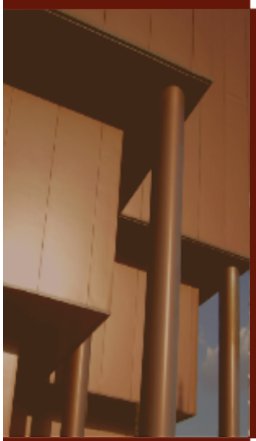
Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Resolución de un VRP con ventanas móviles mediante el uso del algoritmo Cuckoo Search

Autor: Carlos Uribe Astolfi
Tutor: Alejandro Escudero Santana



Dep. Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Máster
Máster Universitario en Ingeniería Industrial

Resolución de un VRP con ventanas móviles mediante el uso del algoritmo Cuckoo Search

Autor:
Carlos Uribe Astolfi

Tutor:
Alejandro Escudero Santana
Profesor ayudante doctor

Dep. Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Máster: Resolución de un VRP con ventanas móviles mediante el uso del algoritmo Cuckoo Search

Autor: Carlos Uribe Astolfi

Tutor: Alejandro Escudero Santana

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Resumen

En este Trabajo Fin de Máster se trata de resolver un problema de rutado de vehículos destinado al e-commerce (EC-VRP) mediante un algoritmo bioinspirado, el algoritmo de la Búsqueda Cuco o Cuckoo Search (CS). El EC-VRP cuenta con las mismas características que el VRPTW (con ventanas temporales) tradicional, pero con la peculiaridad de que cada cliente cuenta con tres localicaziones distintas donde se les puede entregar la mercancía, teniendo que realizar la entrega en tan solo una de ellas. Además, estas localizaciones pueden contar con ventanas temporales disntintas.

Abstract

This work aims to solve a vehicle routing problem for e-commerce (EC-VRP) using a bioinspired algorithm, the Cuckoo Search (CS) algorithm. The EC-VRP has the same characteristics as the traditional VRPTW (with time window constraints), but with the peculiarity that each customer has three different locations where the goods can be delivered, having to make the delivery in only one of them. In addition, these locations have different time windows.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS DEL TRABAJO

Dentro de los costes logísticos de una empresa, los de transporte de mercancías constituyen una de las partes más importantes, representando un porcentaje mayoritario. Esto se aprecia en el siguiente gráfico, extraído del informe "Observatorio de la logística en España" realizado por la Dirección General de Transporte Terrestre del Ministerio de Fomento, donde se recoge el porcentaje de los costes operativos de las empresas logísticas.

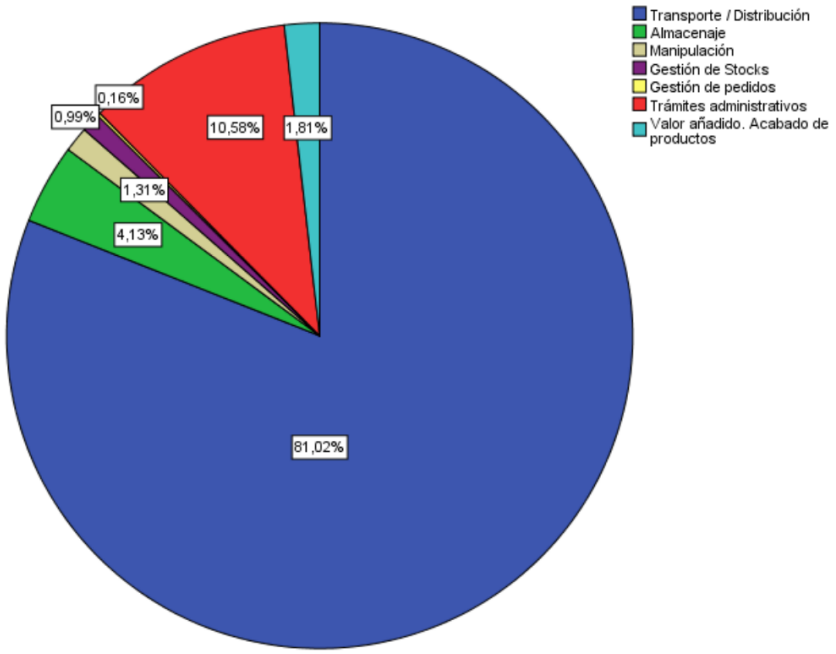


Figura 1. Porcentajes de Costes Operativos por servicio ofrecido. Empresas con actividades de transporte y almacenamiento (una de ellas como actividad primaria). Encuesta Opinómetre (2011).

El sector de la logística debe enfrentarse a una serie de desafíos complejos entre los que se encuentran: el precio del petróleo; costes medioambientales; peajes en autopistas/impuestos; congestión en carreteras; falta de capacidad en infraestructuras; coste material; envejecimiento de la flota de transporte; logísticas cada vez más compleja; procesos cada vez más globales; etc.

Además de estos factores, en el transporte de mercacias asociados a negocios del tipo ecommerce, aparece la posibilidad de que el cliente reciba la mercancía en localizaciones y espacio de tiempos (ventanas temporales) distintos. Esto implica que el grado de dificultad para la resolución de este tipo problema sea aún mayor. A la vista de estos datos y factores, se observa la necesidad de programar rutas de transporte mejor diseñadas. Aquí es donde reside el fin del presente trabajo, tratando de proporcionar soluciones numericamentes exactas, en un tiempo razonable.

Para tratar de proporcionar estas soluciones, se llevará a cabo la implementación de un algoritmo bioinspirado, concretamente basado en la estratégia de reproducción del pájaro cuco,

conocido como *Búsqueda Cuco* o *Cuckoo Search (CS)*.

Además, se implementará una posible mejora al método CS tradicional y se compararán los resultados obtenidos.

1.1 Objetivos

Mediante la realización de este trabajo se persigue la minimización de los costes de transporte mediante la implementación de herramientas de optimización de rutas. Concretamente, se trata de resolver un problema de rutado de vehículos, añadiéndole características propias del comercio electrónico (la posibilidad de entregar la mercancía a un cliente, en varias localizaciones y con espacios de tiempo distintos).

Además de perseguir una reducción de costes, también se trata de reducir el consumo de gasolina y, por tanto, la emisión de gases a la atmósfera debido a que los vehículos estarán menor tiempo en funcionamiento. Consecuentemente, la planificación de rutas implicará mejorar la calidad del servicio al cliente. Por lo tanto, se pueden establecer como objetivos secundarios la reducción de emisiones de gases, así como aumentar de la calidad del servicio.

1.2 Estructura del trabajo

Tratando de cumplir con estos objetivos, el trabajo se ha estructurado en 7 capítulos que se describen a continuación:

En el primer capítulo, el presente, se realiza una descripción del contexto del problema, se explican los objetivos y la estructura a seguir para cumplir con estos.

Mediante el segundo, se pretende realizar una breve presentación sobre el comercio electrónico, a fin de aclarar en qué consiste y proporcionar algunos datos que resalten su importancia. Para ello se realiza una descripción de los distintos tipos que existen, se exponen los beneficios y limitaciones del mismo y, por último, se analizan distintos datos del comercio electrónico *Business-to-Consumer* en España.

El tercer capítulo recoge un estado del arte sobre el problema que se va a resolver, el VRP, y el algoritmo aplicado para su resolución, la Búsqueda Cuco.

En el cuarto, se desarrolla el problema de rutados de vehículos con las características propias del ecommerce (EC-VRP), explicando detalladamente en que sonsiste. Concretamente, se profundizará en el VRPTW (con ventanas temporales), ya que el problema a resolver es similar a este tipo. La característica adicional que presenta el EC-VRP es que un cliente cuenta con tres localizaciones diferentes donde se le puede entregar la mercancía, cada una de ellas con una ventana temporal distinta. Donde se establece la condición de visitar a cada cliente en tan solo una de estas localizaciones.

Tras detallar en que consiste el problema a resolver, mediante el quinto capítulo, se explica en detalle y de forma teórica en que consiste y se sustenta el algoritmo elegido para resorverlo, la Búsqueda Cuco o *Cuckoo Search* (CS). Además, se recoge cómo se adapta dicho algoritmo para resolver el problema mencionado.

Mediante el sexto capítulo se procede a la obtención de resultados y análisis de los mismos, tras la aplicación del algoritmo CS al problema EC-VRP. Se comparán los resultados alcanzados frente a los obtenidos en otros trabajos, tras la aplicación de otras meta-heurísticas tradicionales (Búsqueda Tabú, Recocido Simulado y Algoritmo Genético) a la misma batería de problemas. Finalmente, en este capítulo también se presentan los resultados obtenidos por un algoritmo CS

modificado respecto a la formulación tradicional, con la intención de mejorar los resultados de este.

Por último, se recogen las conclusiones más significativas derivadas del trabajo.

2 EL COMERCIO ELECTRÓNICO

"*El comercio electrónico, también conocido como e-commerce (electronic commerce en inglés) o bien negocios por Internet o negocios online, consiste en la compra y venta de productos o de servicios a través de medios electrónicos, tales como Internet y otras redes informáticas.*" (wikipedia.org)

"*El e-commerce o comercio electrónico es un método de compraventa de bienes, productos o servicios valiéndose de internet como medio, es decir, comerciar de manera online.*" (debitoor.es)

"El e-commerce consiste en la distribución, venta, compra, marketing y suministro de información de productos o servicios a través de Internet." (marketingdigital.bsm.upf.edu)

Sobre comercio electrónico o *e-commerce* se pueden encontrar decenas de definiciones en la red, como se puede notar en las citas anteriores. Entre todas, se aprecia un denominador común: se trata de la compra-venta de un producto o servicio a través de medios electrónicos. Mediante este capítulo se pretende realizar una breve presentación sobre el comercio electrónico, a fin de aclarar en qué consiste y proporcionar algunos datos que resalten su importancia.

2.1 Tipos de comercio electrónico

Existen diversos tipos de comercio electrónico dependiendo de la naturaleza de sus transacciones y la forma de generar sus ingresos. A continuación, se encuentra una muestra de entre los distintos tipos que existen:

- *Business-to-Business* (B2B): Empresas que comercian con otras empresas u organizaciones. Un ejemplo sería el aprovisionamiento de materiales de una empresa mediante un proveedor.

- *Business-to-Consumer* (B2C): Empresas que comercian con consumidores. Es el más habitual. Un ejemplo, sería la compra de un libro por un consumidor final a una empresa.

- *Business-to-Government* (B2G): Empresas que comercian con instituciones del gobierno.

- *Consumer-to-Consumer* (C2C): Comercio entre particulares, es decir, consumidores que compran y venden a otros clientes.

- *Consumer-to-Business* (C2B): Consumidores que venden a negocios, muy popular en productos de segunda mano.

2.2 Beneficios y limitaciones del comercio electrónico

El comercio electrónico presenta un gran número de ventajas, tanto para las empresas como para los consumidores. A continuación, se encuentran algunas de ellas:

- Desaparición de los límites geográficos y temporales: Acceso al catálogo de productos desde cualquier parte y a cualquier hora, 24 horas los 365 días del año.
- Reducción de costes de producción, administración y almacenamiento.
- Aumento de visibilidad: mayor número de clientes, tanto *online* como *offline*.
- Eliminación de intermediarios: Contacto directo con clientes.
- Métodos de pagos flexibles.
- Facilidad para ofrecer una comparativa entre productos, incluyendo características y precios.

En contraposición, también se encuentran algunas desventajas. Entre ellas se encuentran:

- Inseguridad en las transacciones: Desconfianza por parte del cliente por el método de pago y la calidad del producto. La frecuencia del fraude económico no ayuda a este tipo de comercio creando ciertas inseguridades.
- Intangibilidad: La imposibilidad de poder comprobar y tocar los productos afecta a la decisión de los clientes a la hora de realizar las compras.
- Gastos de envío: Pueden llegar a ser un inconveniente para ciertos negocios con volúmenes de ventas pequeños.
- Gastos de promoción: Situar el producto entre la competencia es un trabajo complejo que a veces requiere recurrir a un profesional, aumentado los gastos.
- Retos logísticos: Se han realizado diversos modelos logísticos específicos para el comercio electrónico, debido a que se espera de él un menor tiempo de distribución frente al comercio tradicional.

2.3 El comercio electrónico B2C en España

Entre los diversos tipos de comercio electrónico que existen, el B2C es el más habitual. En España, el Observatorio Nacional de las Telecomunicaciones y de la SI, ONTSI en adelante, ha realizado un informe sobre el mismo. Este informe se denomina "*Estudio sobre Comercio Electrónico B2C 2015*" y, a continuación, se muestran algunos de los resultados que se recogen en el Resumen Ejecutivo de dicho documento, destacando la importancia de este tipo de negocios.

