número de clases para la
generación de nuevas soluciones:
Para i = kMin,…, kMax
      P[i] = cantXclase[i] / cantNoDominadas
Fin para

Calcular probabilidades acumulativas para selección aleatoria de
     la cantidad de clases con que se va a trabajar en la
     siguiente iteración (ruleta)

//Segunda fase
Para it = 1,…, 2/3.iter
      Generar num: número aleatorio uniforme en (0,1)
      Seleccionar K  según num (aplicando ruleta sobre P)
      Repetir de (1) a (20)
Fin para

Calcular el Índice de Silhouette para cada una de las particiones
     en la lista de no-dominadas.
```

Figura 10: Seudocódigo de la estrategia propuesta para resolver el problema de clasificación automática

[20] Algoritmo descrito en la sección 2.1.

3.4 Resultados experimentales

Para el estudio del algoritmo de clasificación automática presentado en la sección anterior, se utilizaron los juegos de datos Gráfico 1 y Gráfico 3 descritos en 2.3.

El problema Gráfico 3 presenta 3 posibles agrupaciones de 3, 5 y 10 clases (figuras 13, 14 y 15), que se pueden considerar como las más naturales, si bien existen otras con cantidades de clases diferentes que también pueden ser admitidas como aceptables.

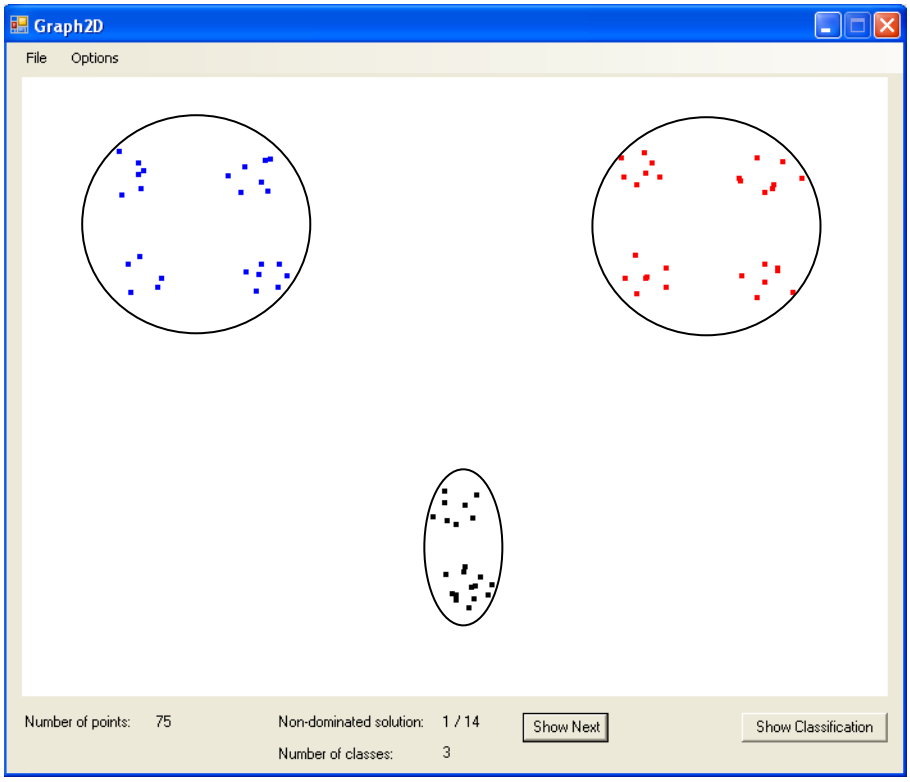


