Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería de Organización Industrial

Desarrollo y aplicación del algoritmo de Optimización basado en Colonia de Hormigas (ACO) para la resolución del Problema del Viajante Asimétrico (ATSP)

Autor:

Revuelta Martínez, Tamara

Tutor:

Pérez Vázquez, María Elena

Departamento de Organización de Empresas y CIM.

Valladolid, Septiembre 2015.

A mis padres, por haberme permitido iniciar este camino y a todas aquellas personas que se han unido a él durante esta etapa de mi vida.

RESUMEN

Los algoritmos evolutivos, concretamente los basados en *Colonia de Hormigas*, ACO, están cobrando cada vez más importancia en el campo de la metaheurística, así como su aplicación para la resolución de problemas de optimización combinatoria. Uno de los problemas más comunes y difíciles de resolver que se clasifica como NP-duro, es el *Problema del Viajante*, TSP. El interés en el estudio de las técnicas metaheurísticas para la resolución de este problema radica, principalmente, en el gran número de aplicaciones prácticas en las que se encuentra.

El presente documento recoge, explícitamente, el desarrollo y aplicación de un algoritmo ACO para la resolución del problema del viajante asimétrico, ATSP, cuyo fin es encontrar una solución que, satisfaciendo las condiciones iniciales del problema, proporcione una ruta o circuito cerrado cuya longitud sea la mínima. Para ello, se realizarán una serie de pruebas, a través de las cuales se obtendrán resultados que, posteriormente, se evaluarán y valorarán mediante el uso de técnicas estadísticas adecuadas.

PALABRAS CLAVE

ACO, ATSP, AS, EAS, ASrank..

ABSTRACT

Evolutionary algorithms, particularly those based on Ant Colony, ACO, are becoming increasingly important in the area of metaheuristic and their application for solving combinatorial optimization problems NP-hard type. One of the most common and difficult problem which are classified as NP-hard, is the Travelling Salesman Problem, TSP. The interest of studying metaheuristics techniques in order to solve this problem is mainly focus, in the large number of practical applications and decision problems which it is part.

This paper includes the development and implementation of an ACO algorithm in order to solve the problem of asymmetric traveling, ATSP, whose final result is to find a solution satisfying the initial conditions of the problem and providing a path or closed circuit with the minimum length. For this, a series of tests will be conducted and their results are subsequently evaluated and assessed using appropriate statistical techniques.

KEY WORDS

ACO, ATSP, AS, EAS, ASrank.

Índice General

Índice de Tablas

Índice de Figuras

INTRODUCCIÓN

Antecedentes.

Las hormigas muestran comportamientos sociales complejos que han captado la atención del ser humano. Uno de los comportamientos que, probablemente, sea el más notable para nosotros es la formación de lo que se denomina caminos de hormigas. Si observamos una colonia de hormigas y el desplazamiento que ésta realiza desde el hormiguero hasta la fuente de alimentación, se aprecia cómo todas y cada de las hormigas circulan en línea recta, una detrás de otra. Todo esto, nos lleva a formularnos multitud de preguntas acerca del comportamiento que presentan: ¿Cómo construyen el camino?, ¿cuál será su reacción ante la presencia de obstáculos en el recorrido?, entre otras.

Numerosos investigadores, principalmente biólogos, han realizado y prestado especial atención al comportamiento de algunas especies de hormigas, concretamente destaca la capacidad que poseen para hallar los recorridos de longitud más corta. Esto se ha demostrado experimentalmente, y es posible gracias a la explotación de la comunicación basada en el rastro de una sustancia química que estos insectos depositan en el camino y detectan mediante el olfato, denominada feromona. Ésta será la base y el fundamento que inspire a los informáticos para desarrollar algoritmos matemáticos implementados en lenguajes de alto nivel para resolver problemas de optimización combinatoria.

Los primeros desarrollos de algoritmos basados en colonia de hormigas (ACO) datan de los años 90, en los que se constataba la validez del enfoque para resolver el problema. Desde entonces, se han formulado y planteado numerosas variantes y modificaciones que han permitido obtener mejores resultados de los problemas sobre los que se planteaba, principalmente sobre el TSP, el cual se considera un paradigma dentro de los problemas de optimización combinatoria. No obstante, el éxito de los ACO (en inglés, *Ant Colony Optimization*) radica en la amplia gama de problemas diferentes a los que se ha aplicado y la alta calidad de sus resultados.

Motivación.

Como ya se ha mencionado en el apartado anterior, las mejoras en los algoritmos ACO se suceden a medida que los investigadores desarrollan nuevos algoritmos. El fin último es desarrollar una técnica metaheurística basada en colonia de hormigas que permita resolver un problema de optimización combinatoria, concretamente, El Problema del Viajante Asimétrico (ATSP), es decir, encontrar la ruta que corresponda

con el recorrido de menor longitud. Puesto que el TSP es un problema NP-duro, la implementación de algoritmos de aproximación como son los ACO, proporciona soluciones de calidad aceptable aunque éstas pueden optimizarse siempre y cuando las técnicas implementadas se mejoren o se desarrollen nuevas variantes.

Éste es el motivo fundamental por el que los investigadores estudian este tipo de algoritmos. No obstante, en el presente documento se realiza de acuerdo con lo que se ha expuesto anteriormente: evaluar los algoritmos ACO para encontrar aquel que nos proporcione los mejores resultados.

Alcance.

El campo de la optimización de colonia de hormigas ha contribuido a numerosas mejoras dentro de la comunidad científica gracias a un gran número de investigadores que han realizado y desarrollado numeroso avances. Principalmente, destaca el pionero de los ACO, Marco Dorigo que junto con Alberto Colorni y Vittorio Maniezzo formularon por primera vez la definición de los ACO.

El alcance del presente documento implementar un algoritmo que sea capaz de proporcionar los mejores resultados para el problema del viajante asimétrico, ATSP, es decir, qué combinación de parámetros de los ACO, así como sus variantes es la que mejor se comporta para este tipo de problemas.

Objetivos.

En los apartados anteriores ya se ha mencionado, en diversas ocasiones, cual es el objetivo fundamental de los algoritmos ACO.

En el presente documento, el objetivo fundamental se centra en el análisis de los algoritmos basados en colonias de hormigas para resolver el problema del viajante asimétrico, ATSP: su comportamiento cuando se estudian problemas de diversos tamaños (número de ciudades), implementación y posterior evaluación de resultados. Los objetivos se pueden resumir en lo siguiente:

- Implementación y desarrollo de los algoritmos ACO y variantes más significativas buscando una optimización del problema a resolver.
- Cuantificación de la eficiencia de las soluciones proporcionadas por estos algoritmos ACO para la resolución de problemas de rutas mínimas, ATSP.
- Realizar un estudio estadístico sobre las evaluaciones y simulaciones de todos los algoritmos implementados.
- Extraer las conclusiones pertinentes en base a los resultados y el estudio estadístico realizado.

Capítulo 1.

El Problema del Viajante, TSP (*Traveling Salesman Problem*). El ATSP.

En este capítulo, se va a abordar el problema del viajante, TSP (en inglés, *Traveling Salesman Problem*) como objetivo fundamental, más concretamente la versión asimétrica del mismo, ATSP, que será la que se resolverá en capítulos posteriores. En primer lugar, se explicará la teoría de la complejidad computacional de los problemas de optimización combinatoria, y una vez realizado esto, se clasificará el TSP como un problema NP-completo y los motivos por los que se le considera. Posteriormente, profundizando con más detalle en el TSP, se explicará su definición, origen, formulación matemática y modelización del mismo. Por último, sus principales variantes y aplicaciones en los distintos campos de la industria, logística y distribución.

1.1. COMPLEJIDAD COMPUTACIONAL DE LOS PROBLEMAS DE OPTIMIZACIÓN COMBINATORIA.

Una forma sencilla de resolver problemas de optimización combinatoria puede ser la búsqueda exhaustiva, la cual consiste en enumerar todas las posibles soluciones del problema en particular y, posteriormente, la elección de la mejor solución entre todas ellas. Pero desafortunadamente, en la mayoría de los casos, esta sencilla aproximación resulta ineficiente debido a que el número de posibles soluciones aumente exponencialmente con el tamaño del problema.

Para algunos problemas de optimización combinatoria, si se tiene un profundo conocimiento de la estructura de los mismos y sus características específicas, esto permite resolverlos mediante la aplicación de los denominados algoritmos de aproximación que puedan encontrar una solución óptima o próxima a la óptima en una región del espacio dentro del conjunto de las posibles soluciones del problema, las cuales son de buena calidad (esta región del espacio contiene al óptimo) y lo hacen mucho más rápido que la búsqueda exhaustiva. Pero hay veces, que el problema comprende una complejidad computacional tan elevada que ni los mejores algoritmos de esta clase son capaces de proporcionarnos soluciones factibles y viables.

1.1.1. Teoría computacional de los problemas de optimización combinatoria.

Cuando nos enfrentamos a un problema de optimización combinatoria, es útil y necesario conocer su dificultad para encontrar la solución óptima. Existen diferentes formas de medir o cuantificar esta dificultad que viene dada por la idea de la complejidad del caso más desfavorable (en inglés, *worst-case complexity*) (Dorigo y Stützle, 2004).

1.1.1.1. Complejidad computacional de los problemas según el caso más desfavorable.

La complejidad del tiempo de ejecución de un algoritmo para un problema dado, éste será $\prod$, indica para cada posible tamaño del problema (el tamaño de la instancia se denota como n), el tiempo máximo que el algoritmo necesita para encontrar una solución al problema de ese mismo tamaño. A esto se le denomina complejidad del tiempo computacional del caso más desfavorable (en inglés, *worst-case time complexity*). Por ejemplo, un problema de este tipo es el problema del viajante, TSP.

La complejidad del tiempo computacional del caso más desfavorable de un algoritmo se formaliza utilizando la notación $O(\cdot)$.

Se consideran las siguientes funciones: $g(n)$ y $h(n)$ las cuales son funciones de números enteros positivos y pertenecientes al conjunto de números reales. Una función $g(n)$ e dice que es $O(h(n))$si existen dos constantes positivas las cuales se denominan como $const$ y n_0, y se cumple la expresión (1.1):

$$g(n) \leq const \cdot h(n) \quad \forall n \geq n_0 \qquad (1.1)$$

Dicho de otra forma, la notación $O(\cdot)$ lo que hace es definir límites superiores asintóticos de la complejidad del tiempo computacional del caso más desfavorable de un algoritmo.

En función de lo que se acaba de definir, se dice que un algoritmo es de tiempo polinómico si su función de complejidad de tiempo es $O(h(n))$ para alguna función polinómica g(·). Si existe un algoritmo que tiene una función de complejidad de tiempo que no puede ser limitada por un polinomio, entonces nos encontramos ante un algoritmo de tiempo exponencial. Nótese que este tipo también incluye funciones como n·log (n), que son referidas algunas veces como sub-exponenciales; en definitiva, las funciones de tipo exponencial crecen mucho más rápido que cualquier función de tipo polinómico. Un problema se dice que es intratable si no existe un algoritmo de tiempo polinómico capaz de resolverlo.

1.1.2. Teoría de los problemas NP. Problema de decisión.

La teoría de los problemas NP se caracteriza por determinar la dificultad de los problemas de optimización combinatoria, no sólo los determina sino que los clasifica en dos grupos principales que son: problemas de combinatorias tratables, aquellos en los que se encuentra una solución en un tiempo polinómico, son resueltos mediante aproximaciones polinómicas; y los problemas intratables, ya enunciados en el párrafo anterior, aquellos que no pueden ser resueltos mediante un algoritmo de tiempo polinómicos por lo que no existe o no se ha encontrado ningún tipo de algoritmo capaz de resolverlos.

Según la teoría de los problemas NP-completos, cada problema de optimización combinatoria ∏ tiene asociado un *problema de decisión,* el cual se define de la siguiente forma: dado un problema o instancia ∏, definido por los siguientes elementos, el triple (S,f,Ω) (ver tabla 1.1)

S	Conjunto de soluciones candidatas
f	Función objetivo
Ω	Conjunto de restricciones

Tabla 1.1. *Elementos que definen un problema NP- completo*

y un parámetro δ, se dice que existe una solución factible para dicho problema, tal que la función f tiene un valor inferior al parámetro δ cuando se calcula la función para el valor de s (solución factible),en el caso de ser un problema de minimización. En términos matemáticos, ver la expresión (1.2):

$$\prod con\,(S, f, \Omega) \quad y \quad \delta \quad \rightarrow \quad \exists\, s \in S \;\; tal\ que\ f(s) \leq \delta \quad (1.2)$$

La búsqueda de la solución de un problema de optimización combinatoria implica ser capaz de dar solución al problema, mientras que lo contrario, por lo general, no tiene por qué ser cierto. Esto quiere decir que el problema ∏ es al menos tan difícil o complicado de resolver como la versión de decisión de ∏. Si se demuestra que el problema de decisión es intratable, esto es, si no puede ser resuelto por algún algoritmo eficiente, implicará que también lo será el problema de búsqueda original.

Dicho esto y conociendo los elementos de los que consta un problema de combinatoria, modelización matemática y restricciones que se dan en ellos, ahora vamos a explicar, propiamente, la teoría de los problemas NP-completos.

1.1.2.1. Teoría de la NP-Completitud: naturaleza de los problemas.

La teoría de la NP-completitud realiza una distinción entre dos clases de problemas:

- Los de tipo P, para los cuales existe un algoritmo polinómico que proporciona soluciones factibles en un tiempo computacional adecuado, razonable.
- Los de tipo NP (polinómico no-determinístico), para los que se puede aplicar un algoritmo polinómico para comprobar si una posible solución es válida o no. Esta característica permite métodos de resolución dónde se va aceptando o desestimando soluciones hipotéticas.
- Problemas NP-completos, son problemas de tipo NP pero éstos son de extrema complejidad. En el mundo real, muchos de los problemas que nos encontramos se encuentran dentro de este conjunto: los NP-completos. Para los cuales, se cree que no existen algoritmos polinómicos capaces de resolverlos, pero esto todavía no está demostrado.

Esta teoría se incluye dentro de una más general, teoría de los problemas NP-duros (en inglés, *NP-hard*). Éstos engloban todos los tipos de complejidad de los problemas que existen actualmente y sobre los cuales se sigue investigando. En la figura 1.1, puede verse el diagrama de Euler para esta teoría, la intersección entre los distintos conjuntos de problemas, y por tanto, si sabemos a qué tipo de problema nos enfrentamos, sabremos qué técnicas emplear para resolver el mismo. En este caso, el diagrama representa el conjunto de problemas para los que los problemas P son distintos de los NP.

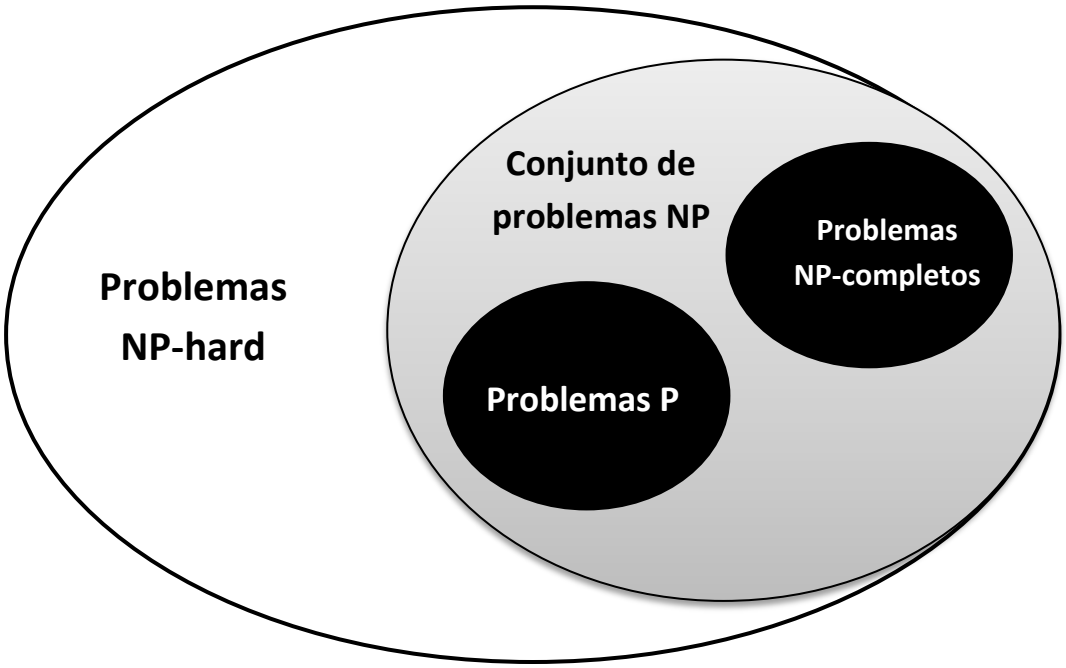


Figura 1.1. *Diagrama de Euler de la teoría de la complejidad computacional.*

$$P \neq NP - completos \ \ (no\ demostrado)$$

Los problemas NP-duros son, como mínimo, igual de complejos que los problemas NP pero no necesariamente tienen que ser problemas NP. El espacio que abarcan es más grande que los NP; los NP-completo, como ya se apuntó antes, son los más complejos de resolver dentro del conjunto de problemas NP.

Existen relaciones entre las dos clases de problemas; de acuerdo con la figura 1.1, es lógico decir que los problemas P están contenidos en los problemas NP, dicho de otra forma: los problemas NP engloban a los P.

$$\boldsymbol{P \subseteq NP}$$

Sin embargo, no se puede concluir nada sobre la cuestión de si los problemas NP son iguales a los P: NP=P. Esto significaría que los problemas NP pudiesen ser resueltos en un tiempo polinómico de computación.

Es un problema abierto que hasta nuestros días todavía no se ha demostrado y constituye el más importante de la teoría de la computación.

Según las últimas investigaciones y estudios acerca de éste problema, se basan en que no es probable que exista un algoritmo de tiempo polinómico para los problemas NP-completos.

Basándonos en la teoría de la complejidad computacional de los problemas NP-completos, el problema del viajante es un problema NP-completo, no existe un algoritmo de tiempo polinomio conocido para él, de hecho, es un problema O(n!) donde n es el tamaño del problema que en este caso es el número de ciudades a visitar por el viajante.

1.2. El TSP Y SU COMPLEJIDAD.

El problema del viajante de comercio, TSP, es uno de los problemas más complejos que hoy en día son conocidos en la programación matemática actual. Principalmente, esto es debido por su complejidad computacional: pertenece a una serie de problemas que parecen tener una fácil solución pero en la práctica presentan una gran dificultad de resolución.

1.2.1. Clasificación según la teoría de la complejidad computacional.

Dentro de clasificación de complejidad de los problemas, el TSP es un problema de optimización combinatoria, el cual se encuentra dentro de los que se conocen como NP-completos. Esto quiere decir que un aumento del tamaño de los problemas hace que el tiempo de resolución (o computación) aumente exponencialmente, por ello, no existe una solución polinómica. Por lo tanto, un aumento de ciudades provoca una elevación exponencial del número de combinaciones posibles. Esto se asocia a que el número de pruebas que hay que realizar para averiguar cuál de las posibles rutas encontradas es la solución óptima del problema, también incrementa exponencialmente el tiempo de resolución. Por ello, en la práctica, si se pretende resolver un problema del viajante con un tamaño muy elevado de datos (ciudades a visitar) es muy probable que pueda suceder que el tiempo de resolución o bien sea muy elevado o incluso infinito, lo cual indicaría que para dicho problema es imposible

encontrar la solución con las herramientas de software que hoy en día están disponibles.

Se va a plantear un problema que presente n ciudades (1) las cuales quieren visitarse. Por lo tanto, tenemos n! rutas posibles. Además, una misma ciudad no puede ser visitada dos veces (2) y, por último, imponemos la condición que la ciudad de partida y de destino final sean la misma (3).

Si tenemos en cuenta la última condición que hemos impuesto (3) el número de rutas posibles a explorar que se había planteado inicialmente puede reducirse a $(n-1)!$. Como la dirección en la que se desplace el viajante no importa, el número de rutas se reduce en un factor 2. Finalmente hay que considerar $\frac{(n-1)!}{2}$rutas posibles. Sin embargo, para el caso del ATSP la dirección de desplazamiento si es relevante por lo que el número de rutas posibles sería $(n-1)!$. .

Efectivamente, para el caso del ATSP con n=4 se cumple la formula deducida anteriormente: el número de combinaciones posibles es $(4-1)! = 3! = 3 \cdot 2 \cdot 1 = 6$

La tabla 1.2, muestra para distintos tamaños del problema TSP el número de combinaciones posibles. Y como ya se explicó anteriormente, la complejidad de este tipo de problemas residía principalmente cuando se aumentaba el tamaño de los mismos, el número de posibles soluciones a evaluar aumentada exponencialmente. Por ejemplo, si quisiéramos visitar sólo 10 ciudades, el número de posibles rutas a evaluar serían 181440, un número muy elevado con el que este problema ya no podría resolverse gráficamente como en el caso de n=4 ciudades. Por tanto, es necesaria la utilización de herramientas de software potentes que permitan realizar implementaciones en leguajes de alto nivel (C, C++, entre otros).

n, ciudades	**Posibles rutas,** $\frac{(n-1)!}{2}$
4	$\frac{(4-1)!}{2} = \frac{3 \cdot \cancel{2} \cdot 1}{\cancel{2}} = 3$
10	181440
100	$4.666 \cdot 10^{155}$

Tabla 1.2. *Número combinaciones para el TSP.*

Para **n=4**, tomando la ciudad- nodo inicial 1**,** las rutas posibles son las siguientes:

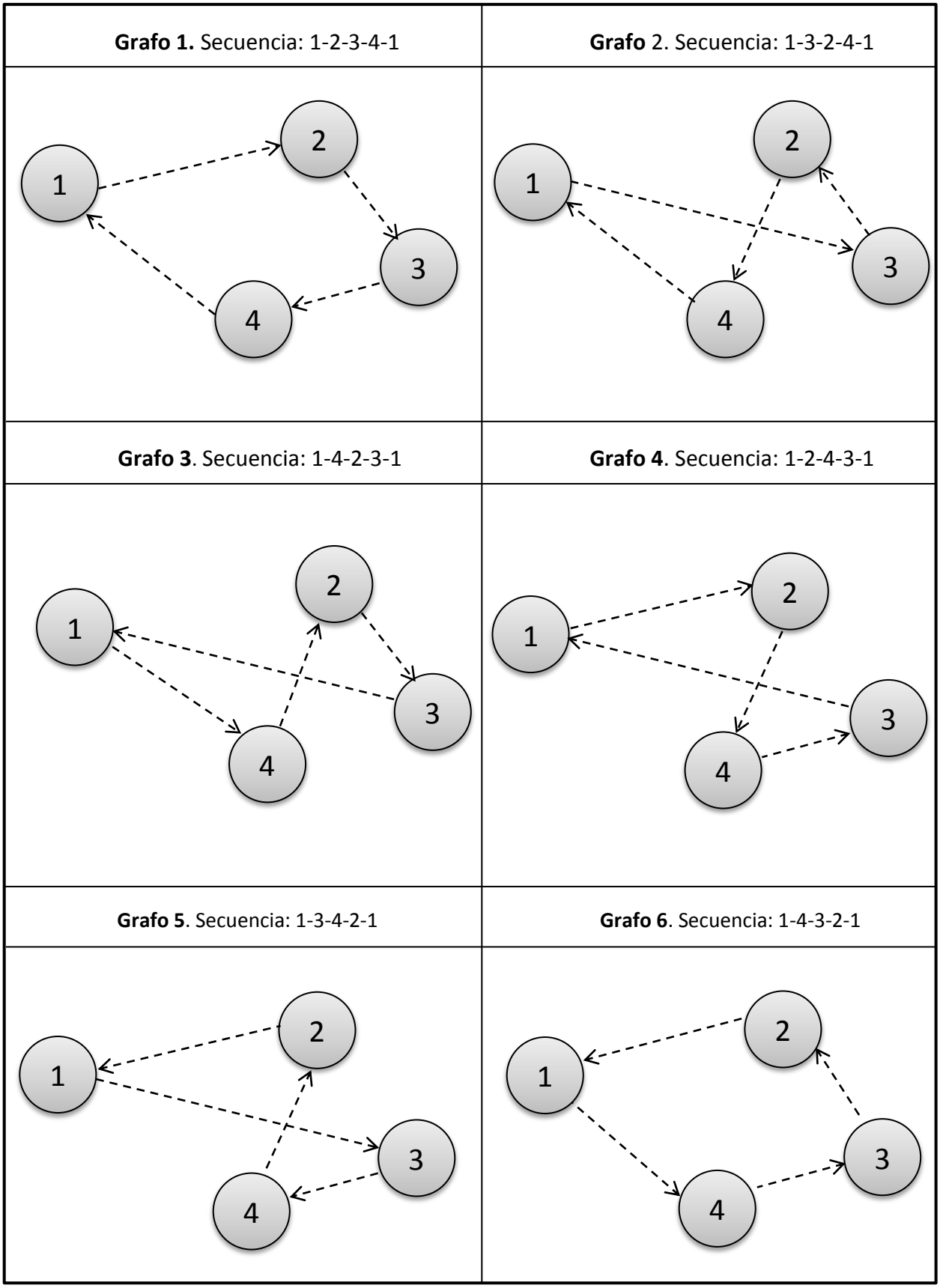


Figura 1.2. *Grafo para TSP con 4 nodos.*

1.3. EL PROBLEMA DEL VIAJANTE DE COMERCIO, TSP.

1.3.1. Problemas de Rutas.

Antes de explicar propiamente el problema del viajante es necesario realizar una breve introducción de los problemas de rutas de forma general, puesto que el TSP pertenece a este tipo de problemas.

Se entiende por un problema rutas como un problema de optimización cuando existe, por parte de unos clientes, una demanda de un servicio y se debe satisfacer ese servicio con el mínimo coste, en este caso, la mejor ruta que es la de menor distancia.

Existen numerosas aplicaciones en la vida real de este tipo de problemas como puede ser el transporte escolar, reparto de correo, en general a problemas de logística y distribución aunque no sólo para ellos, sino también para producción de circuitos electrónicos integrados o la secuenciación de tareas (problemas de tipo programación con restricciones)

1.3.2.1. Clasificación de los problemas de rutas.

Según la tabla 1.3, dependiendo de si el cliente se encuentra en los nodos o en los arcos del grafo o circuito y de si existen restricciones de capacidad se presentan diferentes tipos de problemas:

Demanda	Restricciones de capacidad	Nombre del problema	Otras restricciones
Nodos	NO	Viajante de Comercio, TSP	
	SI	Problema de rutas de vehículos, VRP	Recogida distribución
Arcos	NO	Una componente conexa, Problema del cartero Chino, CPP,	Ventanas de tiempo
		Varias componentes conexas (Problema del cartero rural RPP)	Otras
	SI	Problemas de rutas con capacidades, CARP.	

Tabla 1.3. *Calcificación de los problemas de rutas. (Calviño, 2011).*

El origen de los problemas de rutas cuya demanda se encuentra en los nodos se remonta el siglo XIX, cuando el británico T. Kirkman y el irlandés W.R. Hamilton inventaron un juego denominado "*Icosian Game*" el cual trataba de encontrar una ruta que fuera la de mínima distancia para los 20 nodos del juego teniendo en cuenta las restricciones de origen y fin, es decir, que se debía regresar a la misma ciudad de origen. Pero como es obvio, este juego no trataba de encontrar la ruta óptima, sino que se visitaran los 20 nodos del juego sólo una vez, no repetir visita dando lugar a un circuito cerrado, que años más tarde se conocerá y denominarán a este tipo de ciclos como circuitos *Hamiltonianos*. A partir de esta generalización, surgirá el problema del viajante y, posteriormente, sus variantes.

1.4. TSP (Traveling Salesman Problem).

1.4.1. Definición.

EL problema del viajante o comúnmente conocido como TSP (*Traveling Salesman Problem*) Puede definirse de la siguiente forma:

"Un viajante de comercio tiene que visitar una serie de ciudades pasando solo una vez por cada una, partiendo de su ciudad de origen y volviendo a ésta. ¿Cuál es la ruta óptima que debe elegir?"(Cook, 2012)

Ésta sería una definición informal para poder tener una idea intuitiva del problema que, a continuación, se describirá matemáticamente.

El problema del viajante es uno de los problemas más famosos de la matemática computacional, ya que juega un papel importante en la investigación de la resolución de problemas de complejidad computacional.

1.4.2. Origen e historia.

EL origen del TSP, todavía sigue siendo desconocido, puesto que no existe documentación que se lo atribuya a ningún matemático, varios fueron los que iniciaron este tipo de problema (Hamilton ya a principios del siglo XIX planteó el fundamento de lo que luego se denominó TSP).

En 1832, una guía para agentes viajeros titulado "el viajante de comercio: como debe ser y que debe hacer para conseguir comisiones y triunfar en el negocio" menciona el problema y, además, incluye viajes a través de Alemania y Suiza aplicando dicho


Der
Handlungsreiſende
wie er ſein ſoll
und was er zu thun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in ſeinen Geſchäften gewiß zu ſein.
Von
einem alten Commis - Voyageur.
Mit einem Titelkupfer.
Ilmenau 1832,
Druck und Verlag von B. Fr. Voigt.