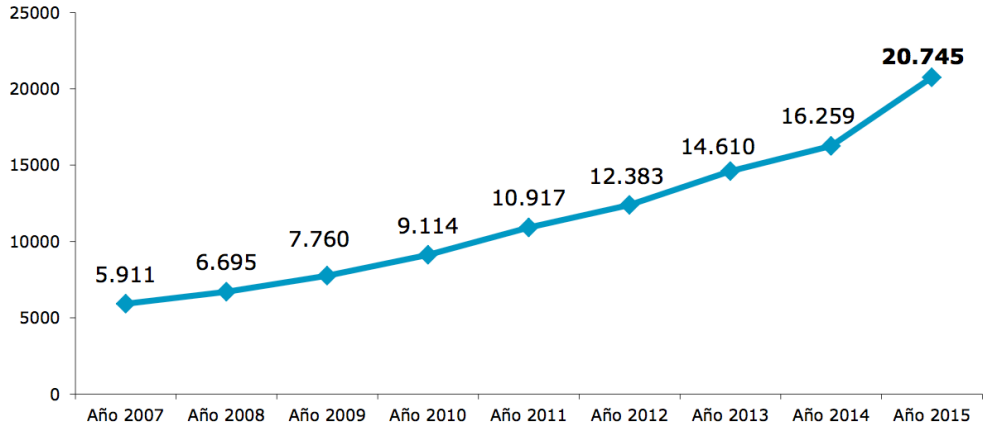


Figura 2. Volumen de comercio electrónico B2C (millones de euros) (Fuente: ONTSI)

El comercio electrónico B2C en España ha seguido creciendo durante 2015. Como se observa, la estimación de la cifra del volumen total del sector es de 20.745 millones de euros, incrementándose un 27,5% respecto a 2014. Este incremento se debe a la evolución de tres variables principales para estimar el volumen del B2C: el número de internautas, la proporción de estos que realizan compras por internet y el gasto medio por comprador.

Pese a que el comportamiento de los compradores online presenta cierta madurez, se observan tendencias que insinuan ciertos cambios de hábitos:

- Aunque la mayoría de compras se realizan desde el hogar (92,2%), la compra en movilidad continua una tendencia ascendente situandose en el 12,9%.
- Se mantiene la irregularidad en la frecuencia. Seis de cada diez compradores no tienen una frecuencia fija en este tipo de operaciones.
- El número medio de categorías de producto compradas por comprador se sitúa en 3,62, ligeramente por encima de la cifra de 2014 (3,53).
- En la búsqueda de información y comparación de precios, la búsqueda online incrementa significativamente su presencia (un 74,1% de los compradores recurren a este medio). Además, aunque el ordenador sigue siendo el principal dispositivo para ello, se observa un crecimiento en el uso de los dispositivos móviles.
- Las webs que venden exclusivamente por Internet se consolidan como primer canal de compra (62,7%), seguidos por las webs de fabricante (42,9%). Las tiendas con establecimiento físico y venta online experimentan un empuje importante respecto a 2014 (41,2% frente a 36,7%).

En cuanto a la experiencia concreta de la compra en Internet:

- Un 14,8% de los compradores ha tenido problemas con las compras realizadas en 2015 (retraso de entrega o producto en mal estado, entre otros).
- Se incrementa ligeramente el número de compradores que han devuelto algún producto (16,6% en 2015 frente a 14% en 2014) y casi 8 de cada 10 han puesto una reclamación cuando han tenido problemas con su compra.
- El conocimiento de los sellos de calidad o códigos de confianza en Internet no se ha incrementado en 2015: un 40,5% de los compradores online declara tener en cuenta si la página web está adherida a un sello de confianza en Internet, mientras que 6 de cada 10 desconocen lo que son.
- Los costes de envío y la garantía de devolución, así como de cambio del producto son aspectos cuya mejora se demanda con mayor intensidad entre los usuarios. Garantizar un buen uso de los datos personales también es un aspecto susceptible de mejora.

A partir de estos datos, se intenta demostrar la importancia actual del comercio electrónico y que es un sector consolidado que seguirá creciendo. También se aprecia que muchos de los problemas están relacionados con la logística de este tipo de comercio. Este último factor, resalta la importancia de trabajos como el que se presenta en este documento, intentando mejorar y facilitar el proceso logístico de este tipo de comercios.

3 ESTADO DEL ARTE

En el presente capítulo, se presenta un breve estado del arte sobre el problema que se va a resolver, el VRP, y el algoritmo aplicado para su resolución, la Búsqueda Cuco.

3.1 El Problema de rutado de vehículos - VRP

El Problema de Rutado de Vehículos (*Vehicle Routing Problem*) o VRP se puede definir de distintas formas. La manera más clásica de hacerlo es como un grafo G=(N,L), siendo N un conjunto de nodos N={0,1,…,i,…,n} y L un conjunto de arcos que conectan los nodos. El nodo 0 se conoce como almacén y cuenta con *m* vehículos idénticos con una capacidad *W*. El número de vehículos *m* puede ser, o no, una variable del problema. El resto de nodos $i > 0$, representan a un cliente con una demanda positiva q_i. Cada arco (*i,j*) cuenta con un coste asociado $c_{ij} = c_{ji}$, que normalmente representa la distancia existente entre nodos. El VRP consiste en encontrar las rutas que satisfagan la demanda de ciertos clientes al menor coste posible, de manera que la ruta comience y acabe en el almacén, se visite a cada uno de los clientes exactamente una vez y la demanda total de cada ruta no supere la capacidad *W* de los vehículos. Esta variante del VRP, conocida como *capacitated* VRP (CVRP), es una de las más importantes (Teymourian et al. 2016) y ha sido objeto de estudio de diversos autores (Alssager, M., Othman 2013; Teymourian et al. 2016).

Otra de las variantes a destacar es el VRP con ventanas temporales (VRPTW), también analizado por distintos artículos académicos (Cordone & Calvo 2001; Bräysy & Gendreau 2005; Pisinger & Ropke 2007). Esta variante se puede considerar como una ampliación del CVRP, dado que cuenta con las mismas restricciones. Además, se le añaden las denominadas "ventanas temporales", que aumentan la complejidad del problema. Estas ventanas temporales consisten en que el cliente impone una franja horaria en la que el pedido puede ser entregado. Además de estas, existen otras variantes como pueden ser el *multi-depot* VRP (MDVRP) o el *Pick-up and Delivery* VRP (PDP), entre otros.

En contraste con la definición clásica del VRP, al llevarlo a la práctica, aparecen dos nuevas dimensiones: evolución y la calidad de la información (Pillac et al. 2013). Una vez diseñada la ruta y llevándola a cabo, puede aparecer un nuevo pedido por parte de un cliente. A esto hace referencia a la evolución de la información, de manera que puede cambiar la información da la ruta durante su transcurso. En cuanto a la calidad de la información, se cuenta con un rango estimado de la demanda real y esto genera incertidumbre. Debido a estos factores y dependiendo del problema, se puede clasificar los Problema de Rutado de Vehículos como dinámicos o estáticos. Estático si los datos de entrada (demandas, tiempos entre desplazamientos, etc) no dependen explícitamente del tiempo, dinámicos si lo hacen. Además, el problema será determinista, si los datos de entrada se conocen a la hora de diseñar las rutas. En caso contrario, será estocástico. En la Tabla 1, se recoge esta clasificación que se comenta. Añadir, además, que aquellos problemas que son dinámicos y deterministas se conocen como *online* o *real time* VRP (Ghiani et al. 2003).

		Calidad de la información	
		Determinista	Estocástico
Evolución de la información.	No depende del tiempo	Estático y determinista	Estático y estocástico
	Dependiente del tiempo	Dinámico y determinista	Dinámico y estocástico

Tabla 1. Clasificación VRP. (Fuente: Pillac et al. (2013))

El VRPTW ha sido objeto de intensas investigaciones. A la hora de resolverlo, se han estudiado tanto métodos heurísticos como exactos. Debido a la complejidad que presenta este problema y a la aplicabilidad en situaciones reales, son necesarios métodos capaces de resolverlo con una cierta calidad en sus resultados y en un tiempo limitado. Es por esto último, que los métodos exactos se han ido dejando de lado, y las últimas investigaciones se han centrado en métodos heurísticos, principalmente en metaheurísticas (Braysy & Gendreau 2005).

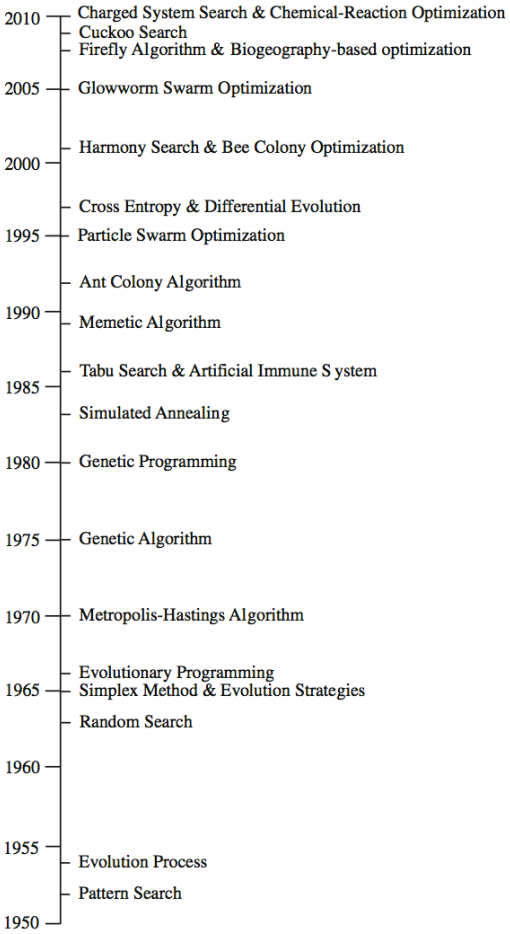


Figura 3.Cronología de las metaheurísticas. (Fuente: (Gandomi et al. 2013))

Tradicionalmente, las metaheurísticas que más se han utilizado para la formulación de los algoritmos y la resolución de este tipo de problemas han sido la Búsqueda Tabú (TS) (Taillard et al. 1997) y el Algoritmo Genético (GA). Esto no quiere decir que hayan sido los únicos métodos utilizados, como se observa en la Figura 3. A su vez, se han empleado otras que no aparecen en dicha

figura, como el *Adaptive Large Neighborhood Search* (ALNS) (Pisinger & Ropke 2007) u otras que hacen uso combinado de TS y GA (Prins 2004). Además en los últimos años, se han desarrollado otros algoritmos inspirados en la naturaleza, *nature inspired* (Yang & He 2016). Entre ellos, destacar la Búsqueda Cuco o *Cuckoo Search* (CS). Estudios preliminares, muestran que esta metaheurística es muy prometedora, obteniéndose resultados que incluso superan a algoritmos como el GA (Yang & Deb 2009).

Finalmente, añadir que para la validación de los distintos algoritmos se recurre a unos "problemas de referencias", que consisten en distintos escenarios donde se aplican los algoritmos, obteniéndose unos resultados. De esta forma se puede comparar la efectividad de un algoritmo frente a otro. En el VRPTW, por ejemplo, el escenario de referencia más utilizado es el de Solomon (Solomon 1987), empleado por distintos autores (Taillard et al. 1997; Bräysy & Gendreau 2005). En él se generan seis series de problemas variando diversos factores. Estos pueden ser: datos geográficos, número de clientes en una ruta o alguna característica de las ventanas temporales.

3.2 La Búsqueda Cuco – Cuckoo Search (CS)

En todas o casi todas las aplicaciones industriales o ingenieriles, se intenta alcanzar una solución óptima. Bien minimizando costes o el consumo energético, o bien maximizando el beneficio. Para ello existen herramientas y métodos que tratan de alcanzar dicho óptimo. Estas son conocidas como algoritmos de optimización. Alcanzar dicha solución es algo complejo en los problemas reales, debido a la incertidumbre que los rodea.

Un algoritmo, desde un punto de vista matemático, se puede definir como un procedimiento que genera una solución a partir de una serie de datos (Yang & Deb 2014). Si para resolver un problema el algoritmo que se utiliza no tiene en cuenta ningún tipo de información sobre el mismo, será más versátil a la hora de abordar otros problemas. Esto implica, en muchos casos, la pérdida de eficacia del mismo. Cuanto mayor se adapta un algoritmo a un problema, más eficiente será.

Los algoritmos se pueden clasificar en deterministas o estocásticos (Yang & Deb 2014). Deterministas serán aquellos que trabajan de forma mecánica y sin ningún factor aleatorio en su aplicación. Se caracterizan porque siempre alcanzan la misma solución ante los mismos datos de entrada. Por el contrario, en cuanto exista algún factor aleatorio, el algoritmo normalmente generará distintas soluciones cada vez que se ejecuta, aún en el caso en que los datos iniciales sean los mismos. Estos últimos se denominan estocásticos. En concreto, conocidos como algoritmos heurísticos o metaheurísticas.

En la última década, las metaheurísticas que han tomado mayor importancia son aquellas basadas en comportamientos propios de la naturaleza (*nature inspired*). Entre ellas se encuentra la que será objeto de estudio en este trabajo, la Búsqueda Cuco.