Figura 11: Partición de tres clases para Gráfico 3

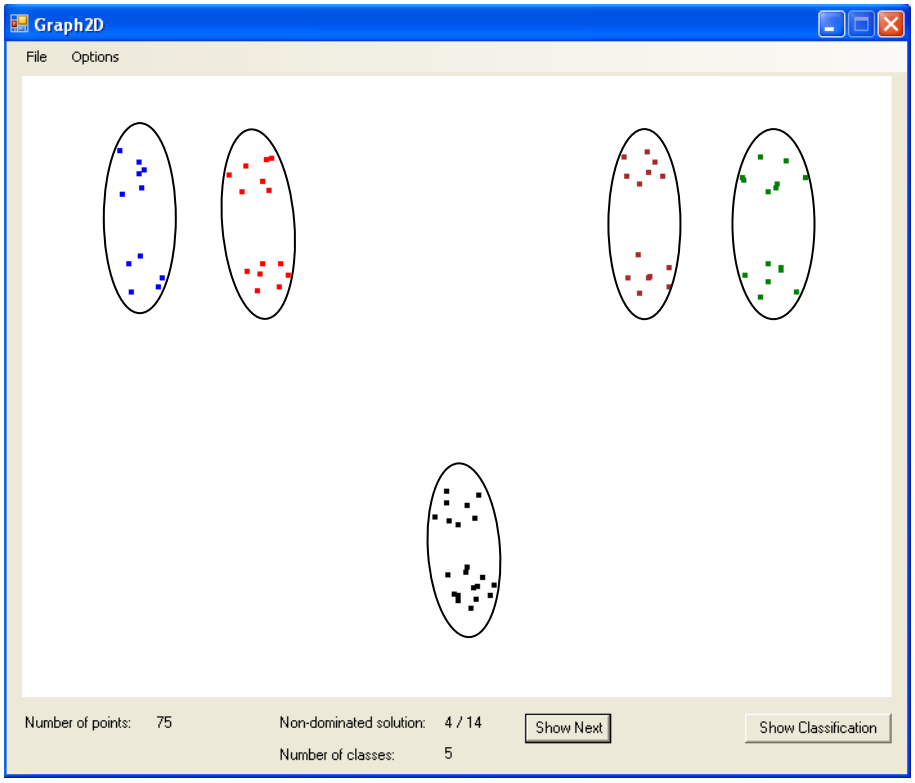


Figura 12: Partición de cinco clases para Gráfico 3

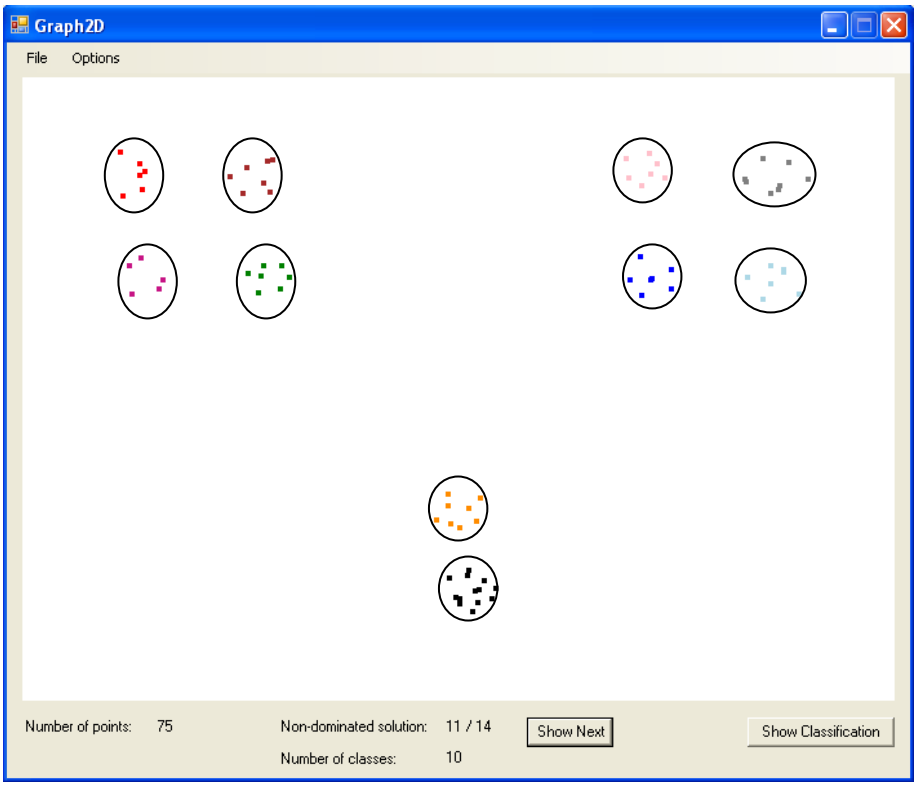


Figura 13: Partición de 10 clases para Gráfico 3

La tabla 3.4.1 muestra los resultados en 20 corridas del algoritmo para el problema Gráfico 3, con un rango de la cantidad de clases entre 2 y 10. En cada columna aparece el número de soluciones no-dominadas obtenidas por el algoritmo para la cantidad de clases que se indica en el encabezado. Se destacan las columnas correspondientes a 3, 5 y 10 clases, en las que se centrarán los análisis en lo adelante.

Para estas tres columnas se indica con las celdas sombreadas más oscuras que en la corrida correspondiente a esa fila se obtuvo, entre las soluciones no-dominadas, la clasificación óptima para la cantidad de clases señalada en el encabezado de la columna. Esto sucede en todos los casos para las soluciones de tres clases, con alta frecuencia para las de cinco y en pocas ocasiones para 10 clases, como refleja el porciento de éxito que se muestra en la tabla 3.4.2. O sea, para 10 clases la partición óptima se obtiene un número de veces mucho menor que para tres y cinco clases.