Figura 1.3. *Portada del libro "El viajante de comercio", 1832.*

problema, pero esta guía no contempla un planteamiento más formal y, por lo tanto, matemático del TSP, a penas el último capítulo sí que resuelve el problema para 33 ciudades.

Sin embargo, la primera referencia que se hizo a este problema como TSP fue en 1949 en un artículo escrito por Julia Robinson "*On the Hamiltonian game (a traveling salesman problem*)". Aun así, parece que la Universidad de Princeton atribuyó el origen en torno a 1930 y no al artículo por Julia Robinson (Flood, 1984).

A partir de 1930, la Universidad de Harvard comienza a trabajar sobre este problema, en concreto Merill Flood, y también la Universidad de Viena de la mano de Karl Menger. Éste fue el que enunció, por primera vez, y definió la complejidad computacional del problema del viajante.

En los años 50 y 60, se plantean problemas de tamaño mucho mayor (para un número más grande de ciudades). En la publicación del artículo Dantzig del año 1954 "*Solution of a large scale traveling-salesman problema*" (Dantzig et al., 1954), en el que se resuelve por primera vez el problema del viajante para 49 ciudades, las cuales correspondía una por cada estado de EE.UU. Todo ello, supuso un gran avance en la investigación del TSP. A partir de la publicación de Dantzig, investigadores y matemáticos comenzaron a implantar y desarrollar algoritmos que permitiesen resolver el problema y que fuesen viables cuando se aumentaban el número de ciudades.

En la figura 1.4, se puede ver con facilidad el incremento de complejidad de las soluciones cuando se produce un aumento del tamaño del problema. En la imagen, aparecen tres rutas establecidas en Alemania. La más pequeña es la de color verde de 33 nodos que fue elaborada en 1832. La de color azul, elaborada por Grötschel, alcanzaba las 120 ciudades, y por último, la ruta en rojo contemplaba la mejor solución para 15112 ciudades, cuya solución se obtuvo gracias al empleo del Concorde, herramienta informática muy potente. Era un código en C de más de 13000 sentencia. Sus mentores fueron William J. Cook y V. Chvátal junto con el asesoramiento de unos técnicos de IBM.

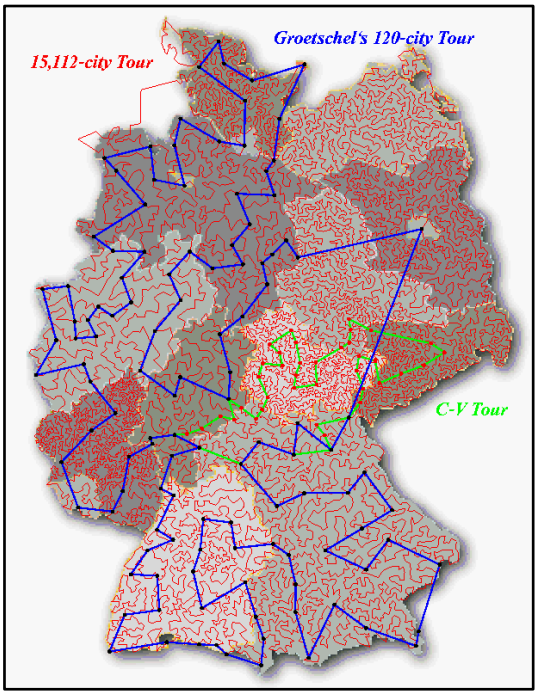


Figura 1.4. *Solución del TSP para tres rutas establecidas en Alemania (Cook, 2012)*

En la tabla 1.4 (Cook, 2012), se puede ver la evolución en el tiempo, cómo ha aumentado el número de ciudades a resolver en el TSP y de la mano de qué investigadores lo han resuelto:

Año	Investigadores	Número de ciudades
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held, R.M. Karp	57
1971	M. Held, R.M. Karp	64
1975	P.M. CAmerini, L. Frantta, F Maffioli	67
1975	P. Miliotis	80
1977	M. Grötschel	120
1980	H. Crowder y M. W, Padberg	318
1987	M. Pardberg y G. Rinaldi	532
1987	M. Grötschel y O. Holland	666
1987	M. Pardberg y G. Rinaldi	1002
1987	M. Pardberg y G. Rinaldi	2392
1992	Vasek Chvátal y William J.Cook	3038
1998	Vasek Chvátal y William J.Cook	13509
2001	Vasek Chvátal y William J.Cook	15122
2004	Vasek Chvátal y William J.Cook	24978
2006	Vasek Chvátal y William J.Cook	85900

Tabla 1.4.*Evolución en el tiempo de la resolución del TSP.*

1.4.3. Formulación matemática y modelización.

1.4.3.1. Representación gráfica del problema. TSP simétrico y asimétrico (ATSP).

El TSP es un problema de enrutamiento que se puede modelar mediante un grafo, puesto que el fin del mismo es encontrar una ruta o circuito cerrado cuyo coste sea mínimo, sujeto a una serie de restricciones: no se puede repetir visita a cualquiera de las ciudades posibles; y la ciudad de fin del circuito tiene que ser la misma que la de inicio (circuito cerrado o *hamiltoniano*)

Formalmente, se define de la siguiente forma aplicando la teoría de los grafos (Laporte, 1992):

"Sea un grafo definido como G=(N,A), siendo N el conjunto de n=|N| nodos que en nuestro caso son ciudades (N=1,…,n) y A es el conjunto de arcos que conectan a los nodos entre sí. Cada arco $(i,j) \in A$ se le asigna un coste con valor no negativo, en este caso el coste representado por la distancia entre nodos, d_{ij} (distancia entre el nodo i y j). El uso de los arcos (i,i), es decir ,la distancia del nodo i a si mismo no existe, es cero o, por lo tanto, un número lo suficientemente grande para que sea inviable ese arco (asignar a ese arco un coste muy elevado): $d_{ij} = \infty$ ó $d_i = 0$, $\forall \ i \in N$."

Si G es un grafo no dirigido, entonces la matriz de distancias será simétrica, d_{ij}=d_{ji} , y el problema al recibirá el nombre de STSP (en inglés, *Symmetric Travel Salesman Problem*), TSP simétrico. Mientras que si el grafo G es dirigido, depende de la dirección en la que se atraviesen los arcos, la matriz de distancias no será simétrica, por tanto, $d_{ij} \neq d_{ji}$ y el problema a resolver será TSP asimétrico (en inglés, *Asymmetric Travel Salesman Problem, ATSP*). Esto puede verse en la figura 1.5:

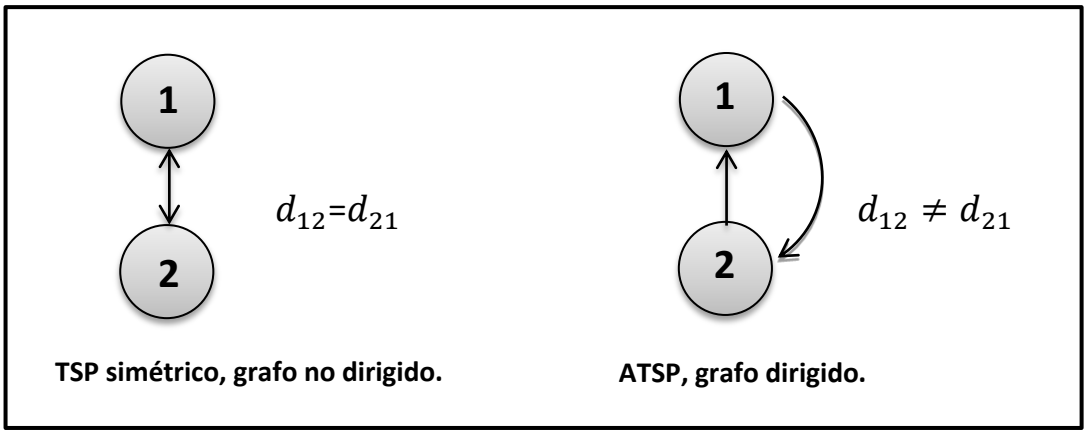


Figura 1.5. *Variantes del TSP en función de la dirección de los arcos.*

EL ATSP es una variante del TSP en la que como se puede ver en la figura 1.5, la dirección de una ciudad desde la i a la j no es misma distancia que de la j a la i. Es una versión del problema general del TSP, la única diferencia entre ambos es la restricción de direccionalidad: el ATSP da lugar a un grafo dirigido.

Nótese que esta versión es la que se utilizará en el capítulo 4 cuando se realice la implementación del algoritmo ACO (en inglés, *Ant Colony Optimization*) para resolverlo.

1.4.3.2. Formulación matemática.

El objetivo fundamental para el problema del viajante es encontrar una ruta que, satisfaciendo las restricciones asociadas a dicho problemas, ya comentadas anteriormente, minimice la distancia total recorrida. Si formulamos el problema planteándolo como un problema de programación lineal, es necesario definir una seria de variables de decisión, x_{ij} para todo $(i,j) \in A$, de tal forma que si las variables x_{ij} valen 1 significará que el arco (i,j) sí que forma parte del recorrido final, mientras que si toma 0 no lo harán. Según esto, la función objetivo, también denominado Makespan o Cmáx, que tenemos que minimizar para resolver el problema del viajante como un problema de programación lineal es la siguiente (1.3):

$$Cmáx = Min \sum_i \sum_j d_{ij} \cdot x_{ij} \quad (1.3)$$

Sujeto a las siguientes restricciones:

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in N \qquad (1.4)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in N \qquad (1.5)$$

La restricción (1.4) quiere decir que únicamente solo puede entrar en un nodo un único arco, mientras que la (1.5) indica que sólo un único arco puede salir de cada nodo. Pero aun así, el problema del viajante con este par de restricciones no queda completamente definido cumpliendo sólo la restricciones de visitar una sola vez cada nodo, pueden dar lugar a subcircuitos dentro del grafo y, con ello, se estaría violando la restricción de un único circuito cuyo nodo de fin es el mismo que el de inicio.

La siguiente figura nos muestra la posibilidad de subcircuitos definiendo únicamente las restricciones (1.4) y (1.5).

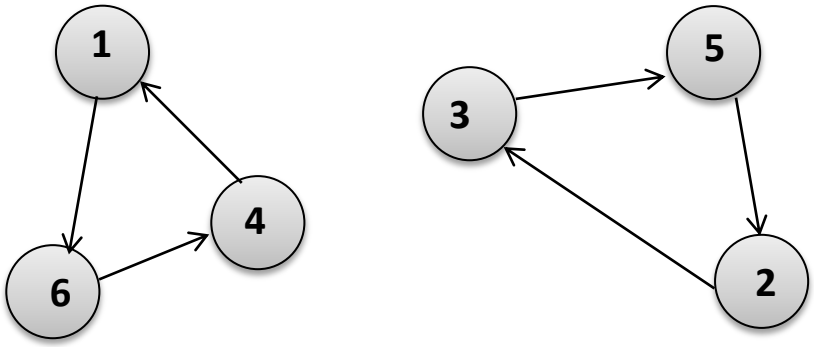


Figura 1.6. *Subcircuitos en el TSP.*

Para romper la formación de subcircuitos, es necesario definir otra variable de decisión. Sea $u_{i,}, \forall i \in N$, variable de decisión que nos va a mostrar el lugar de la secuencia en la que visitamos la ciudad i. Para definir un nodo de origen, tenemos que fijar un valor de u, el cual será $u_0 = 1$, ya que por ser el primer nodo se visita. Para el resto de nodos, la u puede tomar el valor desde 2 hasta n. Por tanto, al problema de programación lineal definido anteriormente le tenemos que añadir la siguiente restricción (1.6):

$$u_i - u_j \leq (n-1)\left(1 - x_{ij}\right) - 1 \quad \forall (i,j) \in A, j \geq 1 \quad (1.6)$$

Esta restricción define que si el agente va desde i hasta j, entonces la variable decisión $x_{ij} = 1$, esto hace que $u_i - u_j \leq -1$. Como se ha visitado antes, el nodo i que el j hace que el valor de $u_i < u_j$ de ahí que valga -1 la diferencia. Por contraposición, si el arco (i,j) no es recorrido por el viajante, entonces $x_{ij} = 0$, y la ecuación (1.6) resultará: $u_i - u_j \leq (n-2)$, pero en este caso, no sabemos si se ha visitado primero el nodo i o el j. Dependiendo de si se ha visitado uno u otro se cumplirá inecuación tomando un valor mínimo (visitamos j antes que i) o máximo (i antes que j) la diferencia $u_i - u_j$.

Finalmente, la formulación matemática para el ATSP sería la siguiente (Bais et al., 2012):

$$Min \sum_i \sum_j c_{ij} \cdot x_{ij} \quad (1.7)$$

Sujeto a:

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in N \quad (1.8)$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in N \quad (1.9)$$

$$\sum_{(i,j)\in E(W)} x_{ij} \leq |W| - 1 \quad \forall i \in N \qquad \forall W \; C \; V, \qquad W \neq \emptyset \;\; (1.10)$$

$$x_{ij} \in \{0.1\}, \; \forall (i,j) \in E \quad (1.11)$$

- (1.7): función objetivo. Minimiza la distancia total del problema.
- (1.8) y (1.9): restricciones que aseguran visitar una vez cada uno de los nodos.
- (1.10): restricción que asegura que no existan subcircuitos.
- (1.11): restricciones que asegura que las variables de decisión sean binarias.

Nótese que la restricción (1.10) es una inecuación, ya que para el caso del ATSP se cumple, pero sin embargo, si el problema que estuviésemos modelando será el TSP simétrico tendríamos que establecer la igualdad. Ésta es la diferencia que radica en la modelización de un tipo u otro.

1.5. VARIANTES DEL TSP.

El problema del viajante explicado en apartados anteriores se trata de la versión general, pero existen numerosas versiones y tipologías del mismo. A continuación, van a enunciarse algunas de las más importantes entre todas las existes (Calviño, 2011):

- **MAX-TSP:** problema del viajante que consiste en encontrar un circuito *hamiltoniano* cuya distancia o coste sea máximo.

$$Máx \sum_i \sum_j c_{ij} \cdot x_{ij}$$

- **TSP gráfico:** se trata de encontrar un circuito de mínima distancia o mínimo coste en el que se visiten, como mínimo, una vez todas las ciudades.

- **TSP con cuello de botella**: cuyo objetivo es encontrar un circuito *hamiltoniano* que minimice el mayor coste de entre todos los arcos del circuito, en vez de minimizar el coste total del mismo.

- **TSP agrupado:** en este problema lo que se realiza es una agrupación de los nodos o ciudades *clusters,* de tal forma que el objetivo final sea encontrar un circuito cerrado (*hamiltoniano*) cuyo coste sea el mínimo y el cual visite todos los nodos de cada *cluster,* consecutivamente.

- **TSP generalizado**: al igual que el TSP agrupado, los nodos también se encuentran divididos en *cluster* o grupos, pero en este caso, el objetivo final es

encontrar un circuito *hamiltoniano* cuyo coste sea mínimo en el que se visite de cada grupo únicamente un solo nodo, ver la figura 1.7:

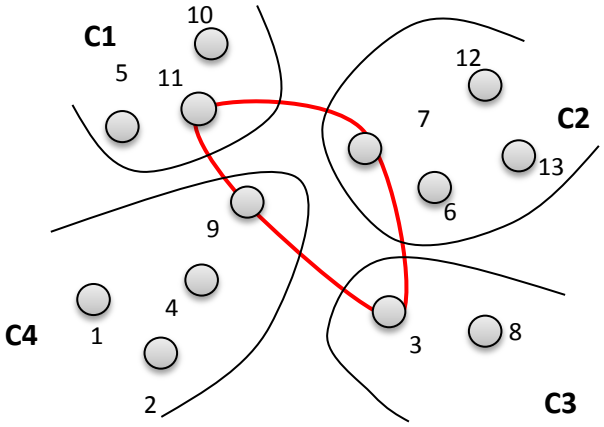


Figura 1.7. *TSP generalizado.*

- **TSP con múltiples viajantes (mTSP):** el punto de partida del problema es que consta de diferentes viajantes, un numero m, cada uno de ellos tiene que visitar algunas de las ciudades no todas, en la figura 1.8 aparece representado. El problema consiste en la búsqueda de una participación de las ciudades a visitar X1,…, Xm y de m ciclos, cada uno de los cuales para cada Xi, de tal forma que el sumatorio de todas y cada una de las distancias que los m viajantes recorren tiene que ser mínima.

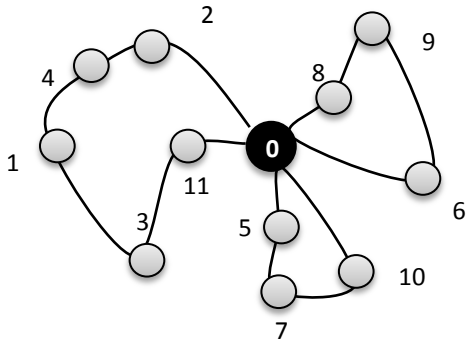


Figura 1.8. *Grafo para el TSP con múltiples viajantes.*

- **El TSP dinámico** (DTSP, *Dynamic Traveling Salesman Problem*), (Gunts et al., 2001) consiste en resolver el problema del viajante en el que las ciudades pueden añadirse o eliminarse en el tiempo de ejecución. La meta es encontrar tan rápido como sea posible la ruta de mínima distancia después de cada transición. Al igual que el TSP, tiene las mismas restricciones que éste. En cuanto a la construcción del grafo, se define un grafo G=(C,L), donde C=C(t) es

el conjunto de ciudades y L=L(t) es el conjunto de arcos. La dependencia de C y L con el tiempo es debido a la naturaleza dinámica del problema. Realmente no se le considera una variante sino un problema distinto.

1.6. APLICACIONES DEL TSP.

EL TSP como ya se introdujo al comienzo de este capítulo, suscita un gran interés debido a las numerosas aplicaciones que tiene. Por ello, se sigue realizando investigaciones para optimizar las técnicas y algoritmos de resolución que puedan ofrecernos una solución óptima y fiable. Hasta nuestros días, el TSP se ha aplicado a numerosas actividades, una amplia variedad de problemas que abarca desde grandes problemas de logística y distribución hasta los circuitos eléctricos incluso la genética. A continuación, se expondrán aquellas consideradas más importantes:

- **Logística y distribución**. La aplicación más importante y común y una de las primeras que se realizo fue en el campo de la logística. La esencia del TSP es transportar personas, vehículos, mercancías de una ciudad a otra y así sucesivamente, realizando un circuito cerrado que sea el de mínimo coste, esto en líneas generales coincide con el fundamento de la logística. Entre la amplia variedad de aplicaciones logísticas del problema del viajante, se encuentran:

  - *Vendedores y turistas*: los turistas suelen utilizar un itinerario o guía para planificar sus rutas y determinar cuál de todas es el mejor camino para realizar un circuito cerrado en el que se parta y finalice en la misma ciudad. Un ejemplo común puede ser cuando los turistas que desean visitar los monumentos más significativos de una ciudad y, para ello, parten del hotel y al finalizar el recorrido regresan al mismo.

  - *Rutas escolares,* siendo ésta una de las primeras aplicaciones del TSP por Merrill Flood. En la actualidad, existen empresas que han adquirido software muy potente capaz de resolver el TSP encontrando una ruta que permita reducir los costes de transporte y con ello, costes económicos obteniendo un margen de ganancia significativo.

- **Industria**. También muy comunes dentro de este campo y entre las cuales se encuentra:

  - *Secuenciación y programación de tareas en una máquina* (González y Ríos, 1999). En muchas ocasiones, sucede que en un taller de fabricación o manufactura se dispone de una sola máquina en la que se pueden procesar diferentes tareas, de tal forma que una vez que una tarea ha sido finalizada, se necesita preparar la máquina para procesar la siguiente tarea. Para realizar esto, se necesita invertir un tiempo que dependerá de la tarea que

se acaba de procesar y de la que se procesará a continuación. Este tiempo que se invierte impide que se ejecute operación alguna en la máquina, por lo que supone tiempo perdido y un costo de oportunidad para la empresa. Por lo tanto, hay que encontrar un orden de procesamiento de las tareas con el fin último de reducir al mínimo el tiempo total perdido. Este problema puede plantearse como el TSP: cada tarea se considera como una ciudad a visitar, y el tiempo necesario de preparación de la maquina se corresponde de forma análoga con la distancia entre dos ciudades. Encontrar la forma óptima de procesar las tareas en la maquina se corresponde a encontrar la ruta de mínima distancia para visitar todas las ciudades.

- *Producción de circuitos electrónicos*. Muy curioso y característico en este campo. El empleo del TSP para esta actividad se base principalmente en la secuencia óptima en la que se taladran las placas y los caminos necesarios (rutas) para conectar los chips entre sí.

Capítulo 2.

Métodos de resolución para los problemas NP-hard.

En este capítulo, se expondrán las diferentes técnicas existentes para la resolución de los problemas NP-hard. En primer lugar, se clasificarán en tres grupos fundamentales: método de resolución exactos como pueden ser las basadas en el cálculo o las enumerativas (estos métodos no serán descritos puesto que no son objeto de estudio); método de aproximación o algoritmos de aproximación (algoritmos basados en el comportamiento social organizativo de agentes: comportamiento colectivo. Dentro de este tipo, se encuentran los algoritmos genéticos, algoritmo basado en colonia de hormigas, enjambre de partículas,... Y por último, otras técnicas distintas a las anteriores, las cuales serán descritas brevemente.

Dentro de las distintas técnicas para la resolución de este tipo de instancias, se otorgará mayor importancia y, por tanto, se explicarán detalladamente los algoritmos de aproximación y, dentro de éstos, las metaheurísticas. Éstas contemplan a los algoritmos basados en colonia de hormigas, ACO que será la herramienta que utilizaremos para resolver el problema ATSP en el capítulo 4.

2.1. TÉCNICAS DE RESOLUCIÓN PARA LOS PROBLEMAS NP- HARD. CLASIFICACIÓN.

Los problemas de optimización combinatoria se relacionan con los denominados problemas NP-hard. Estos problemas, en función del tamaño de la instancia que se trate podrán ser resueltos por unos métodos u otros. A continuación, se enuncian las diferentes técnicas de resolución de un problema de optimización combinatoria NP-hard:

- **Métodos exactos**
- **Métodos de aproximación o algoritmos de aproximación**.
- **Otras.**

Para poder tener una idea intuitiva de la clasificación de las técnicas de optimización, se muestra el siguiente esquema (ver figura 2.1). Posteriormente, se explicarán cada una de ellas aunque no todas, sólo las que se consideran de mayor importancia por su aplicación y utilización en el ámbito de la ingeniería.

2.1.1. Métodos exactos.

Los métodos exactos nos garantizan encontrar una solución óptima y probar su optimización para cualquier tamaño que tenga el problema de optimización combinatoria a resolver en un tiempo de ejecución aceptable (Dorigo y Stützle, 1997, 2004). En el caso de los problemas NP-hard o NP- duros, los métodos exactos necesitan, en el peor de los casos, un tiempo exponencial para encontrar el óptimo. Aunque para algunos problemas específicos el método exacto ha sido mejorado significativamente en los últimos años, para la mayoría de los problemas NP-hard la representación de los métodos exactos no es satisfactoria. Por ejemplo, para el problema de asignación cuadrática (en inglés, *quadratic assignment problema*) QAP, un problema importante que surge en aplicaciones reales, cuyo objetivo es encontrar una asignación óptima para los n elementos de n lugares que consta el problema a tratar. En este problema, cuando tratamos casos en los que el tamaño excede los 30 elementos, resolverlo mediante técnicas basadas en algoritmos exactos resulta prácticamente imposible, por lo que ése se considera el límite de aplicación de estas técnicas.

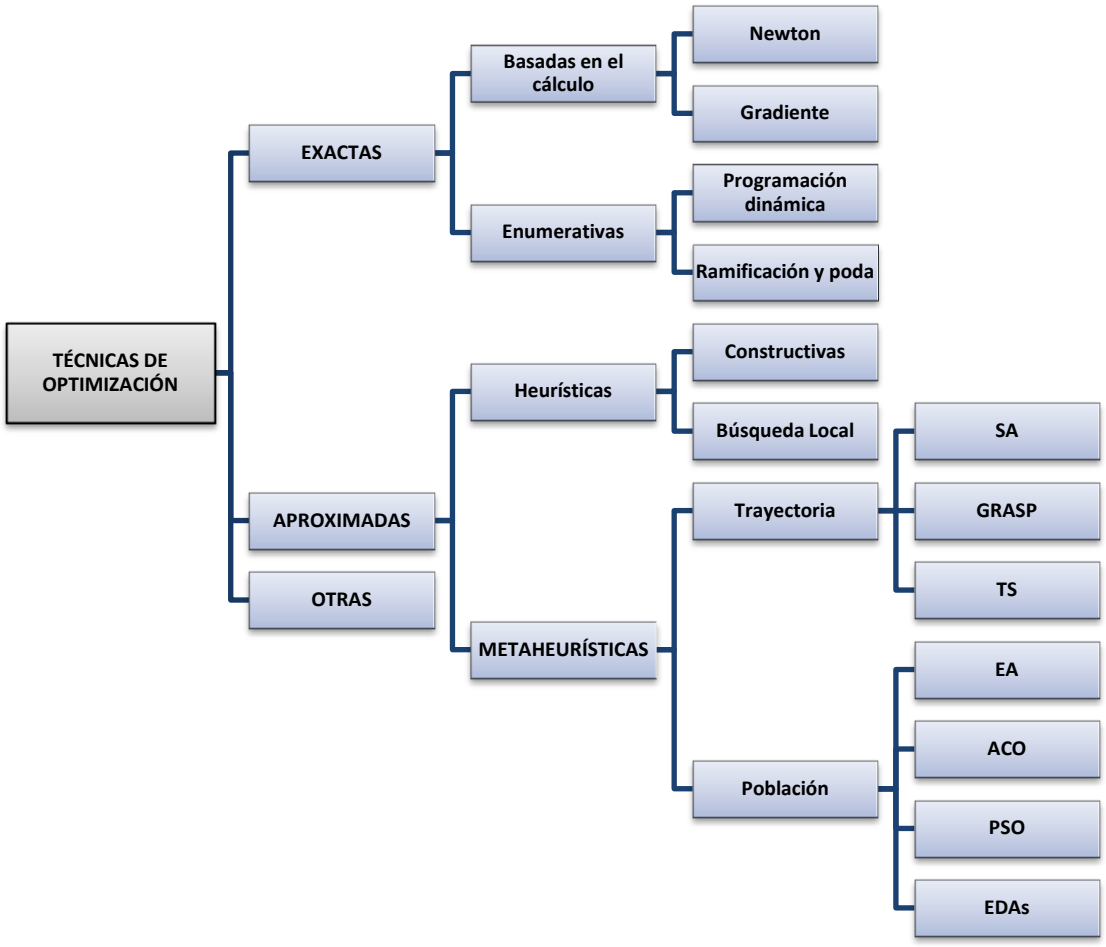


Figura 2.1.*Clasificación de las técnicas de optimización (Luna, 2008).*

Además, para la complejidad exponencial, la aplicación de un método exacto o algoritmo para los problemas NP-hard, en la práctica, también sufren un elevado aumento del tiempo de computación cuando el tamaño del problema incrementa, y con frecuencia, su uso rápidamente llega a ser ineficiente y no fiable: no proporcionan una buena calidad de las soluciones.

Si las soluciones óptimas no pueden ser obtenidas de una forma eficiente en la práctica, la única posibilidad es realizar una optimización basada en la eficiencia del algoritmo en cuanto a la búsqueda y cálculo del óptimo o de una solución próxima al óptimo. En otras palabras, la garantía de encontrar soluciones óptimas puede ser sacrificada por conseguir una buena solución de alta calidad en tiempo polinómico. Para ello, se emplearán los métodos basados en algoritmos de aproximación que son descritos en el apartado siguiente.

2.1.2. Métodos basados en algoritmos de aproximación.

Como ya apunte en el apartado anterior, primero se explicarán dentro de los algoritmos de aproximación, las heurísticas entre las que se encuentran los métodos constructivos o heurísticas constructivas y la búsqueda local.

Frecuentemente, los métodos de aproximación (Dorigo y Stützle, 2004) también son conocidos como métodos heurísticos o simplemente heurísticos. Estos métodos, lo que proporcionan, es obtener soluciones cercanas o muy próximas al óptimo en cuestión, sin infligir en una mala calidad de la solución encontrada. Dentro de este tipo de métodos, encontramos distintos subtipos: métodos constructivos o búsqueda local.

2.1.2.1. Métodos constructivos.

Los algoritmos constructivos construyen una solución para problemas de optimización combinatoria de una forma incremental. Paso a paso y sin volver atrás, añaden componentes a la solución hasta que la solución completa es generada. Aunque el orden en el que se añaden los componentes puede ser aleatorio, normalmente algunas clases de reglas heurísticas es empleada. A veces, heurísticas de construcción *Greedy* son usadas, donde en cada paso de la construcción, se añade un componente de solución.