La Búsqueda Cuco o *Cuckoo Search* (CS) es un algoritmo desarrollado por Yang y Deb en 2009 (Yang & Deb 2009). Es un algoritmo relativamente nuevo y viene trayendo muy buenos resultados en su aplicación a problemas de optimización (Yang & Deb 2010). Dicho algoritmo se caracteriza por (Alssager, M., Othman 2013):

i. Una estrategia de selección sencilla.

ii. Uso de *Lévy Flight*, que proporciona un equilibrio entre los factores de exploración y explotación del algoritmo.
iii. Uso de pocos parámetros para ajustar el algoritmo. Lo cual lo hace más genérico y adaptable a diversidad de problemas.

La Búsqueda Cuco está basada en la agresiva estrategia de reproducción que tienen los pájaros cucos. Algunas especies de esta ave depositan sus huevos en nidos de otros pájaros (normalmente otras especies), llegando incluso a expulsar los huevos del nido invadido para aumentar la probabilidad de que nazcan los suyos. Si el pájaro anfitrión se percata que el huevo no es el propio, lo sacará del nido o directamente lo abandonará y construirá otro nido. Otra característica del pájaro cuco, es que deposita sus huevos en nidos donde las crías de la otra ave tardan más en nacer. De esta manera, cuando nace la cría de cuco, ésta arroja fuera del nido el resto de huevos, de forma que el pájaro anfitrión solo alimentará a la cría del cuco. Abandonando, la cría, el nido cuando termina de ser criada.

Sobre dichas características se sustenta la CS. Después de cada iteración, las peores soluciones son desechadas y se generan nuevas. Como si los huevos fueran identificados y abandonados y se buscan nuevos nidos. El algoritmo se podría resumir de la siguiente forma (Alssager, M., Othman 2013); cada cuco pone un huevo, en cada iteración, y lo deposita aleatoriamente en un nido. Los mejores nidos, aquellos con los huevos de mayor calidad, se conservarán para la siguiente generación. El número de nidos disponible es fijo y siempre existe la probabilidad de que el huevo sea descubierto por el pájaro anfitrión. Para complementar el algoritmo, se debe añadir la forma de generar los huevos, que se realiza mediante *Lévy Flight*.

La Búsqueda Cuco (CS) ha sido empleada en campos muy diversos de optimización e inteligencia computacional, obteniéndose resultados de gran eficiencia en muchos de ellos. Además, se han realizado otras versiones del mismo con la intención de mejorar el rendimiento del algoritmo estándar. Por ejemplo, el realizado por Walton en al. (Walton et al. 2011), *modified cuckoo search* (MCS), que demuestra ser muy eficiente para resolver problemas no lineales. Yildiz (Yildiz 2013) realiza la primera aplicación de la CS a un problema de producción, optimizando los parámetros de una fresadora. En el libro *Nature-Inspired Computation in Engineering* (Yang & He 2016) se muestran distintos tipos de problemas que se han resuelto mediante la Búsqueda Cuco, con la intención de mostrar la diversidad de su aplicación. Estos son: el problema de asignación cuadrática, el problema de mochila, el problema del agente viajero, el de asignación y el problema de rutado de vehículos (VRP).

Analizando el algoritmo CS, Civicioglu & Besdok (2013) comparan la CS con el algoritmo de Optimización por Enjambre de Partículas (PSO), Evolución Diferencial (DE) y *Artificial Bee Colony* (ABC). Deducen que los algoritmos de la Búsqueda Cuco y la Evolución Diferencial generan mejores resultados que los PSO y ABC. Por otro lado, Gandomi et al. (Gandomi et al. 2013) aplica el algoritmo CS para resolver una serie de problemas estructurales. Además, para su validación, compara la CS con otros algoritmos aplicados en este campo. Estos también acaban concluyendo que con la Búsqueda Cuco se obtienen mejores resultados que con otros algoritmos, como el PSO o el algoritmo genético (GA).

Por último, añadir que los autores que desarrollaron originalmente de la CS, Yang y Deb,

también han adaptado el algoritmo para problemas de optimización multiobjetivo (Yang & Deb 2013). Denominándolo como *Multiobjective Cuckoo Search* (MOCS). Este lo aplican a problemas de diseño, como el de un freno de disco o de una viga soldada.

4 ECOMMERCE-VRP

En este capítulo se pretende explicar en detalle el problema objeto de resolución del presente trabajo, el *E-commerce Vehicle Routing Problem* (EC-VRP). Debido a que este problema es una variante del VRP con ventanas temporales (VRPTW) y para lograr su mejor comprensión, las explicaciones que se presentan a continuación, se apoyaran en la base teórica de este último. Para ello, se expone a continuación la base teórica que presentan Cordone & Calvo (2001).

El problema de rutado de vehículos surge cada vez que un conjunto de vehículos se dispone a servir una serie de pedidos. Cada pedido establece una localización distinta a la que debe acudir alguno de estos vehículos, además suelen existir una serie de condiciones que restringen el orden con el que realizar estos pedidos. Entre las restricciones más comunes se encuentran el tiempo o la propia capacidad de los vehículos. Con capacidad se hace referencia a la cantidad total de pedidos que puede cargar un vehículo en un instante determinado. Por otro lado, el tiempo expresa la posibilidad de realizar un servicio en una determinada franja horaria, conocida como ventana temporal o *time window*.

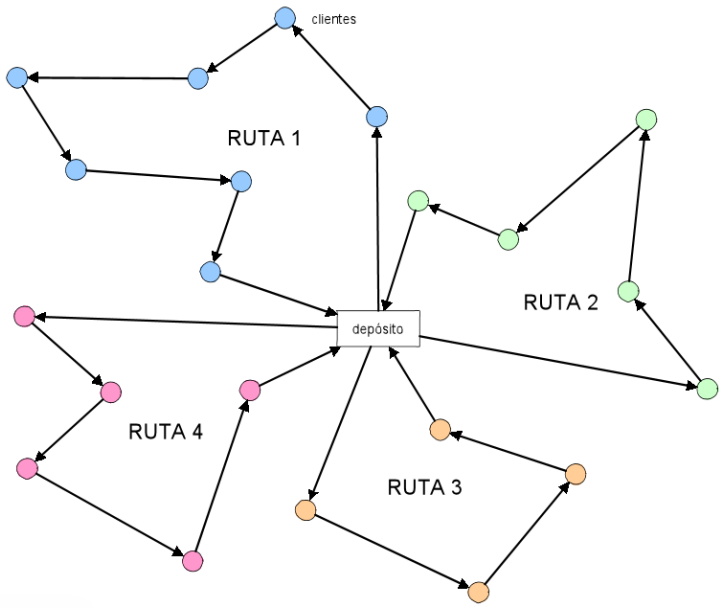


Figura 4. Representación genérica VRP (Fuente: Wikipedia)

De forma general, el VRP consiste en encontrar una serie de rutas que satisfaga una sola vez los distintos pedidos al menor coste posible. Cada ruta comienza y acaba en una localización específica, conocida como almacén o *depot*, y debe cumplir con las restricciones que plantee el propio problema, ya sea de capacidad, de tiempo o ambas. En el caso del VRPTW, se han de considerar una estricta ventana temporal, dado en forma de intervalo $[e_i, l_i]$. El vehículo que vaya a servir al nodo *i* debe llegar en un instante de tiempo anterior a l_i; es posible llegar antes del instante e_i pero en este caso no podrá empezar el servicio hasta que se alcance dicho instante. Una vez comience el servicio, se debe considerar un tiempo por el mismo denominado tiempo de servicio, s_i.

A continuación, se presentará la formulación matemática del VRPTW y su relación con el EC-VRP. Finalmente, se explica en que consiste el EC-VRP.

4.1 VRP con ventanas temporales

Mediante la presente sección, se pretende presentar la formulación matemática del problema de rutado de vehículos con ventanas temporales (VRPTW).

Sea $G = (N, A)$ un grafo dirigido, donde $N = \{0,1, \dots, n\}$ es el conjunto de nodos y $A = \{(i,j): i,j \epsilon N, i \neq j\}$ el conjunto de arcos. El nodo 0 representa el almacén o *depot* y $N' = \{1, \dots, n\}$ representan los nodos que han de ser servidos. Cada arco (i,j) lleva asociado un coste de recorrido $c_{ij} \geq 0$ y un tiempo de viaje $t_{ij} \geq 0$. Cada nodo i lleva asociado una demanda q_i, un tiempo de servicio s_i, y una ventana temporal $[e_i, l_i]$. Todos los vehículos cuentan con la misma capacidad Q y el mismo coste $h \geq 0$. En adelante, se asume que $c_{ij} = t_{ij}, \ \forall (i,j) \in A$ y que h es suficientemente grande para garantizar que minimizar el número de vehículos es el objetivo principal. Sin pérdida de generalidad, se debe estrechar la ventana temporal estableciendo $e_i = \max(e_0 + t_{0i}, e_i)$ y $l_i = \min(l_0 - t_{0i}, l_i)$.

Estableciendo que $x_{ij} = 1$ si el arco (i,j) es usado, $x_{ij} = 0$ en otro caso; p_i representa el inicio del servicio en el nodo i; y_i es la carga del vehículo al abandonar el nodo i. La formulación matemática del problema VRPTW es (Cordone & Calvo 2001):

$$\min \sum_{j \in N'} h x_{0j} + \sum_{(i,j) \in A} c_{ij} x_{ij} \quad (1)$$

sujeto a:

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in N' \quad (2)$$

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in N' \quad (3)$$

$$\text{si } x_{ij} = 1 \rightarrow p_i + s_i + t_{ij} \leq p_j \quad \forall (i,j) \in A \quad (4)$$

$$e_i \leq p_i \leq l_i \ \forall i \in N' \quad (5)$$

$$\text{si } x_{ij} = 1 \rightarrow y_i + q_j \leq y_j \quad \forall (i,j) \in A \quad (6)$$

$$q_i \leq y_i \leq Q \ \forall i \in N' \quad (7)$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in A \quad (8)$$

Las restricciones (2) y (3) aseguran que cada cliente es servido en una sola ruta. Las ecuaciones (4) y (5) representan las restricciones de las ventanas temporales y las (7) y (8) las de capacidad.

El problema a resolver, por el presente trabajo, se basa en el modelo anteriormente desarrollado del problema de rutado de vehículos con ventanas temporales, pero presenta una serie de particularidades que serán descritas en la siguiente sección.

4.2 E-commerce VRP

Teniendo en cuenta las explicaciones anteriores sobre el VRPTW, el EC-VRP presenta además las siguientes características: se tiene una serie de clientes, situados de forma dispersa geográficamente, cuyos pedidos deben ser satisfechos. Se dispone de una serie de vehículos, que deben partir de un depósito y volver a él tras finalizar su ruta, para llevar a cabo las entregas a dichos clientes. Se considera una flota homogénea, teniendo todos los vehículos las mismas características y contando con la misma capacidad, sin ser esta limitante (un único vehículo podría servir a todos los clientes si se cumpliesen las restricciones de tiempo, los productos se consideran de tamaño pequeño). Se supone una cantidad de vehículos suficiente para poder servir a todos los clientes.

A la hora de servir a los clientes esto se puede realizar, para cada uno de ellos, en tres localizaciones distintas. Cada una de estas localizaciones tiene asociada una ventana temporal, que se conoce como el intervalo de tiempo en el que podrá ser atendido en dicha localización. El hecho de que cada cliente tenga tres posibles lugares donde poder entregarle el pedido se debe a que estos pueden cambiar su situación a lo largo del tiempo. Por ejemplo, un cliente podría encontrarse en su casa a primera hora de la mañana, transcurrido cierto tiempo podría desplazarse a su lugar de trabajo, cambiando de esta forma su posición. Si un cliente ha sido atendido en una de sus posibles localizaciones, las otras dejan de estar activas, ya que dicho cliente ya ha sido servido y solo es necesario visitar una vez a cada uno de ellos. El depósito desde el que parten los vehículos también tiene asociada una ventana temporal.

En la figura que se muestra a continuación se puede ver un ejemplo de lo explicado previamente. Se presenta un problema con 3 clientes, cada uno de ellos con 3 ventanas temporales. El depósito aparece denotado por ‘0’. También se puede apreciar una posible solución, donde se visita al cliente uno en su primera localización, después se pasa a servir al cliente tres, también en su primera localización y se acaba visitando al cliente dos en su tercera localización, volviendo finalmente al depósito.