		Número de clases									
		2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Número de corrida	**1**	0	**1**	1	**4**	4	0	2	0	3	**15**
	2	0	**1**	1	**3**	1	2	0	0	2	**10**
	3	0	**1**	1	1	2	2	2	0	**1**	**10**
	4	0	**1**	1	**3**	4	3	0	1	1	**14**
	5	1	**1**	1	**2**	4	0	1	0	3	**13**
	6	1	**1**	1	**3**	3	3	1	1	1	**15**
	7	1	**1**	1	**2**	3	1	1	3	**1**	**14**
	8	1	**1**	1	**2**	3	3	2	0	**2**	**15**
	9	0	**1**	1	**4**	1	2	1	1	0	**11**
	10	0	**1**	1	**1**	2	1	1	1	1	**9**
	11	1	**1**	1	**4**	2	2	1	2	**2**	**16**
	12	1	**1**	1	**2**	3	3	2	1	**2**	**16**
	13	0	**1**	1	**3**	3	2	1	2	**1**	**14**
	14	0	**1**	1	**3**	1	0	0	2	0	**8**
	15	1	**1**	1	1	1	1	1	0	3	**10**
	16	0	**1**	1	2	1	1	1	2	1	**10**
	17	0	**1**	1	**3**	3	0	2	1	2	**13**
	18	1	**1**	1	**2**	2	5	1	1	2	**16**
	19	0	**1**	1	**2**	2	3	1	2	2	**14**
	20	1	**1**	1	**2**	2	3	0	4	0	**14**
	Total	**9**	**20**	**20**	**49**	**47**	**37**	**21**	**24**	**30**	**257**

Tabla 3.4.1: Distribución de las soluciones no-dominadas por cantidades de clases para el problema Gráfico 3 en 20 corridas de la estrategia propuesta para clasificación automática

En todos los casos la partición con mayor índice de Silhouette global fue la de tres clases (véase tabla 3.4.2), que como se puede apreciar en la tabla 3.4.1, fue además la única solución no-dominada con esa cantidad de clases en todas las corridas (en todas las filas aparece un uno, lo que indica que solamente se obtuvo una solución no-

dominada con esa cantidad de clases y en todos los casos está señalada más oscura la celda, o sea que fue la solución esperada). Esto indica que posiblemente entre las soluciones con ese número de clases no haya ninguna otra no-dominada, ya que todas las soluciones de tres clases que se obtuvieron durante la ejecución del algoritmo resultaron dominadas por esta.

	Gráfico 3		
	3 clases	**5 clases**	**10 clases**
W mínima	3 280,33	1 626,74	214,37
F máxima	324,76	136,20	76,65
GS máxima	0,79	0,61	0,76
% éxito	100	85	30

Tabla 3.4.2: Valores óptimos de W, F y GS para el problema Gráfico 3 con 3, 5 y 10 clases (W se minimiza, F y GS se maximizan) y porciento de éxito en 20 corridas

En la tabla anterior se muestran los valores óptimos alcanzados para las tres funciones utilizadas con cada una de las cantidades de clases estudiadas, además del porciento de éxito en 20 corridas. Nótese que a la partición de tres clases corresponde el mayor porciento de éxito y el mayor valor de GS, que mide el grado de pertenencia de los elementos a las clases.

Puede observarse también en la última fila de la tabla 3.4.1 que de estos tres grupos de soluciones la mayor cantidad de no-dominadas corresponde a particiones de cinco clases (49, contra 20 y 30 para 3 y 10 clases respectivamente). Esto puede deberse a que hay más posibilidades de agrupaciones de los elementos en cinco clases con valores aceptables de la inercia intraclase, que para los otros dos casos comparados.

Para el problema Gráfico 1 la partición que resulta más natural consta de cuatro clases (figura 14). Los resultados, de modo similar a lo expuesto anteriormente para Gráfico 3, se muestran en la tabla 3.4.3, señalándose la columna correspondiente a dicha cantidad de clases. Llama la atención que en ningún caso se obtuvieron soluciones no-dominadas con dos clases, a pesar de que de acuerdo al algoritmo siempre se generan soluciones con ese número de clases, lo cual indica que estas soluciones no fueron buenas, o que particularmente para las funciones que están siendo utilizadas son siempre mejores las particiones con mayor número de clases.

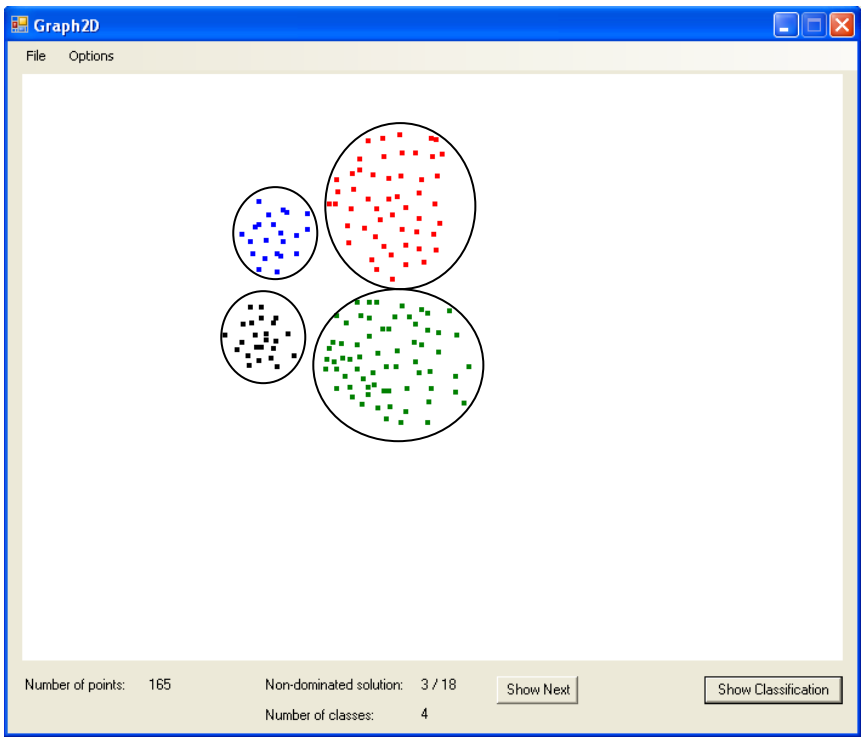


Figura 14: Partición de cuatro clases para Gráfico 1

		Número de clases							
		2	**3**	**4**	**5**	**6**	**7**	**8**	**Total**
Número de corrida	**1**	0	1	**3**	4	4	4	2	**18**
	2	0	1	**2**	3	5	3	2	**16**
	3	0	1	**2**	1	5	4	3	**16**
	4	0	1	**1**	1	5	3	3	**14**
	5	0	1	**2**	3	5	1	2	**14**
	6	0	1	**2**	4	7	3	2	**19**
	7	0	1	**3**	5	4	3	5	**21**
	8	0	1	**2**	2	2	3	2	**12**
	9	0	1	**2**	2	6	2	5	**18**
	10	0	1	**4**	1	6	2	5	**19**
	11	0	1	**3**	3	5	2	2	**16**
	12	0	1	**2**	3	6	2	3	**17**
	13	0	1	**4**	4	4	2	2	**17**
	14	0	1	**3**	4	4	1	3	**16**
	15	0	1	**2**	3	4	1	3	**14**
	16	0	1	**5**	7	5	1	2	**21**
