Universitat Oberta de Catalunya

uoc.edu

Optimización de rutas de distribución de bicicletas entre las estaciones de BiciMAD aplicando el algoritmo de la colonia de hormigas

Ana Hidalgo Boix
Grado de Ingeniería Informática
Inteligencia Artificial

Profesor colaborador: David Isern Alarcón
Profesor responsable: Carles Ventura Royo

Junio 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Optimización de rutas de distribución de bicicletas entre las estaciones de BiciMAD aplicando el algoritmo de la colonia de hormigas*
Nombre del autor:	*Ana Hidalgo Boix*
Nombre del consultor/a:	*David Isern Alarcón*
Nombre del PRA:	*Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	06/2020
Titulación::	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Inteligencia Artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Optimización, Ant Colony Optimization, grafos*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

La finalidad del presente trabajo es optimizar la ruta de reposición de bicicletas entre las estaciones del servicio BiciMAD. Para ello se elige implementar el algoritmo de optimización de la colonia de hormigas, ya que este algoritmo ha demostrado su eficacia para la resolución de problemas similares de optimización de rutas.

La implementación de esta solución, así como de otras soluciones alternativas no basadas en este algoritmo que se utilizarán para evaluar su eficacia por comparación, se ha desarrollado en Python, tomando como conjunto de pruebas un archivo CSV que contiene todas las estaciones de BiciMAD correspondientes a junio de 2018.

El estudio de los resultados obtenidos mediante este algoritmo comparados con otros no basados en la colonia de hormigas, demuestra que se obtienen rutas más cortas y, por lo tanto, mejores. En conclusión, el algoritmo desarrollado en el presente trabajo es adecuado y efectivo para la resolución de este problema.

Abstract (in English, 250 words or less):

The main purpose of this research paper is to find the best route for restoring the bicicles in all of the BiciMAD stations. In order to do so, the Ant Colony Optimization algorithm is chosen, on the grounds that it's known to work well with route optimization problems.

This algorithm, as well as other non ACO solutions developed for evaluation purposes, is coded in Python using as a data set a CSV file containing all the BiciMAD stations for June 2018.

The comparative analysis of the results of both, ACO and non ACO solutions, shows that the Ant Colony Optimization algorithm finds shorter and thus, better routes. In conclusion, the ACO algorithm developed for this paper proves to be suitable and effective to solve this problem.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

1.1.1. Temática

En los últimos años se han establecido en las grandes ciudades servicios de alquiler de bicicletas, que facilitan a los ciudadanos un medio de transporte económico y respetuoso con el medio ambiente. Estos servicios constan de una serie de estaciones distribuidas por distintas zonas de la ciudad y permiten que cada usuario recoja una bicicleta de la estación que más le convenga y la devuelva en la que desee.

1.1.2 Problemática a resolver

Dado que los usuarios pueden devolver las bicicletas en las estaciones que les resulten más convenientes, es habitual que algunas de las estaciones queden vacías al final del día mientras que otras terminan llenas, ya sea por ser más céntricas, encontrarse cuesta abajo o cualquier otro motivo. Por ello se hace necesario disponer de vehículos que redistribuyan las bicicletas por todas las estaciones.

Realizar este servicio siguiendo una ruta óptima no solo tendría beneficios para la empresa, que podría ahorrar tiempo y costes, sino que también reportaría beneficios para el medio ambiente, ya que se reduciría el tiempo que están estos vehículos en circulación. Hay que tener en cuenta, además, que uno de los principales motivos por los que se promueve el uso de bicicletas como transporte en ciudad es disminuir la contaminación ambiental.

El algoritmo de la colonia de hormigas (ACO) es un algoritmo de optimización basado en el comportamiento de las hormigas, que utilizan la comunicación a través de feromonas para ir perfeccionando sus rutas. Este algoritmo tiene aplicaciones en problemas de recorridos en grafos, como el problema del viajante, o problemas de optimización de rutas de vehículos, por lo que se plantea como una buena solución para encontrar una ruta óptima para el servicio de reposición de bicicletas.

1.2 Objetivos del Trabajo

El objetivo general de este trabajo es encontrar una ruta óptima de reposición de bicicletas para el servicio BiciMAD aplicando el algoritmo de la colonia de hormigas.

Objetivos específicos:

- Demostrar que el algoritmo ACO es aplicable a este problema.
- Conseguir un modelo lo más parecido posible al caso real con el que sea factible trabajar.
- Estudiar posibles mejoras que proporcionen un resultado óptimo.
- Analizar los resultados y deducir conclusiones en base a ellos.

1.3 Enfoque y método seguido

Un posible enfoque sería realizar el trabajo en dos fases: una primera donde se realizaría una estimación de la demanda de bicicletas en cada estación, basándose en los datos reales del uso de BiciMAD y utilizando mecanismos de *machine learning,* y una segunda fase donde se aplicaría el algoritmo de colonia de hormigas para obtener la ruta óptima de redistribución.

Ya que la estimación de la demanda es un problema extenso en sí mismo y, a efectos del presente trabajo el interés reside en la eficacia del algoritmo de la colonia de hormigas, se ha decidido partir de unos datos de demanda inicial de bicicletas ficticios para simplificar el problema.

El trabajo se plantea con una estrategia gradual, en la que en un principio se aplicará el algoritmo a un caso simplificado y, una vez que se demuestre que es apropiado para resolver el problema, se irán añadiendo modificaciones que acercarán el problema al caso real de BiciMAD e intentarán buscar mejoras en la eficiencia.

1.4 Planificación del Trabajo

1.4.1 Recursos necesarios:

Para la implementación del algoritmo de la colonia de hormigas se hará uso de la librería ACO-Pants 0.5.2. El proyecto se realizará en Python 3 con la distribución Anaconda y el IDE Spyder.

Como fuente de datos para crear el modelo en el que se implementará el algoritmo, se consultarán los datos públicos del uso de BiciMAD en la página web del Ayuntamiento de Madrid.

Para el estudio del algoritmo de la colonia de hormigas y sus aplicaciones y posibles mejoras, se consultará principalmente el libro *Ant Colony Optimization* de Marco Dorigo y Thomas Stützle.

1.4.2 Tareas a realizar:

- PEC0: Definición de contenidos (20/2 - 2/3)

- PEC1: Plan de trabajo (3/3 - 16/3)
- PEC2: Desarrollo del trabajo. Fase 1 (17/3 - 20/4)

    - Introducción al problema TSP (17/3 - 23/3)
    - Problema de reposición de bicicletas (24/3 - 30/3)
    - Algoritmo de la colonia de hormigas y sus aplicaciones (31/3 - 6/4)
    - Instalación librería ACO-Pants 0.5.2. y preparación del entorno de proyecto (7/4)
    - Aplicación ACO a problema de reposición de bicicletas (8/4 - 19/4)

- PEC3: Desarrollo del trabajo. Fase 2 (21/4 - 18/5)

    - Estudio de posibles mejoras (21/4 - 16/5)
    - Conclusiones (17/5)

- PEC4 - Redacción de la memoria (17/3 - 2/6)
- PEC5a - Elaboración presentación (3/6 - 10/6)
- PEC5b - Defensa pública (22/6)

1.4.3 Planificación temporal

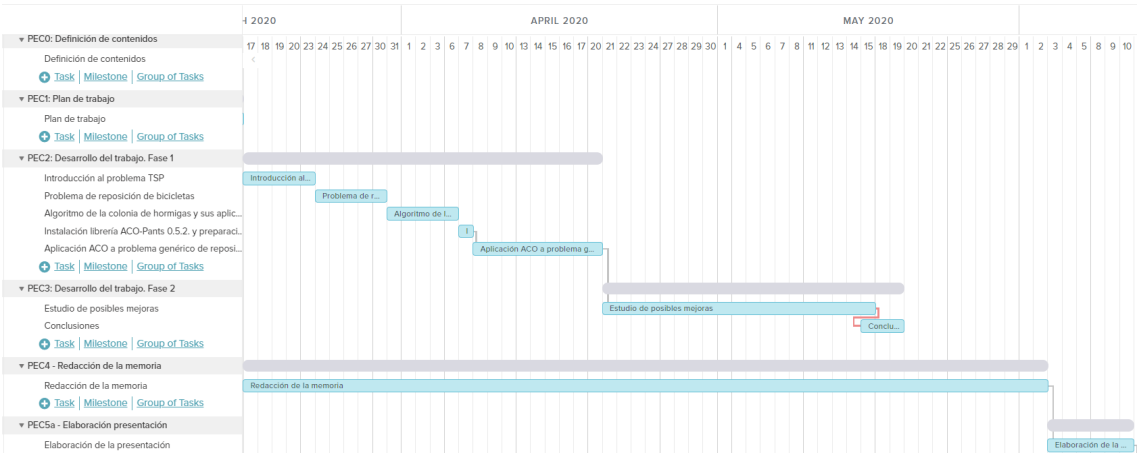

Fig. 1: Planificación temporal

1.4.4 Posibles riesgos

Riesgo	Acciones de mitigación
La librería ACO-Pants 0.5.2 presenta algún problema o fallo	Búsqueda de otra librería disponible o implementación propia del algoritmo
Alguna de las posibles mejoras que se intentan implementar resulta ser demasiado compleja y ocupa más tiempo del previsto en la planificación	Acomodar el número o características de las mejoras al tiempo disponible
La situación actual de la pandemia de coronavirus y los cambios de rutina, así como la posibilidad de contagio pueden afectar al tiempo de dedicación al TFG	Medidas de prevención al contagio y reserva de un espacio de tiempo diario dedicado en exclusiva al desarrollo del TFG

1.5 Breve sumario de productos obtenidos

- Memoria del trabajo
- Archivos .py con los resultados de la implementación del problema y sus casos de test

1.6 Breve descripción de los otros capítulos de la memoria

2. Desarrollo del trabajo: Cuerpo del trabajo en el que se describe el problema, las posibles soluciones, los resultados y las conclusiones.