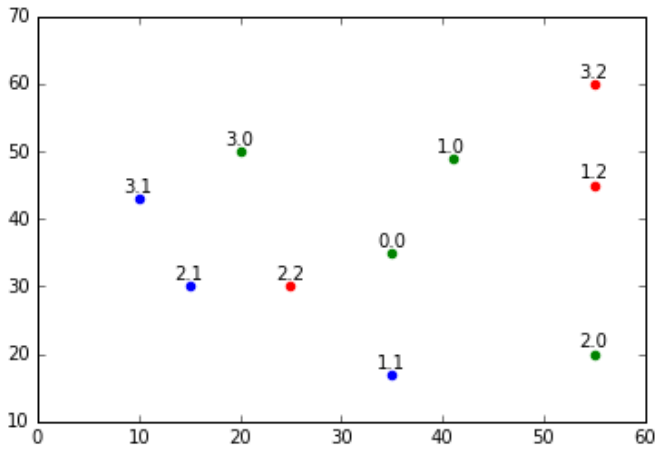


Figura 5. Representación gráfica problema de 3 clientes con 3 ventanas temporales

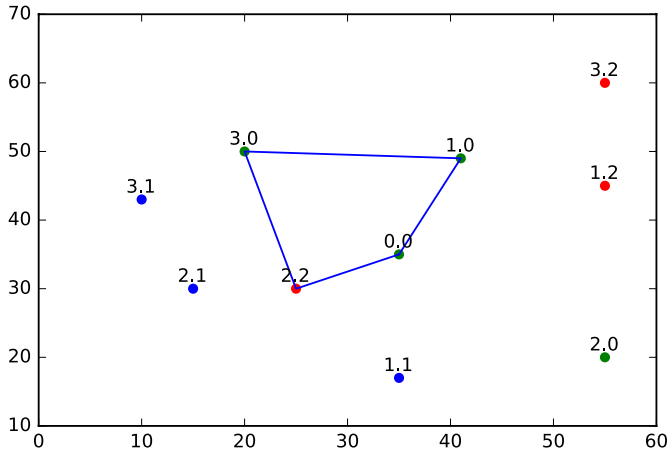


Figura 6. Representación gráfica solución problema de 3 clientes con 3 ventanas temporales cada uno

La resolución del problema conlleva la determinación de las rutas a realizar para servir a todos los clientes, de forma que el coste sea mínimo. Cada una de las rutas corresponde con un vehículo. En la figura que se aprecia a continuación, se observa un ejemplo de resolución para un problema con 25 clientes. En el centro, se encuentra el depósito. Todas las rutas parten de él y vuelven al mismo al finalizar, siendo representada cada una de ellas por un color distinto. Los nodos que no forman parte de ninguna de las rutas, son las localizaciones de los clientes que han sido descartadas a la hora de servirlos.

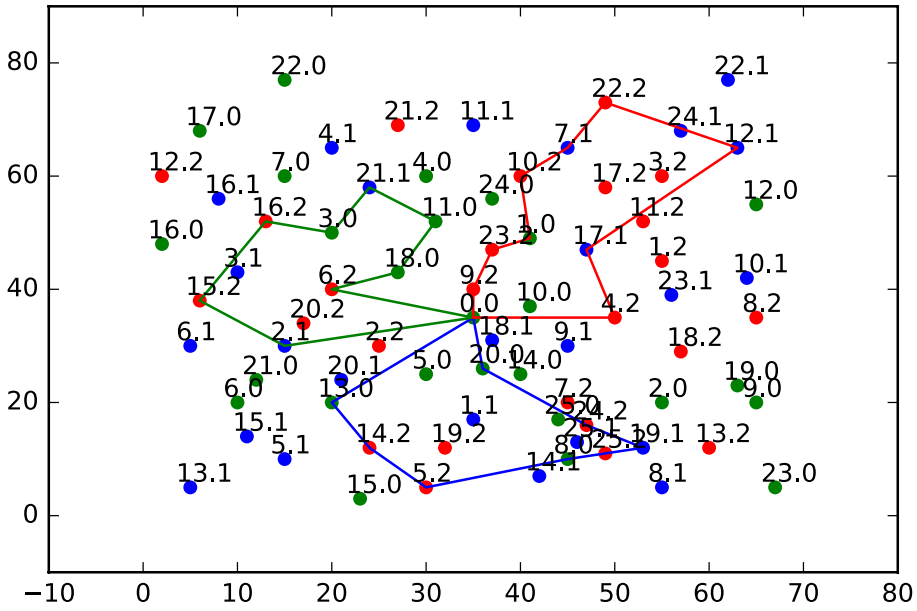


Figura 7. Ejemplo solución problema EC-VRP (Fuente: Elaboración propia)

5 MARCO TEÓRICO DE LA CUCKOO SEARCH

En el presente capítulo se explica en detalle y de forma teórica en que consiste y se sustenta el algoritmo aplicado en este trabajo, la Búsqueda Cuco o *Cuckoo Search* (CS). Para ello, entre otras, se expondrá la explicación teórica que aportan los propios autores (Yang & He 2016).

A la hora de resolver distintos problemas de optimización es común encontrar la siguiente dificultad; son de tiempo polinómico no deterministas o NP-completos y, por lo tanto, no existen algoritmos eficientes para resolver este tipo de problemas. Es decir, no existe un algoritmo que pueda encontrar su solución óptima en un tiempo polinómico. Debido a esto, urge la necesidad de usar métodos alternativos que generen buenas soluciones en una escala de tiempo aceptable. Estos métodos alternativos suelen ser aproximados y/o estocásticos y, entre otros, se puede encontrar aquellos inspirados en comportamientos propios de la naturaleza.

Las técnicas de computación inspiradas en la naturaleza o bio-inspirados tienden a imitar el comportamiento autónomo de individuos en sistemas multi-agentes, donde estos actúan de forma independiente y se rigen por reglas locales. La clave de estas técnicas de optimización se sustenta en la interacción entre los individuos que forman la población o comunidad, considerando que su adaptación al entorno cambia dinámicamente. Esto requiere de un estudio complejo para llegar a comprender completamente dichos comportamientos.

Notar que estas técnicas están sujetas a la propia información que se proporciona sobre el problema, es decir, cuanto mayor sea la adaptación del algoritmo a un determinado problema, las opciones de aplicarlo en otro tipo disminuirán. Por ello, puede convenir considerar estos métodos como "*cajas negras*" o "*black-box"* a la hora de programarlos. De esta forma se gana flexibilidad y se amplía el abanico para poder aplicarlos a otros problemas.

En el presente capítulo se presenta, de forma general, como se puede resolver un problema a través de la inspiración en un comportamiento natural. A continuación, se explica en que consiste una meta-heurística y, finalmente, se presenta en detalle el método utilizado en este trabajo.

5.1 Inspiración de la naturaleza

La naturaleza siempre ha sido fuente de inspiración en el campo del diseño y la optimización ingenieril. De hecho, se puede considerar como uno de los primeros optimizadores porque ella sola, de forma natural, implica que los individuos que la componen evolucionen y adapten su comportamiento de acuerdo al cambiante entorno que los rodea, de no ser así, se van extinguiendo e, incluso, pueden llegar a desaparecer.

Entre los distintos agentes biológicos presentes en la naturaleza destacar, debido al objeto del presente trabajo, aquellos que presentan un comportamiento de enjambre, como los insectos, las hormigas, las abejas o los peces. Dicho comportamiento, favorece la aparición de un sistema

complejo donde múltiples agentes interactúan entre sí, siguiendo unas normas simples y locales, propias de cada sistema. Estas normas están descentralizadas y, en estos sistemas, existen varios líderes. En determinadas condiciones, tales reglas locales pueden conducir a características de auto-organización, casi imposibles de alcanzar por individuos no inteligentes.

5.2 Meta-heurística

¿Cómo se reflejan estos comportamientos naturales en algoritmos a fin de generar una solución aceptable para ciertos tipos de problemas? La vía más común se conoce como *meta-heurística*. Para realizar un buen diseño de una meta-heurística, se requiere de la compresión de las principales características que garantizan el éxito del sistema natural o biológico en el que se basa. A continuación, se deben encontrar los vínculos matemáticos con dicho sistema.

Las meta-heurísticas suelen utilizar una búsqueda estratégica para explorar, de forma efectiva, el espacio de búsqueda de la solución, llegando a centrarse exclusivamente solo en áreas prometedoras. Normalmente, estos métodos parten de una serie de soluciones iniciales del espacio de búsqueda, luego se generan nuevas soluciones basadas en la solución o población actual, realizando movimientos iterativos. Continuando este procedimiento hasta alcanzar un criterio de parada predefinido.

Algunas de las ventajas que se encuentran con el uso de las meta-heurísticas, frente a algoritmos tradicionales, es que los métodos de búsqueda permiten rastrear con mayor eficacia regiones no exploradas y, de esta forma, poder encontrar la solución óptima global, evitando soluciones locales. A su vez, también se observan algunas desventajas. Entre ellas, destaca que este tipo de métodos requieren mayor esfuerzo computacional debido a que no utilizan información propia del problema que se está resolviendo.

No obstante, las meta-heurísticas se caracterizan por su flexibilidad y simplicidad, de forma que suelen ser más sencillas a la hora de implementarlas. Además, pueden resolver cierta cantidad de problemas reales de optimización, desde el ámbito ingenieril hasta aplicaciones en campos como la investigación operativa, la inteligencia artificial o la computacional.

Entre los distintos problemas de optimización, se encuentra aquellos conocidos como *NP-completos*. Hasta la fecha, no existe un algoritmo que resuelva eficientemente este tipo de problemas de forma efectiva. Es por esto, que existe la necesidad de encontrar una solución de forma rápida (sin ser necesariamente la óptima) a este tipo de problemas. Para ello, se recurre y desarrollan diferentes métodos aproximados, entre los que se encuentran los métodos relajados y las meta-heurísticas.

Entre las distintas meta-heurísticas que se conocen, son de interés para el presente trabajo aquellas basadas en poblaciones y, la mayoría, se inspiran en el comportamiento y la forma de relacionarse de grupos, colonias o enjambres de alguna especie biológica. Estos comportamientos colectivos son imitados, de alguna manera, por enjambres artificiales compuestos por múltiples agentes, por ejemplo, en búsqueda de comida o a la hora de evitar depredadores. Las tareas se realizan mediante acciones distribuidas que utilizan reglas locales relativamente simples con capacidad de compartir información.

Por otra parte, desde el punto de vista del análisis del algoritmo, conviene destacar dos factores de importancia a tener en cuenta en casi todas las meta-heurísticas. Estas son la

intensificación y la diversificación.

5.2.1 Diversificación e intensificación

La *diversificación* consiste en llevar a cabo la exploración de una zona nueva o distinta, a la que está siendo explorada, dentro del espacio de la solución. Mientras que la *intensificación* extrae la información recolectada durante la búsqueda para centrarse en regiones prometedoras, donde se pueden encontrar mejores soluciones en su vecindad. Dicho de otro modo, centrará su exploración en buenas soluciones halladas en búsquedas anteriores. De esta forma, se acelera el proceso de convergencia de las soluciones admisibles.

Esto dos factores deben combinarse, de forma que, mediante la diversificación se evite caer en un óptimo local. No obstante, dicha diversificación no siempre es fácil de lograr. Dado que la diversificación requiere muchos movimientos exploratorios iterativos, normalmente puede ralentizar el proceso de búsqueda con una tasa de convergencia más lenta, pero a su vez, aumentará la probabilidad de encontrar la solución óptima global. En los siguientes apartados, se verá en qué medida y cómo se tiene en cuenta estos factores en el algoritmo aplicado.

5.3 Cuckoo Search

Hoy en día, los algoritmos meta-heurísticos son desarrollados con el objetivo de realizar una búsqueda global con tres metas principales: resolver problemas rápidamente, resolver grandes problemas y obtener algoritmos robustos. Los algoritmos genéticos (GA) o la optimización por enjambre de partículas (*Particle swarm optimization*, PSO) son ejemplos de ellos. La eficiencia de estos algoritmos se atribuye a que imitan las mejores características de la naturaleza, especialmente en la selección de aquellos sistemas biológicos que han evolucionado por selección natural durante miles de años. Un caso particular de PSO es la meta-heurística aplicada en el presente trabajo: la Búsqueda Cuco o *Cuckoo Search* (CS), basada en la reproducción del pájaro cuco en combinación con el comportamiento *Lévy Flight* presente en algunas especies de pájaros (Gandomi et al. 2013) .

La CS fue presentada por primera vez por Yang & Deb (2009). Para poder conocer el algoritmo en profundidad se explica en esta sección en qué consiste la estrategia de reproducción de ciertas especies de cucos. A continuación, se describen las ideas básicas y los pasos del algoritmo CS.

5.3.1 Comportamiento de reproducción del pájaro cuco

Los pájaros cucos, además de ser conocidos por el peculiar sonido que realizan, lo son por su peculiar y agresiva estrategia de reproducción. Alguna de sus especies, como el *ani* o el *guira*, ponen sus huevos en nidos de otros pájaros (pájaro anfitrión), llegando incluso a eliminar los huevos de la otra especie para aumentar la probabilidad de que sus huevos sean incubados.

Durante el proceso evolutivo, los rasgos genéticos de una población biológica evolucionan y cambian, llevando a la diversidad y adaptación de las especies. En el caso del cuco, algunas de sus especies han evolucionado desarrollando una estrategia para ahorrar esfuerzos mediante el uso de parasitismo de puesta o de nido y aprovechándose de las aves huésped. Con el fin de que la incubación sea un éxito, los cucos suelen poner sus huevos en nidos de aves con huevos similares en color y textura. Al mismo tiempo, las aves anfitrionas también evolucionan su habilidad de

identificar los huevos "intrusos", depositados por los cucos.