	17	0	1	**3**	3	3	4	2	**16**
	18	0	1	**2**	3	4	1	4	**15**
	19	0	1	**3**	4	2	1	3	**14**
	20	0	1	**3**	4	5	1	3	**17**
	Total	**0**	**20**	**53**	**65**	**91**	**44**	**58**	**330**

Tabla 3.4.3: Distribución de las soluciones no-dominadas por cantidades de clases para el problema Gráfico 1 en 20 corridas de la estrategia propuesta para clasificación automática

Con este ejemplo también se obtuvo la clasificación esperada en todas las corridas, siendo los valores óptimos de W, F y GS los que se muestran en la tabla 3.4.4. Esta partición fue, también en todas las ejecuciones, la que alcanzó mayor índice de Silhouette global. En cambio, se obtuvieron otras soluciones con mejores valores de W y F, lo cual puede atribuirse a la influencia que tiene el número de clases en los valores de dichas funciones, de modo que al igual que para el problema Gráfico 3, las soluciones con menor valor de W fueron aquellas no-dominadas con mayor cantidad de clases y las de mayor valor de F, aquellas con menos clases.

	Gráfico 1
	4 clases
W	1 361,18
F	94,14
GS	0,57
% éxito	100

Tabla 3.4.4: Valores de W, F y GS para el problema Gráfico 1 con cuatro clases (W se minimiza, F y GS se maximizan) y porciento de éxito en 20 corridas

Conclusiones

En el presente trabajo se ofrece una alternativa al problema de clasificación automática basada en la Optimización con Colonia de Hormigas. El problema es visto desde una perspectiva multiobjetivo, teniendo en cuenta dos funciones objetivo para su solución.

Además de los algoritmos desarrollados para resolver el problema antes mencionado se obtuvo una herramienta que permite crear instancias de problemas en $\mathbb{R}^2$ y visualizar las soluciones obtenidas para las mismas.

Cabe destacar que se realizó de forma automática la configuración de los parámetros de la metaheurística utilizada, opción a la que se puede acceder en la aplicación desarrollada.

Respecto a los algoritmos implementados se pudo concluir que:

- La Optimización con Colonia de Hormigas es una alternativa apropiada para resolver el problema de clasificación de datos.
- Los parámetros de la metaheurística tienen gran influencia en el desempeño de los algoritmos para resolver el problema dado, en particular se pueden mejorar los resultados aumentando el número de ciclos del algoritmo de clasificación, aunque ello implica un mayor costo en tiempo.
- La adición de un mecanismo de mejora local al algoritmo de hormigas proporciona un incremento en la eficacia del mismo, logrando alcanzar el óptimo un número de veces superior.
- La estrategia propuesta para resolver el problema de clasificación automática proporciona clasificaciones que corresponden a la distribución natural de los datos.
- El índice de Silhouette es una buena medida de la calidad de las particiones obtenidas, aunque puede favorecer a aquellas con mayor número de clases.

Recomendaciones

Se recomienda:

- Diseñar algoritmos multiobjetivo de Optimización con Colonia de Hormigas para resolver el problema de clasificación automática, ya que por las características observadas en dicho problema es posible hacer un diseño que tenga en cuenta múltiples objetivos desde la base.
- Valorar la implementación de otras métricas para la similitud y disimilitud intra e interclases.
- Implementar otras medidas de calidad de particiones y comparar los resultados obtenidos.

Referencias Bibliográficas

[1] BERRY, Michael J. A. y LINOFF, Gordon S. *Data Mining Techniques For Marketing, Sales, And Customer Relationship Management*. 2da ed. Indianápolis: Wiley, 2004. ISBN: 0-471-47064-3.

[2] FREITAS, Alex A. y LAVINGTON, Simon H. *Mining very large databases with parallel processing*. Kluwer Academic Publishers, 1998. ISBN: 978-0-7923-8048-1.

[3] GORUNESCU, Florin. *Data Mining Concepts, Models and Techniques*. Springer-Verlag Berlin Heidelberg, 2011. ISBN: 978-3-642-19720-8.

[4] SUMATHI, S. y SIVANANDAM, S.N. *Introduction To Data Mining And Its Applications*. Springer-Verlag Berlin Heidelberg, 2006. ISBN: 3-540-34350-4.

[5] WITTEN, Ian H. y FRANK, Eibe. *Data Mining. Practical Machine Learning Tools and Techniques*. 2da ed. San Francisco: Morgan Kaufmann Publishers, 2005. ISBN: 0-12-088407-0.