El siguiente pseudocódigo (ver figura 2.2) nos muestra cómo se realiza una construcción heurística Greedy. A continuación, se explicará cada una de las funciones empleadas en el mismo.

PSEUDOCÓDIGO HEURÍSTICA GREEDY

```
Procedure HeuristicaConstGreedy {
Sp ← ElegirPrimerComponente
        Do While (Sp no sea una solución
completa){
                c ← ComponenteGreedy (Sp)
                Sp ← Sp ∗c
        } End-while
        S ← Sp
        Return s
} End-procedure
```

Figura 2.2. *Pseudocódigo de la heurística de Greedy. (Dorigo y Stützle,2004)*

- La función ***ElegirPrimerComponente*** elige el primer componente de la solución (esto puede realizarse de forma aleatoria o de acuerdo a la heurística de la construcción)
- ***ComponenteGreedy*** devuelve un componente c de solución con la mejor heurística estimada.
- La adicción del componente c a la solución parcial Sp es denotado por el operador *.
- El procedimiento devuelve una solución completa de s.

Un ejemplo de algoritmo constructivo para el problema del viajante es el procedimiento de elegir la ciudad más cercana, en el que se trata las ciudades como componentes. El procedimiento que se lleva a cabo es el siguiente:

Se elige, de forma aleatoria, la ciudad inicial y posteriormente se añade, iteración tras iteración, la ciudad más próxima entre el resto de ciudades posibles por visitar a la solución bajo construcción.

Para poderlo ver de una forma más visual se adjunta la siguiente figura en la que la ciudad de inicio es la a y, secuencialmente, se va añadiendo la ciudad más próxima:

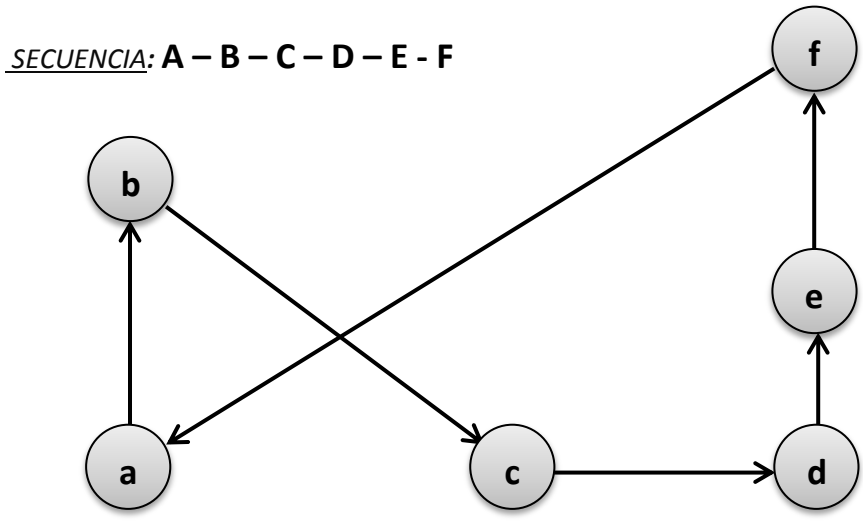


Figura 2.3. *Grafo resuelto mediante la heurística de Greedy.*

2.1.2.2. Búsqueda local.

La búsqueda local es una aproximación general para encontrar soluciones de alta calidad para problemas de optimización combinatoria duros en un tiempo computacional razonable. Este método está basado en una exploración iterativa de vecindarios de soluciones intentando mejorar la solución actual mediante cambios locales. Los tipos de cambios locales que pueden ser aplicados a una solución están definidos por la estructura de vecindario.

Definición (1). Estructura de vecindario (en inglés, Neighborhood structure)

Una estructura de vecindario es una función $N: S \rightarrow 2^S$ que asigna un conjunto de vecinos $N(s)\ contenido\ en\ S \quad \forall\ s \in S.\ \ N(s)$ también se denomina vecindario de s.

La elección de una estructura de vecindario es crucial para la representación de un algoritmo de búsqueda y es específica para cada problema. La estructura de vecindario define el conjunto de soluciones que puede ser investigada para s en un único paso del algoritmo de búsqueda local. Típicamente, una estructura de vecindario está definida implícitamente por los posibles cambios locales que pueden ser aplicados para una solución, y no por la explicación enumerada del conjunto de todos los posibles vecinos.

En definitiva, las estructuras de vecindario definen para cada solución actual, el conjunto de soluciones posibles, para las cuales, los algoritmos de búsqueda local pueden aplicarse e implementarse resultando eficaces.

Otro concepto que aparece en este método es el de óptimo local.

Definición (2). Optimo local.

Un óptimo local para un problema en el que se quiere maximizar o minimizar es una solución s tal que $\forall s' \in N(s): f(s) \leq f(s')$. De forma análoga, para un en un problema en el que se quiere maximizar la función objetivo, un óptimo local es una solución *s* tal que $\forall s' \in N(s): f(s) \geq f(s')$.

2.1.3. Otro tipo de técnicas.

Dentro de este grupo (Brito et al., 2004) se pueden agrupar a todas aquellas técnicas de optimización que corresponden a tipos intermedios entre las anteriores. Entre ellas, destacan las metaheurísticas de descomposición y las de memoria a largo plazo.

Las primeras, se conforman subproblemas para resolver el problema principal. A partir de las soluciones obtenidas de los subproblemas, se construye la solución del problema original. El objetivo principal es obtener subproblemas que resulten más

fáciles de resolver que el problema original de tal forma que se simplifique la dificultad del mismo. Este tipo de metaheurística es muy importante para la aplicación de estrategias de paralización en las que el equilibrio entre los subproblemas obtenidos a partir del problema original es crucial.

En cuanto a las metaheurísticas de memoria a largo plazo, pueden considerarse como derivadas, en una parte, de la búsqueda tabú. Son capaces de utilizar información obtenida de la aplicación del propio procedimiento, ya sea un problema específico o bien, una clase concreta de problemas: problemas tipo. En muchas ocasiones, se las considera el caso más importante de las denominadas metaheurísticas de aprendizaje

2.2. METAHEURÍSTICA.

Una desventaja de los algoritmos (Blum y Roli, 2003) como de los métodos constructivos o iterativos es que o generan únicamente un número muy limitado de soluciones diferentes, como es el caso de una construcción de heurística Greedy, o se paran en un óptimo local de poca calidad, en el caso de los métodos iterativos de mejora. Desafortunadamente, los algoritmos de búsqueda local no producen mejoras significativas en la práctica. Varias aproximaciones generales que hoy en día se conocen como técnicas metaheurísticas han sido propuestas para intentar eludir este problema.

En 1977, Glover fue uno de los pioneros en introducir por primera vez el término de metaheurística (Glover, 1977). Este término lo empleó para designar todos aquellos métodos que de diversas formas, según su implementación, integran procedimientos y estrategias de alto nivel para la mejora local de soluciones, evitar el estancamiento en los óptimos locales y, con ello, realizar una búsqueda eficiente en el espacio de búsqueda de soluciones de los problemas a tratar.

La optimización en el sentido de encontrar la mejor solución, o una solución que se considere lo suficientemente buena (calidad aceptable) para un problema en concreto es uno de los campos más importantes del mundo real, y más en particular, de la ingeniería. Sin que a veces no seamos conscientes, constantemente estamos resolviendo problemas de optimización relativamente sencillos pero que rápidamente se complican cuando el tamaño del problema a resolver aumenta; por ejemplo, como el camino más corto para ir de un lugar a otro, la organización de una agenda entre muchos otros. Como acabo de apuntar, estos problemas son lo suficientemente pequeños y podemos resolverlos sin ninguna ayuda adicional, pero cuando se van haciendo más complejos, es decir, se va incrementando el tamaño del problema, el uso de los ordenadores es necesario e imprescindible para su resolución.

2.2.1 Definición.

Una metaheurística (Aardal et al., 2007) es un conjunto de conceptos algorítmicos que pueden ser usados para definir métodos heurísticos aplicables a un conjunto de problemas diferentes. En otras palabras, una metaheurística puede ser vista como un método heurístico general diseñado para guiar un problema heurístico específico (algoritmo búsqueda local o método constructivo) hacia regiones del espacio de búsqueda que contienen soluciones de alta calidad.

Una definición más formalmente es la siguiente:

Un problema de optimización se formaliza como un par (S,f) donde S es un conjunto distinto del conjunto vacío y representa el espacio de soluciones (o de búsqueda) del problema, mientras que f es una función denominada función objetivo o función fitness que se define de la siguiente forma (2.1):

$$f: S \rightarrow R \quad (2.1)$$

Donde R es el conjunto de números reales.

Así, resolver un problema de optimización consiste en encontrar una solución, i∈ S, que satisfaga la siguiente desigualdad (2.2):

$$f(i^*) \leq f(i), \quad \forall\, i \in S \quad (2.2)$$

Que un problema sea de maximización o minimización no restringe la generalidad de los resultados, ya que se puede establecer una igualdad entre tipos de problemas de maximización y minimización de la siguiente forma, ver expresión (2.3):

$$máx\{f(i)|i \in S\} \equiv mín\{-f(i)|i \in S\} \quad (2.3)$$

Cuando hablamos de técnicas metaheurísticas, nos referimos a los algoritmos de aproximación de propósito general, los cuales consisten en procedimientos iterativos que guían a una heurística subordinada combinando, de forma inteligente, distintos conceptos para explorar y explotar, adecuadamente, el espacio de búsqueda.

Las ventajas e inconvenientes que presenta la metaheurística, en líneas generales, son las siguientes (ver tabla 2.1):

VENTAJAS	INCOVENIENTES
Algoritmos de propósito general	Son algoritmos aproximados no exactos
Gran éxito en la práctica, rapidez de ejecución de los programas	Son altamente no determinísticos (probabilísticos)
Fácilmente implementarles (utilizan lenguajes de alto nivel) → flexibilidad en cuanto a cambios en el programa.	Presentan poca base teórica

Tabla 2.1. *Ventajas e inconvenientes de las Metaheurísticas*.

Como ya se ha mencionado al principio de este apartado, el uso de la metaheurística es interesante por diversos motivos, principalmente cuando no hay un método exacto de resolución o éste requiere mucho tiempo de cálculo; y memoria, por lo que se cataloga el método exacto como ineficiente; también, cuando no se necesita la solución óptima, basta con una buena calidad de la misma (búsqueda de óptimos locales, etc.).

2.2.2. Funcionamiento de las metaheurísticas.

Hay que tener en cuenta que el funcionamiento de las metaheurística se basa en dos principios necesarios y en el equilibrio entre ambos: hay que identificar rápidamente aquellas regiones del espacio de búsqueda con soluciones de buena calidad y, además, no consumir mucho tiempo en aquellas regiones del espacio no prometedoras o ya exploradas, lo cual resultaría perdida de la eficiencia del algoritmo de búsqueda y una pérdida de la calidad de la solución obtenida.

2.2.3. Clasificación de las metaheurísticas.

Varias son las formas que existen para clasificar y describir las técnicas metaheurísticas. Dependiendo de las características a las que nos ciñamos, vamos a tener unas clasificaciones u otras: aquellas que están basadas en la naturaleza y las no basadas en la naturaleza, con memoria o sin ella, con una o varias estructuras de vecindario, etc. Teniendo en cuenta la clasificación que aparece en la figura 2.4 nos encontramos con metaheurísticas basadas en trayectoria y las basadas en población. Las primeras manipulan, en cada paso, un único elemento del espacio de búsqueda, mientras que las segundas, trabajan sobre un conjunto de poblaciones. Más esquemáticamente, se resume en la siguiente figura (2.4) que además, incluye aquellas técnicas que son las más representativas dentro de cada tipo de metaheurísticas:

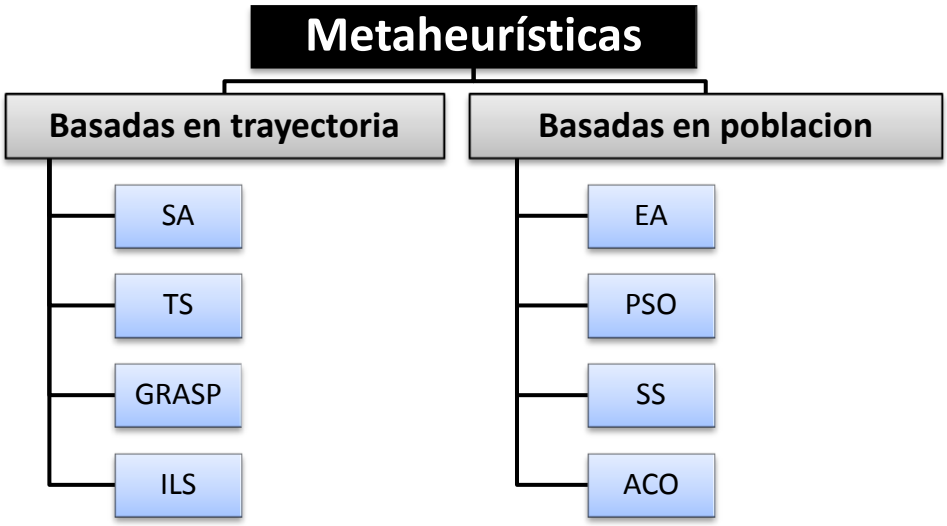


Figura 2.4.*Clasificación de las Metaheurísticas.*

Para saber a qué tipo de metaheurística nos estamos refiriendo, en la tabla 2.2 se muestra el significado de las siglas con las que se conoce comúnmente a las técnicas que aparecieron en la figura anterior.

Sigla	Descripción
SA	*Simulated Anneling*
TS	*Tabu Search*
GRASP	*Greedy Randomized Adaptative Search Procedure*
ILS	*Iterated Local search*
EA	*Evolutionary Algorithm*
PSO	*Particle Swarn Optimization*
SS	*Scatter Search*
ACO	*Ant Colony Optimization*

Tabla 2.2. *Nomenclatura de las principales*

2.2.3.1. Metaheurísticas basadas en trayectorias.

La principal características de este tipo de metaheurísticas es que parten de una solución, y mediante la exploración del vecindario, van actualizando la solución actual formando una trayectoria. Normalmente, se termina la búsqueda cuando se alcanza un número máximo de iteraciones prefijado, entonces se encuentra una solución con una calidad aceptable o, por el en contrario, se detecta un estancamiento de proceso

iterativo. A continuación, aparece descritas los siguientes tipos que engloban las metaheurísticas basadas en trayectoria:

- ***Recocido simulado*** (en inglés, *Simulated Annealing*), (Cerný, 1985) es una de las técnicas más antiguas entre las metaheurísticas. La idea principal de la que se basa esta técnica es simular el proceso de enfriamiento del metal y del cristal. El SA para evitar el estancamiento y quedarse atrapado en un mínimo local, el algoritmo lo que permite es elegir una solución con una cierta probabilidad cuyo valor Cmáx sea peor que el de la solución actual. Por tanto, en cada iteración se elige, a partir de una solución actual s, una solución s' del vecindario N(s). Si s' es mejor que s, entonces se sustituirá el valor de s por el de s'. En caso contrario, se acepta con una determinada probabilidad que depende de la temperatura actual T y de la diferencia de Cmáx entre ambas soluciones, f(s')-f(s) cuando se trata de minimizar el problema en cuestión.

- ***Búsqueda Tabú*** (en inglés, *Tabu Search*). En 1989, Glover y Laguna fueron los pioneros y los que introdujeron por primera vez lo que se conoce como búsqueda tabú. Es una de las metaheurísticas que se aplican con más éxito en los problemas NP-hard. Su idea principal consiste en el uso de un "historial de búsqueda" para evitar que se produzca un estancamiento y escapar de los mínimos locales y evitar buscar varias veces en la misma región del espacio de soluciones. El historial de búsqueda o también puede ser llamado memoria a corto plazo es implementada con una lista tabú, donde se mantienen las soluciones visitadas más recientemente para excluirlas de los próximos movimientos. En cada iteración, se elige la mejor solución entre las posibles y ésta es añadida a la lista tabú.

- ***GRASP.*** El procedimiento de búsqueda aleatorizado y adaptativo (en inglés, *Greedy Randomized Adaptative Search Procedure*), introducido por Feo y Resende en 1989, es una metaheurística simple que combina heurísticos constructivos con búsqueda local. GRASP es un procedimiento iterativo que consiste en dos fases: fase construcción y fase búsqueda local. La solución mejorada es el resultado del proceso de a fase búsqueda local. El mecanismo de construcción de soluciones es mediante una heurística aleatoria. Lo que se hace es añadir en cada paso, un componente c a la solución parcial Sp, que antes de implementar el algoritmo, como es lógico, está vacía. Los componentes que se añaden en cada paso son elegidos aleatoriamente de una lista restringida de candidatos. Esta lista es un subconjunto de N(Sp), el conjunto de componentes permitidos para la solución parcial Sp.

- ***Búsqueda Local Iterada*** (en inglés, Iterated Local search) (Martin et al., 1991) es una metaheurística basada en un concepto simple pero muy efectivo. En cada iteración, la solución actual es perturbada y, a esta nueva solución, se le aplica un método de búsqueda local para mejorarla. El mínimo local obtenido por el método de mejora puede ser aceptado como una nueva solución actual si pasa un test de aceptación. Si la perturbación es demasiado pequeña, entonces puede que el algoritmo no sea capaz de escapar del mínimo local; mientras que si es demasiado grande, la perturbación puede hacer que algoritmo sea como un método de búsqueda local con reinicio aleatorio. Además, el método de perturbación debe generar soluciones que sirvan como inicio a la búsqueda local, pero que no debe estar muy lejos de la solución actual para que no sea una solución aleatoria. El criterio de aceptación actúa como contra-balance, ya que filtra la aceptación de nuevas soluciones dependiendo de la historia de búsqueda y de las características del nuevo mínimo local.

2.2.3.2. Metaheurísticas basadas en población.

A diferencia de los anteriores, estas metaheurísticas se caracterizan por trabajar con un conjunto de soluciones, los anteriores manipulaban sólo una solución del espacio de búsqueda por cada iteración.

- ***Algoritmos evolutivos*** (en inglés, *Evolutionary Algorithm*). La característica principal de estas metaheurísticas es que están inspiradas en la teoría de la evolución natural como son los Algoritmos Genéticos creados por Holland en 1975 y, posteriormente, desarrollados también por Goldberg en 1989. Esta familia de técnicas sigue un proceso iterativo y estocástico que opera sobre una población de soluciones a las que se va a denominar individuos. Inicialmente, tenemos que generar la población de individuos, lo cual generalmente, se hará de forma aleatoria (a veces con la ayuda de una heurística constructiva). Todo algoritmo evolutivo consta de tres fases principales: selección, reproducción y sustitución. Este proceso es repetido en cada iteración y será necesario repetirlo hasta que se cumpla el criterio de parada que hayamos prefijado (máx. numero de iteración, alcanzar un valor concreto de solución,…)

    - *Selección:* se eligen mediante diversas técnicas aleatorias aquellos individuos de la población inicial más aptos. La selección suele realizarse por Torneo o Ruleta son las técnicas aleatorias más usadas y comunes.

- *Reproducción:* los individuos seleccionados son aquellos que se reproducen y se recombinan entre sí mediante el operador de cruce entre parejas de individuos y luego mutación.

- *Sustitución:* por último, a partir de la población actual y/o los mejores individuos generados, de acuerdo con el valor de cada uno de ellos de la función objetivo, se forma la nueva población, que será la población inicial de la siguiente iteración del algoritmo.

- ***Búsqueda Dispersa*** (en inglés, *Scatter Search*). El algoritmo de esta metaheuristica se basa en mantener un conjunto relativamente pequeño de soluciones "tentativas" (este conjunto es denominado como conjunto de referencia o *RefSet*) y su característica principal es que contiene soluciones diversas, quiere decir distantes en el espacio de búsqueda, y de calidad. Para crear un algoritmo SS, tenemos que seguir los siguientes pasos: creamos una población inicial, aplicamos un método de combinación de soluciones y, por último, método de mejora.

- **Optimización basada en colonia de hormigas** (en inglés, *Ant Colony Optimization*). Los algoritmos basados en colonias de hormigas están inspirados en el comportamiento real de las hormigas cuando se trata de organizarse en comunidades y cuando quieren buscar comida para llevar a su hormiguero. En líneas generales, ya que se detallará en profundidad esta metaheurística en el capítulo 3, el comportamiento de las hormigas es el siguiente: primero exploran el área cercana a su nido (*neast*) de forma aleatoria. Cuando una hormiga encuentre comida (*Food*) la lleva a su nido. Mientras está realizando este camino, la hormiga va depositando una sustancia química denominada feromona. La feromona lo que hace es ayudar al resto de las hormigas a encontrar la comida. Gracias a esta comunicación entre las hormigas, a través del rastro de feromona, lo que las permite es encontrar el camino más corto entre el nido y la comida. Este comportamiento es en lo que se basa el algoritmo, por tanto, va a estar formado de las siguientes partes: *Inicializar parámetros, Creación de hormigas, Búsqueda local, Actualización de feromona, Obtención de mejores resultados.*

- ***Optimización basada en enjambre de partículas*** (en inglés, Particle Swarm Optimization). Los algoritmos de optimización basados en enjambre de partículas están fundamentados en el comportamiento social del vuelo de las abejas. El conjunto de soluciones de este tipo de algoritmo se denomina partículas, las cuales son inicializadas aleatoriamente en el espacio que estamos considerando de búsqueda. A diferencia de la metaheurística ACO, en el PSO tenemos que tener en cuenta que cada partícula se encuentra en

movimiento pero es aquí donde radica la diferencia: dicha partícula posee una posición y velocidad que influirá, lógicamente, en el movimiento de la partícula. A diferencia del término estructura de vecindario, descrito en el capítulo 2, aquí se emplea este término con otro sentido: el vecindario de una partícula puede ser global o local. El primero quiere decir que todas las partículas del enjambre se consideran vecinas, mientras que en el segundo quiere decir que sólo son vecinas aquellas que se encuentran más cercanas.

Capítulo 3.

Metaheurística Colonia de Hormigas. ACO (*Ant Colony Optimization*).

En este capítulo se va a tratar y describir tanto el fundamento de la técnica metaheurística Colonia de Hormigas (ACO), así como su aplicación y desarrollo para los problemas de optimización combinatoria. En primer lugar, se abordará el origen y fundamento teórico de esta técnica, su comportamiento social organizativo, y sobretodo la importancia que tiene la comunicación indirecta que tiene lugar entre ellas (estimergia), basada en el rastro de feromona (sustancia química que la hormiga segrega y deja por el camino que recorre). Posteriormente, se plantearán las analogías y diferencias que existen entre el comportamiento de un agente real (hormiga natural) y una hormiga artificial), y a continuación, la metaheurística ACO propiamente dicha, es decir, su pseudocódigo, desarrollo e implementación para problemas que puedan representarse mediante un grafo, G. Por último, sus aplicaciones en problemas de optimización combinatoria y en la industria en general.

3.1. ORIGEN E INTRODUCCIÓN A LA METAHEURÍSTICA ACO.

3.1.1. Origen.

En los últimos años, se ha venido desarrollando una nueva técnica metaheurística de inspiración biológica que se basa en el comportamiento real de las hormigas naturales para la resolución de problemas de optimización combinatoria. Esto es debido a que, por desgracia, los algoritmos de búsqueda local presentan facilidad para estancarse en óptimos locales, que globalmente no son buenas soluciones. Por ello, en las últimas décadas investigadores en este tipo de técnicas de optimización, se han implicado en desarrollar algoritmos que mejoren la efectividad de los métodos de búsqueda local basándose en la heurística para la construcción de soluciones, es decir, la meta heurística.

Su origen se remonta a 1989 de la mano de Goss, Aron, Deneubourg y Pasteels, quienes realizaron un estudio sobre el comportamiento colectivo de las hormigas argentinas. Este estudio será el origen e inspiración de lo que se conoce como algoritmos de optimización de colonias de hormigas. En 1991, Marco Dorigo (Dorigo, 1991) propone su tesis doctoral el método Ant System. A partir de este hecho, se sucedieron numerosas publicaciones sobre los algoritmos de colonia hormigas por varios investigadores como Stützle, Bonabeau, Gambardella, Maniezzo y Colorni.

3.1.2. Introducción.

En la naturaleza, existen numerosas especies que presentan comportamientos auto-organizativos, originando lo que se ha denominados como "*swarm intelligence*" (inteligencia de enjambres). La inteligencia de enjambres engloba un campo de la investigación que estudia algoritmos inspirados en el comportamiento de colonias de agentes/individuos. Dentro de este grupo de algoritmos, se encuentra el que es objeto de estudio e implementación en este documento: ***Ant Colony Optimization***, ACO -Algoritmos basados en Colonia de hormigas-.

Las hormigas son insectos "sociales", se agrupan en colonias formando comunidades. El comportamiento de una hormiga individual apenas tiene mayor interés, mientras el del conjunto de la colonia sí. Ellas coordinan sus actividades vía *stimergy*. La *stimergy* es una forma de comunicación indirecta mediada por modificaciones del entorno. Por ejemplo, una hormiga deposita una cantidad de una sustancia química en el suelo, la cual incrementa la probabilidad de que otras hormigas sigan el mismo camino.

Biólogos han demostrado que el principal comportamiento observado en estos insectos sociales puede explicarse a través de modelos bastantes simples en los que sólo la comunicación 'estimérgica' está presente. Principalmente, este interés radica en el sistema organizativo que llevan a cabo estos insectos. Ésta es la idea principal sobre la que se construye un algoritmo basado en colonia de hormigas.

3.2. EL COMPORTAMIENTO DE UNA HORMIGA NATURAL.

Las colonias de hormigas, y de forma más general, las comunidades de insectos sociales, son sistemas distribuidos que, a pesar de la simplicidad de los agentes que las componen, presentan una organización social altamente estructurada. Como resultado de esta organización, las colonias de hormigas pueden desempeñar complejas tareas que en algunos casos distan mucho de las capacidades individuales que presenta una hormiga sola.

La facultad perceptiva visual de muchas especies de hormigas está desarrollada rudimentariamente, incluso existen especies de hormigas que son completamente ciegas. De hecho, una investigación muy importante sobre el comportamiento de las hormigas fue que la mayoría de las comunicaciones entre las hormigas individualmente, o entre las hormigas y el entorno, está basada en el uso de sustancias químicas producidas por las mismas (comunicación *stimergy*). Estas sustancias químicas se denominan feromonas. Esto es diferente a lo que ocurre con los seres humanos y otro tipo de especies superiores, las cuales poseen sentidos más importantes y desarrollados como son el oído y la vista. Por ello, muy importante para la vida social de algunas especies de hormigas es el denominado rastro de feromona. El rastro de feromona es un tipo específico de feromona que algunas especies de hormigas, como *Lasius niger, la argentina* (*Iridomyrex humilis*) (Deneubourg et al., 1990), utilizan para realizar rutas o caminos sobre el suelo, por ejemplo, caminos desde las fuentes de alimento (*Food*) hasta el nido (*Nest*). Gracias al rastro de la feromona, los agentes pueden seguir el camino descubierto por otras hormigas. Esto explica la manera en la que una hormiga está influenciada por la sustancia química que dejan tras sí el resto de las hormigas: ***rastro de feromona***. Así es, como las hormigas se comunican entre sí. Esta idea será la fuente de inspiración del ACO.

3.2.1. Experimento del doble puente.

En el apartado anterior, se explicó cómo el comportamiento de varias especies de hormigas (*Iridomyrmex humilis, Linepithema humile y Lasius niger*) está basado en una comunicación indirecta mediada por las feromonas. Mientras una hormiga camina desde su nido-colonia hasta su fuente de comida o viceversa, ésta deposita feromonas en el suelo, dejando a través del camino que realiza un rastro de feromona. Las

hormigas pueden oler la feromona, es decir, detectarla, y esto hace que tiendan a elegir un camino u otro de los posibles a realizar.

Probabilísticamente, una hormiga elegirá aquella ruta o camino que contenga mayor concentración de feromonas, es decir, un mayor rastro de feromonas.

El rastro de feromona, que tanto las hormigas dejan tras sí en su camino como el que el resto de hormigas sigue, ha sido estudiado mediante varios experimentos controlados y llevados a cabo por investigadores. Particularmente, destaca el ***experimento del doble puente*** (Deneubourg et al., 1990) (ver figura 3.1).