En el caso que el ave anfitriona descubra que los huevos no son suyos, esta se deshace de los huevos intrusos o, simplemente, abandona el nido construyendo otro, cambiando de localización. A fin de evitar que los huevos sean detectados, algunas especies de cuco son especialistas en imitar el color y el aspecto de los huevos de la especie anfitriona. De esta forma se reduce la posibilidad de que sus propios huevos sufran abandono, aumentando la reproductividad del cuco.

Es de esperar que este comportamiento de reproducción pueda tener ventajas evolutivas, ahorrando tiempo y esfuerzos a la hora de la crianza de polluelos de cuco y, por lo tanto, los cucos pueden concentrarse en maximizar la puesta de huevos en múltiples nidos de acogida, aumentando su capacidad reproductiva.

5.3.2 Vuelo de Lévy

En la naturaleza, los animales buscan alimentos de manera aleatoria o cuasi aleatoria. En general, la trayectoria de alimentación de un animal es un camino aleatorio porque el siguiente movimiento se basa en la ubicación y la probabilidad de transición a la siguiente ubicación. La dirección que elija depende implícitamente de una probabilidad que pueda ser modelada matemáticamente (Gandomi et al. 2013).

En este contexto, aparece una forma estadística para caracterizar este movimiento. Muchos de estos movimientos locales están regidos por caminos aleatorios. Ahora bien, a la hora de buscar comida muchos individuos recorren mayores distancias que otros, de una forma casi aleatoria. Diversos estudios demuestran que algunos animales y pájaros realizan un "*vuelo de Lévy"* cuando están en busca de presas (Yang & He 2016). Un vuelo de Lévy o *Lévy Flight*, nombrado así en honor al matemático francés Paul Pierre Lévy, es un tipo de camino aleatorio donde los incrementos son distribuidos de acuerdo a una distribución de probabilidad *heavy-tailed* o cola pesada. En concreto, sigue una distribución *power-law* con la siguiente forma:

$$y = x^{-a}$$

donde $1 < a < 3$ y por tanto su varianza será infinita.

5.3.3 La Búsqueda Cuco

Inspirado en la agresiva estrategia de reproducción de algunas especies de cuco y para simplificar la aplicación del algoritmo, este se sustenta en tres reglas ideales:

1. Cada cuco pone un huevo a la vez en un nido elegido al azar.
2. Los huevos depositados en nidos de mejor calidad se transmiten a las siguientes generaciones.
3. El número de nidos de acogida es fijo y un huevo puesto en un nido puede ser descubierto por el ave anfitriona con una probabilidad de $p_a \in [0,1]$.

Por simplicidad, esta última suposición puede ser aproximada reemplazando una fracción p_a de los *n* nidos por otros nidos nuevos (con soluciones aleatorias en ubicaciones nuevas). Para problemas de maximización, la calidad o el fitness de los huevos es proporcional a la función objetivo del problema.

Siguiendo estas reglas idealizadas, es necesario establecer algunas relaciones entre la fuente

de inspiración y el espacio de búsqueda. A continuación, se presentan una serie de interpretaciones:

- Cada huevo en un nido representa una solución y cada nido solo contendrá un huevo. Múltiples huevos en un nido podrían ser posible, pero para simplificar la aplicación solo se contempla la posibilidad de uno.
- Cada pájaro cuco pondrá un solo huevo a la vez, y elegirá el nido de forma aleatoria. Por lo tanto, cada individuo en la población de cucos tiene el derecho de generar al azar una sola solución. Esto es principalmente la diversificación que se ha comentado anteriormente.
- Los mejores nidos, con las mejores soluciones, avanzarán a la siguiente generación. Esto sería la intensificación, también comentada con anterioridad.
- Nuevas soluciones deben ser generadas a través de *Lévy Flight,* alrededor de las mejores soluciones.
- El número de nidos de acogida es fijo, y un huevo puesto en un nido puede ser descubierto por el ave huésped con una probabilidad de $p_a \in [0,1]$. En esta situación, el ave huésped elige entre abandonar el nido o eliminar el huevo.
- En caso de que un nido contenga múltiples huevos, representará un conjunto de soluciones.

A continuación, siguiendo estas interpretaciones, se presenta el algoritmo seguido en el presente trabajo (Alssager, M., Othman 2013):

Inicialización

Parámetros iniciales (criterio de parada, n, p_a)

Generar una población inicial con n nidos anfitriones de forma aleatoria, NESTS

Calcular el FITNESS de los nidos anfitriones, FITNESS(NESTS)

Ordenar nidos anfitriones según el FITNESS

Calcular el número de nidos que serán abandonados: $p_a * n$

Mejoras

while (criterio de parada)

Seleccionar aleatoriamente un huevo (EGG) de entre NESTS

Generar una nueva solución (EGG') usando Lévy Flight a partir del EGG seleccionado

Calcular fitness de la nueva solución, FITNESS(EGG')

If FITNESS(EGG') < FITNESS(EGG)

EGG=EGG'

FITNESS(EGG)=FITNESS(EGG')

Para todos los huevos que van a ser abandonados $p_a * n$:

Generar nuevas soluciones (EGG') usando Lévy Flight a partir del EGG seleccionado

Calcular FITNESS de los nuevos huevos y reordenar

End while

5.4 Adaptación de la Cuckoo Search a EC-VRP

Para la adaptación de la CS al problema EC-VRP se tomará como base, la que realizan los autores Ouaarab et al. (2014) al problema del agente viajero, TSP. El proceso de adaptación de la CS al EC-VRP se basa en la interpretación de la terminología explicada en el apartado anterior. La CS y sus fuentes de inspiración pueden estructurarse y explicarse mediante los siguientes elementos principales: huevo, nido, función objetivo, espacio de búsqueda y *Lévy Flight*. A continuación, se explican en detalle.

5.4.1 El huevo

Si se asume que un cuco pone un solo huevo en un nido, se puede dar a los huevos las siguientes propiedades:

- Un huevo en un nido es una solución y representa un individuo de la población.
- Los huevos de los cucos son soluciones candidatas a ubicar en la población.

En la CS el número de nidos es fijo y este número representa el tamaño de la población. Un nido es un contenedor de un individuo de la población y su abandono implica que el huevo debe ser reemplazado por uno nuevo. Los nidos podrían contener mayor número de huevos, pero por simplicidad se considera que tan solo tendrá uno.

Añadir que, en el EC-VRP se puede decir que un huevo es el equivalente de un camino hamiltoniano de rutas servidas que comienzan y terminan en el almacén central.

5.4.2 El nido

En la CS, se pueden imponer las siguientes características con respecto a un nido:

- El número de nidos es fijo.
- Un nido es un individuo de la población y el número de nidos es el tamaño de la población.
- El abandono de un nido implica el reemplazo del individuo de la población por uno nuevo.

Notar que, para inicializar los nidos, se realiza de forma que se les asignan huevos generados de forma aleatoria.

5.4.3 La función objetivo

Cada solución en el espacio de búsqueda está asociada con un valor objetivo numérico. Así que la calidad de una solución es proporcional al valor de la función objetivo. En la CS, los nidos de mayor calidad avanzaran entre las distintas generaciones. Esto significa que la calidad de los huevos está directamente relacionada con su capacidad de producir un nuevo cuco. Para el EC-VRP, la calidad de la solución está relacionada con la distancia total de las rutas que componen un huevo. La mejor solución será aquella de menor distancia.

5.4.4 Espacio de búsqueda

Suponiendo dos dimensiones, el espacio de búsqueda representa la posición de los nidos. Esta posición se define como $(x, y) \epsilon \mathbb{R}x\mathbb{R}$. Para modificar la posición de un nido, tan solo se debería cambiar las coordenadas de su posición actual. Parece obvio que mover los nidos o su localización no representa una restricción real del problema. Este es el caso de la mayoría de los

problemas de optimización constantes, esto puede considerarse como una ventaja que evita muchos problemas técnicos tales como la representación de las coordenadas en el espacio de solución del problema. Notar que hay que hacer diferencia entre las coordenadas de los nidos, las cuales no se van a considerar por las razones presentadas anteriormente, y las de los clientes que forman parte del huevo/solución. Estas coordenadas si son fijas y no se pueden modificar, aunque si el orden de visita a los distintos clientes. Es en este punto donde entra en juego el concepto de vecindad, la cual modificará el orden de visitas a los clientes en las distintas rutas. En concreto, en el presente trabajo, se utilizan varias estructuras de vecindades a fin de mejorar los huevos de cuco para imitar la forma de los del nido anfitrión.

Debido a que las coordenadas de los clientes a los que servir son fijas, los movimientos, como se ha visto, consistirán en modificar el orden de visita a los clientes. Existen varios métodos o perturbaciones que generan nuevas soluciones a partir de otras existentes, variando el orden de visita.

En la adaptación que se realiza de la CS al EC-VRP, las perturbaciones usadas para cambiar el orden de visitas son SHIFT-1-0 (Alssager, M., Othman 2013), movimientos 2-OPT y DOUBLE-BRIDGE (Ouaarab et al. 2014). Los métodos SHIFT-1-0 y 2-OPT se usan para pequeñas alteraciones, mientras que DOUBLE-BRIDGE realiza variaciones mayores.

SHIFT-1-0 selecciona un cliente al azar y lo transfiere a otra posición, en la misma ruta o en otra.

2-OPT elimina dos arcos de un camino (solución o ciclo hamiltoniano) y los reconecta creando unos nuevo, como se aprecia en la siguiente figura. La solución inicial se representa por A y el camino creado mediante 2-OPT es el correspondiente a B.

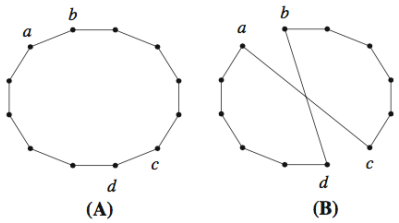


Figura 8. Movimiento 2-OPT (Fuente: (Ouaarab et al. 2014))

El movimiento DOUBLE-BRIDGE elimina cuatro arcos y los reconecta creando otros nuevos, como se aprecia en la figura a continuación.

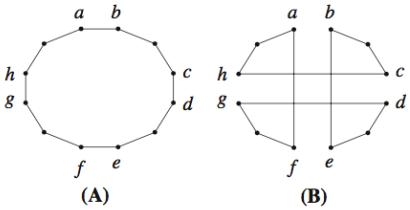


Figura 9. Movimiento DOUBLE-BRIDGE (Fuente: (Ouaarab et al. 2014)

Cabe destacar que, debido a la naturaleza propia del problema que se está resolviendo, se añade un fenómeno de intensificación cuando se realiza alguno de estos movimientos. Debido a que en el EC-VRP cada cliente dispone de tres ventanas temporales distintas, cada vez que se va a realizar uno de los movimientos anteriormente explicados, a su vez, se comprueba que ventana

temporal es la mejor, es decir, cuál es la que implica una menor distancia total (mejor fitness).

5.4.5 Lévy Flight

El paso de un movimiento es la distancia entre dos soluciones. La longitud de este paso marcará que tipo de movimiento, de los explicados anteriormente, se utilizará. El paso se basa en la tipología del espacio de búsqueda y el concepto de vecindad. Así, en el presente trabajo, las longitudes de paso menores se representan mediante SHIFT-1-0 y 2-OPT, mientras que los saltos mayores lo harán por el movimiento DOUBLE-BRIDGE. Para determinar dicha longitud de paso se recurre al vuelo de Lévy.

Los vuelos de Lévy se caracterizan por realizar una búsqueda intensiva alrededor de una solución, seguida de grandes saltos ocasionales. Según Yang & Deb (2009), en algunos problemas de optimización, para la búsqueda de nuevas y buenas soluciones, lo más eficiente es el uso de *Lévy Flight*. A fin de mejorar la calidad de la búsqueda, se asocia la longitud del paso al valor generado por los vuelos de Lévy.

6 RESULTADOS

6.1 Calibración de la Cuckoo Search

Para la calibración de los parámetros de la CS se han tenido en cuenta los siguientes valores de configuración:

- Número de nidos ≡ N.
- Probabilidad de abandono del nido ≡ p_a.
- Número de iteraciones.

Para llevar a cabo la calibración se toma una muestra de 30 clientes, pues se consideró representativa. El parámetro p_a se establece como 0,25, es decir, en cada iteración el 25% de los nidos serán abandonados. Este valor se establece en función de la bibliografía analizada donde todas las aplicaciones del algoritmo se fija dicho valor. Además, en un análisis previo, se realizan distintas simulaciones modificando el valor de p_a comprobando que no mejoran los resultados.