2.1. Algoritmo de la colonia de hormigas (ACO): Descripción de los tipos de problemas existentes desde el punto de vista de la complejidad computacional, introducción al concepto de heurístico y metaheurístico y explicación detallada del algoritmo de la colonia de hormigas y sus aplicaciones.

2.2. Problema de la reposición de bicicletas: Descripción del problema, valoración de posibles estrategias y propuesta de las diferentes soluciones.

2.3. Resultados: Análisis comparativo de los resultados obtenidos con las distintas soluciones propuestas para el conjunto total de estaciones de BiciMAD.

3. Conclusiones: A partir de los resultados obtenidos se realiza una conclusión final sobre la idoneidad del algoritmo ACO para resolver el problema de reposición de bicicletas y se discuten la mejores rutas obtenidas. Finalmente, se valora si se han cumplido los objetivos del trabajo y se proponen futuras áreas de mejora o ampliación sobre el tema.

4. Glosario: Definición de términos relevantes para la comprensión del trabajo.

5. Bibliografía: Fuentes de información consultadas.

6. Anexos: código fuente correspondiente a las implementaciones de las soluciones planteadas y lista de las estaciones de BiciMAD utilizada como conjunto de prueba.

2. Desarrollo del trabajo

2.1. Algoritmo de la colonia de hormigas (ACO)

El algoritmo de la colonia de hormigas es una técnica basada en el comportamiento de las hormigas y su comunicación a través de feromonas, que les permite optimizar el trayecto entre el hormiguero y la fuente de alimento, y tiene como aplicación principal la resolución de problemas que requieren encontrar las mejores rutas en grafos.

2.1.1. Tipos de problemas según su complejidad computacional

En teoría de la complejidad computacional, se diferencian varios tipos de problemas según el tiempo que tardan en resolverse. Un problema *C* puede ser [1]:

- P: se puede resolver en tiempo polinómico con métodos deterministas.
- NP: se puede resolver en tiempo polinómico mediante métodos no deterministas (Incluye P). Se puede verificar en tiempo polinómico.
- NP-Hard: cualquier problema en NP se puede reducir a *C* en tiempo polinómico. Un problema NP-Hard es como mínimo tan difícil como los problemas más difíciles de NP.
- NP-Completo: se puede resolver en tiempo polinómico solo mediante métodos no deterministas (no incluye P). *C* es tanto NP como NP-Hard.

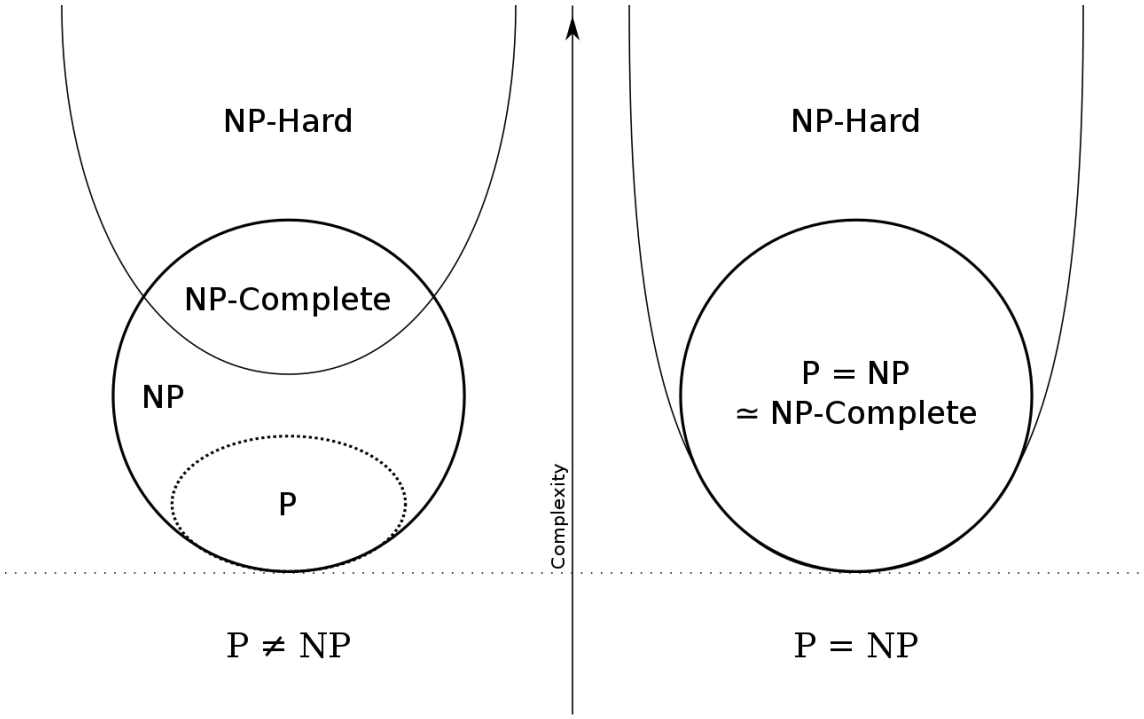

Fig. 2: Diagrama de Euler para los problemas P, NP, NP-completos y NP-hard si se asume que P ≠ NP y si se asume que P = NP [2]

Todos los problemas de optimización pueden convertirse en problemas de decisión y viceversa. Por ejemplo, el problema del viajante de comercio (*Travelling Salesman Problem* o TSP) plantea el problema de un viajante de comercio que debe recorrer una serie de ciudades por el camino más corto posible de tal manera que visite cada ciudad una vez y regrese a la ciudad de partida. En este caso la versión de optimización corresponde a calcular la ruta más corta y sería un problema de tipo NP-Hard, mientras que la de decisión – dada una longitud L, decidir si el grafo tiene una ruta de como máximo longitud L – es un problema de tipo NP-Completo [3].

2.1.2. Heurísticos y metaheurísticos

Los problemas de optimización combinacional a menudo son difíciles de resolver por la complejidad computacional que conllevan. Existen algoritmos que utilizan métodos aproximados para encontrar soluciones casi óptimas en un tiempo relativamente corto. Estos algoritmos se conocen como *heurísticos.*

Un *metaheurístico* es un conjunto de conceptos algorítmicos que se puede utilizar para definir métodos heurísticos. Según Fred Glover y Manuel Laguna (1998) [1]:

"Un metaheurístico se refiere a una estrategia maestra que guía y modifica a otros heurísticos para encontrar soluciones más allá de las que son normalmente generadas en una búsqueda de optimalidad local".

El algoritmo basado en la colonia de hormigas o ACO (*Ant Colony Optimization*) es una metaheurística basada en el comportamiento observado en las hormigas y es aplicable a cualquier problema de optimización combinatoria que pueda definirse con una heurística constructiva y es particularmente interesante en [1]:

- Problemas NP-Hard que no pueden resolverse eficientemente mediante algoritmos tradicionales.
- Problemas dinámicos de camino más corto en los que algunas propiedades van cambiando concurrentemente al proceso de optimización.
- Problemas en los que la arquitectura computacional está distribuida espacialmente.

Algunos ejemplos son los problemas de cálculo de rutas, como el ya mencionado TSP o el problema de ordenamiento secuencial SOP, problemas de asignación de tareas, como el problema de asignación generalizada GAP, problemas de programación de tareas como el Job Shop Scheduling o problemas de subconjuntos como el problema de la mochila múltiple.