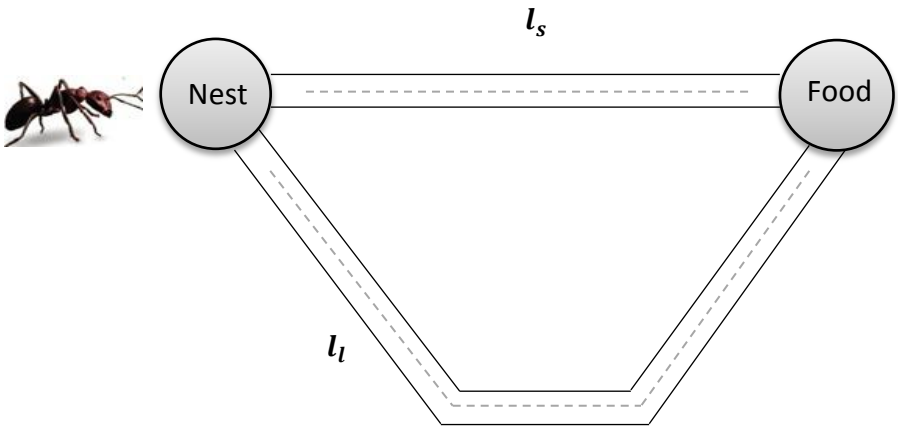


Figura 3.1. *Experimento del doble puente (1).*

Dicho experimento consta de lo siguiente: se realizó con una hormiga de las especie *Argentine I. humilis,* la cual tenía que recorrer el camino desde la colonia (*Nest*) hasta la fuente de comida (*Food*). Dicha hormigas podrían escoger dos caminos para llegar a la fuente. Se llevaron a cabo diferentes pruebas variando las longitudes de ambos caminos (denominadas ramas). Para ello, variaba el ratio entre la rama de longitud más larga y la de la rama más corta (ver figura 3.2):

$$ratio = \frac{l_l}{l_s}$$

$l_s = longitud\ del\ camino\ más\ largo$

$l_l = longitud\ del\ camino\ más\ corto$

Figura 3.2. *Ratio del experimento del doble puente*

- *1er Experimento (figura 3.3):*

Los dos puentes tienen la misma longitud: $\boldsymbol{ratio = 1 \rightarrow l_s = l_l}$

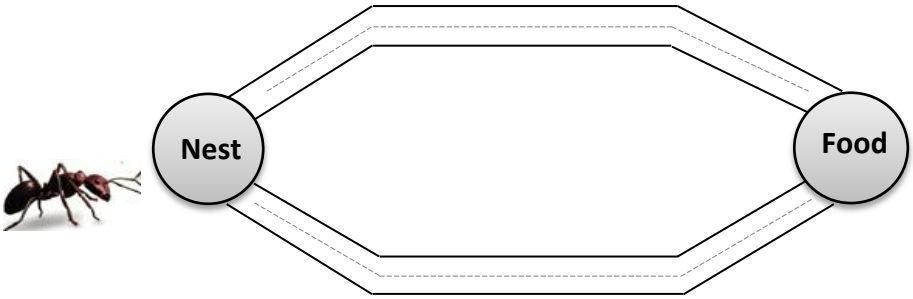


Figura 3.3. *Experimento del doble puente (2).*

Al comienzo del experimento, las hormigas se movían libremente entre el nido y la comida durante un tiempo determinado. Se calculó el porcentaje de hormigas que elegían un puente u otro. El resultado fue el siguiente: todas las hormigas solían usar el mismo arco aunque, en la fase inicial la elección de una rama u otra era aleatoria. Esto sucede porque cuando el ensayo comienza no hay feromona en ninguno de los dos caminos, por ello, las hormigas no tienen ninguna preferencia y seleccionan con la misma probabilidad cualquiera de los dos caminos.

- *2º Experimento (ver figura 3.4).*

Un puente tiene el doble de longitud que el otro: $\boldsymbol{Ratio = 2 \rightarrow l_l = 2 \cdot l_s}$

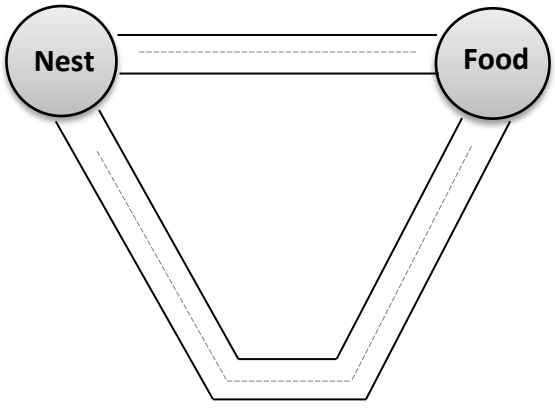


Figura 3.4. *Experimento del doble puente (3).*

En este caso, en la mayoría de los ensayos realizados, se demostró que, finalmente, todas las hormigas eligen el camino más corto, es decir, el puente cuya longitud es ½ de la rama de mayor longitud. De la misma forma que se realizó en el primer experimento, las hormigas dejan el Nest para explorar el entorno y llegan al punto decisivo donde tienen que elegir uno de los dos caminos. Puesto que, inicialmente, los dos caminos parecen idénticos para las hormigas, éstas seleccionan uno de ellos aleatoriamente. Además, puede esperarse, de media, que la mitad de las hormigas eligen el camino corto y la otra mitad el largo, aunque hay que tener en cuenta las oscilaciones estocásticas pueden favorecer un camino sobre otro de manera ocasional. Esto puede verse en la figura 3.5 que aparece a continuación:

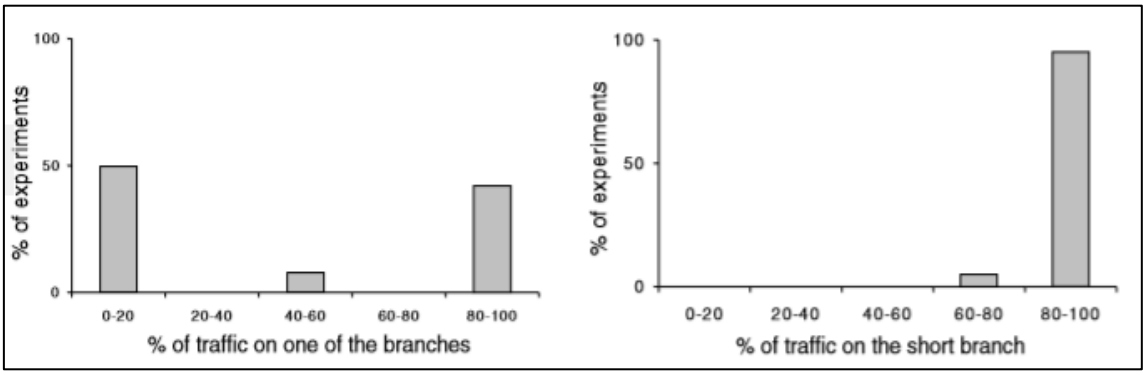


Figura 3.5. *Resultados del experimento del doble puente (Dorigo y Stützle, 2004)*

En la figura 3.5 se puede ver los resultados obtenidos para el experimento del doble puente utilizando la hormiga *Iridomyrmex humili*. En el caso de los dos puentes de igual longitud (1), gráfico de la izquierda, es decir, para r=1, las hormigas usan una rama u otra de manera indistinta, en el mismo número de intentos para cada rama. Es decir, no muestra ninguna tendencia por alguna de las ramas. Mientras que para el caso (2), gráfico de la derecha donde r=2 (doble longitud de rama), la gran mayoría de las hormigas eligen el camino más corto, es decir, la rama de menor longitud en todos los ensayos, como ya se explicó anteriormente.

3.2.2. Un modelo estocástico.

Deneubourg, en 1990, (Dorigo y Stützle, 2004) propuso un modelo estocástico simple que describía, adecuadamente, las dinámicas de las colonias de hormigas como se observaron en el experimento del doble puente. En este modelo, existen ψ hormigas por segundo que cruzan a través del puente en cada dirección con una velocidad constante, siendo ésta *v* cm/s, depositando una unidad de feromona en el puente o rama. Dada la longitud l_s y l_l (en cm) de la rama más corta y la más larga,

respectivamente, si una hormiga elige la rama más corta, ésta tardará en recorrerá un tiempo t_s que se calcula como el cociente entre la longitud de la rama l_s y la velocidad a la que la atraviesa, *v*, que como se ha dicho anteriormente esta es constante, v. El tiempo vendrá dado en segundos. Para la hormiga que elija la rama larga, ocurrirá lo mismo, en este caso tenemos que utilizar la longitud de la rama larga, el tiempo que tardará en recorrerla será t_l.

La probabilidad $p_{ia}(t)$ con la que una hormiga llegando al punto de decisión $i \in \{1,2\}$ selecciona la rama $a \in \{s, l\}$ (esto quiere decir que *a* solo puede corresponderse o con el camino corto *s,* o el largo *l,* como vimos en el experimento del doble puente, (figura 3.1), en el instante t está dispuesto para ser función de la cantidad total de feromona denotada como $\varphi_{ia}(t)$ en la rama, la cual es proporcional al número de hormigas que usan esa rama hasta el tiempo t. Por ejemplo, la probabilidad $p_{is}(t)$ de escoger el camino corto, es decir, la rama de menor longitud viene dada por la expresión (3.1):

$$p_{is}(t) = \frac{(t_s+\varphi_{is}(t))^{\alpha}}{(t_s+\varphi_{is}(t))^{\alpha}+(t_l+\varphi_{il}(t))^{\alpha}} \quad (3.1)$$

Donde la ecuación (3.1), así como el valor de $\alpha = 2$, fue deducido de forma empírica y experimental por Deneubourg en 1990. Para las probabilidades halladas, se tiene que cumplir, en este caso, que $p_{is}(t) + p_{il}(t) = 1$.

Este modelo asume que la cantidad de feromona en la rama es proporcional al número de hormigas que hayan usado, anteriormente, esa rama. Dicho de otra forma, este modelo no considera la evaporación de la feromona.

Retomando de nuevo el modelo, las ecuaciones diferenciales que lo definen son las siguientes (3.2) y (3.3):

$$\frac{d\varphi_{iS}}{dt} = \psi p_{js}(t - t_s) + \psi p_{is}(t), \qquad (i = 1, j = 2; i = 2, j = 1), \quad (3.2)$$

$$\frac{d\varphi_{iL}}{dt} = \psi p_{jL}(t - t_s) + \psi p_{iL}(t), \qquad (i = 1, j = 2; i = 2, j = 1), \quad (3.3)$$

La ecuación (3.2) puede interpretarse de la siguiente manera: las variaciones instantáneas, en un tiempo t, de la feromona en la rama s y en el punto de decisión i viene dada por el flujo de hormigas, ψ, el cual, se ha asumido que es constante, multiplicado por la probabilidad de elegir la rama más corta en el punto de decisión j,

en un tiempo $t - t_s$. A esto, hay que añadirle el flujo de hormigas multiplicado por la probabilidad de escoger el camino más corto, pero en este caso, desde el punto de decisión i y en un tiempo t. La constante t_s representa el tiempo necesario para que las hormigas atraviesen la rama de longitud más corta. De forma análoga, la ecuación (3.3) representa lo mismo pero para la rama de mayor longitud, l_l.

3.3. LA HORMIGA ARTIFICIAL.

Basándonos en el experimento del doble puente y en los resultados que éste nos ofrece, se evalúa el comportamiento organizativo de las hormigas. Inspirándonos en este fundamento, es posible diseñar hormigas artificiales, que moviéndose en un grafo, encuentren el camino más corto entre dos nodos, los cuales corresponden con el nido y la fuente de comida, como se ha venido refiriendo todo el capítulo.

Una hormiga artificial es un agente simple, desde el punto de vista computacional, que lo que intenta es construir soluciones (una solución global) a un problema explorando el grafo por el que se mueve y siguiendo el rastro de feromona que se encuentra en dicho camino.

3.3.1. Características de una hormiga artificial.

- Busca soluciones de mínimo coste para el problema a resolver de buena calidad, válidas.

- La hormiga artificial dispone de una memoria en la que almacena el camino recorrido, de esta forma evalúa las soluciones construidas y si éstas no son válidas aún, las reconstruye realizando un nuevo camino.

- El movimiento de la hormiga a través del grafo se realiza aplicando una regla de transición que es función del rastro de feromona que existe en cada uno de los arcos que recorre, del valor de la heurística aplicada y de las restricciones del problema.

- El procedimiento de construcción del camino que realiza la hormiga se finaliza o termina siempre y cuando se satisfaga algún criterio de parada, como puede ser el número de iteraciones, ejecuciones o alcanzar un valor objetivo.

- Cuando la hormiga haya construido la solución, el camino recorrido por la misma puede ser "reconstruido" y actualizar los rastros de feromona τ_{ij} de los arcos que hayan sido visitados por la hormiga. Éste es el único mecanismo que las hormigas utilizan para comunicarse entre sí.

3.3.2. Las hormigas artificiales y rutas de mínimo coste.

Mediante las hormigas artificiales, se quiere definir algoritmos que pueden usarse para resolver problemas de minimización de costes con grafos mucho más complejos que los que representan el experimento del doble puente (ver figura 3.1).

Consideramos (Islam et al., 2006) un grafo G=(N,A), donde N es el conjunto de n=|N| nodos y A es el conjuntos de arcos no direccionados conectados entre sí. Los dos puntos entre los cuales se establece la ruta o el camino de mínimo coste se denominan *Nest* y *Food*.

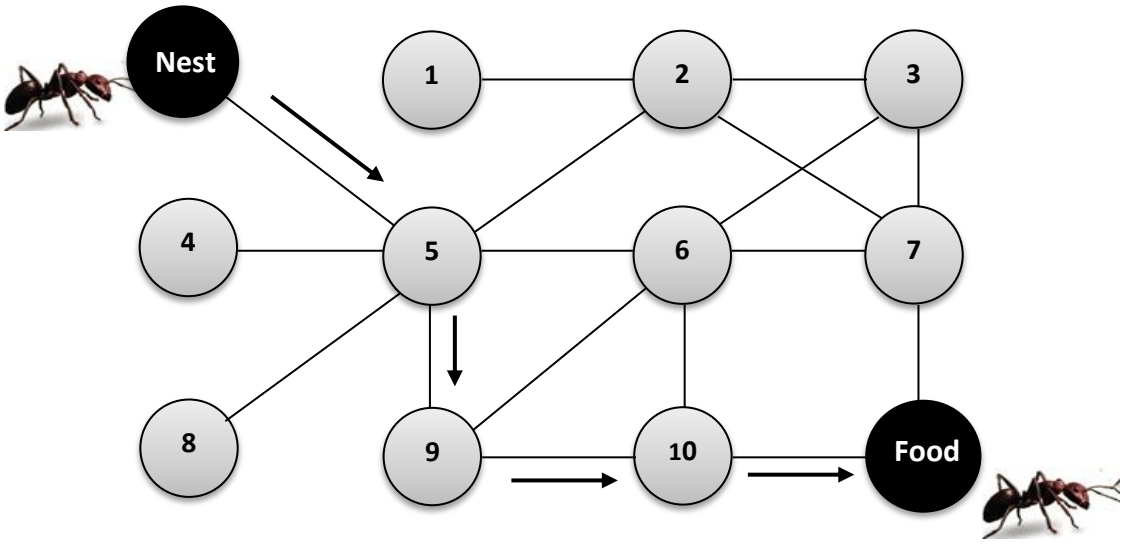


Figura 3.6. *Grafo recorrido por hormigas (Dorigo y Stützle, 2004)*

La hormiga artificial recorrerá los arcos con el fin de llegar al nodo destino (Food). Para moverse a través de los arcos que conectan los nodos del grafo, ésta lo hará de acuerdo con una regla de decisión basada en probabilidades que se explicará a continuación.

Desafortunadamente, si intentamos resolver el problema hallando el camino de mínimo coste en el grafo G usando hormigas artificiales cuyo comportamiento deriva del comportamiento de las hormigas naturales (descrito en el apartado anterior), surge el siguiente problema: las hormigas, mientras construyen una solución, pueden generar bucles. Como consecuencia del mecanismo de actualización del rastro de feromona, los bucles tienden a llegar a ser más y más atractivos y esto hace que las hormigas puedan quedar atrapadas en ellos.

Éste es el fundamento del que se basa el algoritmo colonia de hormigas para resolver problemas que puedan plantearse mediante un grafo como puede ser el TSP (en este documento se verá en el capítulo 4 implantación para el ATSP).

3.4. SIMILITUDES Y DIFERENCIAS ENTRE LAS HORMIGAS NATURALES Y ARTIFICIALES.

Como es lógico las colonias de hormigas naturales y artificiales tienen que compartir una serie de características puesto que las segundas se basan y fundamentan en el comportamiento de las primeras. Estas diferencias y similitudes aparecen enunciadas a continuación:

- Una de las características principales es la capacidad que presenta una colonia de hormigas tanto artificiales como naturales de colaborar e interaccionar entre sí para construir soluciones.

- Ambas modifican su entorno gracias a la comunicación estimérgica (en inglés, *stimergy*) basada en la feromona. Para las hormigas artificiales, los rastros de feromonas son valores numéricos que se encuentran en los arcos que son recorridos por ellas.

- Una de las similitudes más importantes es la finalidad de la hormiga artificial y natural en su construcción de camino: ambas, buscan el camino más corto, lo cual se lleva a cabo una construcción iterativa del coste mínimo desde un origen a un destino. En el caso de la hormiga natural, desde el hormiguero hasta la comida.

- Entre las diferencias, la principal radica en la memoria de la que disponen las hormigas artificiales para almacenar en ella el camino realizado y por tanto, los nodos visitados en el mismo; esto se explicó en el apartado 3.3.

3.5. LA METAHEURÍSTICA ACO.

Ant Colony Optimization es una metaheurística en la que una colonia de hormigas artificiales coopera en encontrar soluciones buenas para problemas complejos de optimización. La cooperación es un componente clave en el diseño de los algoritmos ACO. Éstos pueden emplearse para resolver tanto problemas estáticos como dinámicos de optimización combinatoria. En el capítulo 2, apartado 2.2.3.2., se hizo una breve introducción, en lo que consistía la metaheurística ACO. A continuación, se va a profundizar en el tema, explicando los pasos de los que consta.

3.5.1. Pseudocódigo.

En la figura 3.7, aparece representado el pseudocódigo para el algoritmo colonia de hormigas que ya se comentó en líneas generales en el capítulo 2. En este apartado (Saka et al., 2013), se explicará paso a paso cada uno de los bloques que componen dicho pseudocódigo.

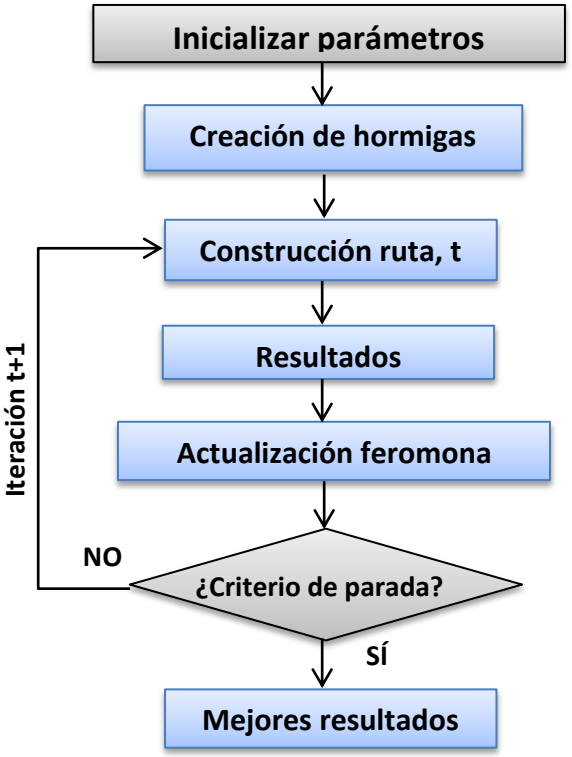


Figura 3.7. *Pseudocódigo de un ACO*

- ***Inicializar parámetros y creación de las hormigas***: seleccionar el número de hormigas, *m*, que se necesitarán para el problema de optimización que estamos resolviendo. Una vez que tenemos esto, será necesario definir un rastro de feromona inicial, τ_0 el cual será una cantidad constante e idéntica para todo y cada uno de los arcos que conectan el grafo. Este rastro inicial de feromona se calcula como el cociente entre el número de hormigas, m, prefijado y la longitud del camino más corto del grafo aplicando la heurística de *Greedy*, C^{nn}, (esto es para el caso del TSP, para otro tipo de problemas se consideran y calcula de otra forma el rastro de feromona inicial). Fundamental será la matriz de datos de coste los arcos, de tamaño *n* que determinará el tamaño del problema para las que se calculará la inversa, esto es $\eta_{ij} = \frac{1}{c_{ij}}$, donde c_{ij} es el costo del nodo i hasta el j. Para poder aplicar la regla de decisión para la construcción de la secuencia, es necesario definir los parámetros α y β, en donde el primero, representa la influencia relativa del rastro de feromona y el

segundo, la información e importancia de la heurística empleada. Muy importante en la actualización de la feromona resulta ser el coeficiente de evaporación de la feromona, ρ, que lo explicaremos a continuación. Por último, hay que definir el criterio de parada del algoritmo este puede ser el número de iteraciones que se haya prefijado o simplemente encontrar un resultado objetivo que hayamos prefijado. Normalmente, suele ser el número de iteraciones. En definitiva, todos estos parámetros aparecen definidos en la tabla 3.1 :

Parámetro	Descripción
$\mathbf{m}$	Número de hormigas
$\mathbf{n}$	Número de nodos
$\boldsymbol{c_{ij}}$	Costo que existe desde el nodo i hasta el j
$\boldsymbol{\tau_0}$	Rastro de feromona inicial
$\boldsymbol{\eta_{ij}}$	Inversa de c_{ij}
$\boldsymbol{\alpha}$	Influencia relativa del rastro de feromona
$\boldsymbol{\beta}$	Importancia de la información heurística
$\boldsymbol{\rho}$	Coeficiente de evaporación de la feromona

Tabla 3.1. *Definición de parámetros.*

- ***Construcción de la ruta.*** Para la construcción del camino, es necesario aplicar la regla de decisión basada en probabilidades cuyo mentor fue Dorigo en 1991. Mediante esta regla, se podrá hallar la probabilidad de elegir un nodo u otro perteneciente al grafo, G (ver figura 3.8):

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}} \qquad si \;\; j \in N_i^k$$

Figura 3.8. *Regla de decisión basada en probabilidades (Dorigo, 1991).*

Donde todos los parámetros aparecen definidos en la tabla 3.1, a excepción de N_i^k , que representa el vecindario o conjunto de nodos factibles para visitar por la hormiga k-ésima cuando ésta se encuentra en el nodo i, es decir, el conjunto de nodos que la hormiga k-ésima todavía no ha recorrido o visitado cuando se trata del problema del agente viajante.

- ***Resultados.*** Los resultados de la búsqueda local corresponderán con el coste mínimo hallado por cada hormiga que recorra el gráfico, es decir, C^k, así como la secuencia de los nodos que haya visitado, esto es, el camino realizado por cada hormiga, T^k.

- ***Actualización de la feromona***. Una vez que todas las hormigas, m, finalizan su recorrido en la iteración t, se aplica la regla de actualización del rastro de la feromona. Asociado a ésta, es necesario definir un nuevo parámetro, el cual, se mantiene constante en la ejecución del algoritmo: ρ, coeficiente de evaporación del rastro de feromona. ρ es un ratio cuyo valor está comprendido entre 0 y 1, esto es $\rho \in (0,1]$. Por lo tanto, el nuevo rastro de feromona hallado en el arco (i,j) será: $\tau_{ij}^{t+1} \leftarrow (1-\rho)\tau_{ij}^{t}$. A ésto, se le añade el rastro de feromona que las hormigas k-ésimas hayan dejado tras sí en todos los arcos (i,j) por los que hayan realizado su recorrido. En función del problema que estemos resolviendo, la actualización de feromona se realizará de una forma u otra. En el capítulo 4, cuando se aplique al problema ATSP se explicará detenidamente todos y cada uno de los parámetros y términos que intervienen en la actualización del rastro. Sin embargo, para la evaporación de la feromona, $\tau_{ij}^{t+1} \leftarrow (1-\rho)\tau_{ij}^{t}$, siempre será la misma y se realizará igual independientemente del problema que estemos resolviendo y del método que estemos aplicando para dicha resolución. Dicho esto, el rastro de feromona puede incrementar, cuando las hormigas depositan feromona en los arcos o conexiones por los que se mueven, o disminuir debido a la evaporación de la feromona. Desde un punto de vista práctico, el depósito de nuevo rastro incrementa la probabilidad de que estos componentes que fueron visitados o recorridos por un gran número de hormigas, serán recorridos de nuevo por futuras hormigas ya que producen buenos resultados. Por otro lado, la evaporación de la feromona impide, de cierta manera, el estancamiento en una región del espacio de soluciones y favorece la exploración de nuevas áreas del espacio de búsqueda. Si no se evaporara la feromona, esto daría lugar a una convergencia demasiado rápida del algoritmo hacia una región sub-óptima.

- ***Criterio de parada***. Normalmente, suele ser el número de iteraciones que el usuario haya prefijado. Una vez alcanzado este número, se presentan los resultados finales del algoritmo (***mejores resultados***), los cuales corresponderán con la ruta de mínimo coste realizada por una hormiga k-ésima así como el coste mínimo asociado a esa ruta: T^{best} y C^{best}, respectivamente.

En definitiva, éstos son los pasos que representan la aplicación de un ACO para cualquier problema de optimización combinatoria que pueda ser representado mediante un grafo G, ya sea dirigido o no. En el capítulo 4, se adecuaran para la resolución del ATSP, por lo tanto, la explicación del pseudocódigo será distinta ya se

verá sobretodo en la regla de construcción del tour (ruta) y en la actualización de la feromona.

3.6. VARIANTES DEL ACO APLICADAS PARA EL TSP.

Existen numerosas variantes de la metaheurística ACO que en los últimos años han ido surgiendo. La principal característica que diferencia un método de otro reside en la actualización de la feromona: las variantes existentes de los ACO aplican la misma regla de construcción del camino o tour pero, sin embargo, en función del problema que se esté resolviendo la actualización de la feromona se realizará de una forma u otra. Esto podrá comprobarse en el capítulo 4 cuando se implemente el ACO para la resolución del ATSP y las variantes que se emplearán para ello. A continuación, sólo se va a realizar una breve introducción de algunas de las variantes que existen.

Tres versiones diferentes fueron propuestas por Marco Dorigo en 1991 (Colorni et al., 1991). Éstas se denominaron *ant-density*, *ant quantity,* y *ant-cycle*. Las dos primeras versiones, las hormigas actualizaban la feromona inmediatamente después de moverse desde un nodo i hasta un nodo j, mientras que en la versión *ant-cycle* la actualización de la feromona depositada por cada hormiga era función de la calidad del camino y se realizaba al final de recorrer el mismo. Hoy en día, cuando se emplea el AS (en inglés, *Ant System*) nos referimos a la versión *ant-cycle* puesto que las otras dos fueron abandonadas, apenas se utilizaron para la representación del problema en general.

En 1992, Dorigo introdujo lo que se conoce como ***Elistist Ant System*** (EAS), la versión elitista del modelo AS. La principal diferencia de la anterior es que en este caso cuando se realiza la actualización de la feromona se otorga mayor peso a la hormiga cuyo recorrido y coste realizado es el mínimo: éstos son C^{best} y T^{best}. Para ello, se define un parámetro nuevo, e.

Años más tarde y tras numerosas investigaciones y publicaciones realizadas acerca de los algoritmos de colonia de hormigas, Bullnheimer en 1997, publicó la versión ***Rank-Based Ant System*** (ASrank), en el que cada hormiga deposita una cantidad de feromona que decrece con el rango definido. La diferencia con la versión EAS radica también en la actualización de la feromona. Se verá en las experimentaciones realizadas en el capítulo 4, que el ASrank presenta mejores resultados que el AS al igual que la versión elitista EAS. Sin embargo, la diferencia entre EAS y ASrank no es muy significativa.

Estas son las tres versiones que más repercusión han tenido y las que se implementarán, en el capítulo 4, para resolver el problema del viajante asimétrico. No

obstante, cabe destacar que existen muchas más como son el ***MAX-MIN Ant System*** basado en unos límites de rastro de feromona que se definen al comienzo ($\tau_{mín}, \tau_{máx}$); o el ***Ant Colony System*** (ACO) (Dorigo y Gambardella, 1997) el cual difiere del AS en cuatro puntos principales: la transición de estados, la evaporación de la feromona y la cantidad depositada tiene lugar sólo en los arcos pertenecientes a la mejor ruta o camino realizado; y por último, cada vez que una hormiga utilice un arco (i,j) para moverse del nodo i hasta el j, elimina cierta cantidad de feromona en el arco lo que favorece la exploración y la incrementa de otros caminos o rutas alternativas.

Como ya he comentado al principio de este apartado, más adelante, en el capítulo 4, se explicarán en profundidad todas estas variantes aplicadas al ATSP.

3.7. APLICACIONES DEL ACO.

Los algoritmos basados en colonia de hormigas, ACO, toman como referencia el comportamiento de este tipo de insectos para la resolución de problemas de optimización combinatoria como ya se ha descrito en el apartado 3.1 y sucesivos.