[6] TREJOS, Javier, CASTILLO, William y GONZÁLEZ, Jorge, *Análisis multivariado de datos. Métodos y Aplicaciones*. 2011, Escuela de Matemática, Universidad de Costa Rica.

[7] FRAKES, William B. y BAEZA-YATES, Ricardo. *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992. ISBN: 0-13-463837-9.

[8] *Jim Gray*. [en línea] [ref. de 8 de noviembre de 2011]; Disponible en Web: http://es.wikipedia.org/wiki/Jim_Gray.

[9] *Jim Gray Summary Home Page*. [en línea] [ref. de 8 de noviembre de 2011]; Disponible en Web: http://research.microsoft.com/en-us/um/people/gray/.

[10] EVERITT, Brian S., et al. *Cluster Analysis*. 5ta ed. John Wiley & Sons, Ltd, 2011. ISBN: 978-0-470-74991-3.

[11] GAREY, Michael R. y JOHNSON, David S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Bell Telephone Laboratories, Incorporated, 1979. ISBN: 0-7167-1044-7.

[12] GLOVER, Fred y KOCHENBERGER, Gary A. *Handbook of Metaheuristics*. Dordrecht: Kluwer Academic Publishers, 2003. ISBN: 1-4020-7263-5.

[13] NOA, Yenny. "Una aplicación de algoritmos genéticos al problema de Categorización". Director: Luciano García. Trabajo de Diploma. Universidad de La Habana, Facultad de Matemática y Computación, 2007.

[14] DELGADO, Yeneit. "Clasificación automática mediante Colonia de Hormigas. Configuración automática de parámetros". Director: Juan Manuel Otero. Trabajo de Diploma. Universidad de La Habana, Facultad de Matemática y Computación, 2004.

[15] LOZANO, Jose A. "Algoritmos genéticos aplicados a la clasificación no supervisada". Tesis de Doctorado. Universidad del País Vasco, Departamento de Ciencias de la Computación e Inteligencia Artificial, 1998.

[16] TREJOS, Javier, *Clasificación Automática: Métodos para el Análisis de Grupos. Notas de Curso*, Universidad de Costa Rica: San José.

[17] HO, Tsang Chi. "Network-Based Anomaly Intrusion Detection using Ant Colony Clustering Model and Genetic-Fuzzy Rule Mining Approach". Tesis de Maestría. City University of Hong Kong, Department of Computer Science, 2006.

[18] JAIN, Anil K., MURTY, M. Narasimha y FLYNN, Patrick J. *Data clustering: a review.* ACM Computing Surveys. 1999, vol. 31, núm. 3, p. 264-323.

[19] KAUFFMAN, Leonard y ROUSSEEUW, Peter J. *Finding Groups in Data: An Introduction to Cluster Analysis*. New Jersey: John Wiley & Sons, Inc., 1990. ISBN: 0-471-73578-7.

[20] DUDA, Richard O., HART, Peter E. y STORK, David G. *Pattern classification*. 2da ed. John Wiley & Sons, Inc., 2001. ISBN: 0-471-05669-3.

[21] TREJOS, Javier, et al. "*Comparación de metaheurísticas en clasificación automática*". En I Congreso Internacional Computación y Matemática. Heredia, Costa Rica, 2008. ISBN: 978-9968-9961-1-5.

[22] PACHECO, Alexia, et al. *Evaluación de heurísticas de optimización combinatoria en clasificación por particiones.* Revista Investigación Operacional. 2006, vol. 27, núm. 2, p. 124-128.

[23] TREJOS, Javier, MURILLO, Alex y PIZA, Eduardo. "*Clustering by Ant Colony Optimization*". En Classification, Clustering, and Data Mining Applications. Proceedings of the Meeting of the International Federation of Classification Societies (IFCS). Springer-Verlag Berlin Heidelberg, 2004. ISBN: 3-540-22014-3.

[24] HAN, Jiawei y KAMBER, Micheline. *Cluster Analysis*, en *Data Mining: Concepts and Techniques*. 2006, Morgan Kaufmann Publishers.

[25] ABONYI, János y FEIL, Balázs. *Cluster Analysis for Data Mining and System Identification*. Birkhäuser Verlag AG, 2007. ISBN: 978-3-7643-7987-2.

[26] DORIGO, Marco y DI CARO, Gianni A. *The ant colony optimization meta-heuristic*, en *New Ideas in Optimization*, D. CORNE, M. DORIGO y F. GLOVER (ed.). 1999, McGraw-Hill: Londres, Reino Unido. p. 11-32.

[27] DORIGO, Marco, DI CARO, Gianni A. y GAMBARDELLA, Luca Maria. *Ant algorithms for discrete optimization.* Artificial Life. 1999, vol. 5, núm. 2, p. 137-172.

[28] DORIGO, Marco y STÜTZLE, Thomas. *Ant Colony Optimization*. MIT Press, 2004. ISBN: 0-262-04219-3.

[29] DORIGO, Marco y SOCHA, Krzysztof. *An Introduction to Ant Colony Optimization*, en *Handbook of Approximation Algorithms and Metaheuristics*, T. F. GONZÁLEZ (ed.). 2007, CRC Press. p. 26-1.