Problem name	*Authors*	*Algorithm name*	*Year*	*Main references*
Traveling salesman	Dorigo, Maniezzo & Colorni	AS	1991	[29, 37, 38]
	Gambardella & Dorigo	Ant-Q	1995	[41]
	Dorigo & Gambardella	ACS & ACS-3-opt	1996	[33, 34, 42]
	Stützle & Hoos	$\mathcal{MM}$AS	1997	[84, 82, 85]
	Bullnheimer, Hartl & Strauss	AS_{rank}	1997	[14]
	Cordón, et al.	BWAS	2000	[18]
Quadratic assignment	Maniezzo, Colorni & Dorigo	AS-QAP	1994	[65]
	Gambardella, Taillard & Dorigo	HAS-QAP[a]	1997	[46]
	Stützle & Hoos	$\mathcal{MM}$AS-QAP	1997	[79, 85]
	Maniezzo	ANTS-QAP	1998	[62]
	Maniezzo & Colorni	AS-QAP[b]	1999	[64]
Scheduling problems	Colorni, Dorigo & Maniezzo	AS-JSP	1994	[17]
	Stützle	AS-FSP	1997	[80]
	Bauer *et al.*	ACS-SMTTP	1999	[2]
	den Besten, Stützle & Dorigo	ACS-SMTWTP	1999	[21]
	Merkle, Middendorf & Schmeck	ACO-RCPS	2000	[66]
Vehicle routing	Bullnheimer, Hartl & Strauss	AS-VRP	1997	[12, 13]
	Gambardella, Taillard & Agazzi	HAS-VRP	1999	[45]
Connection-oriented network routing	Schoonderwoerd *et al.*	ABC	1996	[77, 76]
	White, Pagurek & Oppacher	ASGA	1998	[89]
	Di Caro & Dorigo	AntNet-FS	1998	[26]
	Bonabeau *et al.*	ABC-smart ants	1998	[10]
Connection-less network routing	Di Caro & Dorigo	AntNet & AntNet-FA	1997	[23, 25, 28]
	Subramanian, Druschel & Chen	Regular ants	1997	[86]
	Heusse *et al.*	CAF	1998	[54]
	van der Put & Rothkrantz	ABC-backward	1998	[88]
Sequential ordering	Gambardella & Dorigo	HAS-SOP	1997	[43, 44]
Graph coloring	Costa & Hertz	ANTCOL	1997	[19]
Shortest common supersequence	Michel & Middendorf	AS-SCS	1998	[67, 68]
Frequency assignment	Maniezzo & Carbonaro	ANTS-FAP	1998	[63]
Generalized assignment	Ramalhinho Lourenço & Serra	MMAS-GAP	1998	[73]
Multiple knapsack	Leguizamón & Michalewicz	AS-MKP	1999	[59]
Optical networks routing	Navarro Varela & Sinclair	ACO-VWP	1999	[71]
Redundancy allocation	Liang & Smith	ACO-RAP	1999	[60]
Constraint satisfaction	Solnon	Ant-P-solver	2000	[78]

24

[a] HAS-QAP is an ant algorithm which does not follow all the aspects of the ACO metaheuristic.
[b] This is a variant of the original AS-QAP.

Fig. 3: Aplicaciones del algoritmo de la colonia de hormigas [4]

2.1.3. Hormigas reales y hormigas artificiales

Hormigas reales

Las hormigas coordinan su actividad mediante una comunicación indirecta basada en la modificación del ambiente en el que se mueven. Este concepto se conoce como *estimergia* (*stigmergy*) y fue introducido por Pierre-Paul Grassé (1959), que originalmente se basó en el comportamiento de las termitas en la construcción de sus nidos y lo definió como [1]:

"Estimulación de los trabajadores mediante el trabajo que han obtenido".

Es decir, en el caso de las hormigas no son estas las que se adaptan al medio para llevar a cabo su tarea, sino que modifican el medio mediante el depósito de feromonas y cambian la manera en la que otras hormigas perciben el problema.

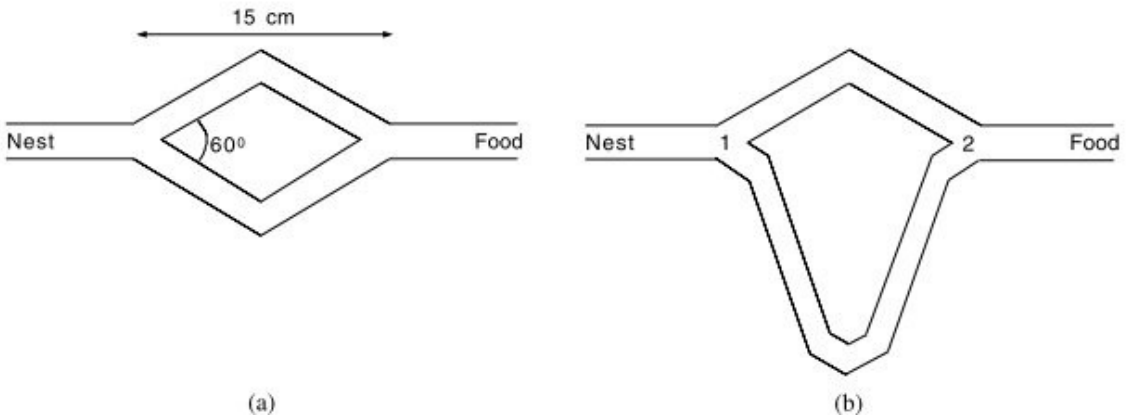

Fig. 4: Experimento de doble puente. Con ambos caminos de la misma longitud (a) y con un camino más largo que otro (b) [5]

Cuando las hormigas tienen que encontrar una ruta desde el hormiguero hasta el alimento, comienzan eligiendo caminos al azar y mientras los recorren van depositando feromonas, tanto a la ida como a la vuelta. Debido a que el camino de ida y vuelta por una ruta corta puede realizarse más veces en el mismo tiempo que siguiendo rutas más largas, se acumula en ellos una mayor concentración de feromonas, lo que va a provocar que las siguientes hormigas se sientan más atraídas por los caminos cortos haciendo que poco a poco la ruta se optimice.

Este comportamiento se ha podido observar en los experimentos conocidos como "experimentos de doble puente", en los que se ofrecen dos caminos a las hormigas para llegar a la comida. Si uno de ellos es más largo que el otro (b) siempre terminarán todas las hormigas recorriendo el camino corto.

Si por el contrario se ofrecen dos caminos iguales (a), por simple azar por uno de ellos pasarán más hormigas que por otro, aunque sean muy pocas, lo que

hará que ese camino quede marcado con mayor cantidad de feromonas y finalmente todas las hormigas convergerán en él. En este caso en la mitad de los experimentos todas las hormigas acaban convergiendo en uno de los caminos y en la otra mitad, en el otro.

Si se ofrece un único camino y una vez que las hormigas han marcado esa ruta con sus feromonas se añade un camino alternativo más corto, las hormigas seguirán eligiendo el largo, ya que es el que tiene mayor concentración de feromonas acumuladas. Esto se debe a que la evaporación de feromonas es tan lenta que para cuando se ha producido las hormigas ya han convergido en un camino. Es decir, a todos los efectos es como si no existiera evaporación.

Deneubourg et al. (1990) y Goss et al.(1989) propusieron un modelo estocástico basado en los experimentos de doble puente. Este modelo considera que [1]:

- La cantidad de feromona en cada rama es proporcional al número de hormigas que han pasado por esa rama.
- No se produce evaporación de feromonas.
- Las hormigas depositan feromona en el camino de ida y en el de vuelta.

En este modelo, de la misma manera que ocurre con las hormigas reales, la probabilidad de que una hormiga seleccione una rama concreta dependerá de la cantidad total de feromonas acumuladas en esa rama, que es proporcional al número de hormigas que han pasado por allí hasta ese momento. Por ejemplo, la probabilidad de que una hormiga cuando llega a un punto de bifurcación seleccione el camino más corto viene representada por la siguiente fórmula [1]:

$$p_{is}(t) = (t_s + \varphi_{is}(t))^{\alpha} / ((t_s + \varphi_{is}(t))^{\alpha} + (t_s + \varphi_{il}(t))^{\alpha})$$

Donde $p_{is}(t)$ es la probabilidad de escoger el camino más corto en una bifurcación i y un momento de tiempo t, t_s es el tiempo que tarda una hormiga en atravesar la rama corta , $\varphi_{is}(t)$ es la cantidad total de feromona acumulada en la rama corta a tiempo t, $\varphi_{il}(t)$ es la cantidad total de feromona acumulada en la rama larga a tiempo t y α es un valor experimental de valor $\alpha = 2$.

Hormigas artificiales

Si se intenta aplicar directamente un algoritmo basado en los experimentos con hormigas reales para encontrar el camino de coste mínimo en un grafo, se corre el riesgo de que en los primeros trayectos se generen bucles y, debido al refuerzo de las feromonas, que las hormigas queden atrapadas en estos bucles. Para optimizar el comportamiento de las hormigas artificiales, se pueden aplicar una serie de mejoras. Las principales son [1]:

- Implementación de una memoria: se almacena información sobre las rutas parciales que se han seguido hasta el momento y el coste de los enlaces. Esta memoria permite tres comportamientos:

- Construir soluciones probabilísticas a partir de rastros de feromonas sin realizar actualización de feromonas. Cada hormiga tiene dos modos: *forward* (del nido a la comida) y *backward* (de la comida al nido). En cada nodo la hormiga elige el siguiente nodo a visitar basándose en el rastro de feromonas que dejaron las hormigas anteriores. Las hormigas *forward* no dejan feromona.
- Retroceder por el camino de forma determinista eliminando bucles y dejando feromonas.
- Evaluar la calidad de las soluciones generadas y basarse en ella para determinar la cantidad de feromona a depositar. Dejar mayor cantidad de feromona en caminos más cortos lleva a encontrar las mejores soluciones.