Desde un punto de vista general, los algoritmos ACO se pueden aplicar a un amplio conjunto de problemas de optimización. Especialmente, este tipo de algoritmos son muy utilizados en problemas de rutas debido a su alto grado de adaptación al problema. Su empleo para la resolución se debe a que son muy complejos de resolver óptimamente mediante métodos exactos (muy costosos en cuanto a términos de tiempos de ejecución de programa). A veces, aunque el algoritmo resulte de gran ayuda y, en muchas ocasiones, nos aporta soluciones muy próximas a la óptima, incluso la óptima, en otros casos no lo hace. Es decir, no siempre se encuentra la solución óptima empleando estas técnicas. A continuación, se presenta en la tabla 3.2 los principales problemas de optimización combinatoria (tipología y nombre) que pueden resolverse mediante la implementación de un ACO:

Tipología del problema	Nombre	Referencias
Rutas	Problema del viajante (TSP)	Dorigo, Maniezzo, y Colorni (1991a,b ,1996)
		Dorigo (1992)
		Gambardella y Dorigo (1995)
		Dorigo y gambardella (1997a,b)
		Stützle y Hoos (1997, 2000)
		Bullnheimer, Hartl, y Strauss (1999a, b)
		Cordón, de Viena, Herrera, y Morena (2000)
	Enrutamiento de vehículos (VRP)	Bullnheimer, Hartl, y Strauss (1999a, b)
		Gambardella, Taillard, y Agazzi (1999)
		Reimann, Stummer, y Doerner (2002)
Asignación (Assignment)	Sequencial ordering (SOP)	Gambardella y Dorigo (1997,2000)
	Quadratic assignment (QAP)	Maniezzo, Colorni, y Dorigo (1994)
		Stützle (1997b)
		Maniezzo y colorni (1999)
		Maniezzo (1999)
		Stützle y Hoos (2000)
	Graph coloring	Costa y Hertz (1997)
	Generalized assignment	Lourenço y Serra (1998, 2002)
	Frequency assignment	Maniezzo y Carbonaro (2000)
Programación (Scheduling)	Job shop	Colorni, dorigo, Maniezzo, y trubian (1994)
	Open shop	Pfahringer (1996)
	Flow shop	Stützle (1998a)
	Total tardiness	Bauer, Bullnheimer, Hartl, y Strauss (2000)
	Total weighted tardiness	Den Besten, Stützle, y Dorigo (2000)
Subconjuntos (Subset)	Project scheduling	Merkle, Middendorf, y Schmeck (2000ª, 2002)
	Group shop	Blum (2002a, 2003)
	Multiple knapsack	Leguizamón y Michalewicz (1999)
	Max independent set	Leguizamón y Michalewicz (2000)
	Set covering	Leguizamón y Michalewicz (2000)

Tabla 3.2. *Aplicaciones en la actualidad del ACO. (Dorigo y Stützle, 2004)*

Capítulo 4.

Implementación y Experimentación.

En este capítulo, se implementará el algoritmo ACO para la resolución del problema del viajante asimétrico (ATS). Para lo cual, será necesario explicar el pseudocódigo que se ha realizado y, por lo tanto, se aplica al problema ATSP. Antes de adentrarnos con la implementación, propiamente dicha, en el ordenador, se describe el fundamento teórico de los ACO para el problema del viajante, así como los parámetros empleados y los valores de los mismos. A continuación, se definirá el lenguaje empleado para la programación del algoritmo que en este caso es C++, un lenguaje muy potente de alto nivel que permitirá programar el algoritmo y optimizar, en la mediad de lo posible, el tiempo de ejecución del mismo. Después, se explicará cada uno de los ficheros que han sido creados. No obstante, para poder entender de manera sencilla la implementación, se resolverá un problema de pequeño tamaño y se describirán, detalladamente, cada uno de los pasos a realizar. Finalmente, y como ya se apuntó en el capítulo 3, se explicarán cada una de las versiones del AS que pueden implementarse como son la versión elitista, EAS, y la versión basada en rangos, ASrank.

4.1. IMPLEMENTACIÓN DE LOS ALGORITMOS ACO PARA EL ATSP.

Como ya se introdujo en el capítulo 3, la aplicación de los algoritmos ACO para los problemas de enrutamiento, como es el TSP, es calcular la ruta de mínima distancia. Ahora se va aplicar y desarrollar directamente al ATSP.

A continuación, se describe un ejemplo para comprender mejor la lógica del algoritmo para la resolución del ATSP. Posteriormente, se explicará cada uno de los algoritmos ACO que se han realizado en el proyecto .cpp en C++ así como el pseudocódigo del programa y las relaciones que existen entre los distintos ficheros creados.

4.1.1. Ejemplo de la implementación del ACO. Variantes del ACO.

El problema que se va a resolver en este apartado constará de cuatro nodos, es decir, la construcción del grafo será de acuerdo con G=(N,A), donde N=4 (ver figura 4.1) y A el número de arcos va a ser: n(n-1)=4x3=12. Las restricciones que se impone a este problema son las siguientes:

- Visitar una única vez cada una de las ciudades.
- La ciudad de fin tiene que ser la misma que la de inicio (circuito *hamiltoniano*). Se considera la ciudad 1 como la inicial.
- La distancia de la ciudad i a la j es distinta que de la j a la i → $d_{ij} \neq d_{ji}$ (grafo dirigido).
- El criterio de parada del algoritmo será cuando se realicen 3 iteraciones, es decir cuando la variable iteración sea igual a 4.

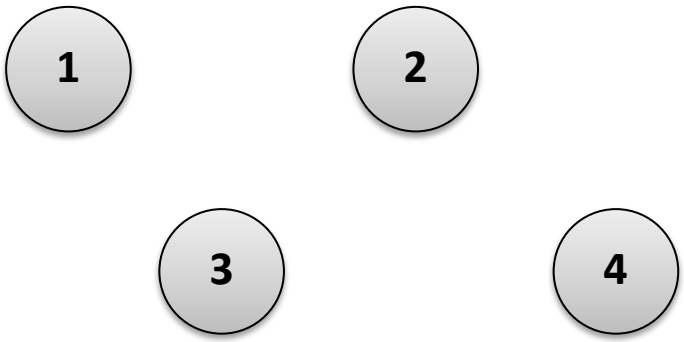


Figura 4.1. *Ejemplo de ATSP. Grafo, G=(4,12)(1).*

Una vez impuestas las restricciones, ahora se Inicializan los parámetros y se presentan los datos necesarios para su resolución.

La matriz de distancias es la siguiente (ver tabla 4.1):

Distancias, d_{ij}	**1**	**2**	**3**	**4**
1	0	2	3	1
2	1	0	2	1
3	2	1	0	4
4	2	2	1	0

Tabla 4.1. *Distancias entre las ciudades.*

Los parámetros de los que se van a necesitar son los siguientes (ver tabla 4.2):

Parámetro	**Descripción**
m	Número de hormigas
n	Tamaño del problema; Número de nodos
$\boldsymbol{\tau_0}$	Rastro de feromona inicial
$\boldsymbol{\eta_{ij}}$	Heurística utilizada
$\boldsymbol{\alpha}$	Influencia relativa del rastro de feromona
$\boldsymbol{\beta}$	Importancia de la información heurística
$\boldsymbol{\rho}$	Coeficiente de evaporación de la feromona

Tabla 4.2. *Parámetros del algoritmo ACO.*

El rastro de feromona asociado a cada arco, τ_{ij}, que en este caso iteración=1, el rastro de feromona es igual a la inicial. Dependiendo del tipo de algoritmo ACO que se aplique, se calcula de una forma u otra. Para ello, ver tabla 4.4.

La información heurística para cada arco, η_{ij}, se calcula de la siguiente forma: $\eta_{ij}={}^{1}/_{d_{ij}}$. Por tanto, hay que calcular la inversa de las distancias. En el caso de ser $d_{ij} = 0$, para un arco (i,j) como viajar desde la ciudad i hasta la i es imposible, η_{ij} se fija un número muy grande, implica que el costo sería muy elevado y por tanto, inviable. La matriz de la información heurística aparece en la tabla 4.3:

$\boldsymbol{\eta_{ij}}$	**1**	**2**	**3**	**4**
1	99999	0.5	0.33	1
2	1	99999	0.5	1
3	0.5	1	99999	4
4	0.5	0.5	1	99999

Tabla 4.3. *Información heurística de cada arco.*

De acuerdo con los estudios experimentales llevados a cabo por Marco Dorigo y Stützle (Dorigo y Stützle, 2004) para el problema TSP general, han identificado una serio de rangos y valores para los parámetros que interviene en la implementación del ACO, para los que se obtiene una buena representación del problema. La tabla 4.4 recoge lo descrito anteriormente:

Algoritmo ACO	α	β	ρ	m	τ_0
AS (Ant System)	1	2-5	0.5	n	m/C^{nn}
EAS (Elitist Ant System)	1	2-5	0.5	n	$(e+m)/\rho C^{nn}$
ASrank (Rank-based Ant System)	1	2-5	0.1	n	$0.5w(w-1)/\rho C^{nn}$

Tabla 4.4. *Valores óptimos de cada parámetro (Dorigo y Stützle, 2004).*

El ejemplo va a resolver mediante el método AS. Para el resto de variantes que se han implementado en el programa informático, se explicarán a continuación.

4.1.1.1. Ant System (AS).

Como ya se describió brevemente en el capítulo 3, las variantes más características de los algoritmos ACO, hoy en día, cuando nos referimos a la variante AS se trata de lo que se conocía, anteriormente, como *ant-cycle*, esto quiere decir, que la actualización del rastro de feromona será realizada únicamente después de que todas y cada una de las hormigas hayan construido sus rutas y depositado una cantidad de feromona.

Las dos principales fases de las que consta un ACO AS son: la construcción de la ruta y la actualización de la feromona. En el AS, una Buena heurística para inicializar los rastros de feromona, es establecer un valor de la misma en cada arco ligeramente superior a la cantidad esperada de feromona depositada por las hormigas en una iteración; una estimación aproximada de este valor podría establecerse mediante la siguiente expresión: $\forall\ (i,j),\ \tau_{ij}^{t=1} = \tau_0 = {}^{m}/_{C^{nn}}$ (ver tabla 4.4).

La razón por la que se elige esta expresión es debida a que si los rastros de feromona iniciales fuesen demasiado bajos, la búsqueda estaría sesgada por las primeras rutas generadas por las hormigas, las cuales, generalmente conducen hacia la exploración de zonas inferiores del espacio de búsqueda de soluciones.

- **Construcción de la ruta.**

En AS, m hormigas artificiales construyen una ruta para el problema ATSP. Como ya indicamos anteriormente, las hormigas parten de la ciudad que se ha elegido como la de inicio-fin (en el ejemplo que se ha considerado, ésta será la 1). En cada paso, para la construcción de la ruta, cada hormiga k, siendo k=1,2,…,m, aplica una regla probabilística de elección para decidir la siguiente ciudad a visitar. Esta regla ya fue mencionada en el capítulo 3. La probabilidad con la que la hormiga k-ésima, que actualmente se encuentra en la ciudad i, elige visitar la ciudad j viene dada por la expresión 4.1:

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}} \qquad si \;\; j \in N_i^k \qquad (4.1)$$

Para poder comprender mejor la aplicación de esta regla probabilística de elección, se aplicará al ejemplo que se ha ido desarrollando en este capítulo.

Iteración 1, hormiga k=1.

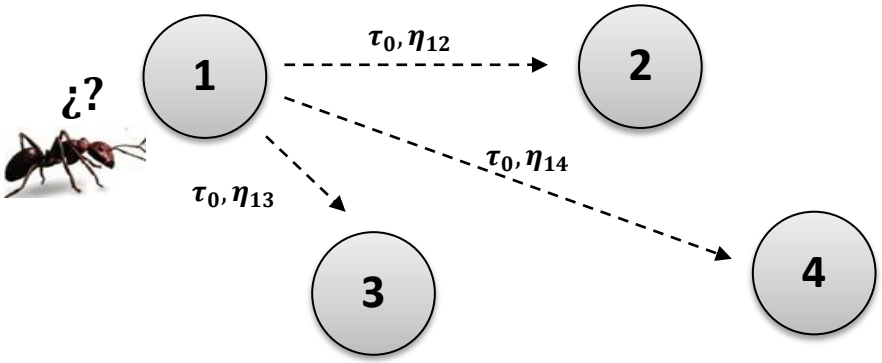


Figura 4.2. *Ejemplo de ATSP. Grafo, G=(4,12)(2).*

En la figura 4.2, una hormiga k=1 parte de la ciudad 1 quiere visitar la siguiente ciudad. Las posibles para visitar son las ciudades 2, 3 y 4. Para saber cuál se visitará, se aplica la regla probabilística de elección para cada ciudad posible a visitar y que todavía no ha sido visitada. Una vez hecho esto, se haya la probabilidad acumulada de las probabilidades calculas y se genera un número aleatorio entre (0,1). El rango al que pertenezca este número aleatorio será, por tanto, la ciudad próxima a visitar.

Datos.

Para calcular el rastro de feromona inicial aplicamos la siguiente expresión ya comentada anteriormente. C^{nn}, es la distancia total recorrida por una hormiga siguiente la heurística de Greedy, quiere decir que la siguiente ciudad a visitar, y que todavía no ha sido visitada, será aquella cuya distancia sea mínima a la ciudad actual en la que la hormiga k se encuentra. Por tanto, teniendo en cuenta las distancias entre las ciudades de nuestro ejemplo, en la figura 4.3 se calcula C^{nn} :

Paso 1

Distancias, d_{ij}	1	2	3	4
1	~~0~~	~~2~~	~~3~~	1
2	1	0	2	1
3	2	1	0	4
4	2	2	1	0

Paso 2

Distancias, d_{ij}	1	2	3	4
1	~~0~~	~~2~~	~~3~~	1
2	1	0	2	1
3	2	1	0	4
4	~~2~~	~~2~~	1	0

Paso 3

Distancias, d_{ij}	1	2	3	4
1	~~0~~	~~2~~	~~3~~	1
2	1	0	2	1
3	~~2~~	1	0	4
4	~~2~~	~~2~~	1	0

Paso 4

Distancias, d_{ij}	1	2	3	4
1	~~0~~	~~2~~	~~3~~	1
2	1	0	2	1
3	~~2~~	1	0	4
4	~~2~~	~~2~~	1	0

Figura 4.3. *Aplicación de la heurística Greedy.*

Por tanto, la ruta que se seguiría para visitar las ciudades siguiendo la heurística de Greedy sería 1-4-3-2-1, cuya distancia total es $C^{nn} = 1 + 1 + 1 + 1 = 4$

En definitiva, los datos a utilizar en el algoritmo AS son los que aparecen en la tabla 4.5:

Algoritmo ACO	α	β	ρ	m	τ_0
AS (Ant System)	1	3	0.5	4	m/C^{nn} =4/4=1

Tabla 4.5. *Valores de los parámetros para el AS.*

La importancia relativa de la heurística se extrae de la tabla 4.3.

Ahora, se aplica la expresión (4.1), es decir, la regla probabilística de elección para todas las posibles ciudades a visitar para construir la ruta (ver tabla 4.6).

Hay que tener en cuenta el conjunto de posibles ciudades a visitar, siendo éste: $N^1 = [2,3,4]$.

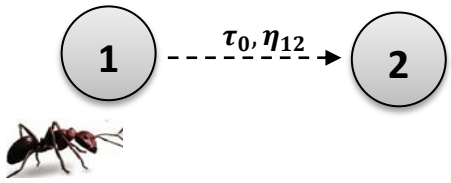
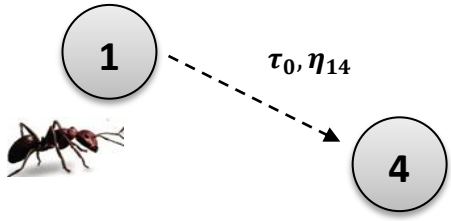
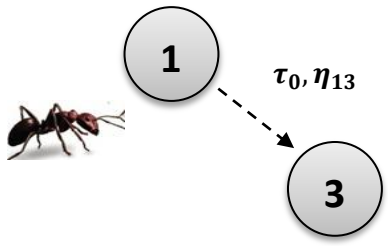
Posible ciudad a visitar	Regla probabilística de elección
	$P_{12}^1 = \frac{[\tau_0]^\alpha \cdot [\eta_{12}]^\beta}{[\tau_0]^\alpha \cdot [\eta_{12}]^\beta + [\tau_0]^\alpha \cdot [\eta_{13}]^\beta + [\tau_0]^\alpha \cdot [\eta_{14}]^\beta} =$ $= \frac{[1]^1 \cdot [0.5]^3}{[1]^1 \cdot [0.5]^3 + [1]^1 \cdot [0.3333]^3 + [1]^1 \cdot [1]^3} = 0.1076$
	$P_{14}^1 = \frac{[\tau_0]^\alpha \cdot [\eta_{14}]^\beta}{[\tau_0]^\alpha \cdot [\eta_{12}]^\beta + [\tau_0]^\alpha \cdot [\eta_{13}]^\beta + [\tau_0]^\alpha \cdot [\eta_{14}]^\beta} =$ $= \frac{[1]^1 \cdot [1]^3}{[1]^1 \cdot [0.5]^3 + [1]^1 \cdot [0.3333]^3 + [1]^1 \cdot [1]^3} = 0.8606$
	$P_{13}^1 = \frac{[\tau_0]^\alpha \cdot [\eta_{13}]^\beta}{[\tau_0]^\alpha \cdot [\eta_{12}]^\beta + [\tau_0]^\alpha \cdot [\eta_{13}]^\beta + [\tau_0]^\alpha \cdot [\eta_{14}]^\beta} =$ $= \frac{[1]^1 \cdot [0.3333]^3}{[1]^1 \cdot [0.5]^3 + [1]^1 \cdot [0.3333]^3 + [1]^1 \cdot [1]^3} = 0.0318$

Tabla 4.6. *Aplicación de la regla probabilística de elección (1).*

Se calcula la probabilidad acumulada y se genera el número aleatorio → **0,7957**

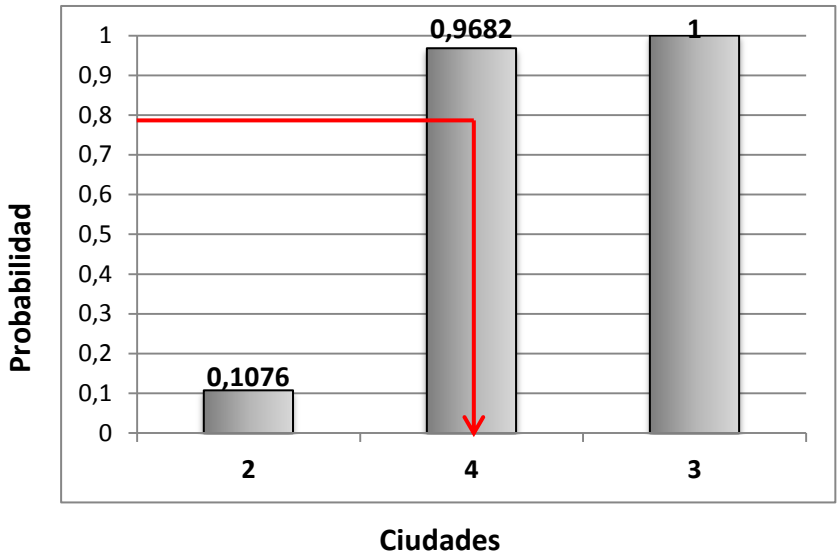


Figura 4.4. *Probabilidades acumuladas (1).*

Según lo que se ilustra en la figura 4.4, la ciudad a visitar por la hormiga k=1 es la **4.**

El proceso descrito anteriormente se vuelve a realizar, es decir, para las dos ciudades restantes que quedan por visitar, pero ahora, teniendo en cuenta que la hormiga se encuentra en la ciudad 4. Se aplica la regla probabilística (ver tabla 4.7).

La ciudad 4 ya ha sido visitada, y por tanto, no aparecerá en la formula ni la importancia relativa de la cantidad de feromona ni de la heurística para la misma. Por tanto, la memoria de la hormiga tendrá el valor de $M^1 = [1,4]$. Y el conjunto de posibles ciudades a visitar será $N^1 = [2,3]$.

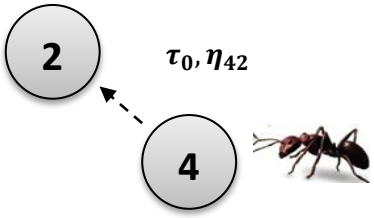
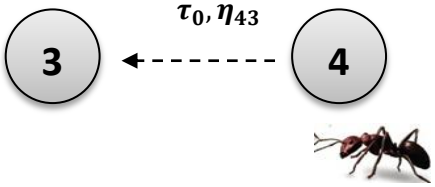
Posible ciudad a visitar	Regla probabilística de elección
	$P_{42}^1 = \frac{[\tau_0]^\alpha \cdot [\eta_{42}]^\beta}{[\tau_0]^\alpha \cdot [\eta_{42}]^\beta + [\tau_0]^\alpha \cdot [\eta_{43}]^\beta} =$ $= \frac{[1]^1 \cdot [0.5]^3}{[1]^1 \cdot [0.5]^3 + [1]^1 \cdot [1]^3} = 0.1111$
	$P_{43}^1 = \frac{[\tau_0]^\alpha \cdot [\eta_{43}]^\beta}{[\tau_0]^\alpha \cdot [\eta_{42}]^\beta + [\tau_0]^\alpha \cdot [\eta_{43}]^\beta} =$ $= \frac{[1]^1 \cdot [1]^3}{[1]^1 \cdot [0.5]^3 + [1]^1 \cdot [1]^3} = 0.8889$

Tabla 4.7. *Aplicación de la regla probabilística de elección (2).*

De nuevo, se calcula la probabilidad acumulada y se genera el número aleatorio → **0,4874**

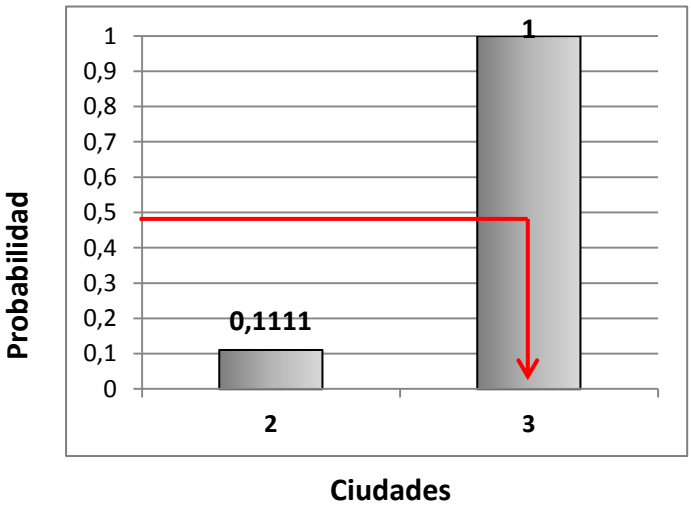


Figura 4.5. *Probabilidades acumuladas (2).*

La siguiente ciudad a visitar es la 3, según la figura 4.5. Por tanto, la ciudad 2 será la última antes de finalizar la ruta en la ciudad 1 (inicio).

Finalmente, la ruta realizada por la hormiga k=1 es:

$$T^1 = [1,4,3,2,1]$$

T^k como ya se comentó en el capítulo 3, es el vector que contiene la secuencia de ciudades visitadas por la hormiga k-ésima.

Y la distancia total recorrida por la hormiga k=1 en la iteración 1 es:

$$C^1 = d_{14} + d_{43} + d_{32} + d_{21} = 1 + 1 + 1 + 1 = 4$$

Como se habían prefijado 4 hormigas para este algoritmo, el procedimiento que se ha descrito, ha de realizarse idénticamente para k=2, 3,4.

Una vez, que las 4 hormigas han recorrido una ruta y su distancia, se realiza la actualización del rastro de feromona de la primera iteración. A continuación, se explica de forma exhaustiva para el problema ATSP, aplicando el método AS.

- **Actualización del rastro de feromona.**

Después de que todas las hormigas hayan construido sus rutas, el rastro de feromona se actualiza. El rastro que había en los arcos en la primera iteración se le multiplica por un factor ρ, denominado, coeficiente de evaporación de la feromona, donde $0 < \rho \leq 1$. El parámetro ρ se utiliza para evitar la acumulación ilimitada de los rastros de feromona y permite que las decisiones no estén sesgadas por las anteriores tomadas, las cuales pueden no ser siempre buenas. De hecho, si un arco no es elegido por las hormigas, su valor de feromona asociado disminuye con el número de iteraciones de forma exponencial.

Después de la evaporación, todas las hormigas depositan feromona en aquellos arcos que pertenezcan al recorrido realizado por las mismas, de acuerdo con la expresión (4.2):

$$\tau_{ij}^{t+1} \leftarrow (1-\rho)\tau_{ij}^{t} + \sum_{k=1}^{m} \Delta\tau_{ij}^{tk}, \qquad \forall(\mathrm{i,j}) \in L \quad (4.2)$$

- $(1-\rho)\tau_{ij}^{t}$, este primer término alude a la cantidad de evaporación de la feromona.
- $\sum_{k=1}^{m} \Delta\tau_{ij}^{tk}$, sumatorio de la cantidad de feromona que deposita cada hormiga k en la iteración t, en cada arco (i,j) perteneciente al recorrido de las mismas, T^k. El término $\Delta\tau_{ij}^{tk}$ se define en la expresión (4.3):

$$\Delta\tau_{ij}^{tk} = \begin{cases} 1/C^k & \forall(i,j) \in T^k \\ 0 & en\ otro\ caso \end{cases} \qquad (4.3)$$

donde C^k, es la longitud de la ruta T^k construida por la hormiga k-ésima y se calcula como la suma de las longitudes de los arcos pertenecientes a T^k

Generalmente, los arcos que son cruzados por la mayoría de las hormigas y que forman parte, por lo tanto, de las rutas cuya longitud es corta, reciben más cantidad de feromona, y tienen más posibilidades de ser elegidos por las hormigas que los crucen en iteraciones posteriores del algoritmo.

Retomando el ejemplo que se está resolviendo, y teniendo en cuenta las rutas, T^k, y las longitudes de las mismas, C^k de las 4 hormigas realizadas en la primera iteración (ver tabla 4.8), se calculará la actualización del rastro de feromona para todos los arcos: se actualiza la matriz de feromonas asociadas a cada arco.

Hormiga, k	T^k	C^k
1	1-4-3-2-1	4
2	1-4-2-3-1	1+2+2+2=7
3	1-4-3-2-1	4
4	1-4-3-2-1	4

Tabla 4.8.*Resultados obtenidos en la primera iteración.*

Para el arco (1,4):

$$\tau_{14}^2 \leftarrow (1-\rho)\tau_{14}^1 + \sum_{k=1}^{m} \Delta\tau_{14}^{1k} = (1-0.5)\cdot 1 + \frac{1}{4} + \frac{1}{7} + \frac{1}{4} + \frac{1}{4} = 1.3929$$

Como se puede comprobar en la tabla 4.8 todas las hormigas cruzan el arco (1,4), sin embargo, para el arco (1,2), ninguna hormiga lo atraviesa, por tanto su actualización de feromona será:

$$\tau_{12}^2 \leftarrow (1-\rho)\tau_{12}^1 + \sum_{k=1}^{m} \Delta\tau_{12}^{1k} = (1-0.5)\cdot 1 + 0 = 0.5$$

Este proceso realizado para todos y cada uno de los arcos pertenecientes al grafo, G, lo que dará lugar a una nueva matriz de rastros de feromona (tabla 4.9) que será la utilizada en la iteración 2 cuando se aplique la regla probabilística de elección.

τ_{ij}	1	2	3	4
1	0	0.5	0.5	1.3929
2	1.25	0	0.6429	0.5
3	0.5	1.25	0	0.5
4	0.5	0.6429	1.25	0

Tabla 4.9.*Matriz del rastro de feromona.*

En la tabla 4.9, se puede comprobar lo que se explicó anteriormente cuando en cuanto a la actualización de la feromona: aquellos arcos pertenecientes a las rutas de menor longitud aparecen reforzados, es decir, tienen una cantidad mayor de feromona.

En definitiva, como ya se ha indicado en capítulos anteriores, el rendimiento del algoritmo AS en comparación con otras metaheurísticas tiende a disminuir dramáticamente a medida que el tamaño de los problemas a tratar aumenta, en este caso el número de ciudades. Por lo tanto, una parte importante de la investigación de este tipo de técnicas metaheurísticas, ACO, se ha centrado en cómo mejorar el AS.

4.1.1.2. Elitist Ant System (EAS).

Una primera mejora del método AS es la denominada estrategia elitista para el Ant System (EAS). Fue introducida por Marco Dorigo en 1992 (Dorigo, 1992). La característica principal de este nuevo algoritmo ACO radica en proporcionar un fuerte refuerzo adicional a los arcos que pertenezcan a la mejor ruta encontrada desde el comienzo del algoritmo; a esta ruta se la denota como T^{bs} $(best\ so\ far\ tour)$.

La construcción de las rutas que realiza cada hormiga se lleva a cabo de la misma forma que se hizo para el método AS.