[30] *Ant Colony Optimization. Bibliography*. [ref. de 16 de diciembre de 2011]; Disponible en Web: http://www.aco-metaheuristic.org/publications.html.

[31] DORIGO, Marco. *Ant Colony Optimization.* Scholarpedia. 2007, vol. 2, núm. 3.

[32] DORIGO, Marco, MANIEZZO, Vittorio y COLORNI, Alberto. *The ant system: an autocatalytic optimizing process*. Milán, Italia: Politecnico di Milano, 1991.

[33] COLORNI, Alberto, DORIGO, Marco y MANIEZZO, Vittorio. "*Distributed optimization by ant colonies*". En ECAL'91 European Conference on Artificial Life. Elsevier Publishing, 1991.

[34] DORIGO, Marco. "Organization, learning and natural algorithms". Tesis de Doctorado. Politecnico di Milano. Dipartimento di Elettronica, 1992.

[35] DORIGO, Marco y GAMBARDELLA, Luca Maria. *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem.* IEEE Transactions on Evolutionary Computation. 1997, vol. 1, núm. 1, p. 53-66.

[36] *ANTS International Conference on Swarm Intelligence*. [ref. de 16 de diciembre de 2011]; Disponible en Web: http://iridia.ulb.ac.be/~ants.

[37] DORIGO, Marco, STÜTZLE, Thomas y DI CARO, Gianni A. *Special Issue: "Ant Algorithms".* Future Generation Computer Systems. 2000, vol. 8, núm. 16, p. 851-956.

[38] DORIGO, Marco, et al. *Special Section on "Ant Colony Optimization".* IEEE Transactions on Evolutionary Computation. 2002, vol. 4, núm. 6, p. 317-365.

[39] *Ant Algorithms: Third International Workshop, ANTS 2002*. Springer-Verlag Berlin Heidelberg, 2002. ISBN: 3-540-44146-8.

[40] *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004*. Springer-Verlag Berlin Heidelberg, 2004. ISBN: 3-540-22672-9.

[41] *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006*. Springer-Verlag Berlin Heidelberg, 2006. ISBN: 3-540-38482-0.

[42] *Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008*. Springer-Verlag Berlin Heidelberg, 2008. ISBN: 3-540-87526-3.

[43] *Swarm Intelligence: 7th International Conference, ANTS 2010*. Springer-Verlag Berlin Heidelberg, 2010. ISBN: 3-642-15460-3.

[44] DORIGO, Marco y STÜTZLE, Thomas. *Ant Colony Optimization: Overview and Recent Advances*, en *Handbook of Metaheuristics*, Michel GENDREAU y Jean-Yves POTVIN (ed.). 2010, Springer. p. 227-263.

[45] BRANKE, Jürgen, et al., (eds.) *Multiobjective Optimization: Interactive and Evolutionary Approaches*. 2008, Springer-Verlag Berlin Heidelberg: Berlín. ISBN: 3-540-88907-8.

[46] DEB, Kalyanmoy. *Multi-Objective Optimization using Evolutionary Algorithms*. Gran Bretaña: John Wiley & Sons, Ltd, 2001. ISBN: 0-471-87339-X.

[47] LEGUIZAMÓN, Guillermo y COELLO, Carlos A. *Multi-Objective Ant Colony Optimization: A Taxonomy and Review of Approaches*, en *Integration of Swarm Intelligence and Artificial Neural Networks*, Satchidananda DEHURI, Susmita GHOSH y Sung-Bae CHO (ed.). 2011. p. 67-94.

[48] MARÍN, Tomás. "Colonia de hormigas: una nueva heurística para la solución de problemas de optimización combinatorios". Director: Juan Manuel Otero. Trabajo de Diploma. Universidad de La Habana, Facultad de Matemática y Computación, 2002.

[49] GUERRERO, Danilo. "Algoritmos basados en optimización por colonia de hormigas para el problema de clasificación automática por particionamiento". Director: Juan Manuel Otero. Trabajo de diploma. Universidad de La Habana, Facultad de Matemática y Computación, 2003.

[50] DELGADO, Yeneit. "*Clasificación mediante colonia de hormigas y configuración de parámetros*". En IV Congreso Internacional de Matemática Aplicada a la Ingeniería y Enseñanza de la Matemática en Ingeniería, INMAT 2008 (Libro de resúmenes CD-ROM). 2008. ISBN: 978-950-29-1077-2.

[51] MANIEZZO, Vittorio, GAMBARDELLA, Luca Maria y DE LUIGI, Fabio. *Ant Colony Optimization*, en *New Optimization Techniques in Engineering*, G. C.

ONWUBOLU y B. V. BABU (ed.). 2004, Springer-Verlag Berlin Heidelberg. p. 101-117.

[52] GENDREAU, Michel y POTVIN, Jean-Yves. *Handbook of Metaheuristics*. 2da ed. Springer, 2010. ISBN: 978-1-4419-1663-1.

[53] BIRATTARI, Mauro, et al. "*A racing algorithm for configuring metaheuristics*". En Genetic and Evolutionary Computation Conference (GECCO-2002). 2002. ISBN: 1-55860-878-8.

[54] BILLINGSLEY, Patrick. *Probability and Measure*. 2da ed. Nueva York: John Wiley & Sons, 1986. ISBN: 0471-80478-9.