- Evaporación de feromonas: minimiza la influencia de las feromonas dejadas en las primeras fases, cuando las hormigas pueden conseguir soluciones de baja calidad.

En cuanto a los valores óptimos de las variables, los resultados experimentales han demostrado que :

- Grandes valores de α tienden a amplificar la influencia de fluctuaciones aleatorias iniciales. De experimentos con hormigas reales se dedujo un valor de $\alpha = 2$, pero para artificiales se ha encontrado que el algoritmo llega al camino más corto más frecuentemente cuando α es 1.

- Una tasa de evaporación *p* pequeña lleva a mejores resultados.

- Cuanto mayor número de hormigas, mejor comportamiento del algoritmo, aunque aumenta el coste temporal.

En conclusión, una aplicación directa del comportamiento observado en hormigas reales no llevaría a resultados óptimos, especialmente por el riesgo de que queden atrapadas en bucles. Sin embargo, a las hormigas artificiales se les pueden añadir mejoras como implementación de memoria, evaporación de feromonas y elección de valores óptimos para las variables.

2.2. Problema de la reposición de bicicletas

2.2.1. Descripción y estrategias posibles

En el caso del problema planteado, los usuarios de BiciMAD depositan las bicicletas en las estaciones que encuentran más convenientes, produciendo al final de cada día un desequilibrio en el número de vehículos en cada estación. Por este motivo es necesario disponer de vehículos que redistribuyan las bicicletas por todas las estaciones. En este trabajo se tratará de encontrar las rutas más cortas para realizar esta tarea.

Los algoritmos inspirados por la naturaleza se pueden clasificar en evolutivos y en los conocidos como *inteligencia de enjambre* (*swarm intelligence*). Entre estos últimos se encuentra el algoritmo de la colonia de hormigas y otros como el de la colonia de abejas, los murciélagos o la mariposa monarca. Se ha elegido como más adecuado al problema tratado en este trabajo el algoritmo de la colonia de hormigas, por ser el que aporta una solución en forma de recorrido en un grafo. [6]

Table 1 Comparative study of algorithm

Technique	Evolutionary			Swarm		
Survival	Survival of the next generation			Survival in the society		
Communication of agents	No communication. Information can be passed from one generation to another			Agents can communicate in order to share information		
Algorithm	Genetic algorithm	Genetic programming	Differential evaluation	Ant colony optimization	Particle swarm optimization	Artificial bee colony
Developed by	Holland	Koza	Storm and Price	Marco Dorigo	Eberhart and Kennedy	Karaboga
Year	1975	1992	1995	1992	1995	2005
Basic principle	Survival of the fittest	Survival of the fittest	Survival of the fittest	The cooperative intelligence of the swarm	The cooperative intelligence of the swarm	The collective knowledge of bees
Solution representation	Binary/real valued	Expression tree	Real-valued	Graph structure for path covering of ants	Real-valued	Real-valued
Fitness	Normalized fitness	Normalized fitness	Objective function value	Scaled objective function	Objective function value	Objective function value
Operation	Reproduction, crossover, occasional mutation	Reproduction, crossover	Mutation, other operators or crossover if required	Pheromone updating and evaporation	Velocity update, position update	Probablistic selection of site, rejection of sites
Selection process	Probabilistic, preservative	Probabilistic, extinctive	Deterministic, extinctive	Probabilistic, preservative	Deterministic, extinctive	Probabilistic, preservative
Require ranking of the solution	Yes	Yes	No	No	No	No
Termination criterion	After a specific number of generations or when some acceptable and recognizable result is obtained.	After a specific number of generations or when some acceptable and recognizable result is obtained	After a specific number of generations or when some acceptable and recognizable result is obtained	After a particular number of iterations performed, or a solution with adequate objective function value is found	After a particular number of iterations performed, or a solution with adequate objective function value is found	After a particular number of iterations performed, or a solution with adequate objective function value is found
Result designation	The best individual so far	The best individual so far	The best individual so far	The best path so far	The best position so far	The best location so far

Fig. 5: Estudio comparativo de diferentes algoritmos basados en la naturaleza [6]

Como estrategia para redistribuir las bicicletas, existen varias aproximaciones que se pueden aplicar [7]. Principalmente se han observado las siguientes:

1) Tener en cuenta únicamente la distancia entre estaciones y buscar el recorrido más corto pasando una vez por cada estación. Es decir, nos encontraríamos con un problema de tipo TSP al que podríamos aplicar el algoritmo de la colonia de hormigas u otro aplicable a este tipo de problemas.

El inconveniente que presenta este planteamiento es que no tiene en cuenta desde el principio la demanda de bicicletas en cada estación, por lo que podría

darse el caso de que el camión visitase primero estaciones en las que es necesario depositar bicicletas por encontrarse a menor distancia y que aún no haya cargado las suficientes para reponerlas. Este efecto podría contrarrestarse mediante el depósito de feromonas, de tal modo que si se llega a una estación que demanda más bicicletas de las disponibles no se deposite feromona y en siguientes iteraciones las hormigas se vean más atraídas por otras rutas mejores. No obstante, siguiendo este procedimiento no se tiene total seguridad de que se vaya a encontrar una solución válida, es decir, que todas las estaciones que necesitan ser abastecidas lo sean.

2) Tener en cuenta solo la demanda de bicicletas. Para asegurar que se abastece correctamente a todas las estaciones se podría seguir por ejemplo la estrategia de visitar primero las estaciones con más excedente de bicicletas y dejar para el final aquellas que están desprovistas de vehículos. Suponiendo un camión de redistribución con capacidad infinita, este planteamiento es viable, pero no óptimo, ya que puede dar lugar a rutas más largas de lo necesario. Esta aproximación, no basada en la colonia de hormigas, se desarrollará como un algoritmo ingenuo con el fin de comparar resultados y determinar si la solución ACO es más efectiva.

3) Hallar un equilibrio entre la distancia entre estaciones y la demanda de bicicletas en cada estación para encontrar una solución viable que optimice la distancia que tienen que recorrer los camiones. Este equilibrio se conseguirá en el algoritmo ACO gracias a la regulación del depósito de feromonas y la probabilidad de escoger la siguiente estación a visitar.

2.2.2. Búsqueda de soluciones

Como se ha explicado en el apartado anterior, el primer acercamiento no se considera viable por existir la posibilidad de que algunas estaciones se queden sin abastecer ya que no existe una restricción para que se cumpla esa condición. Por lo tanto, el planteamiento que se seguirá para resolver el problema se centrará primero en cumplir la condición necesaria de abastecer a todas las estaciones y después optimizar la ruta para que el camino recorrido por los camiones sea el más corto posible.

Para una primera versión de la búsqueda de una ruta óptima se reduce al máximo la complejidad del problema, realizando las siguientes suposiciones:

- Se dispone de un solo camión con capacidad infinita. El algoritmo de la colonia de hormigas permite proporcionar una solución tanto para el caso de disponer de un camión de reposición como para varios que operen a la vez. Ya que buscamos en un principio la versión más sencilla del problema, lo plantearemos con un solo camión con capacidad infinita, es decir, no tenemos que preocuparnos de que el camión tenga que cargar más bicicletas de las que caben. En otras implementaciones del algoritmo se podría añadir un límite de capacidad para el camión o incluso tener en cuenta más de un camión.

- No se tiene en cuenta la estación central de donde salen los camiones, sino que se construye un grafo solo con las estaciones de BiciMAD. Ya que no se conoce la ubicación de la estación central y no afecta al planteamiento del algoritmo, es más simple obviarla.

- No se tiene en cuenta el regreso a la estación original, por simplicidad.

- Las distancias entre estaciones se calculan como distancias euclidianas, suponiendo que un camión puede desplazarse en línea recta de una estación a otra. En la realidad, los camiones deben seguir el trazado y sentido de las carreteras, por lo que para poder aplicar el problema al caso real se tendría que implementar una mejora que incluyese las distancias reales, que podrían obtenerse por ejemplo mediante Google Maps.

Las primeras soluciones planteadas se centrarán en encontrar resultados viables y servirán como base para construir el algoritmo de la colonia de hormigas y posteriormente introducir mejoras, así como para valorar por comparación los resultados del algoritmo de la colonia de hormigas.

Solución 1: Búsqueda de ruta viable

Como primera toma de contacto al problema se plantea un algoritmo ingenuo o *naive*, en el que solo se tiene en cuenta la demanda de bicicletas en cada estación para conseguir una solución viable. Este algoritmo se basa en ordenar las estaciones según su demanda, de las que tienen mayor excedente de bicicletas a las que tienen más escasez, con la finalidad de que el camión de reparto siempre tenga bicicletas disponibles para atender todas las demandas.