En cuanto al cálculo del rastro de feromona que se emplea en la primera iteración, se realiza de acuerdo a la expresión (4.4):

$$\tau_0 = (e + m)/\rho C^{nn} \qquad (4.4)$$

- **Actualización del rastro de feromona.**

El refuerzo adicional de la mejor ruta, T^{bs}, se consigue al añadir una cantidad e/C^{bs}, para todos los arcos pertenecientes a T^{bs}. El parámetro e se define como el peso dado para la mejor ruta y su longitud. Por tanto, el rastro de feromona para el método EAS y de acuerdo con lo que se acaba de explicar, viene dado por la expresión (4.5):

$$\tau_{ij}^{t+1} \leftarrow (1-\rho)\tau_{ij}^{t} + \sum_{k=1}^{m} \Delta\tau_{ij}^{tk} + e\Delta\tau_{ij}^{bs} \qquad (4.5)$$

El primer y segundo término son exactamente iguales a la expresión de actualización de feromona del método AS. El término adicional es $e\Delta\tau_{ij}^{bs}$, que se define por la ecuación (4.6):

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & \forall(i,j) \in T^{bs} \\ 0 & en\ otro\ caso \end{cases} \qquad (4.6)$$

Es decir, se deposita rastro de feromona en el arco (i,j) cuando la hormiga k-ésima cruza el arco (i,j), y éste, además, pertenezca a la mejor ruta considerada desde el inicio del algoritmo, T^{bs}. Este depósito es igual al inverso de la longitud de la mejor ruta multiplicado por el parámetro e, el cual es fijado por el usuario. Nótese que el valor más apropiado para el parámetro e suele ser el número de ciudades n del problema a resolver según los resultados experimentales de Dorigo y Stützle (Dorigo y Stützle, 2004).

Los resultados computacionales presentados por Dorigo en 1992 nos sugieren que el uso de la estrategia elitista con un apropiado valor del parámetro e (ya comentado en

el párrafo anterior) permite que el método EAS encuentre mejores resultados en menos número de iteraciones de lo que lo haría el método AS.

4.1.1.3. Rank-Based Ant System (ASrank).

Otra mejora del método AS se basa en el empleo de rangos, denominada Rank-Based Ant System (ASrank) (Bullnheimer et al., 1997).

En el método ASrank, cada hormiga deposita una cantidad de feromona la cual decrece con el rango. Adicionalmente, como en el EAS, también se tiene en cuenta el depósito que realiza la mejor ruta en cada iteración.

La construcción de las rutas que realiza cada hormiga se lleva a cabo de la misma forma que se hizo para el método AS.

En cuanto al cálculo del rastro de feromona que se emplea en la primera iteración, se realiza de acuerdo a la expresión (4.7):

$$0.5w(w-1)/\rho C^{nn} \qquad (4.7)$$

Como se puede ver en la expresión (4.7), entra en juego un nuevo parámetro, w, que significa el número de hormigas que depositan feromonas. Generalmente, el valor de w suele ser 6, para el funcionamiento óptimo del algoritmo.

- **Actualización del rastro de feromona.**

Antes de actualizar el rastro de feromona, las hormigas se ordenan según la longitud de sus rutas, de menor a mayor longitud. La cantidad de feromona que una hormiga deposita viene determinada por el rango r de las hormigas. Esto quiere decir, una vez ordenadas las hormigas de menor a mayor longitud recorrida por cada una de ellas, se escoge un rango de r hormigas, el cual suele ser r=w-1; Por tanto, en cada iteración solo las (w-1) hormigas mejores del rango y la hormiga que ha producido la mejor ruta (que al igual que para el EAS, esta hormiga no tiene que corresponder, necesariamente, con el conjunto de hormigas de la iteración actual del algoritmo) serán las responsables del depósito de feromona. La evaporación del rastro de feromona de la iteración actual se realiza de la misma forma que para el método AS y EAS. En definitiva, la actualización del rastro de feromona para el método ASrank viene determinada por la expresión (4.8):

$$\tau_{ij}^{t+1} \leftarrow (1-\rho)\tau_{ij}^{t} + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{tr} + w\Delta\tau_{ij}^{bs} \qquad (4.8)$$

En donde :

$$\Delta\tau_{ij}^{tr} = \begin{cases} 1/C^r & \forall(i,j) \in T^r \\ 0 & en\ otro\ caso \end{cases} \quad (4.9)$$

$$\Delta\tau_{ij}^{bs} = \begin{cases} 1/C^{bs} & \forall(i,j) \in T^{bs} \\ 0 & en\ otro\ caso \end{cases} \quad (4.10)$$

La expresión (4.9) representa la r-ésima mejor hormiga del rango en la iteración actual que contribuye al depósito de feromona con el valor $1/C^r$ multiplicado por el peso (w-r) como aparece en la expresión (4.8).

La expresión (4.10), la mejor ruta es la que proporciona la cantidad más grande de depósito de feromona. Al igual que en el método EAS, se calcula como la inversa de la longitud del mejor recorrido: el de menor longitud.

Los resultados de una evaluación experimental de Bullnheimer (Bullnheimer et al., 1999), sugieren que el método ASrank presenta resultados ligeramente mejores que los del método EAS y significativamente mejores que los del método AS. Generalmente, para un número de iteraciones elevado, el método ASrank presenta mejores resultados que EAS, sin embargo, cuando no se ejecutan muchas iteraciones el método EAS, puede presentar resultados mejores que éste, aunque como se acaba de apuntar, ligeramente mejores para las condiciones de operación óptimas, es decir, otorgando a los parámetros del algoritmo los valores aportados en la tabla 4.4.

4.1.2. Programación del algoritmo ACO. Implementación en C++.

Se ha empleado un lenguaje de programación de alto nivel que nos permite utilizar funciones y librerías con el fin de optimizar el tiempo de ejecución del algoritmo y flexibilidad en cuanto a la manipulación y cálculo de los datos. Para su implementación y experimentación, hemos empleado el entorno de desarrollo *CodeBlocks The open source- cross.platform IDE 13.12* en C++.

Cada fichero .cpp tiene asociado su fichero de cabecera .h, todos ellos llamados desde una función principal denominada main () donde se realizan las llamadas a otras funciones ubicadas en otros ficheros .cpp. El programa sigue una dinámica funcional: el main hace una llamada a una función, ésta se ejecuta devuelve un resultado y retorna al main el cual volverá a hacer otra llamada a la siguiente función. Cada fichero .cpp tiene en sí escritas funciones en C++ las cuales se van ejecutando de forma secuencial

leyendo una a una las líneas del código Los resultados, así como los parámetros iniciales del algoritmos, serán sacados por pantalla, es decir, por el cmd.

4.1.2.1. Esquema general del algoritmo.

El programa consta de 6 ficheros .cpp: extraccion_datos.cpp, inversa.cpp, C_nn.cpp, ACO_AS.cpp, Actualizar.cpp y el main.cpp. Cada uno de ellos contiene definidas unas funciones que se irán ejecutando según sean llamadas en el main.cpp, el principal. El funcionamiento del algoritmo es el siguiente: desde el main, a través del cmd, se solicita al usuario el nombre del problema .txt que se desea resolver así como una serie de parámetros necesarios para la resolución del algoritmo. Después, el main.cpp llama a la primera función que se denomina *extraer_datos* ubicada en el fichero extracción_datos.cpp, ésta a su vez extrae los datos que se encuentran en los ficheros .txt del problema que se quiere resolver. Una vez ejecutadas las sentencias dentro de *extraer_datos*, esta función devuelve matriz (siendo ésta la matriz de distancias del problema) y vuelve al main donde éste llamará a la siguiente función la cual es *calculoInversa* ubicada en el fichero inversa.cpp. La función *calculoInversa* realiza la inversa de la matriz de distancias, por lo tanto, la entrada de la función es la matriz obtenida en *extraer_datos*. La salida que devuelve esta función es matriz_inv_dist. A continuación, se vuelve de nuevo al main el cual llamará a la funciónsiguiente: *recorrido*, ubicada en el fichero C_nn.cpp. Esta función lo que devuelve es la mínima distancia recorrida según la heurística de Greedy ya explicada en el capítulo 3 como se calculaba.

De nuevo, se vuelve al main que llamará a *funcion* ubicada en al fichero ACO_AS.cpp el cual contiene una serie de funciones creadas para la resolución del algoritmo ACO. Esto se explicará más detalladamente a continuación. En función de los parámetros de entrada, se empleará el método Ant System, Elitist Ant System o el Ranked-Based Ant System. A diferencia de los otros tres ficheros explicados anteriormente, ACO_AS.cpp comprueba mediante un bucle si se satisface el criterio de parada, si no es así, realiza una llamada a la función *actualizar_pher* ubicada en Actualizar.cpp donde se produce la actualización del rastro de feromona y de nuevo se vuelve a ejecutar el algoritmo en ACO_AS.cpp. Por el contrario, si la condición de parada si se ha cumplido se retorna al ACO_AS.cpp el cual escribirá los resultados en ficheros .txt, constituyendo éstos las salidas del programa: Variables.txt, Resultados.txt, ResultadosIter.txt. Finalmente, se retorna al main para acabar la ejecución.

El esquema ubicado en la figura 4.6, muestra la lógica que sigue el programa en .cpp para la resolución del algoritmo, así se podrá tener una visión más conceptual y esquemática de la lógica del algoritmo.

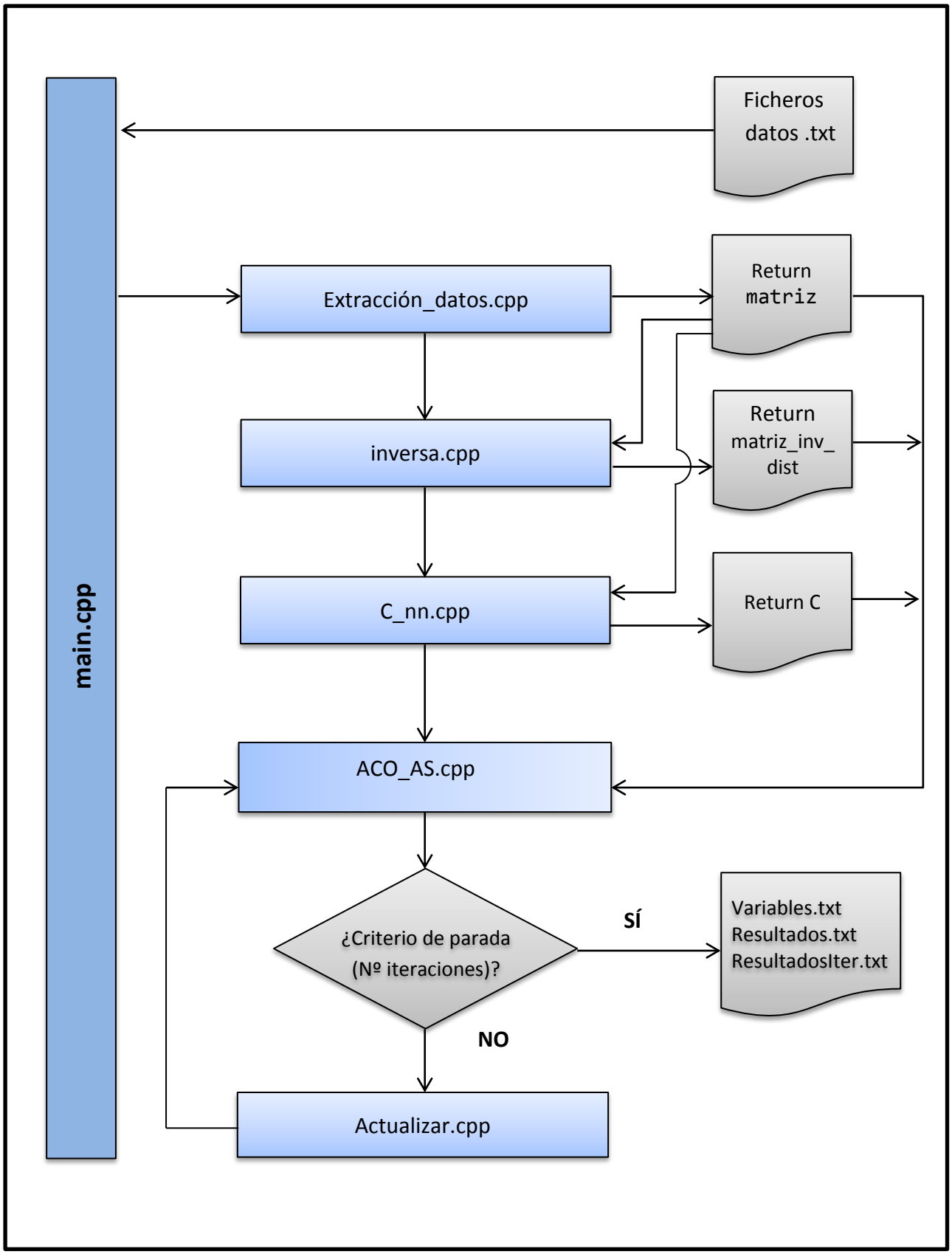


Figura 4.6. *Esquema de los ficheros .cpp del programa.*

4.1.2.2. Entrada de datos.

El main se encarga de llamar al archivo extracción de datos que, como ya se apunta anteriormente, su función principal es la extracción de las cadenas de caracteres que componen el fichero .txt de problemas, y luego, almacenarlas en una matriz de tamaño *dimensionxdimension*, siendo *dimension* el número de ciudades a visitar en el problema dado.

Pero antes de esto, hay que escribir en el programa exactamente el fichero que se desea resolver así como los parámetros de entrada para que el algoritmo pueda ejecutarse. Por tanto, en el main se solicitará al usuario:

- El nombre del fichero .txt que se desee resolver: que se almacena en un array de caracteres denominado *fichero* [50] con posibilidad de almacenar hasta 50 caracteres.
- El método ACO que se quiera emplear, con la variable *opción* siendo ésta un entero. Como ya se ha dicho, existen 3 opciones posibles: AS, *AntSystem* tendrá el valor 1; EAS, *ElitistAntSystem* con el valor 2; y, por último, ASrank, *Ranked-BasedAntSystem*, con el valor 3.
- El número de ejecuciones que queremos que el algoritmo se ejecute y saque resultados por pantalla. El número de ejecuciones se almacena en una variable de tipo entero denominada *ejecuciones*.
- El número de hormigas, m, se almacena en una variable *m* de tipo entero.
- El número de iteraciones n que se realizan para una ejecución, se almacena en una variable *n* de tipo entero.
- EL valor de la influencia relativa del rastro de feromona, es decir, del parámetro α, almacenado en la variable, *a,* de tipo double.
- EL valor de la influencia relativa de la heurística, β, que también será almacenado en una variable de tipo double denominada *b*, en este caso.
- El valor del coeficiente de evaporación del rastro de feromona que como ya se ha mencionado, debe estar comprendido entre $0<\rho\leq 1$. Se almacena en una variable *p* de tipo double.
- En el caso de que el usuario introduzca el método 2, EAS, se le solicitará a mayores el valor de e (peso de la mejor ruta) almacenado en una variable de tipo double, *e.*

- Si, en cambio, se solicita la opción 3, entonces el usuario deberá introducir el valor de w (número de hormigas que depositan feromonas), almacenado en una variable de tipo double, *w*.

En la figura 4.8, puede verse como aparece en el cmd lo anteriormente explicado:

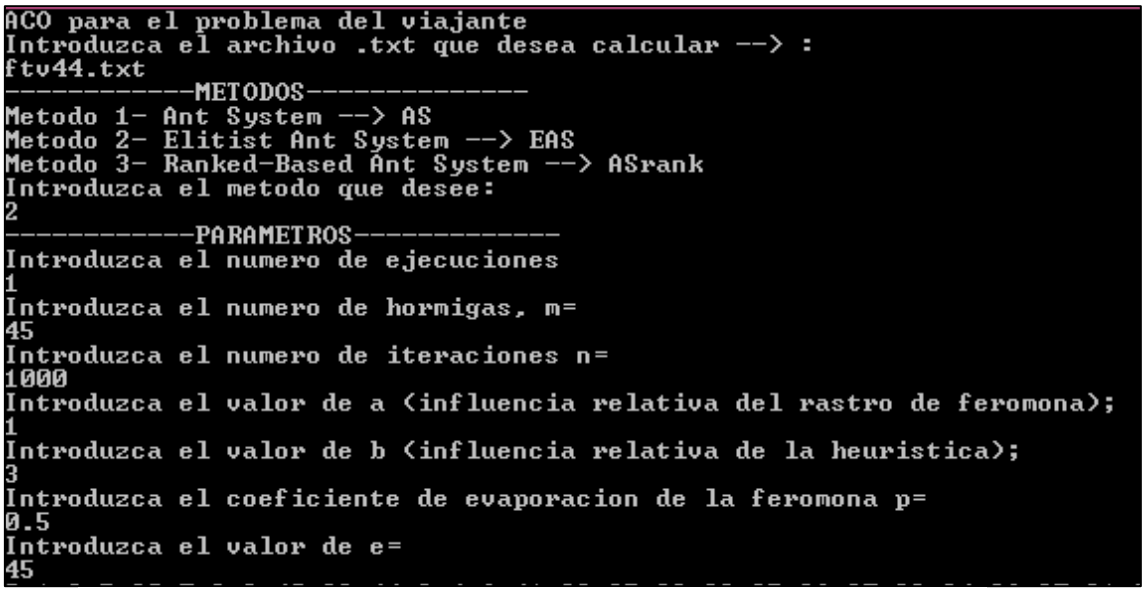


Figura 4.7.*Entrada de datos por pantalla.*

En este caso el usuario ha solicitado el método EAS.

4.1.2.3. Salida de datos.

La salida de datos por pantalla se realiza a través de ficheros de texto. Una vez se haya completado cada ejecución del algoritmo, es decir, el programa haya realizado todas las iteraciones que el usuario haya solicitado para una ejecución, esos resultados se habrán escrito en unos ficheros .txt de salida.

Existen tres ficheros .txt como salida de resultados, en los que el programa va escribiendo conforme se va ejecutando. En el ACO_AS.cpp se escriben los resultados en la variable *destino* cuya salida es el fichero Resultados.txt y en la variable *destino2*, en este caso el fichero .txt de salida se denomina ResultadosIter.txt. El tercer fichero de salida de datos se declara en el main en la variable *destino,* cuyo fichero asociado es Variables.txt. A continuación, se explica detalladamente cada uno de estos ficheros y que datos contienen.

- **Resultados.txt.**

Este fichero .txt contiene la secuencia de la ruta mejor que se ha obtenido en cada ejecución seguida de la distancia total de la misma. Esto quiere decir que si el usuario ha solicitado 1 ejecución de 1000 iteraciones, entre esas 1000 iteraciones el mejor resultado de las mismas corresponderá con el resultado de la ejecución número 1. En la figura siguiente, se puede ver que para el problema ftv33.txt, cuando se introducen 10 ejecuciones de 500 iteraciones con el método AS, m=34, α=0, β=3, ρ=0.5, los resultados por pantalla son (ver figura 4.8):

```
***** RESULTADO DE LA EJECUCION 0 *****
0 13 12 15 14 16 1 3 30 33 2 25 24 23 17 9 32 7 6 5 4 8 10 11 31 18 19 20 21 22 26 27 28 29 0 ---->1480
***** RESULTADO DE LA EJECUCION 1 *****
0 12 13 15 14 16 1 2 3 25 24 23 19 20 21 22 29 26 27 28 33 30 4 6 5 32 7 8 10 11 31 18 17 9 0 ---->1501
***** RESULTADO DE LA EJECUCION 2 *****
0 13 9 32 7 8 10 11 31 18 19 21 20 22 26 29 27 28 25 24 23 17 12 14 15 16 1 30 33 2 3 4 5 6 0 ---->1523
***** RESULTADO DE LA EJECUCION 3 *****
0 13 14 15 16 1 25 24 23 19 17 11 8 9 32 7 4 6 5 30 33 2 3 27 28 29 26 22 20 21 31 18 10 12 0 ---->1456
***** RESULTADO DE LA EJECUCION 4 *****
0 13 9 32 7 4 6 5 29 26 22 20 21 27 28 25 24 23 19 31 18 17 11 8 10 14 15 16 12 1 33 30 2 3 0 ---->1491
***** RESULTADO DE LA EJECUCION 5 *****
0 13 12 14 15 16 25 24 23 20 21 27 28 29 26 22 17 18 19 31 11 10 7 32 8 9 4 5 6 30 33 2 3 1 0 ---->1538
***** RESULTADO DE LA EJECUCION 6 *****
0 13 14 15 16 25 24 23 17 9 32 7 4 6 5 2 3 1 33 30 29 22 27 28 26 20 21 18 19 31 11 8 10 12 0 ---->1533
***** RESULTADO DE LA EJECUCION 7 *****
0 13 16 15 14 12 9 32 7 8 10 4 6 5 2 3 24 23 19 18 17 11 31 20 21 22 26 27 28 29 1 33 30 25 0 ---->1499
***** RESULTADO DE LA EJECUCION 8 *****
0 13 9 25 24 23 19 20 21 31 18 17 11 8 10 32 7 4 6 5 30 33 2 3 26 22 27 28 29 1 15 14 16 12 0 ---->1522
***** RESULTADO DE LA EJECUCION 9 *****
0 25 24 23 19 18 17 32 7 4 6 5 30 33 2 3 13 16 15 14 12 9 8 10 11 31 21 20 22 26 27 28 29 1 0 ---->1455
```

Figura 4.8.*Salida de datos en el fichero Resultados.txt*

En el cmd, también aparece lo mismo que en este fichero, por ejemplo para los valores que aparecen en la figura 4.7 en la que se ilustraba la entrada de datos y parámetros, el resultado es el siguiente (ver figura 4.9):

```
0 1 2 5 20 7 9 8 40 38 44 3 4 6 41 39 35 32 28 25 26 27 33 34 36 37 31 30 29 24
42 23 22 18 16 17 14 43 10 11 12 13 15 19 21 0 ---->1675

Process returned 0 (0x0)   execution time : 105.160 s
Press any key to continue.
```

Figura 4.9.*Salida de datos por pantalla en el cmd.*

- **ResultadosIter.txt.**

Al igual que Resultados.txt, ResultadosIter.txt se declara también en el archivo ACO_AS.cpp. Este fichero de texto contiene los resultados de cada iteración, es decir, la mínima distancia y su ruta correspondiente en cada iteración. En cada iteración, habrá m rutas posibles con sus respectivas distancias y de entre todas ellas, en ResultadosIter.txt, aparece la información de aquella que corresponde con la menor distancia. En la figura 4.10, aparece de nuevo para el problema ftv33.txt los resultados de cada iteración para 1 ejecución de 30 iteraciones con el método AS, m=34, α=1, β=3, ρ=0.5):

```
0 13 12 15 14 16 25 24 23 19 17 32 7 6 4 5 1 3 8 10 9 11 18 31 21 20 22 26 27 28 29 33 30 2 0 ----> 1801
0 13 9 32 7 4 6 5 8 10 11 18 19 21 20 22 24 23 17 12 14 15 16 25 1 33 30 2 3 29 26 27 28 31 0 ----> 1767
0 13 14 15 16 25 24 23 19 20 21 31 18 17 11 10 32 7 5 6 4 22 26 27 28 29 33 30 2 3 1 8 9 12 0 ----> 1658
0 13 7 8 9 32 4 6 5 2 3 33 30 29 26 22 20 21 19 31 18 17 11 10 24 23 27 28 12 14 15 16 1 25 0 ----> 1610
0 13 9 12 15 14 16 1 25 24 23 19 27 28 29 26 22 20 21 31 18 17 11 8 32 7 4 6 5 30 33 2 3 10 0 ----> 1499
0 13 12 9 32 7 8 10 11 31 18 19 17 16 15 14 25 24 23 20 21 22 29 26 27 28 33 30 4 6 5 3 1 2 0 ----> 1541
0 13 16 15 14 12 9 32 7 8 10 11 31 18 19 20 21 22 26 27 28 29 25 24 23 17 4 6 5 30 33 2 3 1 0 ----> 1382
0 13 12 14 15 16 25 24 23 19 17 11 31 18 20 21 22 26 27 28 29 1 3 33 30 2 4 6 5 32 7 8 10 9 0 ----> 1418
0 13 12 14 15 16 1 33 2 3 25 24 23 19 20 21 31 18 17 11 10 9 32 7 8 4 6 5 30 29 26 22 27 28 0 ----> 1417
0 13 9 32 7 4 6 5 2 3 1 33 30 29 26 22 20 21 31 18 19 17 11 8 10 12 14 15 16 25 24 23 27 28 0 ----> 1411
```

Figura 4.10.*Salida de datos en el fichero ResultadosIter.txt*

- **Variables.txt.**

A diferencia de los otros dos anteriores, se declara en el main y recoge los parámetros de entrada, es decir los valores que se introdujeron en el cmd cuando se inició la ejecución del programa. La figura 4.11 muestra el contenido del fichero Variables.txt:

```
Archivo .txt que desea calculado --> : ftv44.txt
------------METODO--------------
Metodo 2- Elitist Ant System --> EAS
------------PARÁMETROS-------------
Numero de ejecuciones 1
Numero de hormigas, m= 45
Numero de iteraciones n= 1000
Valor de a (influencia relativa del rastro de feromona); 1
Valor de b (influencia relativa de la heurística); 3
Coeficiente de evaporacion de la feromona p= 0.5
Valor de e= 45
Valor de w= 0
```

Figura 4.11.*Contenido del fichero Variables.txt*

4.1.2.4. Archivos y funciones principales del algoritmo.

En este apartado se explican detalladamente cada uno de los ficheros fuente .cpp de los que está compuesto el programa en C++, y dentro de los cuales, qué funciones contienen y qué es lo que realizan. Por tanto, teniendo en cuenta lo mencionado anteriormente y siguiendo la lógica del esquema que contempla la figura 4.6 el programa ACO_ATSP *Project file* está compuesto por:

- **Extracción_datos.cpp**. Este fichero .cpp contiene una función que extrae las cadenas de caracteres del fichero .txt de datos para almacenarlas en un vector denominado fichero; también extrae el número de ciudades del problema y lo almacena en una variable de tipo entero denominada *dimension.* Esta variable se utilizará para determinar el tamaño de la matriz de distancias que es lo que se devuelve como resultado esta función.

- **inversa.cpp.** A la función de este fichero, se le pasa como parámetros la matriz de distancias y el tamaño del problema, es decir, el número de ciudades que corresponde, como hemos explicado antes, con la variable entera *dimension.* En definitiva, lo que realiza esta función es calcular la inversa de las distancias, es decir, la importancia relativa de la heurística empleada posteriormente en el archivo ACO_AS.cpp para la construcción de las rutas.

- **C_nn.cpp.** Este fichero contiene la función que calcula distancia de la ruta aplicando la heuristica de Greedy, es decir, siendo la próxima ciudad a visitar la que a menor distancia se encuentre de la actual. Los parámetros que se le pasan a la función son: *dimension*, la *matriz* de distancias y una nueva variable denominada *c*, la cual retornará el resultado de la longitud de la ruta aplicando la heurística Greedy.

- **ACO_AS.cpp.** Este fichero contiene una función que calcula el rastro de feromona inicial de la primera iteración*, c0*. Dependiendo del método ACO, se calcula de una forma u otra (explicado anteriormente). También, se crea otra función para el cálculo de la probabilidad de visitar las ciudades que todavía no se han visitado por la hormiga.

- **Actualizar.cpp.** Este fichero contiene una función para comprobar si el arco (i,j) pertenece o no a la mejor ruta de la mejor hormiga del algoritmo, así como tres funciones más, una para cada actualización del rastro de feromona dependiendo del método que elija el usuario: AS, EAS y ASrank. Este fichero es llamado desde el fichero ACO_AS.cpp a diferencia de los explicados anteriormente que son llamados desde el main.

4.2. EXPERIMENTACIÓN.

4.2.1. Problemas a resolver.

El algoritmo ACO junto con sus variantes ha sido probado para todos los ficheros de problemas del ATSP que se puede encontrar en la librería TSPLIB95 creada por Gerhard Reinelt [1], en la que para diversos tipos de problemas, existen una serie de ficheros sobre los que se investiga para hallar las rutas de mínima distancia para los mismos. La tabla 4.10 muestra todos los problemas sobre los que hemos realizado simulaciones y pruebas, el tamaño de cada uno de ellos y el mejor resultado encontrado hasta el momento:

Tipo de problema	Problema	Número de ciudades	Mejor resultado, C^{best}
Problemas pequeños	**br17**	17	39
	ft53	53	6905
	ft70	70	38673
	ftv33	34	1286
	ftv35	36	1473
	ftv38	39	1530
	ftv44	45	1613
	ftv47	48	1776
	ftv55	56	1608
	ftv64	65	1839
	ftv70	71	1950
	ftv170	171	2755
	kro124p	100	36230
	p43	43	5620
Problemas grandes	**ry48p**	48	14422
	rbg323	323	1326
	rbg358	358	1163
	rbg403	403	2465
	rbg443	443	2720

Tabla 4.10. *Listado de problemas .txt a resolver.*

4.2.2. Simulaciones.

Para cada uno de estos problemas se ha realizado una serie de simulaciones en las que se ha variado el número de hormigas, m, pudiendo ser este igual al número de ciudades o el doble a éste, en el caso de problemas pequeños, para los grande se ha fijado a 50 hormigas; el número de iteraciones, dependiendo de si tratamos un problema pequeño (500, 1000, 5000) o grande (5000, 10000, 30000); y, por último, la importancia relativa del rastro de feromona, α (pudiendo ser 0, en este caso toda la importancia se le da a la heurística cuando se aplica la regla de elección basada en probabilidades para la construcción de la ruta, o por el contrario 1). El resto de parámetros que intervienen se han mantenido fijos para cada método ACO. La elección del valor de los parámetros fijos se ha hecho de acuerdo con la tabla 4.4, en la que Dorigo obtuvo una serie de valores para los cuales los algoritmos ACO proporcionaban las mejores soluciones. La tabla 4.11 resume lo explicado anteriormente:

Variable	Nombre		Valor
n	dimension--> número de ciudades		Tamaño del problema
m	nº hormigas	Problemas pequeños	Puede ser n ó 2n
		Problemas grandes	50 para problemas grandes
método 1	AS (Ant System)		1
método 2	EAS (Elitist Ant System)		2
método 3	Asrank (Rank-based Ant System)		3
α	Importancia del rastro de feromona		1 ó 0
β	importancia relativa de la heurística		3
ρ	coeficiente de evaporación de la feromona ϵ {0,1}		0.5 para AS y EAS
			0.1 para Asrank
e	Peso de la mejor ruta		n
w	número de hormigas que depositan feromonas		6
nº iteraciones	Problemas pequeños		500; 1000; 5000;
	Problemas grandes		5000; 10000; 30000

Tabla 4.11. *Posibles valores de los parámetros de las simulaciones.*

En definitiva, hemos realizado 36 simulaciones para cada problema pequeños y 18 para los grandes, con todas las combinaciones posibles. Para cada simulación realizada, el **número de ejecuciones** será siempre **30.** En el siguiente esquema muestra las simulaciones realizadas para cada problema según el tipo: grande o pequeño.