De este modo, se procede a realizar la calibración fijando un número de iteraciones y variando el número de nidos. Así, para un determinado número de iteraciones se realizan distintas simulaciones variando entre N=25, 50 y 75 nidos. Al tratarse de un algoritmo cuya solución es estadística, para cada escenario presentado se realizan 5 simulaciones con los mismos parámetros. Es decir, 5 simulaciones con el mismo número de iteraciones y de nidos. De este modo, el valor que se utilizará posteriormente para la comparación será el promedio de estas soluciones. Aclarar que, el número de iteraciones se aumentó con un paso de 5000, comenzando por 10000 hasta llegar a 50000 iteraciones. A continuación, se muestran los resultados de dicha calibración mediante dos gráficos. En el primero, se muestran las distancias promedias obtenidas y, en el segundo, los tiempos de ejecución de los mismos, frente al número de iteraciones.

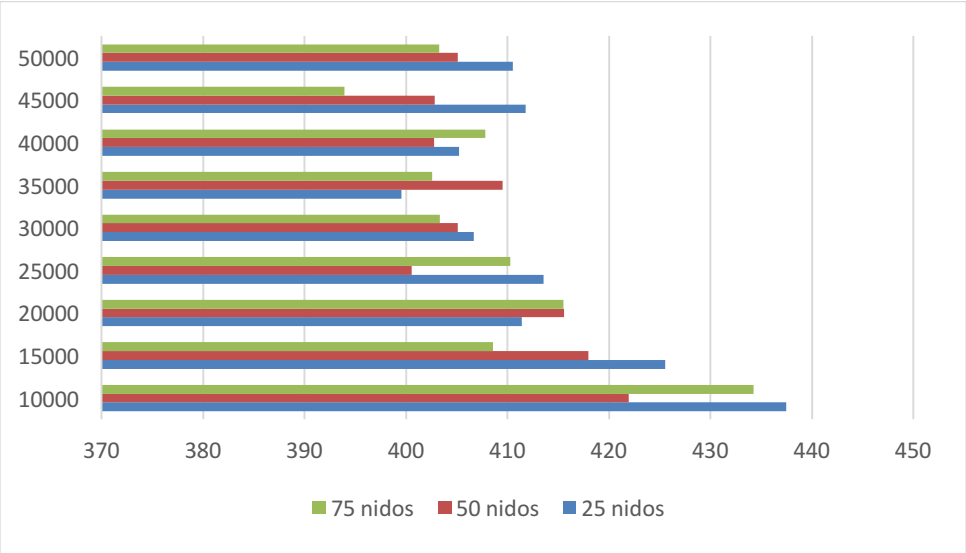


Figura 10. Distancias promedias obtenidas en el proceso de calibración (Fuente: Elaboración propia)

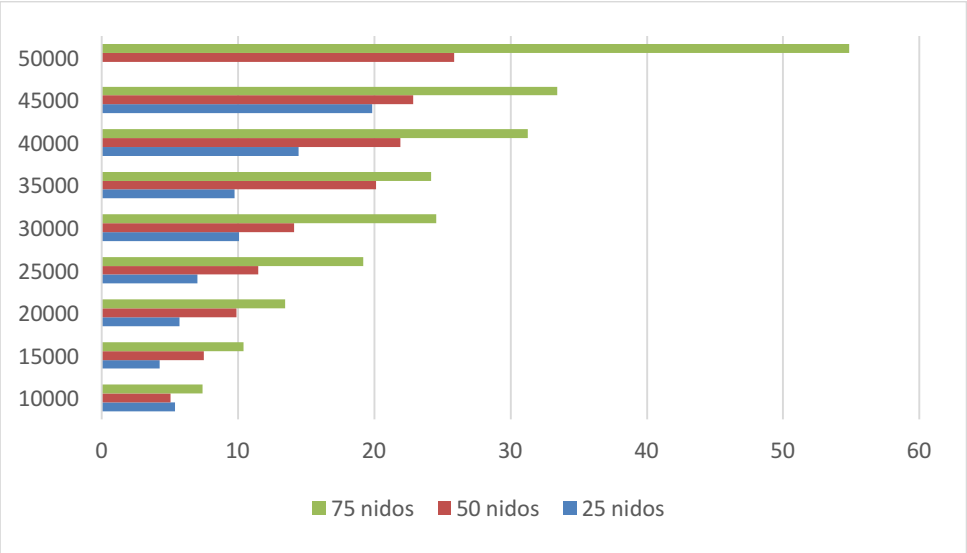


Figura 11. Tiempo (min) promedio simulaciones en proceso de calibración (Fuente: Elaboración propia)

A la vista de estos resultados, se aprecia que el mejor valor se obtiene con **25 nidos** y **35000 iteraciones**. Además, el tiempo medio de ejecución es uno de los más bajos. Por estas razones se establecen estos parámetros para llevar a cabo la resolución del problema.

6.2 Batería de problema elegida para el testeo

Se han generado 3 baterías de problemas para comprobar el funcionamiento de la *Cuckoo Search*. Cada una de las 3 baterías ha sido la compuesta por los 12 problemas de correspondientes al tipo R1 del conjunto de problemas de Solomon.

Solomon (1987) generó seis tipos de problemas. El diseño de estos problemas se hizo de tal forma que destacan varios factores que afectan al comportamiento de las heurísticas de enrutamiento y programación. Estos factores incluyen: datos geográficos, número de clientes servidos por un vehículo y características de la ventana temporal, tales como el porcentaje de clientes con limitaciones de tiempo, y la rigidez y el posicionamiento de las ventanas temporales. Se ha elegido la batería de Solomon debido a su frecuente uso para la validación de técnicas heurísticas en problemas tipo VRPTW.

El problema que se va a resolver, el R1, se ha generado a través de una distribución uniforme aleatoria y, además, posee un corto horizonte de programación. Esto implica que la longitud de la duración de la ruta actúa como una limitación de capacidad que, junto con las restricciones propia de capacidad del vehículo, permite que sólo unos pocos clientes sean atendidos por el mismo vehículo. Esto quiere decir que los clientes no se agrupan por zonas, sino que se sitúan de forma aleatoria y que cuentan con una estrecha ventana temporal.

Para conocer la cantidad de información con la que se cuenta en los distintos escenarios que se van a simular, en la siguiente figura, se muestra un fragmento correspondiente al primer problema R1 con 100 clientes.

r101.txt

```
R101

VEHICLE
NUMBER     CAPACITY
  25         200

CUSTOMER
CUST NO.   XCOORD.   YCOORD.   DEMAND   READY TIME   DUE DATE   SERVICE TIME

    0         35        35         0          0         230           0
    1         41        49        10        161         171          10
    2         35        17         7         50          60          10
    3         55        45        13        116         126          10
    4         55        20        19        149         159          10
    5         15        30        26         34          44          10
    6         25        30         3         99         109          10
    7         20        50         5         81          91          10
    8         10        43         9         95         105          10
    9         55        60        16         97         107          10
```

Figura 12. Fragmento problema R101 Solomon.

La primera columna corresponde al número de cliente, denotándose el depósito como cliente 0. La segunda y la tercera, corresponden a la localización de los clientes. En concreto a las coordenadas *x* e *y* de los mismos. La quinta y sexta columna corresponden a la ventana temporal de cada cliente, concretamente al instante de inicio y de finalización de la misma. Se observa que el depósito cuenta con una ventana temporal, la cual afectará a la resolución de los distintos problemas pues se establece que el vehículo debe regresar al mismo antes del fin de la misma.

La cuarta columna corresponde a la demanda de cada cliente y la séptima al tiempo de servicio de los mismos. En nuestro problema estos datos no se van a considerar, debido a las siguientes hipótesis:

- La capacidad de los vehículos no es limitante.
- La entrega de mercancía es instantánea, no considera tiempo de servicio.

Debido a las características del EC-VRP, explicadas anteriormente, los clientes resultan de agrupar de 3 en 3 los clientes definidos en el fichero de Solomon, debido a que cada uno tiene tres realizaciones, lo que implica que se tienen 3 posiciones y 3 ventanas temporales diferentes.

CUSTOMER	XCOOR	YCOOR	DEMAND	READY TIME	DUE TIME	SERVICE TIME	WINDOW
0	35	35	0	0	230	0	-
1	41	49	10	161	171	10	0
	35	17	7	50	60	10	1
	55	45	13	116	126	10	2
2	55	20	19	149	159	10	0
	15	30	26	34	44	10	1
	25	30	3	99	109	10	2

Tabla 2. Ejemplo de tabla de clientes con los que se trabaja.

6.3 Resolución batería de problemas

Para llevar a cabo la resolución de los problemas, se parte del supuesto que se dispone de un número de vehículos suficiente para servir a todos los clientes. Además, se considera que la velocidad media a la que se desplazan los vehículos es de 60 km/h, facilitando así el cálculo de los tiempos de desplazamientos a partir de las distancias pues se miden en km y, por lo tanto, se tardará un minuto en recorrer un kilómetro.

Las tres baterías consideradas han sido separadas por número de clientes, considerándose las siguientes:

- Batería 1: **25 clientes**.

  12 Escenarios: r101, r102, r103, …, r112.

- Batería 2: **50 clientes**.

  10 Escenarios: R1_2_1, R1_2_2, R1_2_3, …, R1_2_10.

- Batería 3: **100 clientes**.

  10 Escenarios: R1_4_1, R1_4_2, R1_4_3, …, R1_4_10.

Cada escenario ha sido ejecutado 10 veces y se presentará como resultado el valor promedio de esas 10 ejecuciones, el mejor valor encontrado en todas ellas, su desviación típica y el tiempo medio de ejecución (en minutos). Las baterías son ejecutadas con los parámetros calculados en el paso de calibración. A continuación, se presentan los resultados obtenidos:

25 clientes	Iteraciones:	35000	Nidos:	25
	Promedio	**Mejor Valor**	**Desviación**	**Tiempo (min)**
r101	354,5731356	347,2689907	7,23860411	8,298026975
r102	314,718563	298,6246668	10,67267695	7,944270694
r103	295,7120637	281,0568046	12,35285648	7,516573778
r104	254,9938565	232,5069954	12,26727427	7,509454695
r105	328,6058912	311,9577945	10,39941778	7,857571067
r106	296,967907	265,3438512	15,20031168	7,73748552
r107	282,5838008	272,6846114	6,359012033	7,622434707
r108	253,8869156	228,4652059	15,05625968	7,373298704
r109	287,3787179	270,1144905	14,24427387	7,650865541

r110	274,8904573	240,5300518	22,07707678	7,879934861
r111	283,3928491	243,7897976	16,77876806	7,49670165
r112	267,8759743	228,8959121	19,01624308	7,332982939

Tabla 3. Solución mediante CS para 25 clientes.

50 clientes	Iteraciones:	35000	Nidos:	25
	Promedio	**Mejor Valor**	**Desviación**	**Tiempo (min)**
r1_2_1	1095,139471	1010,57918	46,67756274	15,81016596
r1_2_2	945,8235694	852,2635608	47,01398744	15,78146561
r1_2_3	773,8416508	685,4524221	57,37863306	15,45087953
r1_2_4	682,701398	595,55225	58,24320306	15,03752696
r1_2_5	1001,318138	921,9154673	56,26803204	16,49917146
r1_2_6	906,2457163	832,8525343	60,4150448	15,05372303
r1_2_7	762,8683465	678,5696213	44,13430345	15,28040021
r1_2_8	681,5606121	612,8771578	44,91622032	15,65942633
r1_2_9	889,3122157	813,1153812	51,89550128	16,1703762
r1_2_10	829,1892348	783,4462928	22,84342845	15,08169976

Tabla 4. Solución mediante CS para 50 clientes.

100 clientes	Iteraciones:	35000	Nidos:	25
	Promedio	**Mejor Valor**	**Desviación**	**Tiempo (min)**
r1_4_1	3328,36777	3169,426095	92,25352748	31,55760533
r1_4_2	2773,634237	2630,874392	73,72405569	31,73631518
r1_4_3	2305,954972	2057,586288	118,1077752	29,91620143
r1_4_4	2119,263034	1951,536678	124,52888	29,53669873
r1_4_5	3123,899776	2844,027678	126,5475502	31,9321062
r1_4_6	2725,027876	2566,099026	88,02929086	30,60493286
r1_4_7	2317,711771	2138,991802	100,9738609	29,60410242
r1_4_8	2091,015288	1898,91892	96,59849629	28,81006604

r1_4_9	2930,849611	2751,567908	82,79623879	30,55100657
r1_4_10	2646,163477	2553,648011	70,34881356	30,57070353

Tabla 5. Solución mediante CS para 100 clientes.

Se debe especificar de forma detallada las características del ordenador con las que se llevan a cabos las distintas pruebas, ya que esto influirá sobre el tiempo de computación. Para el presente trabajo se ha usado:

- Ordenador: MacBook Pro (Retina 13 pulgadas, principios de 2015)
- Sistema operativo: macOS Sierra
- Memoria RAM: 8 GB 1867 MHz DDR3
- Procesador: 2,7 GHz Intel Core i5

6.4 Análisis de las soluciones

Mediante esta sección se realiza un análisis de los resultados obtenidos. Para ello, se compara la solución obtenida mediante la *Búsqueda Cuco*, que se observa en la sección anterior, con otras obtenidas mediante distintas meta-heurísticas aplicadas en trabajos similares sobre la misma batería de problemas. Concretamente, se comparan con las soluciones obtenidas por el *Algoritmo Genético*, la *Búsqueda Tabú* y el *Recocido Simulado*.