Por claridad y para facilitar la comprensión visual de la ruta obtenida, se utilizará como ejemplo una aplicación del algoritmo únicamente sobre las cinco primeras estaciones de la lista de BiciMAD por orden alfabético [8]. Para ello se añaden demandas ficticias a cada una de las estaciones:

Agustín de Betancourt: long= -3.6956047, lat= 40.4440297, demanda= -4
Alberto Alcocer: long= -3.684715, lat= 40.4585318, demanda= 4
Alcalá, long= -3.6801307, lat= 40.4226906, demanda= -3
Alcántara, long= -3.6738714, lat= 40.4261851, demanda= 4
Almadén, long= -3.693225, lat= 40.4108472, demanda= -1

Donde las demandas negativas se corresponden con una falta de bicicletas en dicha estación y las positivas, con excedente de bicicletas. La suma de todas las demandas cumple la propiedad de ser igual a cero.

Al ejecutar el programa (ver Anexo) se obtiene como resultado:

Ruta: ['Alberto Alcocer', 'Alcántara', 'Almadén', 'Alcalá', 'Agustín de Betancourt']
Distancia total: 0.1028251449577644

En este caso, el camión parte vacío y visita primero Alberto Alcocer, donde carga las cuatro bicicletas sobrantes de esta estación (total camión = 4). De ahí se dirige a Alcántara, donde carga otras cuatro (total camión = 8). Seguidamente va a Almadén, donde descarga una (total camión = 7). Después se dirige a Alcalá, donde deja otras tres (total camión = 4) y finalmente, termina en Agustín de Betancourt, donde deja las últimas cuatro bicicletas (total camión = 0). (ver Fig. 6)

Fig. 6: Solución 1

La distancia total es un valor obtenido mediante la fórmula de la distancia euclidiana sobre los valores de latitud y longitud de las estaciones. No se corresponde con un valor concreto en metros, ya que los camiones en la vida real no pueden circular en línea recta desde unas estaciones a otras, pero sirve como valor de referencia para comparar distintas soluciones y estimar qué ruta es mejor.

Solución 2: Restricciones de viabilidad y acercamiento a la optimización

Al igual que en la solución 1, solo se tiene en cuenta la demanda de bicicletas en cada estación para conseguir una solución viable, pero en este caso no se obliga a seguir un orden de visita según la demanda de bicicletas, sino que se elegirá la siguiente estación a visitar de forma aleatoria y se introducirán una serie de restricciones que aseguren que no se visitarán estaciones de las que no se pueda satisfacer la demanda.

La finalidad de este cambio con respecto a la solución 1 es dejar implementadas las restricciones que asegurarán una solución viable y en futuras modificaciones sustituir la elección aleatoria de la siguiente estación a visitar por una elección basada en el rastro de feromonas. De esta forma se prepara al programa para poder implementar una solución basada en la colonia de hormigas. Otro cambio introducido con respecto a la solución 1 es un bucle que genera 100 rutas aleatorias y elige la que ofrece una menor distancia recorrida. Es decir, añade cierto grado de optimización.

Al ejecutar el programa una vez (ver Anexo) se obtiene el resultado:

Ruta: ['Alberto Alcocer', 'Agustín de Betancourt', 'Alcántara', 'Alcalá', 'Almadén']
Distancia total: 0.07108056080729643

El camión se dirige a Alberto Alcocer, donde carga cuatro bicicletas (total camión = 4) que descarga seguidamente en Agustín de Betancourt (total camión = 0). Después se dirige a Alcántara, donde vuelve a cargar cuatro bicicletas (total camión = 4), en Alcalá deja tres (total camión = 1) y en Almadén deja la última (total camión = 0). (ver Fig. 7)

Fig. 7: Solución 2

Se puede comprobar que es una ruta mejor que la obtenida con la Solución 1, ya que la distancia total a recorrer es menor.

Solución 3: Efecto de las feromonas

El siguiente paso es modificar la manera en la que se seleccionan las estaciones a visitar e introducir el concepto de feromonas y, con ello, el de hormigas. En una primera vuelta las hormigas visitan las estaciones al azar, depositando feromonas por donde pasan y, a partir de la segunda vuelta, van siendo atraídas por la concentración de feromonas acumuladas en cada estación. Cabe destacar que se ha implementado una construcción secuencial de la solución. Es decir, cada iteración corresponde a una hormiga y, por tanto, cada una de ellas construye totalmente su ruta antes de que salga la hormiga siguiente. Otra posible implementación es la implementación paralela, en la que las hormigas se mueven a la vez.

Para calcular el efecto de atracción ejercido por las feromonas se ha utilizado como modelo el propuesto por Deneubourg et al. (1990) y Goss et al.(1989) [1]. La probabilidad de que una hormiga elija un camino *x* vendrá determinada por la relación entre las feromonas acumuladas en *x* y las feromonas totales.

Para implementar esta probabilidad se ha seguido la siguiente estrategia:

Supongamos que se tiene la siguiente relación de caminos y concentración de feromonas:

fer = {‘a’: 1, ‘b’: 3, ‘c’: 2}

Se construye una lista ponderada que contendrá cada camino una vez por cada unidad de feromonas.

caminos = [‘a’, ‘b’, ‘b’, ‘b’, ‘c’, ‘c’]

Finalmente, se selecciona un elemento al azar de esa lista de caminos. De esta manera se mantiene la probabilidad de elegir un camino según la concentración de feromonas.

En un grafo, un camino es un arco que une dos vértices. En este caso estarán representados por una pareja de estaciones. Por ejemplo, un camino podría ser (Álcalá, Almadén). Como estructura de datos para almacenar las concentraciones de feromonas en cada camino se ha utilizado un diccionario doble como el siguiente:

{'Agustín de Betancourt': {'Agustín de Betancourt': 1, 'Alberto Alcocer': 3, 'Alcalá': 1, 'Alcántara': 1, 'Almadén': 1}, …}

En este ejemplo, el camino que une Agustín de Betancourt con Alberto Alcocer tiene acumuladas 3 unidades de feromona.

Al ejecutar el programa una vez (ver Anexo) se generan 100 rutas posibles y se elige la mejor de todas ellas. Por ejemplo, una de las salidas obtenidas es:

['Alberto Alcocer', 'Alcántara', 'Alcalá', 'Almadén', 'Agustín de Betancourt']
Distancia total: 0.09220808688922288

El camión se dirige a Alberto Alcocer, carga cuatro bicicletas (total camión = 4), después va a Alcántara, donde carga otras cuatro (total camión = 8). Seguidamente se dirige a Alcalá, donde deja tres (total camión = 5). Después deja una en Almadén (total camión = 4) y finalmente, deja las cuatro últimas en Agustín de Betancourt (total camión = 0). (ver Fig. 8)

Fig. 8: Solución 3

En este caso la distancia es un poco mayor que en la solución 2, por lo que se considera una ruta peor. Sin embargo, al ser un ejemplo con solo cinco estaciones el resultado no tiene por qué ser representativo de la calidad del algoritmo. Posteriormente se realizará un estudio comparativo de todas las soluciones utilizando el conjunto de datos completo de BiciMAD.

Mejoras en el algoritmo

Como se ha explicado en el apartado 2.1.3., una aplicación directa del comportamiento de las hormigas reales conlleva varios problemas, siendo el principal la acumulación de feromonas en los primeros trayectos, pudiéndose reforzar rutas no óptimas o incluso bucles. Para optimizar el comportamiento de las hormigas se contemplan varias mejoras [1]:

- Implementación de una memoria almacenando información sobre las rutas parciales y el coste de los enlaces.

- Evaporación de feromonas, que minimiza la influencia de las hormonas dejadas en las primeras fases.

En la solución 3 ya se ha implementado una memoria en forma de una lista que almacena las estaciones visitadas y un diccionario doble que guarda los valores de feromona para cada arco. Sin embargo, no se ha tenido en cuenta la evaporación de feromonas, por lo que en la siguiente fase se aplicará la evaporación de feromonas como mejora del algoritmo.

Otras mejoras que podrían implementarse pero quedan fuera de este trabajo son la *Estrategia elitista* (Dorigo (1992) y Dorigo et al., (1991a, 1996)), en la que se provee de un refuerzo adicional a los arcos pertenecientes a la mejor ruta encontrada hasta el momento, o el *Sistema basado en rango* (Bullnheimer et al.(1999c)) en el que la cantidad de feromonas depositada por cada hormiga depende del rango de la misma. Esto es, la mejor hormiga hasta el momento siempre deposita la mayor cantidad de feromona en cada iteración [1].

Solución 3 mejorada: Construcción de la ruta y evaporación de feromonas

La solución 3 añadía el uso de feromonas de una forma muy básica: se comenzaba con una dosis inicial de feromona igual a 1 en cada camino, se depositaba una unidad de feromona en los caminos recorridos y se elegía la siguiente estación a visitar según la cantidad de feromona acumulada. Existen varios cambios que se pueden implementar que han demostrado mejorar los resultados obtenidos. Estos son:

- Inicializar las feromonas con valores un poco mayores de lo que se espera depositar en una iteración. En concreto se ha observado que una buena medida para el problema TSP es inicializarlas con el valor m / C^{nn}, donde m es el número de hormigas y C^{nn} es la longitud del camino obtenido por una heurística de vecino más cercano, aunque valdría cualquier otra heurística [1]. Como en el caso que nos ocupa es necesario tener en cuenta la demanda de bicicletas en cada estación además de la distancia entre ellas, se utilizará el valor de distancia correspondiente a la ruta obtenida por la solución 1.