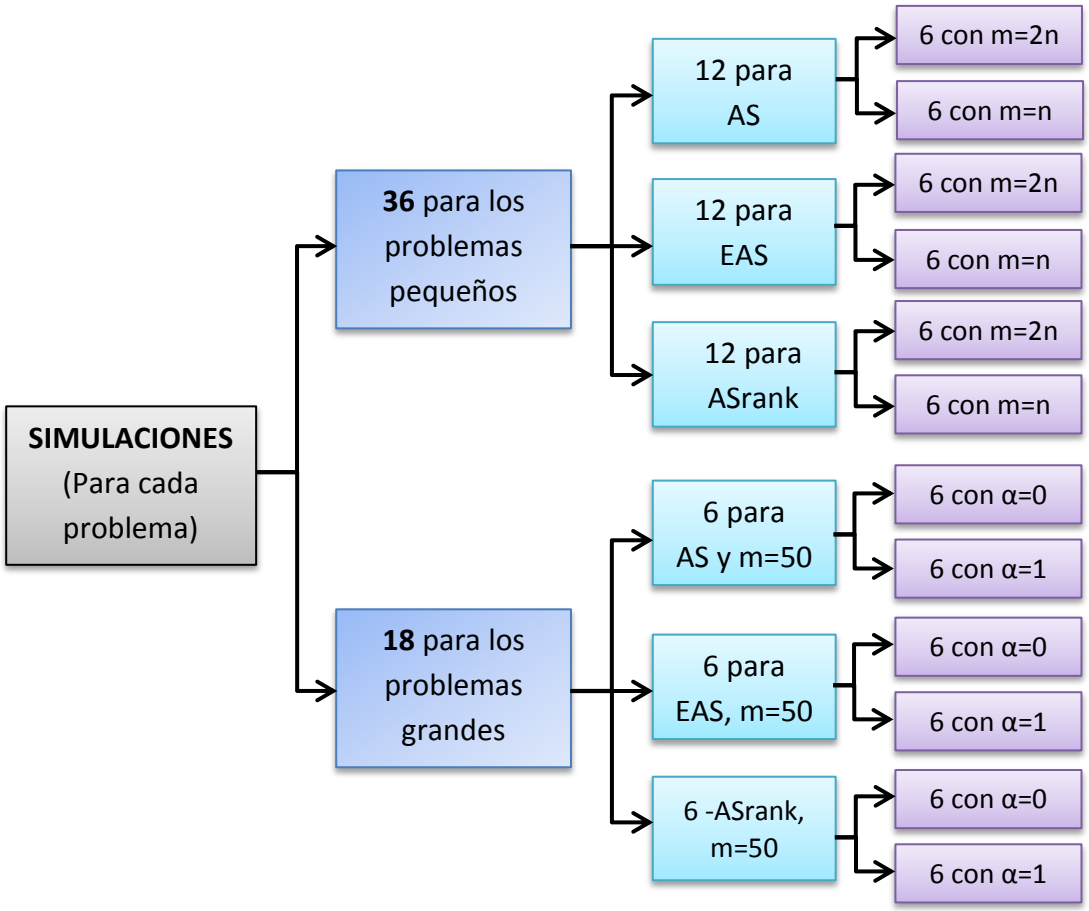


Figura 4.12. *Simulaciones en la experimentación.*

Para cada método ACO, 12 simulaciones se han realizado para los problemas pequeños, en las que 6 de esas 12 se han hecho con el doble de hormigas que el tamaño del problema, y las otras 6 con el mismo valor que el tamaño del problema (para los problemas grandes, el número de hormigas se fija a 50). Dentro de estas 6 simulaciones, tres de ellas se tomará el valor de α=0, mientras que las otras tres con la importancia relativa de la feromona, α=1. Para cada una de esas tres, se fijaran 500, 1000, 5000 iteraciones respectivamente para los problemas pequeños; y, 5000, 10000, 30000 para los grandes.

4.2.3. Presentación de resultados.

Los resultados que se muestran a continuación, corresponden con el error hallado a partir de la media de las 30 ejecuciones obtenidas para cada simulación y con la media de las ejecuciones para cada simulación, según la expresión (4.11). El error se haya como la desviación de la media de las ejecuciones respecto del valor óptimo del problema (mejor resultado, tabla 4.10), ver expresión (4.12):

$$C^{medio} = \frac{\sum_{ej=1}^{30} C^{ej}}{30} \quad \forall\, ej = 1, \dots, 30 \qquad (4.11)$$

$$\%\ \text{Error} = \frac{C^{medio} - C^{best}}{C^{medio}} \cdot 100 \qquad (4.12)$$

		m=2·n; iter=500; $\alpha_{=0}$			m=2·n; iter=1000; $\alpha_{=0}$			m=2·n; iter=5000; $\alpha_{=0}$			m=2·n; iter=500; $\alpha_{=1}$			m=2·n; iter=1000; $\alpha_{=1}$			m=2·n; iter=5000; $\alpha_{=1}$		
Fichero	**n**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**
br17	17	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,09%	0,00%	0,00%	0,00%	0,00%	0,00%
ft53	53	26,98%	26,76%	26,35%	26,29%	26,02%	25,60%	23,96%	24,46%	24,95%	16,45%	10,64%	10,53%	16,64%	10,24%	9,28%	16,75%	8,71%	9,09%
ft70	70	26,73%	26,71%	26,73%	26,36%	26,28%	26,17%	25,49%	25,58%	25,49%	4,65%	2,92%	2,60%	4,66%	3,03%	2,74%	4,85%	3,11%	2,43%
ftv33	34	14,66%	14,64%	14,90%	13,59%	13,61%	13,63%	11,00%	10,95%	4,04%	6,49%	4,61%	3,53%	6,33%	4,50%	3,21%	6,59%	5,40%	4,04%
ftv35	36	15,55%	15,47%	15,05%	14,32%	14,15%	13,84%	11,27%	11,55%	11,73%	5,82%	2,91%	2,02%	5,00%	2,53%	1,69%	5,33%	2,87%	1,55%
ftv38	39	17,25%	17,26%	48,09%	16,49%	16,35%	15,60%	13,84%	13,65%	13,82%	6,60%	4,39%	2,78%	6,43%	3,59%	3,00%	5,89%	3,73%	6,03%
ftv44	45	22,33%	21,87%	21,83%	21,31%	21,07%	20,95%	18,66%	18,07%	18,52%	8,01%	4,91%	3,65%	8,02%	4,42%	4,13%	7,61%	3,44%	3,03%
ftv47	48	23,87%	23,94%	24,19%	22,76%	22,33%	23,02%	20,45%	20,53%	20,83%	8,97%	7,80%	7,04%	8,40%	7,82%	6,95%	8,23%	7,77%	6,95%
ftv55	48	23,87%	24,07%	24,19%	22,89%	21,99%	22,14%	19,73%	20,40%	20,07%	8,09%	5,76%	4,99%	8,04%	6,23%	5,08%	8,41%	5,54%	4,83%
ftv64	65	29,79%	29,53%	30,16%	28,12%	27,31%	28,56%	25,88%	26,01%	25,94%	8,46%	6,79%	5,58%	8,77%	6,83%	5,52%	8,16%	6,09%	5,43%
ftv70	71	31,13%	30,69%	30,97%	27,86%	30,17%	30,27%	27,63%	28,27%	27,95%	11,78%	8,12%	7,67%	11,37%	8,02%	7,10%	10,92%	6,82%	6,97%
ftv170	171	47,05%	46,73%	46,88%	46,32%	46,70%	46,14%	45,91%	46,37%	46,33%	17,07%	14,20%	10,44%	16,85%	11,10%	11,02%	16,50%	9,87%	10,61%
p43	43	0,51%	0,50%	0,49%	0,47%	0,46%	0,45%	0,37%	0,38%	0,35%	0,55%	0,39%	0,31%	0,54%	0,40%	0,34%	0,46%	0,35%	0,28%
ry48p	48	20,39%	20,38%	20,49%	18,90%	19,24%	19,42%	17,60%	17,61%	17,55%	6,85%	5,60%	4,62%	6,54%	5,48%	3,88%	6,75%	5,12%	3,69%
kro124p	100	35,42%	35,09%	35,18%	34,84%	34,91%	34,92%	33,50%	33,45%	33,39%	11,25%	8,09%	7,88%	10,95%	8,03%	35,00%	10,64%	8,15%	7,47%
calidad media error		**22,37%**	**22,24%**	**24,37%**	**21,37%**	**21,37%**	**21,38%**	**19,69%**	**19,82%**	**19,40%**	**8,07%**	**5,81%**	**4,91%**	**7,91%**	**5,48%**	**6,59%**	**7,81%**	**5,13%**	**4,83%**
		m=n; iter=500; $\alpha_{=0}$			**m=n; iter=1000; $\alpha_{=0}$**			**m=n; iter=5000; $\alpha_{=0}$**			**m=n; iter=500; $\alpha_{=1}$**			**m=n; iter=1000; $\alpha_{=1}$**			**m=n; iter=5000; $\alpha_{=1}$**		
Fichero	**n**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**
br17	17	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,17%	0,00%	0,00%	0,17%	1,60%	0,00%	0,59%	0,00%	0,00%
ft53	53	27,63%	27,66%	27,29%	26,72%	27,14%	26,12%	24,70%	24,85%	24,95%	17,23%	11,50%	11,31%	16,96%	12,03%	10,45%	16,56%	9,97%	9,13%
ft70	70	27,23%	27,14%	27,21%	26,72%	26,68%	26,69%	25,75%	26,00%	25,93%	4,91%	4,60%	3,06%	5,23%	4,35%	3,15%	4,89%	4,21%	2,74%
ftv33	34	15,76%	15,38%	15,74%	14,71%	14,70%	14,45%	12,47%	12,45%	11,66%	7,52%	7,28%	4,78%	6,92%	6,46%	3,86%	6,83%	6,34%	4,12%
ftv35	36	16,77%	16,05%	16,78%	14,96%	14,97%	15,72%	12,27%	12,91%	12,87%	6,21%	5,20%	2,52%	6,70%	4,58%	2,07%	6,28%	4,76%	2,22%
ftv38	39	18,30%	17,89%	18,50%	17,65%	17,94%	17,74%	14,87%	15,05%	15,14%	6,51%	5,15%	3,31%	6,40%	4,66%	3,30%	6,50%	5,44%	3,28%
ftv44	45	23,08%	22,61%	23,17%	22,11%	21,71%	21,70%	19,63%	18,99%	19,24%	8,45%	6,54%	4,74%	8,92%	5,86%	4,37%	8,72%	5,33%	3,64%
ftv47	48	24,89%	24,60%	25,50%	23,73%	23,88%	23,67%	21,17%	21,95%	21,38%	9,08%	9,11%	7,75%	9,42%	8,71%	7,22%	8,43%	8,91%	7,53%
ftv55	48	24,59%	24,44%	25,15%	23,82%	23,52%	23,24%	21,32%	20,89%	21,27%	9,01%	7,35%	5,38%	8,76%	7,53%	5,44%	8,82%	6,83%	5,56%
ftv64	65	30,38%	30,17%	30,46%	28,69%	28,73%	29,56%	27,68%	26,93%	27,01%	7,27%	7,27%	6,34%	8,82%	8,18%	5,79%	8,77%	7,34%	6,48%
ftv70	71	32,62%	31,87%	32,13%	30,83%	31,02%	31,27%	29,03%	29,51%	28,85%	11,95%	9,41%	8,19%	11,74%	9,36%	7,78%	11,86%	8,49%	8,00%
ftv170	171	47,70%	47,36%	47,51%	46,88%	47,15%	46,83%	45,59%	45,43%	45,13%	17,72%	12,02%	11,37%	17,43%	11,12%	11,32%	17,30%	9,88%	10,19%
p43	43	0,53%	0,53%	0,51%	0,47%	0,48%	0,50%	0,40%	0,41%	0,40%	0,58%	0,49%	0,38%	0,56%	0,51%	0,34%	0,52%	0,43%	0,34%
ry48p	48	20,82%	20,60%	21,13%	11,27%	19,98%	20,71%	18,11%	18,48%	18,35%	7,35%	6,78%	4,64%	7,31%	7,45%	4,34%	7,31%	6,64%	4,15%
kro124p	100	36,20%	35,86%	36,05%	35,75%	35,11%	35,36%	34,00%	34,02%	33,69%	11,83%	9,39%	9,12%	11,72%	9,56%	8,64%	11,22%	9,40%	8,31%
calidad media error		**23,10%**	**22,81%**	**23,14%**	**21,62%**	**22,20%**	**22,24%**	**20,47%**	**20,52%**	**20,39%**	**8,39%**	**6,80%**	**5,53%**	**8,47%**	**6,80%**	**5,21%**	**8,31%**	**6,26%**	**5,05%**
		m=50; iter=5000; α=0			**m=50; iter=5000; α=1**			**m=50; iter=10000; α=0**			**m=50; iter=10000; α=1**			**m=50; iter=30000; α=0**			**m=50; iter=30000; α=1**		
Fichero	**n**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**
rbg323	323	29,68%	29,46%	29,70%	10,53%	10,33%	9,41%	29,32%	29,28%	29,48%	10,15%	8,76%	8,64%	29,13%	29,19%	29,04%	9,80%	8,50%	8,51%
rbg358	358	28,80%	28,65%	28,71%	13,75%	9,28%	10,74%	28,53%	28,61%	28,47%	13,23%	9,12%	10,43%	28,45%	28,55%	27,97%	13,15%	8,72%	9,96%
rbg403	403	18,87%	19,15%	18,91%	13,82%	11,84%	12,25%	18,80%	19,05%	18,56%	13,60%	10,22%	11,12%	18,67%	18,89%	17,53%	13,12%	9,66%	10,47%
rbg443	443	23,81%	23,93%	23,81%	17,75%	13,64%	15,49%	23,84%	23,87%	23,79%	17,50%	12,03%	15,29%	23,69%	23,87%	23,75%	17,26%	11,83%	15,09%
calidad media error		**25,29%**	**25,30%**	**25,29%**	**13,96%**	**11,28%**	**11,97%**	**25,12%**	**25,20%**	**25,08%**	**13,62%**	**10,03%**	**11,37%**	**24,98%**	**25,13%**	**24,57%**	**13,33%**	**9,68%**	**11,01%**

Tabla 4.12. *% Error de las simulaciones realizadas para cada algoritmo.*

		m=2·n; iter=500; α=0			m=2·n; iter=1000; α=0			m=2·n; iter=5000; α=0			m=2·n; iter=500; α=1			m=2·n; iter=1000; α=1			m=2·n; iter=5000; α=1		
Fichero	**n**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**
br17	17	39,00	39,00	39,00	39,00	39,00	39,00	39,00	39,00	39,00	39,03	39,00	39,00	39,00	39,00	39,00	39,00	39,00	39,00
ft53	53	9456,80	9428,27	9375,53	8264,87	7726,93	7717,53	9368,40	9333,00	9280,83	8283,33	7692,57	7611,50	9081,07	9140,87	9200,80	8293,97	7563,87	7595,03
ft70	70	52780,73	52770,50	52783,03	40557,50	39834,50	39703,93	52518,43	52460,67	52384,53	40564,30	39880,77	39761,23	51903,47	51964,70	51903,47	40642,57	39915,43	39636,93
ftv33	34	1506,87	1506,63	1511,17	1375,30	1348,10	1333,10	1488,23	1488,63	1488,90	1372,93	1346,63	1328,60	1444,97	1444,17	1340,20	1376,70	1359,43	1340,20
ftv35	36	1744,17	1742,63	1734,00	1564,10	1517,13	1503,40	1719,10	1715,77	1709,70	1550,47	1511,23	1498,27	1660,13	1665,27	1668,70	1555,93	1516,60	1496,13
ftv38	39	1848,93	1849,27	2947,60	1638,20	1600,20	1573,70	1832,20	1828,97	1812,87	1635,07	1586,93	1577,27	1775,80	1771,87	1775,40	1625,77	1589,20	1628,20
ftv44	45	2076,83	2064,60	2063,47	1753,40	1696,30	1674,07	2049,93	2043,57	2040,53	1753,70	1687,57	1682,43	1983,03	1968,67	1979,67	1745,87	1670,40	1663,40
ftv47	48	2332,97	2334,97	2342,60	1951,00	1926,27	1910,40	2299,40	2286,67	2307,23	1938,93	1926,60	1908,67	2232,43	2234,90	2243,20	1935,37	1925,63	1908,70
ftv55	48	2112,13	2117,67	2121,17	1749,60	1706,30	1692,50	2085,33	2061,40	2065,27	1748,60	1714,80	1694,00	2003,13	2020,00	2011,80	1755,63	1702,23	1689,60
ftv64	65	2619,47	2609,63	2633,20	2008,87	1972,97	1947,70	2558,30	2530,03	2574,33	2015,70	1973,80	1946,43	2481,07	2485,53	2483,07	2002,43	1958,17	1944,60
ftv70	71	2831,47	2813,47	2824,87	2210,40	2122,27	2112,03	2703,00	2792,63	2796,53	2200,10	2120,03	2099,07	2694,33	2718,60	2706,53	2188,93	2092,80	2096,13
ftv170	171	5203,10	5171,40	5186,70	3322,10	3210,80	3076,00	5131,87	5169,10	5115,20	3313,37	3098,90	3096,13	5093,50	5137,43	5132,80	3299,47	3056,77	3082,10
p43	43	5648,93	5648,30	5647,77	5650,83	5642,23	5637,47	5646,60	5645,87	5645,57	5650,43	5642,33	5639,10	5640,67	5641,67	5639,77	5646,03	5639,87	5635,73
ry48p	48	18115,67	18113,10	18138,57	15482,53	15277,03	15120,37	17783,63	17858,93	17897,37	15430,77	15258,70	15003,80	17501,37	17503,57	17491,77	15466,57	15199,47	14974,23
kro124p	100	56098,80	55818,10	55895,03	40820,67	39419,67	39329,50	55604,57	55664,57	55667,80	40686,63	39393,10	55740,47	54479,73	54438,60	54391,03	40544,30	39446,00	39153,47
		m=n; iter=500; α=0			**m=n; iter=1000; α=0**			**m=n; iter=5000; α=0**			**m=n; iter=500; α=1**			**m=n; iter=1000; α=1**			**m=n; iter=5000; α=1**		
Fichero	**n**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**
br17	17	39,00	39,00	39,00	39,07	39,00	39,00	39,00	39,00	39,00	39,07	39,63	39,00	39,00	39,00	39,00	39,23	39,00	39,00
ft53	53	9540,97	9545,80	9496,50	8342,87	7802,27	7785,20	9423,17	9477,37	9346,43	8315,47	7849,43	7710,90	9170,53	9187,83	9200,80	8275,90	7669,60	7599,13
ft70	70	53145,20	53079,37	53126,53	40670,67	40537,20	39892,53	52777,03	52748,37	52756,07	40805,30	40432,37	39929,43	52084,20	52262,50	52211,93	40660,50	40372,13	39763,40
ftv33	34	1526,60	1519,80	1526,23	1390,60	1386,90	1350,60	1507,87	1507,63	1503,20	1381,67	1374,80	1337,67	1469,13	1468,87	1455,70	1380,20	1373,03	1341,23
ftv35	36	1769,80	1754,63	1770,03	1570,57	1553,77	1511,07	1732,07	1732,30	1747,70	1578,83	1543,63	1504,13	1679,10	1691,37	1690,53	1571,77	1546,57	1506,47
ftv38	39	1872,70	1863,27	1877,20	1636,57	1613,03	1582,40	1857,97	1864,57	1860,07	1634,57	1604,77	1582,20	1797,23	1800,97	1802,87	1636,37	1618,00	1581,90
ftv44	45	2096,87	2084,30	2099,40	1761,87	1725,93	1693,20	2070,83	2060,17	2060,07	1770,97	1713,43	1686,70	2007,07	1991,10	1997,27	1767,03	1703,90	1673,97
ftv47	48	2364,57	2355,57	2384,00	1953,40	1953,97	1925,20	2328,60	2333,07	2326,70	1960,63	1945,53	1914,23	2252,87	2275,50	2259,00	1939,43	1949,70	1920,70
ftv55	48	2132,40	2128,20	2148,23	1767,30	1735,50	1699,40	2110,73	2102,43	2094,87	1762,40	1739,03	1700,50	2043,63	2032,53	2042,33	1763,57	1725,87	1702,63
ftv64	65	2641,37	2633,37	2644,47	1983,23	1983,23	1963,50	2579,03	2580,30	2610,60	2016,80	2002,77	1951,97	2542,97	2516,60	2519,60	2015,70	1984,60	1966,37
ftv70	71	2893,83	2862,33	2873,20	2214,60	2152,53	2123,93	2819,30	2826,73	2837,30	2209,27	2151,37	2114,60	2747,77	2766,27	2740,73	2212,27	2130,93	2119,50
ftv170	171	5267,83	5233,30	5248,60	3348,30	3131,40	3108,47	5185,93	5212,40	5181,57	3336,53	3099,83	3106,83	5063,63	5048,33	5021,03	3331,43	3057,07	3067,60
p43	43	5650,00	5649,97	5648,53	5653,03	5647,43	5641,70	5646,73	5647,33	5648,33	5651,60	5648,83	5639,37	5642,37	5643,07	5642,53	5649,20	5644,50	5639,40
ry48p	48	18214,47	18163,73	18284,70	15566,40	15470,60	15124,00	16254,67	18022,27	18188,47	15560,20	15583,23	15076,00	17610,93	17691,23	17662,30	15560,10	15446,97	15045,67
kro124p	100	56785,13	56482,87	56655,53	41092,23	39982,63	39866,97	56386,03	55829,63	56047,93	41037,60	40057,80	39657,40	54891,83	54908,37	54634,33	40808,73	39990,50	39513,97
		m=50; iter=5000; α=0			**m=50; iter=5000; α=1**			**m=50; iter=10000; α=0**			**m=50; iter=10000; α=1**			**m=50; iter=30000; α=0**			**m=50; iter=30000; α=1**		
Fichero	**n**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**	**AS**	**EAS**	**Asrank**
rbg323	323	1885,70	1879,83	1886,33	1482,00	1478,73	1463,70	1876,07	1874,93	1880,43	1475,80	1453,37	1451,37	1870,90	1872,67	1868,77	1470,03	1449,20	1449,30
rbg358	358	1633,50	1629,93	1631,40	1348,47	1282,00	1302,97	1627,27	1629,13	1625,83	1340,40	1279,73	1298,43	1625,33	1627,67	1614,50	1339,17	1274,03	1291,70
rbg403	403	3038,33	3048,83	3039,93	2860,13	2796,20	2809,13	3035,53	3044,97	3026,67	2852,87	2745,53	2773,37	3030,90	3039,13	2988,90	2837,23	2728,50	2753,20
rbg443	443	3570,23	3575,57	3570,23	3306,90	3149,77	3218,53	3571,37	3572,90	3569,23	3296,80	3091,97	3211,00	3564,30	3572,87	3567,00	3287,47	3084,87	3203,33

Tabla 4.13. *Recogida de la media de las 30 ejecuciones realizadas para cada algoritmo.*

4.3. ESTUDIO ESTADÍSTICO.

Tras la realización de las simulaciones (apartado 4.2), posterior recogida de datos, se procede a la implementación del estudio estadístico de los mismos. Para ello, se realizada un estudio mediante el uso de test no paramétrico. El motivo por el cual se emplea un test no paramétrico frente al paramétrico se explica a continuación.

Sin embargo, antes de adentrarnos en esta materia, es necesario explicar algunos conceptos relacionados con la misma.

4.3.1. Introducción.

En los últimos años, consecuencia de las investigaciones y avanzases realizados en el campo de las técnicas metaheurísticas y, más concretamente, respecto a los algoritmos evolutivos, se ha desatado un creciente interés por el análisis de experimentos en este ámbito.

El objeto último del estudio de los algoritmos evolutivos es encontrar aquel que nos ofrezca los mejores resultados para unas determinadas condiciones y valores del problema a resolver. Por ello, es necesario realizar un estudio estadístico que mediante la comparación de los algoritmos nos proporcione cuál de todos los posibles es el más idóneo.

Por lo que el uso de las técnicas estadísticas cada vez es más frecuente. Su implementación se realiza a partir de los resultados empíricos que obtenemos de los algoritmos que se han simulado. Aunque algunas de estas técnicas presentan limitaciones en cuanto al cumplimiento de una serie de condiciones, es cierto que proporcionan información clara y precisa la cual se empleará para extraer conclusiones.

4.3.2. Test paramétricos y no paramétricos.

Anteriormente [2], la mayoría de los trabajos que han sido publicados en revistas científicas, el estudio estadístico que realizaban no era otro que la comparación de un conjunto de resultados basados en el valor medio de las ejecuciones de cada simulación y la desviación de la media respecto del valor óptimo. Sin embargo, hoy en día es necesaria la utilización de test estadísticos ya sean paramétricos (ANOVA, t-test,..) o no paramétricos (Friedman, χ^2, Holm).

Si se emplean test paramétricos la condición indispensable es que se cumplan las siguientes hipótesis. Si alguna de ellas es violada, esto hace que el análisis estadístico realizado pierda credibilidad y los resultados no sean válidos. Por lo tanto, un test paramétrico debe cumplir lo siguiente:

- *Independencia*: en estadística, se dice que dos sucesos son independientes si al ocurrir uno de ellos no modifica la probabilidad de ocurrencia del otro.
- *Normalidad:* si una observación cumple que su comportamiento sigue una distribución normal o de Gauss con una determinada media μ y varianza σ, se dice que es normal. Para comprobar la normalidad de las observaciones se realiza un test sobre la muestra.
- *Homocedasticidad de la varianza*: las varianzas de las distintas pruebas realizadas deben ser homogéneas.

Basándonos en los estudios realizados por la Universidad de Granada [2]. Se demuestra que la hipótesis de normalidad de las observaciones no se cumple (condición suficiente para invalidar el uso de test paramétricos), por lo que es necesario emplear test no paramétricos para realizar el estudio estadístico de los resultados obtenidos. En el presente documento, se ha empleado el test de Friedman y el test de Holm.

4.3.2.1. Test de Friedman.

Se ha empleado el test de Friedman, propuesto por Milton Friedman [3], el cual se considera un equivalente no paramétrico al test paramétrico ANOVA. Se calcula el orden de los resultados observados por algoritmo, es decir, es decir, r_j para el algoritmo j con k algoritmos. Al mejor, se le asigna el orden 1 , y al peor el orden k.

Para calcular los órdenes de cada algoritmo, se determinan los rankings medios de los algoritmos analizados sobre los n-problemas mediante la expresión (4.13):

$$R_j = \frac{\sum_{i=1}^{n} r_{ij}}{n} \qquad (4.13)$$

Siendo r_{ij} el ranking del algoritmo j en el problema i.

Si el comportamiento de los algoritmos es similar, esto quiere decir, que la hipótesis nula se cumple, sus rankings también serán similares. Basándonos en esta hipótesis se formula el estadístico de Friedman, F_F, correspondiente con la expresión (4.14), a partir de una distribución χ_F^2 con k-1 grados de libertad:

$$F_F = \frac{12n}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \qquad (4.14)$$

Una vez realizado el test de Friedman y comprobando que la hipótesis nula es rechazada, es decir, el comportamiento de unos algoritmos y otros difiere, es necesario realizar otro test a posteriori para detectar las diferencias entre unos algoritmos y otros si las hubiese. Uno de los más recomendables entre los posibles test a utilizar es el Test de Holm.