[55] CONOVER, William Jay. *Practical Nonparametric Statistics*. 3ra ed. Nueva York: John Wiley & Sons, 1999. ISBN: 0471160687.

[56] DEAN, Angela M. y VOSS, Dean. *Design and Analysis of Experiments*. Nueva York: Springer Verlag, 1999. ISBN: 0387985611.

[57] BALAPRAKASH, Prasanna, et al. "*Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement*". En 4th International Workshop on Hybrid Metaheuristics, (HM 2007). Springer Verlag, 2007. ISBN: 3-540-75513-6.

[58] BIRATTARI, Mauro, et al. *Automated Algorithm Tuning using F-races: Recent Developments*, en *Proceedings of MIC 2009, the 8th Metaheuristics International Conference (Proceedings on CD-ROM)*, M. CASERTA y S. VOß (ed.). 2009, University of Hamburg: Hamburgo, Alemania.

[59] BIRATTARI, Mauro, et al. *F-Race and Iterated F-Race: An Overview*, en *Experimental Methods for the Analysis of Optimization Algorithms*, T. BARTZ-BEIELSTEIN, et al. (ed.). 2010, Springer. p. 311-336.

[60] BOLSHAKOVA, Nadia y AZUAJE, Francisco. *Cluster validation techniques for genome expression data.* Signal Processing. 2003, vol. 83, p. 825-833.

[61] ROUSSEEUW, Peter J. *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis.* Journal of Computational and Applied Mathematics. 1987, vol. 20, p. 53-65.

Anexos

Anexo 1: Resumen de tipos de datos y métodos aplicables de clasificación automática

Data type	Method	Notes
Continuous (Ordered data can be treated as continuous, possibly with standardization based on range)	Hierarchical agglomerative or partitioning methods	Ward's method, average linkage, and k-means or methods based on det(W) are popular choices; contiguity-constrained versions are possible
	Density search, mode analysis	
	Mixtures models	Especially for multivariate normal data; may need to restrict number of parameters fitted and/or take account of structure (such as repeated measures)
	Fuzzy k-means	Where clusters other than the 'best' are of interest
	Direct data clustering	Useful for data representing positive associations; both objects and variables clustered; need to consider scaling of data
	Kohonen self-organizing map (SOM)	Useful for large data sets; produces low-dimensional plot of clusters
Binary (Categorical data can be converted to binary)	Hierarchical or partitioning methods using appropriate proximity measure	Need to consider negative match issue; some special proximity measures available for categorical data
	Monothetic divisive method	Useful for developing diagnostic keys
	Latent classes or grade of membership (GOM)	These are 'fuzzy' methods; GOM often used in health applications
	Hierarchical classes (HICLAS)	Objects and variables clustered; overlapping clusters, useful for psychological data
Mixed mode	Hierarchical or partitioning methods using appropriate proximity measure or using ranks	Gower's similarity measure can be used
	Two step (SPSS)	Note possible problems with commensurability
	Model-based models for mixed data types	
Proximity matrix (Either computed from data or measured directly)	Hierarchical agglomerative or partitioning methods that do not require raw data	Examples are single, average and complete linkage, or partitioning around medoids (PAM)
	Additive clustering (ADCLUS and variants)	Overlapping clusters
	Kaufman and Rousseeuw fuzzy method (FANNY)	Similar to fuzzy k-means but raw data not required
	Tree-fitting methods	Dendrograms, additive trees; pyramids (limited overlap)
Large data sets	Data reduction methods such as principal components, spectral analysis	Reduction methods do not generally respect the cluster structure (so may be misleading)
	Clustering a subsample; using the clusters as seeds for further processing	Sampling process may need to be adapted to likely cluster structure

Tabla A.1: "Resumen de tipos de datos y métodos aplicables de clasificación automática". Fuente [10]

Anexo 2: Algunas aplicaciones actuales de algoritmos ACO

Problem type	*Problem name*	*Authors*	*Year*
Routing	Traveling salesman	Dorigo et al.	1991, 1996
		Dorigo and Gambardella	1997
		Stützle and Hoos	1997, 2000
	TSP with time windows	López Ibáñez et al.	2009
	Sequential ordering	Gambardella and Dorigo	2000
	Vehicle routing	Gambardella et al.	1999
		Reimann et al.	2004
		Favoretto et al.	2007
		Fuellerer et al.	2009
	Multicasting	Hernández and Blum	2009