- Aplicar la regla de *proporcionalidad aleatoria* a la construcción de la ruta. La probabilidad de escoger la siguiente estación viene dada por [1]:

$$p_{ij}^{k} = ([\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}) / (\Sigma_{l \epsilon N_{ik}}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}) \quad \text{si } j \epsilon N_i^k$$

Donde p_{ij}^{k} es la probabilidad de ir a la estación j para una hormiga k que se encuentra en la estación i, τ_{ij} representa las feromonas acumuladas en el arco (i,j), η_{ij} es un valor heurístico correspondiente a la inversa de la distancia entre i y j, N_i^k es el conjunto de estaciones que la hormiga k puede visitar a continuación y α y β son parámetros que determinan la influencia relativa de la ruta de feromonas y de la información heurística, respectivamente. Si $\alpha = 0$ hay mayor probabilidad de elegir estaciones más cercanas y si $\beta = 0$ solo se da peso al valor acumulado de feromonas. Experimentalmente se ha encontrado que los valores óptimos para α y β son $\alpha = 1$ y $2 <= \beta <= 5$ [1].

- Añadir evaporación de feromonas. Cuando las hormigas han construido sus rutas y deben actualizar la feromonas, en lugar de simplemente depositar una unidad en los arcos pertenecientes a la ruta recorrida, primero se disminuirá la concentración de feromonas en todos los arcos por un valor constante y después se añadirán feromonas a los arcos que la hormiga ha visitado [1]. Es decir:

$$\tau_{ij} \leftarrow (1 - p)\tau_{ij}$$

Donde $0 < p <= 1$ es la tasa de evaporación. La función de este parámetro es evitar que se acumulen feromonas de forma ilimitada y favorece que se puedan olvidar caminos poco deseables recorridos anteriormente.

Después de esta evaporación, la hormiga dejará una cantidad de feromona $1 / C^k$ en cada arco recorrido, donde C^k es la distancia de la ruta construida por la hormiga k [1]. Al ser el inverso de la distancia, cuanto mejor sea la ruta obtenida por la hormiga, más cantidad de feromona depositará.

Incorporando estos cambios a la solución 3 y eligiendo un valor de evaporación de p = 0.5 (ver Anexo) se obtienen los siguientes resultados para el ejemplo de cinco estaciones:

Ruta: ['Alberto Alcocer', 'Agustín de Betancourt', 'Alcántara', 'Alcalá', 'Almadén']
Distancia total: 0.07108056080729643

El camión carga cuatro bicicletas en Alberto Alcocer (total camión = 4), se dirige a Agustín de Betancourt, donde deja las cuatro (total camión = 0). Seguidamente va a Alcántara, donde coge otras cuatro bicicletas (total camión = 4). Después deja tres de ellas en Alcalá (total camión = 1) y finalmente deja la última bicicleta en Almadén (total camión = 0). (ver Fig. 9)

Fig. 9: Solución 3 mejorada

Una vez descritas las distintas soluciones con el ejempo del subgrupo de cinco estaciones, se realizará un análisis comparativo de las mismas sobre el conjunto original de estaciones de BiciMAD.

2.3. Resultados

El conjunto de estaciones de BiciMAD utilizado corresponde al de junio de 2018 [8]. Del mismo se han eliminado dos estaciones debido a la falta de datos de latitud correspondientes. En total, se trabaja con 170 estaciones repartidas por la ciudad de Madrid.

Comparación en función de la distancia obtenida

En la primera parte del estudio comparativo se ejecutará cada una de las soluciones modificando el número de iteraciones para elegir la mejor ruta y se registrarán y representarán los resultados gráficamente.

Las iteraciones se corresponden con el número de hormigas simulado. Por ejemplo, la iteración número 10 se correspondería con la hormiga número 10

que recorre la ruta y por tanto, se habrían depositado feromonas de 9 hormigas anteriores.

Como valores para las distintas variables aplicadas en la Solución 3 mejorada, se han escogido valores que experimentalmente han demostrado producir buenos resultados [1]. Estos son:

p = 0,5
α = 1
β = 2

Solución 1

Iteraciones	Distancia mejor ruta
5	3,73099598258607
10	3,73099598258607
50	3,73099598258607
100	3,73099598258607
150	3,73099598258607
200	3,73099598258607
500	3,73099598258607
1000	3,73099598258607

Solución 2

Iteraciones	Distancia mejor ruta
5	3,84990787160869
10	3,66843768264665
50	3,62333436873924
100	3,57236441761795
150	3,47329370578576
200	3,5254658976256
500	3,48388177089268
1000	3,44734323186071

Solución 3

Iteraciones	Distancia mejor ruta
5	3,64459678434503

10	3,70127593434251
50	3,57977709367492
100	3,52737812425703
150	3,56473170488315
200	3,50815849555824
500	3,48189651363948
1000	3,45841341824674

Solución 3 mejorada

Iteraciones	Distancia mejor ruta
5	1,34237067444866
10	1,28598889708183
50	0,904733256619248
100	0,952496919156068
150	0,858777013903354
200	0,909908291359288
500	0,910396353453592
1000	0,854679459981572

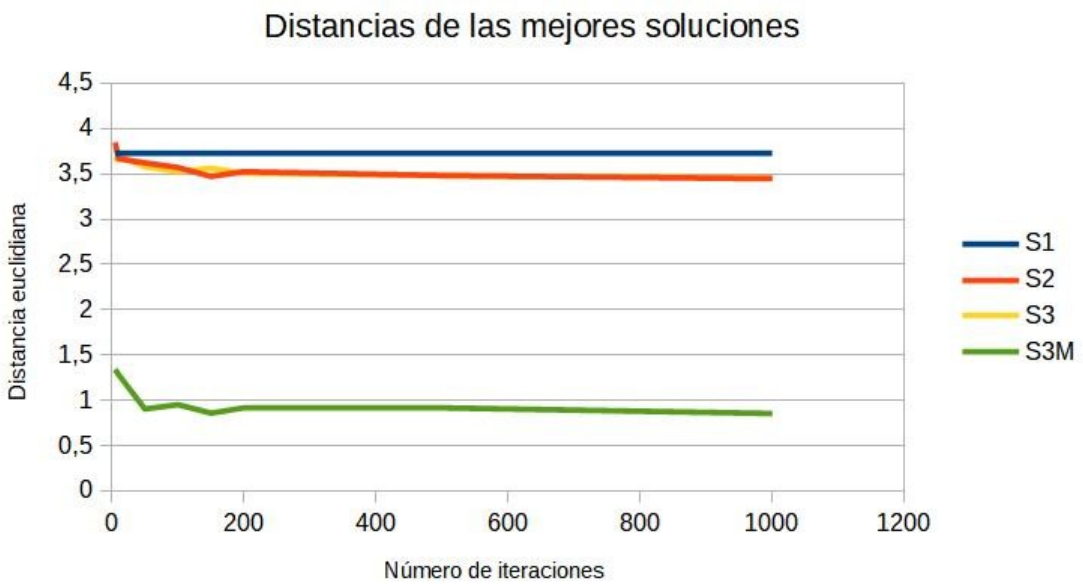

Fig. 10: Distancias de las mejores soluciones encontradas por cada algoritmo según el número de iteraciones

A simple vista se puede apreciar que, aunque para los ejemplos con cinco estaciones la diferencia entre los resultados era despreciable, al trabajar con el conjunto de datos total la solución basada en el algoritmo de la colonia de hormigas con las mejoras incorporadas se desmarca visiblemente del resto de soluciones propuestas. Analizando una a una cada solución se puede observar que:

La solución 1, correspondiente al algoritmo ingenuo que ordena las estaciones por demanda, obtiene los peores resultados de las cuatro. La mejor ruta obtenida es más larga que las obtenidas por las demás soluciones.

La solución 2 y la solución 3 ofrecen un resultado similar entre ellas, siendo a su vez ligeramente mejor que el de la solución 1. Por este motivo, se puede deducir que el simple hecho de añadir feromonas no ofrece beneficios con respecto a otro algoritmo que elige las estaciones a visitar de forma aleatoria.

La solución 3 mejorada, que añade evaporación y mejora las operaciones de inicialización de feromonas, actualización de feromonas y construcción de rutas, obtiene unos resultados notablemente mejores que el resto para cualquier número de iteraciones, mejorando además según aumentan las iteraciones, es decir, el número de hormigas.

Por lo tanto, gracias a este experimento se puede observar que las distancias obtenidas con el algoritmo ACO son menores que con las otras opciones. Sin embargo, esto no quiere decir necesariamente que el algoritmo de la colonia de hormigas funcione de forma eficiente. Para saber si el algoritmo ACO es una opción válida y eficiente en comparación con las demás, se estudiará el tiempo que tarda cada algoritmo en encontrar la mejor ruta.