A continuación, por cada uno de los escenarios, se generan tres gráficas distintas: la primera recoge la distancia (km) promedio de las 10 ejecuciones llevadas a cabo para cada batería. La segunda, la menor distancia obtenida y, la tercera, el tiempo medio de ejecución en segundos.

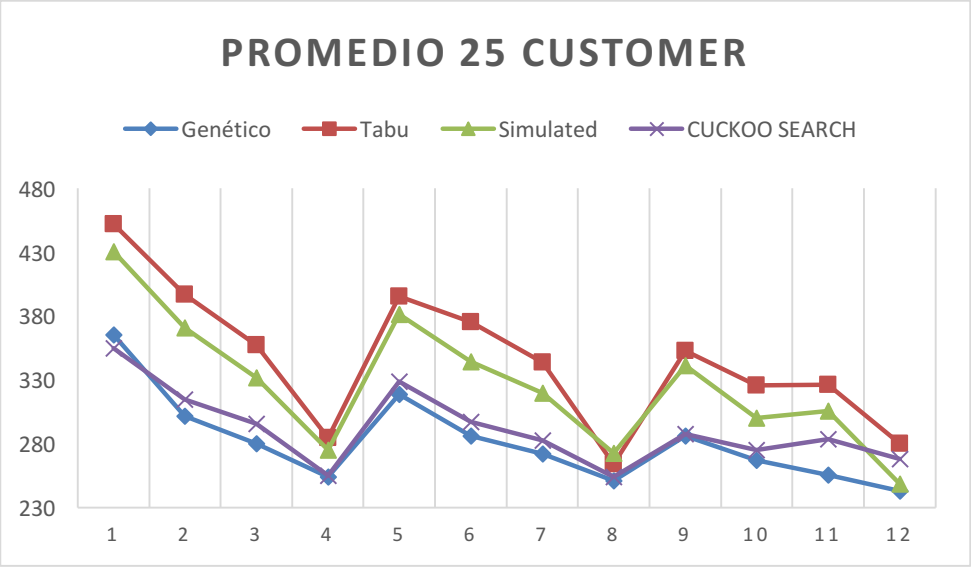


Figura 13. Distancia promedio para 25 clientes.

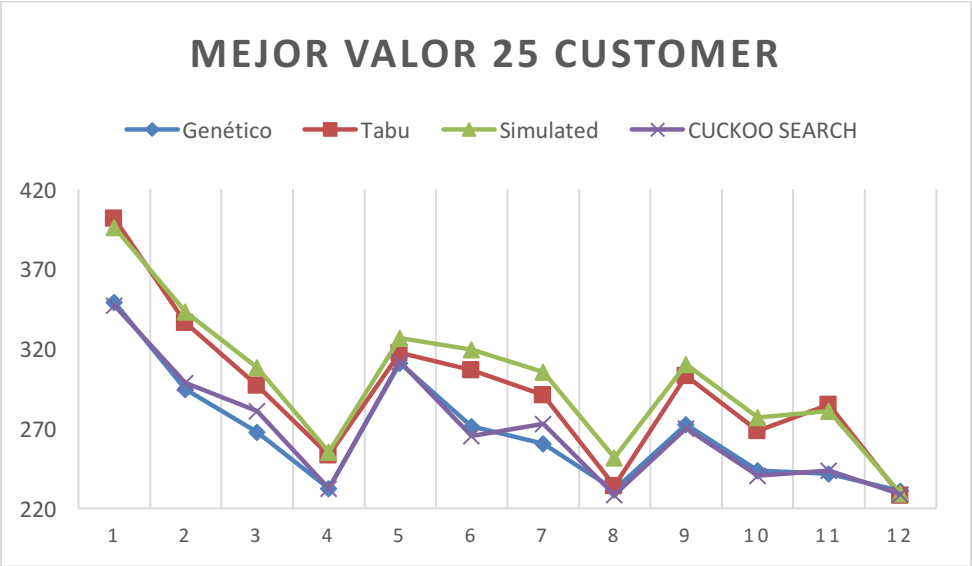


Figura 14. Menor distancia obtenida para 25 clientes.

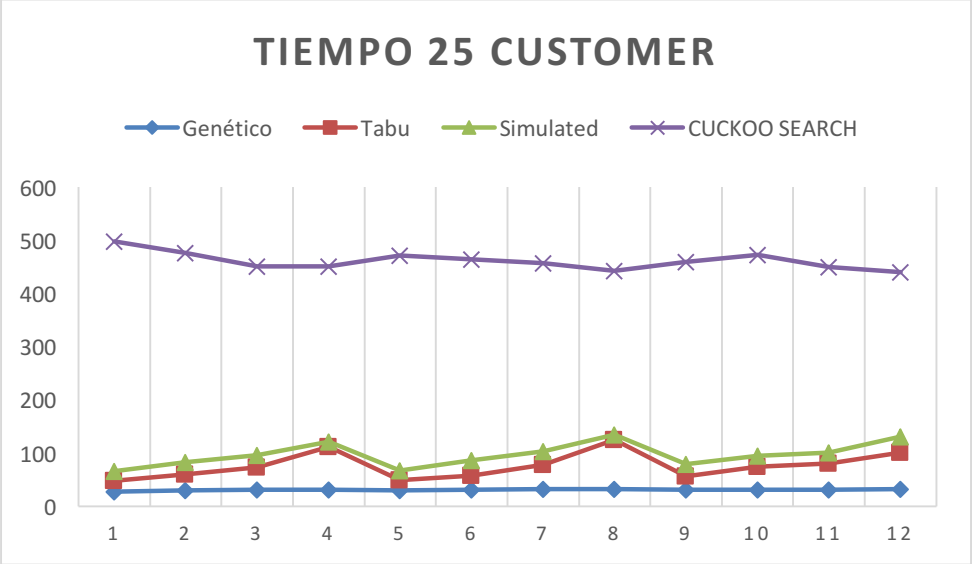


Figura 15. Tiempo promedio de simulación para 25 clientes.

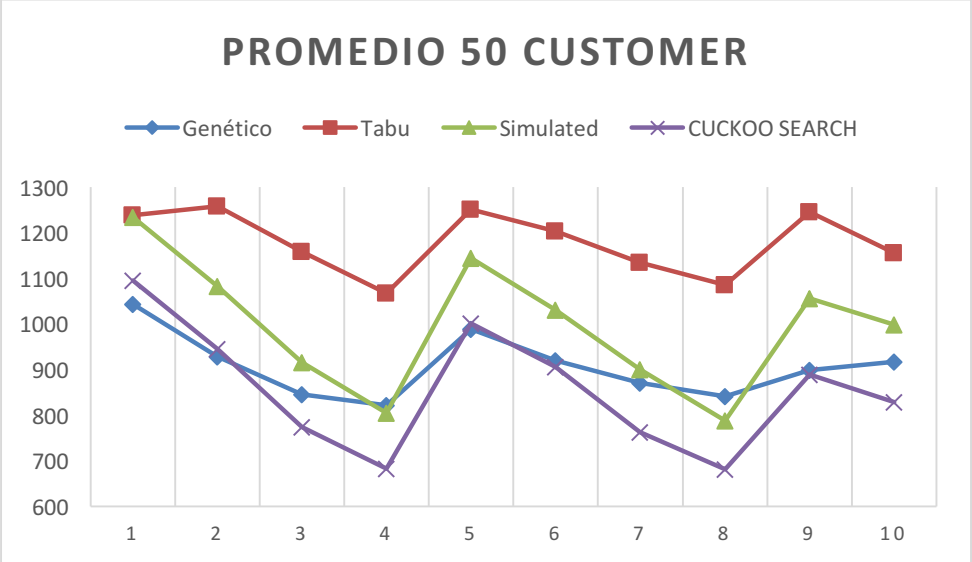


Figura 16. Distancia promedio para 50 clientes.

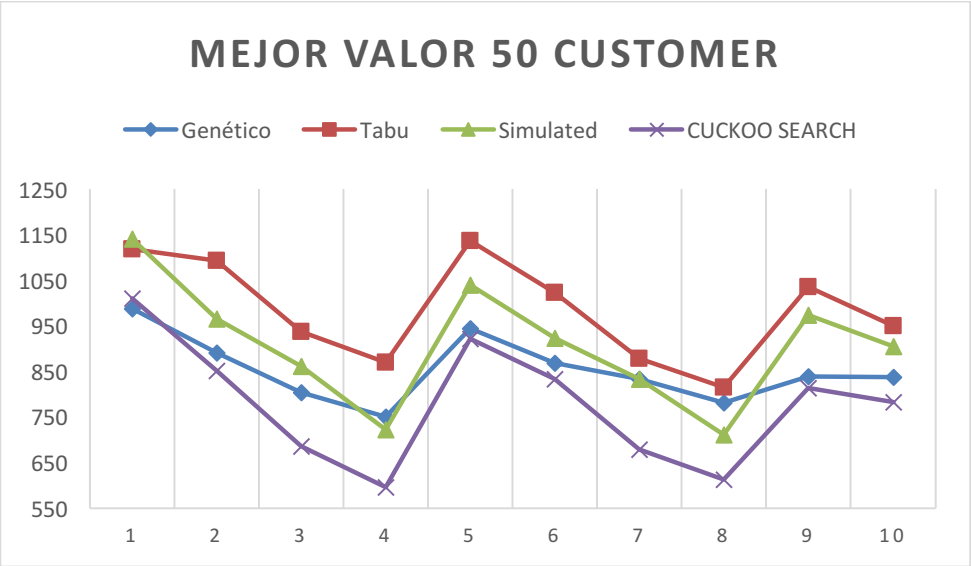


Figura 17. Menor distancia obtenida para 50 clientes.

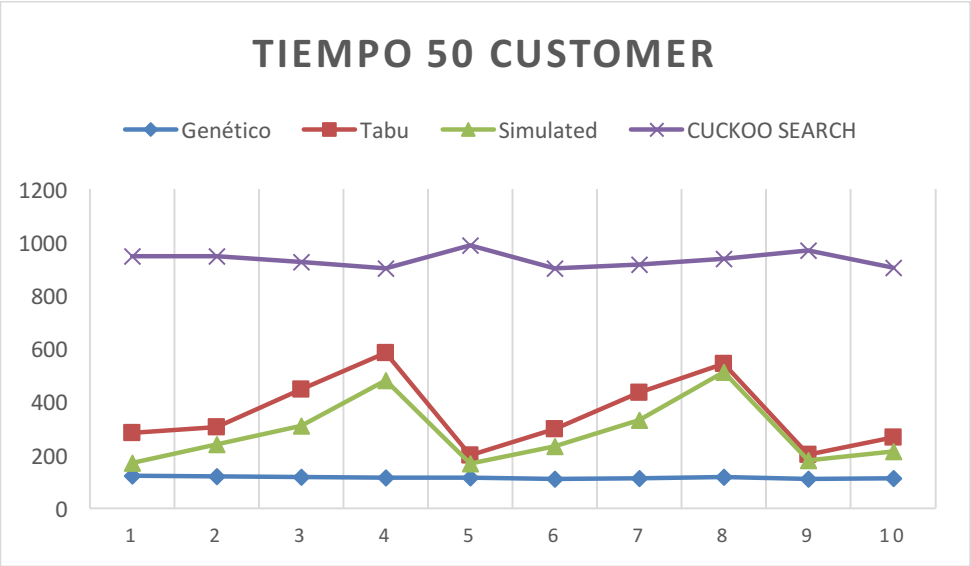


Figura 18. Tiempo promedio de simulación para 50 clientes.

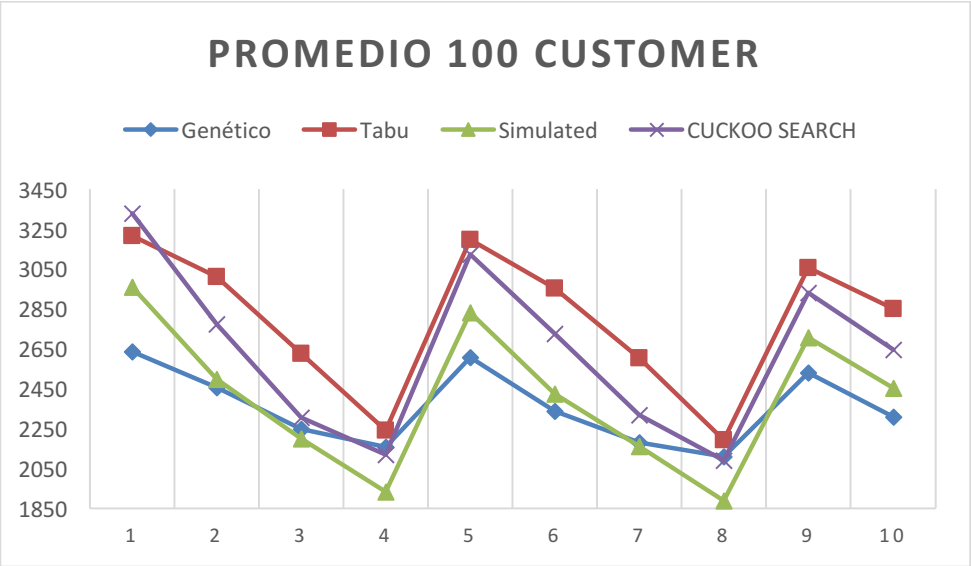


Figura 19. Distancia promedio para 100 clientes.