4.3.2.2. Test de Holm.

Sture Holm, (Holm, 1979), planteó el siguiente test con el fin de mejorar el test de Bonferroni. Este test consiste en probar secuencialmente las hipótesis planteadas según su orden de relevancia. El método compara cada p-valor, p_i ordenado de menor a mayor con la siguiente expresión (4.15):

$$\frac{\alpha}{(k-i)} \qquad (4.15)$$

Comenzando desde el p más significante, es decir, el menor. Si el p-valor $p_1 < \frac{\alpha}{(k-1)}$, entonces, se rechaza la hipótesis nula, H_0, y esto, nos permite comparar el siguiente p-valor, en este caso sería p_2. Hasta que la H_0 no sea aceptada, no se termina de realizar las comparaciones.

En la expresión (4.15), α es el nivel de precisión cuyo valor se encuentra entre 0 y 1 (en el presente documento se emplea un α=0.05) y k es el valor obtenido en el Test de Friedman.

4.3.3. Resultados obtenidos del Estudio Estadístico.

Una vez explicados los test que vamos a emplear para el estudio estadístico, se procede al mismo, para ello será necesario la tabla 4.13 (tabla de medias) cuyos resultados será necesario calcular la inversa puesto que la función objetivo del problema es minimizar la distancia total del recorrido.

Los algoritmos que se van a comparar son aquellos con los que se obtienen mejores resultados para los problemas pequeños, es decir, los 15 primeros de las tablas. Como ya se explicó, se han implementados 36 algoritmos para cada problema. La 12 de los cuales, se implementan con el parámetro α=0, siendo éste la importancia relativa del rastro de feromona. Cuando α=0, los resultados distan del óptimo, esto se debe a que las ciudades más cercanas son más probables de ser seleccionadas por lo que daría lugar resultados que se obtienen con el algoritmo estocástico de Greedy y se perdería la aleatoriedad de elección de las ciudades a visitar. Por ello, se han tomado aquellos algoritmos con α=1 para realizar el test de Friedman. Para cada método ACO se realiza el Test de Friedman y serán analizados individualmente. Posteriormente, se realiza el Test de Holm con el que se extraerán las conclusiones finales.

Para un nivel de precisión del 0.05, los resultados del Test de Friedman para el **método AS** son (ver tabla 4.14):

Algoritmo ACO: nº hormigas-nº iteraciones- α- Método ACO	Ranking medio
2n-5000-1-AS	1.833333
2n-1000-1-AS	2.233333
2n-500-1-AS	2.9
n-5000-1-AS	4.033333
n-1000-1-AS	4.899999
n-500-1-AS	5.100000

Tabla 4.14.*Test de Friedman para los algoritmos AS.*

Con los datos de la tabla 4.14, puede verse perfectamente como el algoritmo 2n-5000-1-AS es el que obtiene el mejor puesto. Tras él, el 2n-1000-1-AS y posteriormente, el AS 2n-500-1-AS. Es decir, los mejores resultados con el método AS, son obtenidos por los algoritmos cuyo número de hormigas es el doble al número de ciudades a visitar, lo cual es coherente puesto que cuantas más veces se recorran las ciudades en cada iteración la probabilidad de encontrar posibles rutas con menor longitud será mayor, y el refuerzo del rastro de feromona de las rutas también será mayor.

A continuación, se va a realizar el Test de Holm para un nivel de precisión del 0.05. Este test nos mostrará si el algoritmo 2n-5000-1-AS enfrentado con los otros 5 restantes es significativamente mejor o por el contrario no lo es (ver tabla 4.15):

Algoritmo ACO	p-valor	Holm
n-500-1-AS	1.736371e-6	0.02
n-1000-1-AS	7.151117e-6	0.025
n-5000-1-AS	0.001279	0.033333
2n-500-1-AS	0.118420	0.05
2n-1000-1-AS	0.558185	0.1

Tabla 4.15.*Test de Holm para los algoritmos AS.*

Basándonos en los resultados que recoge la tabla 4.15, los valores de Holm para los algoritmos n-500-1-AS, n-1000-1-AS, n-5000-1-AS son menores que sus p-valores,

respectivamente. Esto quiere decir que estos algoritmos son significativamente peores que 2n-5000-1-AS. Sin embargo, para los algoritmos 2n-500-1-AS y 2n-1000-1-AS no puede afirmarse que sean peores que el 2n-5000-1-AS, puesto que sus p-valores son mayores que el parámetro de Holm.

Para un nivel de precisión del 0.05, los resultados del Test de Friedman para el **método EAS** son (ver tabla 4.16):

Algoritmo ACO: nº hormigas- nº iteraciones- α- Método ACO	Ranking medio
2n-5000-1-EAS	1.666666
2n-1000-1-EAS	2.2
2n-500-1-EAS	2.733333
n-5000-1-EAS	4.066666
n-1000-1-EAS	5.2
n-500-1-EAS	5.133333

Tabla 4.16.*Test de Friedman para los algoritmos EAS.*

Al igual que para los algoritmos del método AS, el mejor puesto lo ocupa 2n-5000-1-EAS cuyo parámetro de Friedman es menor y más próximo a 1. Le siguen 2n-1000-1-EAS y 2n-500-1-EAS, respectivamente. El salto se produce cuando el número de hormigas lo igualamos al número de ciudades.

Ahora, mostramos los valores del Test de Holm con los que se concluirá si el algoritmo 2n-5000-1-EAS es significativamente mejor o no que el resto de algoritmos (ver tabla 4.17):

Algoritmo ACO	p-valor	Holm
n-1000-1-EAS	2.312666e-7	0.01
n-500-1-EAS	3.881479e-7	0.0125
n-5000-1-EAS	4.426769e-4	0.016666
2n-500-1-EAS	0.118419	0.025
2n-1000-1-EAS	0.434967	0.05

Tabla 4.17.*Test de Holm para los algoritmos EAS.*

Para los algoritmos n-1000-1-EAS, n-500-1-EAS y n-5000-1-EAS se rechaza la hipótesis nula puesto que los p-valores son menores que los valores de Holm, esto es, que el algoritmo 2n-5000-1-EAS es significativamente mejor que los anteriores. Mientras que para los algoritmos 2n-500-1-EAS y 2n-1000-1-EAS no podemos rechazar la hipótesis nula. Ocurre lo mismo que en el caso anterior par el método AS.

Para un nivel de precisión del 0.05, los resultados del Test de Friedman para el **método ASrank** son (ver tabla 4.18):

Algoritmo ACO: nº hormigas-nº iteraciones- α- Método ACO	Ranking medio
2n-5000-1-ASrank	1.9
2n-1000-1-ASrank	2.766666
2n-500-1-ASrank	2.766666
n-5000-1-ASrank	3.900000
n-1000-1-ASrank	4.233333
n-500-1-ASrank	5.433333

Tabla 4.18.*Test de Friedman para los algoritmos ASrank.*

De nuevo, la mejor posición en el ranking la ocupa el algoritmo 2n-5000-1-Asrank cuyo Ranking medio es el menor seguido del 2n-1000-1-Asrank y 2n-500-1-Asrank que ambos presentan el mismo valor del ranking medio de Friedman. Cuando se varía el número de hormigas es entonces cuando se produce el salto en el ranking. Con el doble número de hormigas se consiguen las mejores posiciones.

Una vez analizado esto, el Test de Holm determinará si el algoritmo 2n-5000-1-Asrank es significativamente o no mejor que el resto de algoritmos evaluados (ver tabla 4.19):

Algoritmo ACO	p-valor	Holm
n-500-1-ASrank	2.312666e-7	0.01
n-1000-1-ASrank	6.362991e-4	0.0125
n-5000-1-ASrank	0.003415	0.016666
2n-500-1-ASrank	0.204559	0.025
2n-1000-1-ASrank	0.204559	0.05

Tabla 4.19.*Test de Holm para los algoritmos ASrank.*

En la tabla 4.19 vemos que para los algoritmos n-500-1-ASrank, n-1000-1-ASrank y n-5000-1-ASrank los p-valores son menores que los valores de Holm asociados a cada algoritmo. Por lo que, al igual que en los casos anteriores estudiados, se rechaza la hipótesis nula para estos algoritmos y, con ello, se puede garantizar para un nivel de precisión del 0.05 que el 2n-5000-1-Asrank es, significativamente, mejor que los otros tres. Sin embargo, para los algoritmos 2n-500-1-ASrank y 2n-1000-1-Asrank, los p-valores son mayores que los valores de Holm por lo que se acepta la Hipótesis nula y no podemos decir que 2n-5000-1-Asrank sea el mejor.

Por último, vamos a comparar los tres mejores algoritmos obtenidos en el estudio estadístico realizado anteriormente y el mejor algoritmo PSO (en inglés, *Particle Swarm Optimization*) obtenido como resultado de la experimentación realizada en el Trabajo Fin de Grado de una alumna de la Escuela de Ingenierías Industriales de la Universidad de Valladolid. Para ello, se realiza el Test de Friedman para saber cuál de los 4 algoritmos es el que mejor posición ocupa en el ranking como aparece recogido en la tabla 4.20:

Algoritmo	Ranking medio
2n-5000-1-AS	2.900000
2n-5000-1-EAS	1.766666
2n-5000-1-ASrank	1.433333
PSO	3.899999

Tabla 4.20.*Test de Friedman para los algoritmos ACO y PSO.*

Según los resultados recogidos en la tabla 4.20, la mejor posición la ocupa el algoritmo ACO 2n-500-1-Asrank, seguido de 2n-1000-1-EAS, 2n-5000-1-AS y por último PSO. ES decir, en un primer estudio, los algoritmos evolutivos basados en colonia de hormigas son mejores que el algoritmo PSO. Y dentro de los algoritmos ACO, el mejor es el ASrank, seguido del EAS.

Ahora, se va a realizar el test de Holm (ver tabla 4.21) para verificar si efectivamente el algoritmo 2n-5000-1-ASrank es, significativamente, mejor que el resto de algoritmos para un nivel de precisión del 0.05.

Algoritmo	p-valor	Holm
PSO	1.671510e-7	0.016666
2n-5000-1-AS	0.001863	0.025
2n-5000-1-EAS	0.479500	0.05

Tabla 4.21.*Test de Holm para los algoritmos ACO y PSO.*

Como se muestra en la tabla 4.21, sólo se rechaza la Hipótesis nula para el PSO y 2n-5000-1-AS puesto que sus p-valores son menores que sus valores de Holm. Esto es, que para un nivel de precisión de 0.05, el algoritmo 2n-5000-1-ASrank es significativamente mejor que los algoritmos PSO y 2n-5000-1-AS. Mientras que para el algoritmo elitista ACO, 2n-5000-1-EAS, el p-valor es mayor que el valor de Holm por lo que no podemos garantizar que 2n-5000-1-ASrank sea significativamente mejor que 2n-5000-1-EAS.

4.4. CONVERGENCIA DE LOS MÉTODOS ACO.

A continuación, se presenta una serie de gráficas (figuras 4.13, 4.14 y 4.15) en las que se representa el número de iteraciones frente a los resultados obtenidos en cada iteración para cada método ACO: AS, EAS y Asrank. Así, se comprueba, de forma visual, cómo se comporta cada método, cuál de los tres converge más rápido y cuál es el que nos proporciona mejores resultados. En definitiva, lo que se busca es una comparación entre los métodos ACO, y si los resultados coinciden con lo que ya se anunció en el punto 4.1 acerca del comportamiento de los distintos algoritmos ACO y con las conclusiones extraídas en el apartado 4.3. Los valores que se han otorgado, en este caso, para los problemas son aquellos que Dorigo presenta como condiciones favorables de operación. En la tabla 4.22, se muestran:

Método ACO	Nº Iteraciones	m	α	β	ρ	e	w
AS	500	n	1	3	0.5	-	-
EAS	500	n	1	3	0.5	n	-
ASrank	500	n	1	3	0.1	-	6

Tabla 4.22.*Valores de los parámetros para las representaciones gráficas.*

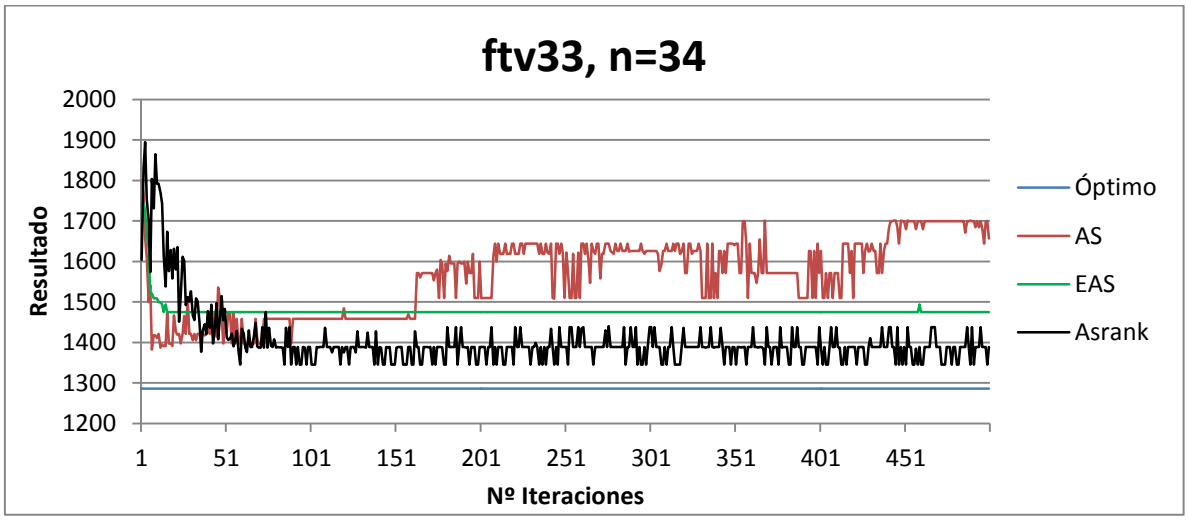


Figura 4.13. *Convergencia de los métodos ACO para el problema ftv33.*

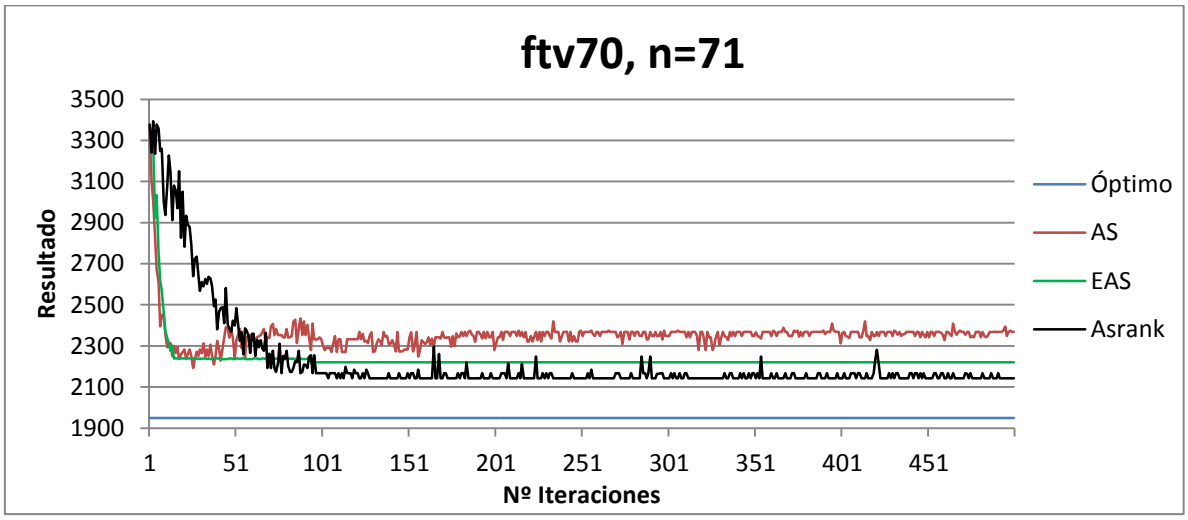


Figura 4.14. *Convergencia de los métodos ACO para el problema ftv70.*

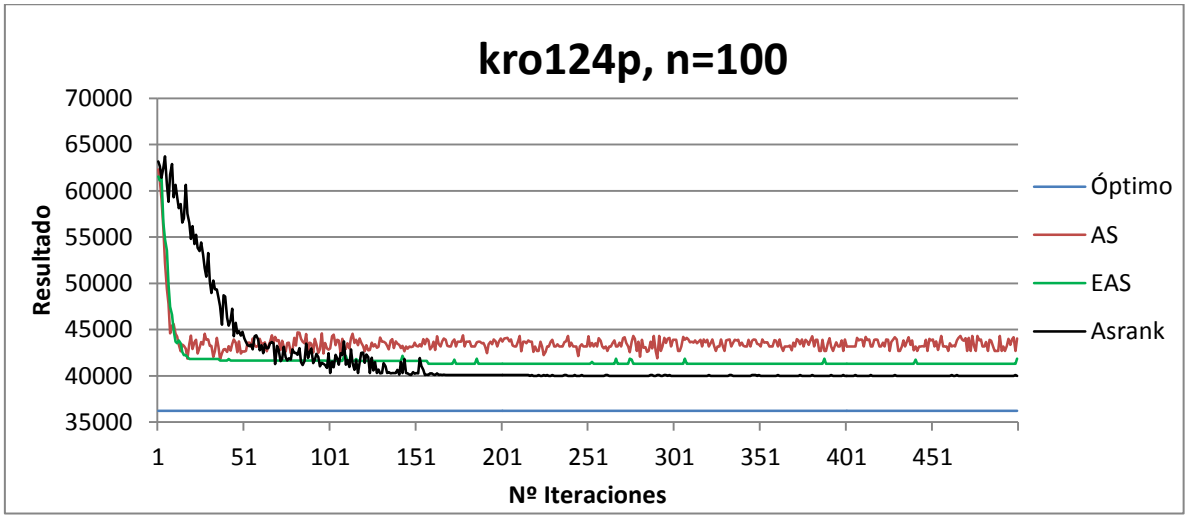


Figura 4.15. *Convergencia de los métodos ACO para el problema kro124p.*

En las tres representaciones gráficas (ver figuras 4.13, 4.14 y 4.15), se comprueba que el método que proporciona mejores resultados y más próximos al óptimo es el ASrank de acuerdo con lo que ya se dedujo en el apartado 4.3. A éste le sigue el método EAS no difiriendo mucho del anterior en cuanto a los resultados obtenidos, lo cual resulta coherente puesto que ambos métodos son elitistas y refuerzan las rutas mejores de cada iteración haciéndolas participe en la actualización del rastro de feromona. Por último, el método AS es el que más rápido converge pero posteriormente, cuando aumenta el número de iteraciones sus resultados oscilan y vuelve a proporcionar resultados no próximos al óptimo. En definitiva, el método de rangos ASrank aunque sea, de los tres el que converge más tarde, es aquel que proporciona mejores resultados.

Capítulo 5.

Conclusiones y líneas futuras.

En este capítulo, se presentan las conclusiones extraídas a partir de la experimentación realizada en el capítulo 4, así como del estudio estadístico posterior. Además, se realizará una valoración conjunta del desarrollo completo del presente documento que se ha elaborado en los capítulos anteriores. Por último, de forma general, se expondrán las líneas futuras abiertas para investigaciones o posteriores Trabajos Fin de Grado relacionadas con los algoritmos ACO y su implementación para la resolución del TSP y sus variantes, como en este caso sucede con el problema ATSP.

5.1. CONCLUSIONES.

Basándonos en los resultados obtenidos de las simulaciones de los problemas pequeños y, principalmente, los que nos proporciona el estudio estadístico, se comprueba que los mejores resultados se obtienen con los algoritmos 2n-5000-1 para los tres métodos ACO. Sin embargo, para los problemas grandes como el número de hormigas es fijo (m=50), los mejores resultados corresponden con los algoritmos 50-30000-1, también para los tres métodos (ver tablas 4.12 y 4.13). Dicho esto, en cuanto a la comparación entre los distintos métodos ACO (AS, EAS y ASrank), el estudio estadístico (Test de Friedman y Test de Holm) nos muestra que para un nivel de precisión del 0.05, los resultados obtenidos son mejores cuando el algoritmo que se implementa es el ASrank, seguido del método elitista EAS y, por último, la versión original y pionera de los ACO, el método AS. Esto, ya se predijo, al inicio del capítulo 4, cuando se explicaron las distintas versiones existentes que hay para la actualización del rastro de feromona.

Resulta coherente y lógico que el método ASrank y EAS, ambos elitistas a diferencia del AS, se centran en reforzar el rastro de feromona con la mejor ruta encontrada hasta el momento en el algoritmo, lo cual hace que aquellas rutas que contengan o pasen por los arcos pertenecientes a la mejor ruta tengan depositada una mayor cantidad de feromona, por lo que serán más atrayentes por el resto de las hormigas. Además, el método ASrank resulta que es mejor que el EAS por su depósito de feromonas no solo por parte de la mejor ruta, sino también por el rango de las w-1 mejores hormigas. Aunque, los resultados del ASrank no despuntan bruscamente respecto al método EAS, el estudio estadístico sí que le otorgan como el mejor.

Basándonos en los resultados gráficos obtenidos del apartado 4.4 del capítulo 4, el método EAS presenta una fase de exploración muy corta, ya que rápidamente pasa a la fase de explotación donde se encuentran las mejores soluciones, es decir, la convergencia de este método es rápida y la calidad de las soluciones mejor que para la versión AS. Por el contrario, se produce un estancamiento de la búsqueda de soluciones cuando aumenta el número de iteraciones. El método ASrank presenta un comportamiento similar al anterior, aunque la fase de exploración es más amplia lo que conlleva una convergencia más lenta que el EAS aunque la calidad de las soluciones obtenidas es mejor.

En definitiva, y realizando una comparativa de los tres algoritmos ACO (2n-5000-1-AS, 2n-5000-1-EAS y 2n-5000-1-ASrank) y el algoritmo evolutivo PSO, aquél que proporciona los mejores resultados es el algoritmo 2n-5000-1-ASrank, seguido del 2n-5000-EAS, 2n-5000-1-AS y, por último, el PSO, según el Test de Friedman (ver tabla 4.21).

5.2. LÍNEAS FUTURAS.

Como línea futura abierta a la investigación y posteriores Trabajos Fin de Grado, se plantea y propone el desarrollo e implementación de la variante ACO MAX-MIN para la resolución del TSP.

Tomas Stützle junto con Holger Hoos (Stützle y Hoos, 1997) son los responsables que introdujeron, en 1997, una nueva variante de los algoritmos ACO: el MAX-MIN Ant System (MMAS). La diferencia de esta versión ACO respecto de las anteriores (ASrank, EAS y AS) radica en las cuatro modificaciones principales que se realizan: primero, solamente la mejor solución encontrada en el algoritmo hasta el momento o la mejor solución de la iteración es la responsable del depósito de feromonas en el rastro. En la expresión (5.1) y (5.2), se refleja lo explicado anteriormente.

$$\tau_{ij}^{t+1} \leftarrow (1-\rho)\tau_{ij}^{t} + \Delta\tau_{ij}^{best} \qquad (5.1)$$

Donde

$$\Delta\tau_{ij}^{best} = 1/C^{best} \qquad (5.2)$$

- Para el caso de la mejor solución de la iteración será: $\Delta\tau_{ij}^{best} = 1/C^{tb}$, donde C^{ib} es la longitud de la mejor ruta de la iteración t.
- Para la mejor solución encontrada hasta el momento: $\Delta\tau_{ij}^{best} = 1/C^{bs}$

Para evitar el estancamiento que podría producirse con éste método, se introduce una segunda modificación: a los valores del rastro de feromona que pueda tener la matiz de feromonas, se les añade un límite superior e inferior del rastro $[\tau_{min}, \tau_{max}]$. La tercera, corresponde con la inicialización de los rastros del límite superior de feromona, τ_{max}. Y por último, la cuarta modificación, también alusiva al rastro de feromonas, expresa que las feromonas han de reinicializarse trascurrido un numero de iteraciones tras comprobar que no ha habido mejora en la solución.

Bibliografía.

Aardal, K., Van Hoesel, S., Koster, A., Mannino, C y Sassano, A. (2007). Models and solutions techniques for frequency assignment problems. Annals of Operations Research. Pág. 79-129.

Bais, J., Gen- Ke Yang, Yu-Wang Chen, Li- Sheng Hu, Chang-Chu Pan, (2012). A model induced max-min ant colony optimization for asymmetric traveling salesman problema. Revista Elsevie, nº 13. Pág. 1365-1375.

Blum, C. y Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys, nº 35. Pág. 268-308.

Brito, J., Campos, C., y García, F. (2004). Metaheurísticas: una revisión actualizada. Grupo de Computación Inteligente. Departamento de Estadística, Investigación Operatica y computación. Universidad de La Laguna. Pág. 9-24.

Bullnheimer, B., Hartl, R. F. y Strauss, C. (1997). A New Rank Based Version of the Ant System- A computacional Study. Technical report, Institute of Management Science, University of Vienna. Pág. 1-14.

Bullnheimer, B., Hartl, R. F. y Strauss, C. (1999). A New Rank Based Version of the Ant System- A computacional Study. Central European Journal for Operations Research and Economics, nº. 7. Pág. 25-38.

Calviño, A. (2011). Cooperación en los problemas del viajante (TSP) y de rutas de vehículo (VRP): una panorámica. Máster universitario en Técnicas Estadísticas. Universidad de Vigo, Coruña y Santiago de Compostela. Pág. 3-22.

Cerný, V. (1985) A thermodynamical approach to the traveling salesman problem. Journal of Optimization Theory and Applications, nº. 45. Pág. 41-51.

Colorni, A., Dorigo, M. y Maniezzo, V. (1991). Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle. Paris. Elsevier Publishing. Pág. 134-142.

Cook, W. J. (2012). In Purssuit of Traveling Salesman: Mathematics at the Limits of Computation. University Press. Chapter 1 y 2. Pág. 2-360.

Dantzig, G., Fulkerson, R., Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. Operations Research, nº. 2. Pág. 393-410.

Deneubourg, J. L., Aron, s., Goss, S. y Pasteels, J.M. (1990). The self-organizing exploratory pattern of the Argentine ant. Journal of Insect Behavior, nº 3. Pág. 159-168.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms, PhD thesis. Dipartamento Electtronica e Informazione, Politecnico di Milano, Italy.

Dorigo, M. y Stützle, T. (2004). Ant Colony Optimization. A Bradford Book. The MIT Press. Cambridge, Massachusetts.

Flood, M.M. (1984). The Princeton Mathematics Community in the 1930s, Transcript number 11 (PMC11). Princeton University.

Glover, F. (1977). Heuristics for integer programming using surrogate constraints. Decision Sciences, nº 8. Pág. 156-166.

González, J.L. y Ríos, Z. (1999). Investigación de operaciones en acción: aplicación del TSP en problemas de manufactura y logística. Revista ingenierías, vol. II nº 4. Pág. 21-22.

Gunts, M., Middendorf, M. y Shmeck, H. (2001). An Ant Colony Optimization Approach to Dynamic TSP. Institute AIFB University of Karlsruhe, Germany.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics, nº 6, Pág. 65–70.

Islam, M., Sadid, W.H., Ar Rashid, M. y Kabir, J. (2006). An implementation of ACO system for solving NP-complete problem; TSP. 4th International Conference on Electrical and Computer Engineering. ICECE. Dhaka, Bangladesh. Pág. 304-307.

Laporte, G. (1992). The Travelling Salesman Problem: An Overview of Exact and Approximate Algorithms, European Journal of Operational Research, nº 59, pág. 231–247.

Luna, F. (2008). Metaheurísticas avanzadas para problemas reales en redes de telecomunicaciones. Tesis Doctoral. Universidad de Málaga. Pág. 27-32.

Martin, O., Otto, S. W. y Felten, E. W. (1991). Large-step markov chains for the TSP incorporing local search heuristics. Journal, Operations Research Letters. nº 11. Pág. 219-224.

Saka, M.P., Dogan, E. y Aydogdu, I. (2013). Analysis of Swarm Intelligence –Based Algorithms for Constrained Optimization. Revista Elsevier, nº 2. Pág. 24-48.

Stützle, T. y Hoos, H. H. (1997). The MAX-MIN Ant System and local search for the travelling salesman problema. Proceedings of the 1997 IEEE International Conference on Evolutionary Computation. Pág. 309-314.

Webs consultadas:

[1] Reinelt, G. (1995). TSPLIB Library of Traveling Salesman Problem. Universität Heidelberg. Recuperado el 12 de julio de 2015 de: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[2] García, S., Fernández, A., Luengo, J. y Herrera, F. (2010). Advanced nonparametric test for multiple comparisons in the design of experiments in computational intellingence and data mining: Experimental analysis of power. Elsevier, nº 180. Pág. 2044-2064. Recuperado el 25 de agosto de 2015 de http://sci2s.ugr.es/sicidm/

[3] Derrac, J., García, S., Hui, S., Suganthan., P. N. y Herrera, F. (2014). Analyzing convergence perfoance of evolutionary algorithms: A statiscal approach. Elsevier, nº 289. Pág. 41-58. Recuperado el 30 de agosto de 2015 de http://sci2s.ugr.es/sicidm/