Comparación en función del tiempo

Manteniendo el número de iteraciones del experimento anterior, se registrará el tiempo tardado por cada solución y se representará de forma gráfica el resultado.

Tiempo (s) de cada solución según el número de iteraciones:

Iteraciones	S1	S2	S3	S3M
5	1	1	4	1,52
10	1	1	7,48	2
50	1	1	34	5,45
100	1	1,26	69	9
150	1	1,4	102	12
200	1	1,63	138	15
500	1	5,67	338	32
1000	1	8	675	60

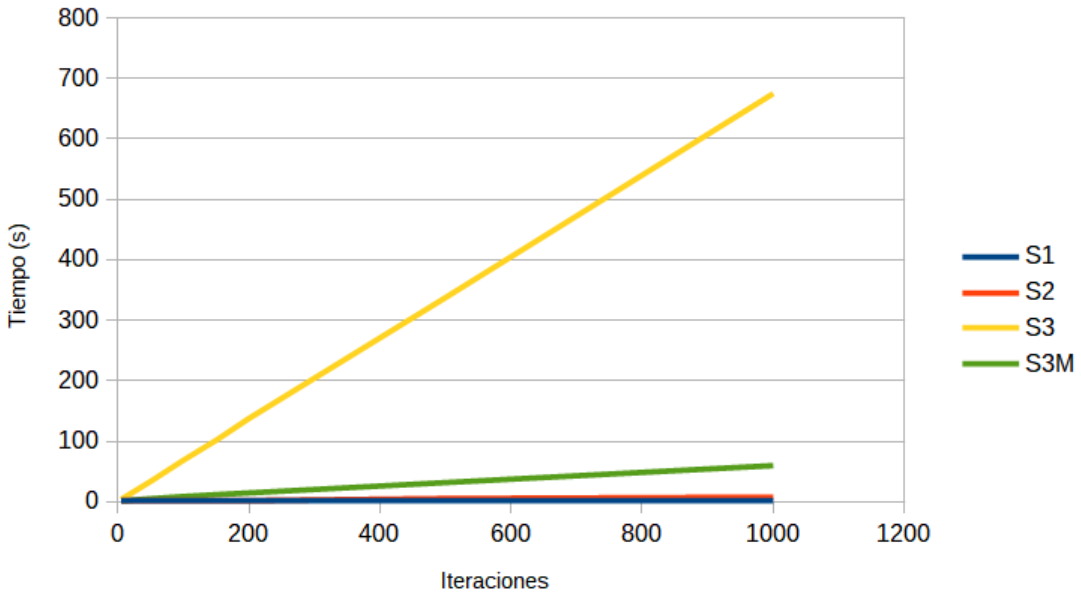

Fig. 11: Tiempo tardado en encontrar la mejor ruta según número de iteraciones

Como se puede ver, la solución 3, que añade feromonas sin mejoras, además de no presentar ninguna ventaja en cuanto a la calidad de las rutas obtenidas respecto a la solución 2, que escoge la siguiente estación a visitar de forma aleatoria, es mucho más lenta. En la solución 3, para elegir la siguiente estación a visitar siguiendo una probabilidad basada en la cantidad de feromonas depositadas, se construía una lista ponderada de todas las estaciones según las unidades de feromona en cada arco –por ejemplo, si un arco tiene cinco unidades de feromonas, lo añadía cinco veces a la lista– . Este proceso para 170 estaciones, según se va recorriendo el grafo y añadiendo feromonas a los arcos hace que la lista se vuelva demasiado grande y lastre todo el programa. Sin embargo, como la solución 3 solo es un paso intermedio

entre la aleatoria y el algoritmo de la colonia de hormigas completo, no se tuvo en cuenta su eficiencia a la hora de programarlo.

La solución 3 mejorada, es decir, el algoritmo ACO, presenta unos tiempos que, a pesar de ser algo mayores que en la solución aleatoria, son muy manejables. Por ejemplo, para 1000 hormigas, que es la prueba más grande realizada, tarda un minuto en calcular la mejor ruta. Por lo tanto, y teniendo en cuenta la diferencia de calidad de las rutas obtenidas –con 1000 hormigas tendríamos una ruta con una distancia euclidiana de 0,96 frente a los 3,58 de la versión aleatoria– podemos concluir que el algoritmo ACO, además de ser eficaz, también es un algoritmo eficiente.

3. Conclusiones

El estudio de la implementación del algoritmo de la colonia de hormigas aplicado al problema de reposición de bicicletas permite concluir que el algortimo ACO es un método eficiente para encontrar soluciones a este problema. Además, también se han obtenido las siguientes conclusiones:

- La observación de la naturaleza es una buena fuente de inspiración para la búsqueda de soluciones a problemas complejos.

- La aplicación directa de los fenómenos observados en la naturaleza no tiene por qué proporcionar los resultados esperados y puede ser necesario introducir modificaciones en su versión artificial que mejoren su comportamiento en los problemas estudiados.

Al inicio de este trabajo se había propuesto como objetivo principal encontrar una ruta óptima de reposición de bicicletas para el servicio BiciMAD aplicando el algoritmo de la colonia de hormigas. Debido a la necesidad de simplificar el problema, no se ha realizado una representación exacta de la realidad, por lo que los resultados obtenidos sirven para comparar los diferentes métodos utilizados y conocer cuál es más adecuado, pero no proporcionan la ruta óptima real.

En cuanto a los objetivos específicos, se habían considerado los siguientes:

• Demostrar que el algoritmo ACO es aplicable al problema estudiado.

• Conseguir un modelo lo más parecido posible al caso real con el que sea factible trabajar.

• Estudiar posibles mejoras.

• Analizar los resultados y deducir conclusiones en base a ellos.

Todos estos objetivos se han alcanzado satisfactoriamente.

Respecto a la planificación del trabajo, ha sido necesario un cambio en el enfoque inicial, ya que la librería que se había propuesto utilizar en un principio no se adaptaba totalmente al problema estudiado en este trabajo, sino que estaba enfocada a un problema TSP clásico. Después de analizar otras librerías similares, se llegó a la conclusión de que lo más conveniente era programar el algoritmo de forma íntegra aplicando los conocimientos obtenidos en la fase de documentación y, especialmente, del libro *Ant Colony Optimization* de Marco Dorigo y Thomas Stützle. A pesar de este imprevisto, el resto de plafinicación y metodología ha podido realizarse sin inconvenientes.

Como posibles líneas futuras de trabajo se proponen dos tipos de mejoras: las relacionadas con el propio algoritmo de la colonia de hormigas y las que proporcionarían una representación más realista del problema. Entre las primeras, el primer paso lógico sería la realización de pruebas modificando los valores de las variables. Por ejemplo, variando el grado de evaporación o cambiando los valores de α y β. Además, podrían implementarse algunas mejoras ya comentadas, como una estrategia elitista o un sistema basado en rango.

En cuanto a la representación del problema existen varias líneas de trabajo que se podrían seguir, como por ejemplo realizar una estimación de la demanda mediante técnicas de machine learning a partir de los movimientos de las bicicletas registrados por BiciMAD o sustituir la distancia euclidiana entre estaciones por una distancia real correspondiente al trazado de las carreteras.

4. Glosario

TSP (Travelling Salesman Problem): Problema clásico en ciencias de la computación. Su complejidad computacional es NP-Hard como problema de optimización y NP-Completo como problema de decisión.

Heurístico: Algoritmo que utiliza métodos aproximados para encontrar soluciones cercanas a las óptimas en un tiempo relativamente corto.

Metaheurístico: Conjunto de conceptos algorítmicos que se puede utilizar para definir métodos heurísticos.

Algoritmo de la colonia de hormigas (Ant Colony Optimization, ACO): Metaheurística basada en el comportamiento de las hormigas aplicable a problemas de optimización combinatoria.

Estigmergia (Stigmergy): Coordinación del trabajo mediante una comunicación indirecta basada en la modificación del ambiente.

Feromona: Sustancia química secretada por los seres vivos que modifica el comportamiento de otros individuos de la misma especie.

5. Bibliografía

[1] DORIGO, Marco; STÜTZLE, Thomas (2004). *Ant Colony Optimization* . Bradford Books.

[2] NP - Completeness [en línea] Wikipedia [Fecha de consulta: 20 de marzo de 2020] <https://en.wikipedia.org/wiki/NP-completeness>

[3] Travelling Salesman Problem [en línea] Wikipedia [Fecha de consulta: 20 de marzo de 2020]
<https://en.wikipedia.org/wiki/Travelling_salesman_problem >

[4] DORIGO, Marco; STÜTZLE, Thomas. «The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances» (2001) [en línea] Researchgate [Fecha de consulta: 25 de marzo de 2020]
<https://www.researchgate.net/publication/2330246_The_Ant_Colony_Optimization_Metaheuristic_Algorithms_Applications_and_Advances >

[5] SALAMA, Khalid Magdy; ABDELBAR, Ashraf M. «Extensions to the Ant-Miner Classification Rule Discovery Algorithm » (2010) [en línea] Researchgate [Fecha de consulta: 1 de abril de 2020]
<https://www.researchgate.net/publication/225868644_Extensions_to_the_Ant-Miner_Classification_Rule_Discovery_Algorithm >

[6] SRIVASTAVA, Sweta; SAHANA, Sudip Kumar. «A survey on traffic optimization problem using biologically inspired techniques» (2019) Springer Nature B.V.