Assignment	Quadratic assignment	Maniezzo	1999
		Stützle and Hoos	2000
	Frequency assignment	Maniezzo and Carbonaro	2000
	Course timetabling	Socha et al.	2002, 2003
	Graph coloring	Costa and Hertz	1997
Scheduling	Project scheduling	Merkle et al.	2002
	Weighted tardiness	den Besten et al.	2000
		Merkle and Middendorf	2000
	Flow shop	Stützle	1997
		Rajendran, Ziegler	2004
	Open shop	Blum	2005
	Car sequencing	Solnon	2008
Subset	Set covering	Lessing et al.	2004
	l-cardinality trees	Blum and Blesa	2005
	Multiple knapsack	Leguizamón and Michalewicz	1999
	Maximum clique	Solnon and Fenet	2006
Machine learning	Classification rules	Parpinelli et al.	2002
		Martens et al.	2006
		Otero et al.	2008
	Bayesian networks	Campos, Fernández-Luna	2002
	Neural networks	Socha, Blum	2007
Bioinformatics	Protein folding	Shmygelska and Hoos	2005
	Docking	Korb et al.	2006
	DNA sequencing	Blum et al.	2008
	Haplotype inference	Benedettini et al.	2008

Tabla A.2: “Algunas aplicaciones actuales de algoritmos ACO”. Las aplicaciones están listadas de acuerdo a los tipos de problemas. Fuente [52]

Anexo 3: Resumen de los principales algoritmos ACO para problemas NP-duros

ACO algorithm	*Year*	*TSP*
Ant System	1991	Yes
Elitist AS	1992	Yes
Ant-Q	1995	Yes
Ant Colony System	1996	Yes
*MM*AS	1996	Yes
Rank-based AS	1997	Yes
ANTS	1998	No
Best-worst AS	2000	Yes
Population-based ACO	2002	Yes
Beam-ACO	2004	No

Tabla A.3: “Resumen de los principales algoritmos ACO para problemas NP-duros que han sido propuestos en la literatura”. Se muestra el nombre de los algoritmos, el año en que se publicaron por primera vez y si han sido probados o no en el problema del viajante de comercio. Fuente [52]

Anexo 4: Configuraciones utilizadas para la configuración automática

Configuración	*α*	*β*	*ρ*	*q0*	*Q*	*ciclos*	*subciclos*
c1	0,1	1	0,9	0,9	1000	10	1
c2	0,1	1	0,9	0,9	1000	10	5
c3	0,1	1	0,9	0,9	1000	10	10
c4	0,1	2	0,9	0,9	1000	10	1
c5	0,1	2	0,9	0,9	1000	10	5
c6	0,1	2	0,9	0,9	1000	10	10
c7	0,1	4	0,9	0,9	1000	10	1
c8	0,1	4	0,9	0,9	1000	10	5
c9	0,1	4	0,9	0,9	1000	10	10
c10	0,5	1	0,9	0,9	1000	10	1
c11	0,5	1	0,9	0,9	1000	10	5
c12	0,5	1	0,9	0,9	1000	10	10
c13	0,5	2	0,9	0,9	1000	10	1
c14	0,5	2	0,9	0,9	1000	10	5
c15	0,5	2	0,9	0,9	1000	10	10
c16	0,5	4	0,9	0,9	1000	10	1
c17	0,5	4	0,9	0,9	1000	10	5
c18	0,5	4	0,9	0,9	1000	10	10
c19	1	1	0,9	0,9	1000	10	1
c20	1	1	0,9	0,9	1000	10	5
c21	1	1	0,9	0,9	1000	10	10
c22	1	2	0,9	0,9	1000	10	1
c23	1	2	0,9	0,9	1000	10	5
c24	1	2	0,9	0,9	1000	10	10
c25	1	4	0,9	0,9	1000	10	1
c26	1	4	0,9	0,9	1000	10	5
c27	1	4	0,9	0,9	1000	10	10

Tabla A.4: Diferentes configuraciones utilizadas para la configuración automática

Anexo 5: Resultados de las corridas del algoritmo de clasificación

	Gráfico 1 (2 clases)	Gráfico 3 (10 clases)		Gráfico 3 (10 clases)
W mínima	3 504,30	214,37		214,37
W promedio	3 504,30	445,95		385,30
Desv. estándar	0	501,45		456,70
% de éxito	100	10		20
T. promedio (s)	0,36	0,16		0,58
				Q=200 ciclos=40

Tabla A.5.1: Resultados en 20 corridas del algoritmo de clasificación para los problemas Gráfico 1, con dos clases, y Gráfico 3, con 10 clases

	Amiard (2 clases)	Iris (2 clases)		Amiard (2 clases)
W mínima	69 368,27	0,999		69 368,27
W promedio	69 512,54	0,999		69 368,27
Desv. estándar	985,54	0		0
% de éxito	70	100		100
T. promedio (s)	0,05	0,33		0,19
				Q=69 000 ciclos=40

Tabla A.5.2: Resultados en 20 corridas del algoritmo de clasificación para los problemas Amiard e Iris, con dos clases

	Amiard (4 clases)	Thomas (4 clases)		Amiard (4 clases)	Thomas (4 clases)
W mínima	18 281,39	235,03		18 281,39	235,03
W promedio	19 766,79	242,45		18 508,15	235,97
Desv. estándar	15 837,96	17,90		2 868,38	6,31
% de éxito	60	5		85	35
T. promedio (s)	0,08	0,07		0,29	0,07
				Q=18 200 ciclos=30	Q=200 ciclos=80

Tabla A.5.3: Resultados en 20 corridas del algoritmo de clasificación para los problemas Amiard y Thomas, con cuatro clases

	Amiard (5 clases)	Thomas (5 clases)		Amiard (5 clases)	Thomas (5 clases)
W mínima	14 497,81	202,58		14 497,81	202,58
W promedio	15 113,16	208,63		14 765,78	203,49
Desv. estándar	3 024,57	21,88		1 147,34	6,54
% de éxito	10	10		30	20
T. promedio (s)	0,08	0,08		0,30	0,78
				Q=14 400 ciclos=30	Q=200 ciclos=100

Tabla A.5.4: Resultados en 20 corridas del algoritmo de clasificación para los problemas Amiard y Thomas, con cinco clases