[7] FAN, Yiwei; WANG, Gang; LU, Xiaoling. «Distributed forecasting and ant colony optimization for the bike-sharing rebalancing problem with unserved demands» (2019) [en línea] Plos One [Fecha de consulta: 25 de febrero de 2020]
<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0226204#sec021 >

[8] Portal de datos abiertos del Ayuntamiento de Madrid [en línea] Ayuntamiento de Madrid [Fecha de consulta: 25 de febrero de 2020].
<https://datos.madrid.es/sites/v/index.jsp?vgnextoid=374512b9ace9f310VgnVCM100000171f5a0aRCRD&buscar=true&Texto=bicimad&Sector=&Formato=&Periodicidad=&orderByCombo=CONTENT_INSTANCE_NAME_DECODE >

6. Anexo

Código fuente de las soluciones

Todo el código se encuentra disponible en el repositorio
https://github.com/Cleotron/TFG

Solución 1

```
import math

from bases_csv import get_bases
import data


def get_euclidean(a, b):
    return math.sqrt(pow(a.long - b.long, 2) + pow(a.lat - b.lat, 2))

def get_distance(base_list):
    distance = 0

    for x in range (0, len(base_list)-1):

        distance = distance + get_euclidean(base_list[x], base_list[x
+ 1])

    return distance

def get_path(bases):
    best = []
    for _ in range(1000):
            ordered = sorted(bases, key=lambda Base: Base.demand,
reverse=True)
        if (not best) or get_distance(ordered) < get_distance(best):
            best = ordered

    return best

def main():
    original = get_bases()

    #remove stations with demand = 0
    original = [b for b in original if b.demand != 0]

    result = get_path(original)
    print([o.name for o in result])
    print("total:" , get_distance(result))


if __name__ == "__main__":
```

```
    main()
```

Solución 2

```
import copy
import math
import random

from bases_csv import get_bases
import data


def get_euclidean(a, b):
    return math.sqrt(pow(a.long - b.long, 2) + pow(a.lat - b.lat, 2))

def get_distance(base_list):
    distance = 0

    for x in range (0, len(base_list)-1):

        distance = distance + get_euclidean(base_list[x], base_list[x
+ 1])

    return distance


#choose the best path in 100 iterations
def get_path(original):
    #truck initially empty
    truck = 0;

    #best path so far
    best = []

    for _ in range(150):
        bases = copy.deepcopy(original)
        visited = []

        while (len(bases) > 0):

            x = random.randrange(len(bases))

            visited.append(bases[x])

            if bases[x].demand > 0:
                truck = truck + bases[x].demand
                bases[x].demand = 0


            if bases[x].demand < 0:
                if abs(bases[x].demand) > truck:
                    bases[x].demand = bases[x].demand + truck
```

```
                    truck = 0
                else:
                    truck = bases[x].demand + truck
                    bases[x].demand = 0

            if bases[x].demand == 0:
                bases.remove(bases[x])

                    if (not best) or (get_distance(visited) <
get_distance(best)):
            best = visited

    return best

def main():
    original = get_bases()

    #remove stations with demand = 0
    original = [b for b in original if b.demand != 0]

    result = get_path(original)
    print([o.name for o in result])
    print("total:" , get_distance(result))

if __name__ == "__main__":
    main()
```

Solución 3

```
      import math
      import random

      from bases_csv import get_bases
      import data


      def get_euclidean(a, b):
            return math.sqrt(pow(a.long - b.long, 2) + pow(a.lat -
      b.lat, 2))

      def get_distance(base_list):
          distance = 0

          for x in range (0, len(base_list)-1):

                  distance = distance + get_euclidean(base_list[x],
      base_list[x + 1])

          return distance


      def set_pheromones(phe, original):
```

```
        for x in original:
            phe[x.name] = {}
            for y in original:
                phe[x.name][y.name] = 1


    def choose_next(now, bases, visited, phe):
        weighed = []
        for i, base in enumerate(bases):
            if base not in visited:
                pherom = phe[bases[now].name][base.name]
                for _ in range(pherom):
                    weighed.append(i)
        return random.choice(weighed)

    def get_path(original):

        #truck initially empty
        truck = 0;

        #best path so far
        best = []

        #pheromone in each branch
        #every branch starts with 1 unit
        phe = {}
        set_pheromones(phe, original)

        #choose the best path in 100 iterations
        for _ in range(50):
            bases = original.copy()
            visited = []

            #first base is chosen at random
            now = random.randint(0, len(bases) - 1)

            while len(bases) > len(visited):
                next = choose_next(now, bases, visited, phe)

                #update pheromones
                phe[bases[now].name][bases[next].name] += 1

                if bases[next].demand > 0:
                    truck = truck + bases[next].demand
                    bases[next].demand = 0


                if bases[next].demand < 0:
                    if abs(bases[next].demand) > truck:
                          bases[next].demand = bases[next].demand +
    truck
                        truck = 0
                    else:
```

```
                        truck = bases[next].demand + truck
                        bases[next].demand = 0


                now = next

                #update visited
                visited.append(bases[next])

                    if (not best) or (get_distance(visited) <
    get_distance(best)):
                best = visited

        return best

    def main():
        original = get_bases()

        #remove stations with demand = 0
        original = [b for b in original if b.demand != 0]

        result = get_path(original)
        print([o.name for o in result])
        print("total:" , get_distance(result))


    if __name__ == "__main__":
        main()
```

Solución 3 mejorada

```
import copy
import math
import random

from bases_csv import get_bases
import data
import solucion1


def get_euclidean(a, b):
    return math.sqrt(pow(a.long - b.long, 2) + pow(a.lat - b.lat, 2))

def get_distance(base_list):
    distance = 0

    for x in range (0, len(base_list)-1):

        distance = distance + get_euclidean(base_list[x], base_list[x
+ 1])

    return distance
```

```
def set_pheromones(phe, original, ants):
    d = get_distance(solucion1.get_path(original))
    init_phe = ants/ d
    for x in original:
        phe[x.name] = {}
        for y in original:
            phe[x.name][y.name] = init_phe

def phe_evaporation(phe, bases, p):
    for x in bases:
        for y in bases:
            phe[x.name][y.name] = phe[x.name][y.name] * (1 - p)


def update_pheromones(phe, visited, deposit):
    for i in range(len(visited) - 1):
        phe[visited[i].name][visited[i + 1].name] = (
            phe[visited[i].name][visited[i + 1].name] + deposit
        )

def choose_from_list(bases, l, n):
    sum = 0
    for i, element in enumerate(l):
        if element != 0:
            sum += element
            if sum >= n:
                return i
    return -1

def choose_next(now, bases, phe, truck):

    prob_list = []
    sum_prob = 0
    for i, base in enumerate(bases):
        phi = phe[bases[now].name][base.name]
        d = get_euclidean(bases[now], base)
       if d == 0 or base.demand == 0 or (base.demand < 0 and -
base.demand > truck):
            prob = 0
        else:
            prob = phi / (d**2)
        sum_prob += prob
        prob_list.append(prob)
    n = random.uniform(0.0, sum_prob)
    return choose_from_list(bases, prob_list, n)

def get_path(original, ants):

    #truck initially empty
    truck = 0;
```

```
    #best path so far
    best = []

    #pheromone in each branch
    #every branch starts with 1 unit
    phe = {}
    set_pheromones(phe, original, ants)

    #choose the best path in 100 iterations
    for _ in range(ants):
        bases = copy.deepcopy(original)
        visited = []

        #first base is chosen at random
        now = random.randint(0, len(bases) - 1)

        while any([b.demand < 0 for b in bases]):
            next = choose_next(now, bases, phe, truck)

            if bases[next].demand > 0:
                truck = truck + bases[next].demand
                bases[next].demand = 0


            if bases[next].demand < 0:
                if abs(bases[next].demand) > truck:
                    bases[next].demand = bases[next].demand + truck
                    truck = 0
                else:
                    truck = bases[next].demand + truck
                    bases[next].demand = 0


            now = next

            #update visited
            visited.append(bases[next])

        #pheromone evaporation
        phe_evaporation(phe, bases, 0.5)

        #update pheromones
        deposit = 1 / (get_distance(visited))
        update_pheromones(phe, visited, deposit)

                    if (not best) or (get_distance(visited) <
get_distance(best)):
            best = visited

    return best

def main():
    original = get_bases()
```

```
    #remove stations with demand = 0
    original = [b for b in original if b.demand != 0]

    result = get_path(original, 1000)
    print([o.name for o in result])
    print("total:" , get_distance(result))


if __name__ == "__main__":
    main()
```