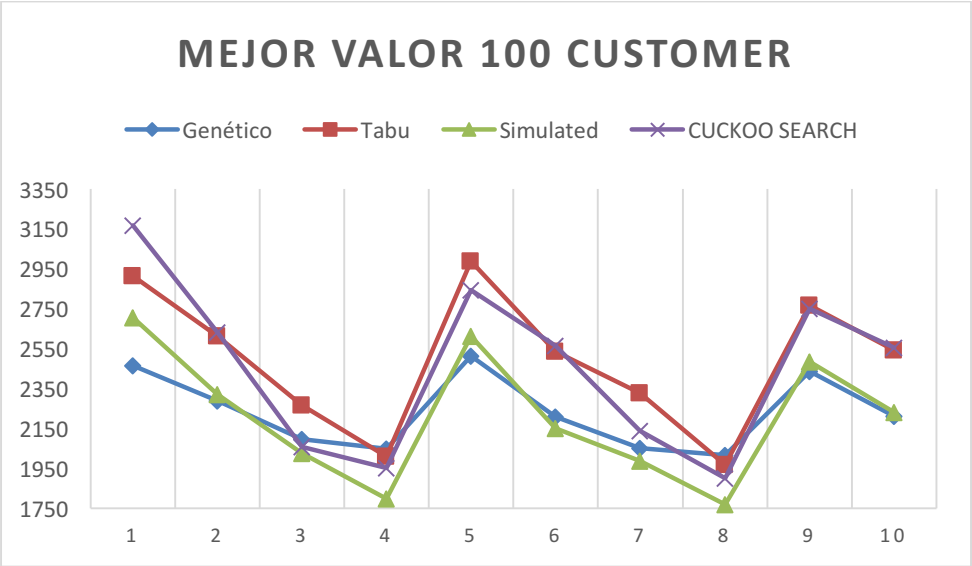


Figura 20. Menor distancia obtenida para 100 clientes.

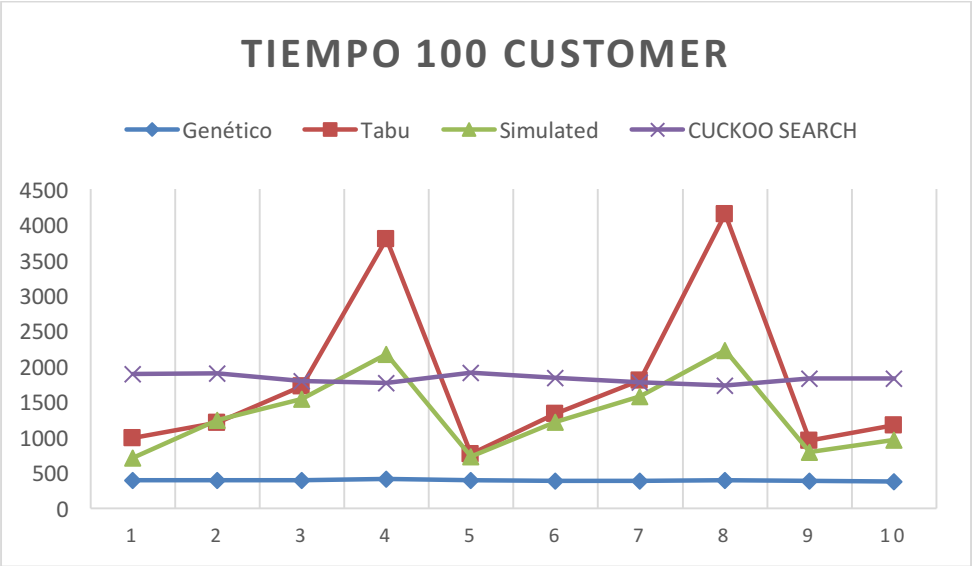


Figura 21. Tiempo promedio de simulación para 100 clientes.

En líneas generales se obtienen soluciones aceptables respecto a las obtenidas mediante otros algoritmos. En cuanto a los resultados particulares de la batería 1 (25 clientes), se aprecia que la distancia promedio obtenida mediante CS se aproxima a los resultados del Genético, siendo este el que proporciona las mejores soluciones, e incluso lo supera en el escenario 1 (r101). En cuanto al mejor valor obtenido (mínima distancia) presenta un comportamiento similar, pero superando en más escenarios al Genético.

En cuanto a la batería 2 (50 clientes), se observa que la CS proporciona los mejores resultados en cuanto a distancia promedio en 7 escenarios y en cuanto a la mínima distancia obtenida en 9 de 10 posibles escenarios. De esta forma, es en esta batería donde mejores resultados se obtienen en la aplicación de la CS.

Por último, en la última batería (100 clientes) es donde se obtienen peores resultados en comparación a los otros métodos. De forma general, tan solo se supera a los resultados proporcionados por la Búsqueda Tabú.

En cuanto al tiempo medio de ejecución se observa que en las baterías 1 y 2 es mayor en la

CS que en el resto, mientras que en la batería 3 se encuentra en el orden.

6.5 Ejemplo de solución

A continuación, para concluir este capítulo, se presenta un ejemplo de solución. En concreto, se muestra la mejor solución obtenida correspondiente al escenario r101 para la betería de 25 clientes. La razón de la elección de esta batería no es otra que la de poder apreciar con mayor claridad la solución gráfica.

```
***** PARAMETROS *****
Nidos=25
Probabilidad abandono=0.25
Iteraciones=35000
***** SOLUTION *****
Tiempo ejecucion (min)=8.49761555195
Distancia Total=347.268990666
Nº vehiculos=5
[(0, 0), (20, 1), (15, 1), (6, 0), (21, 0), (13, 0), (5, 0), (0, 0)]73.8064796088
[(0, 0), (10, 0), (4, 2), (17, 2), (22, 2), (12, 1), (1, 0), (0, 0)]112.12483137
[(0, 0), (18, 1), (0, 0)]8.94427191
[(0, 0), (9, 2), (23, 2), (11, 0), (3, 0), (16, 2), (0, 0)]66.3536868909
[(0, 0), (24, 2), (7, 2), (25, 0), (14, 1), (8, 0), (19, 1), (2, 0), (0, 0)]86.0397208862
```

Figura 22. Solución problema r101 (25 clientes).

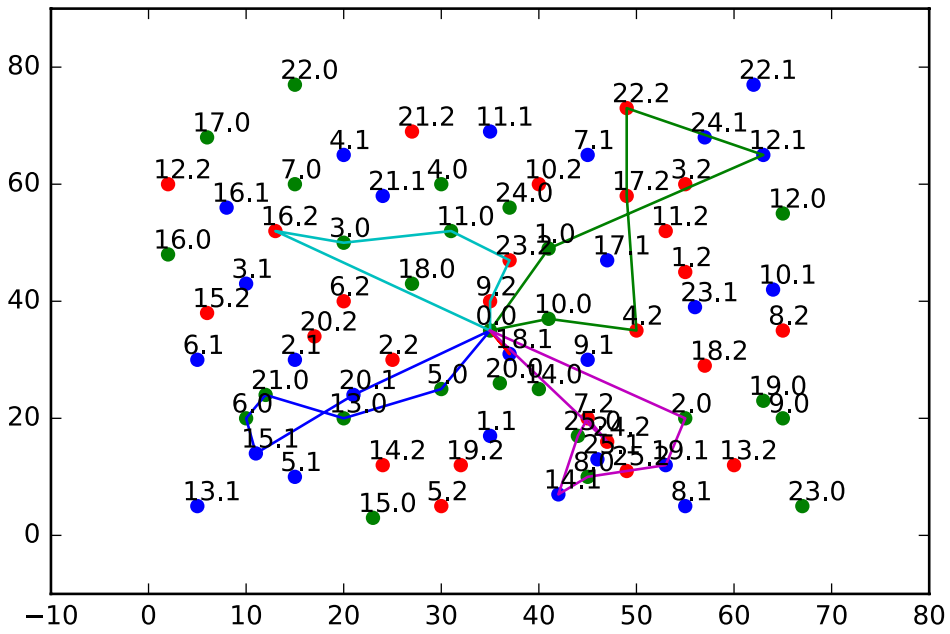


Figura 23. Representación gráfica problema r101 (25 clientes).

Se observa que la representación gráfica corresponde a las distintas rutas que se observan en la solución.

6.6 Mejora de la CS

Siguiendo la propuesta realizada por Ouaarab et al. (2014), mediante este apartado se procede a la implementación de la CS "mejorada". Debido a que la fortaleza de la CS reside en la forma de explotar y explorar el espacio solución por parte del cuco, el autor defiende que una fracción de estos podrán tener "inteligencia" en la búsqueda de nuevas soluciones.

La mejora reside en dos niveles de control; el cuco y la población. El primer nivel consiste en utilizar al cuco para controlar tanto la intensificación como la diversificación. El segundo, reside en la población, pues está formada por distintos cucos. De esta forma, se reestructura el algoritmo tradicional añadiendo una nueva categoría de cucos "inteligentes" más eficientes en sus búsquedas.

El autor defiende que hay estudios que muestran que los cucos realizan una vigilancia previa de los nidos que van a ser huéspedes. Este comportamiento puede servir como inspiración para crear una nueva categoría de cucos que tienen la capacidad de cambiar el nido anfitrión durante la incubación para evitar el abandono de los huevos. De esta forma, se puede hablar de una especie de búsqueda local realizada por ciertos cucos alrededor de las soluciones actuales.

Por simplicidad, el autor resume esta mejora en dos pasos:

1) Un cuco se mueve hacia una nueva solución a través del vuelo de Lévy.
2) De la solución actual, el cuco en la misma zona busca una nueva y mejor solución.

De acuerdo a estos pasos, el autor propone añadir al algoritmo CS tradicional un nuevo parámetro, p_c, que representa la fracción de cucos inteligentes. El objetivo de esta mejora consiste en fortalecer la búsqueda intensiva en torno a las mejores soluciones de la población.

De este modo, la diferencia respecto a la CS tradicional reside en que en vez de considerar todos los nidos del espacio solución a la hora de generar una nueva, tan solo se considerarán soluciones candidatas los p_c nidos con mejores soluciones.

Siguiendo esta mejora y considerando $p_c = 0{,}6$, se muestran a continuación los resultados obtenidos para la misma batería de problemas usadas con el algoritmo CS tradicional. En ellos se aprecia los resultados del algoritmo mejorado frente al tradicional.

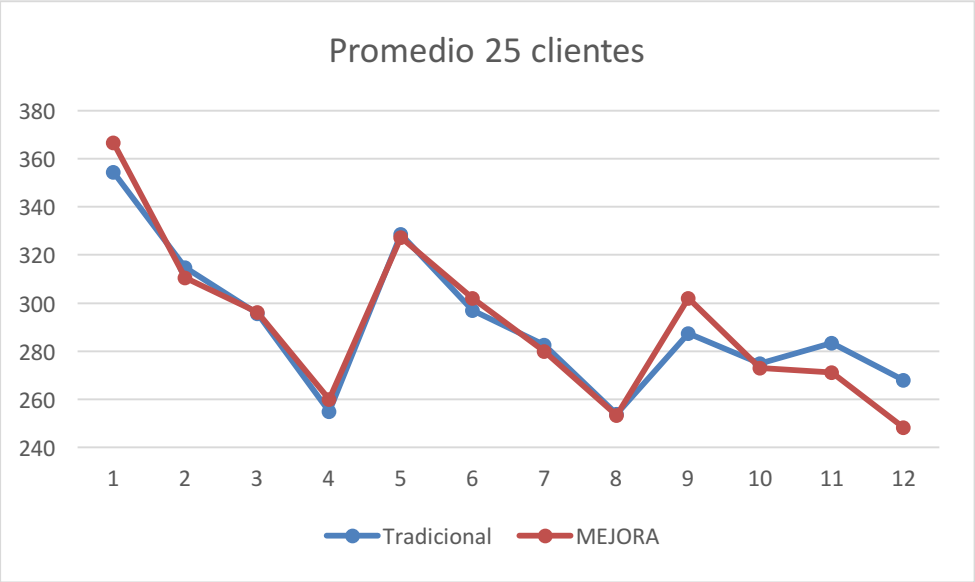


Figura 24. Promedio 25 clientes. CS tradicional vs mejora.

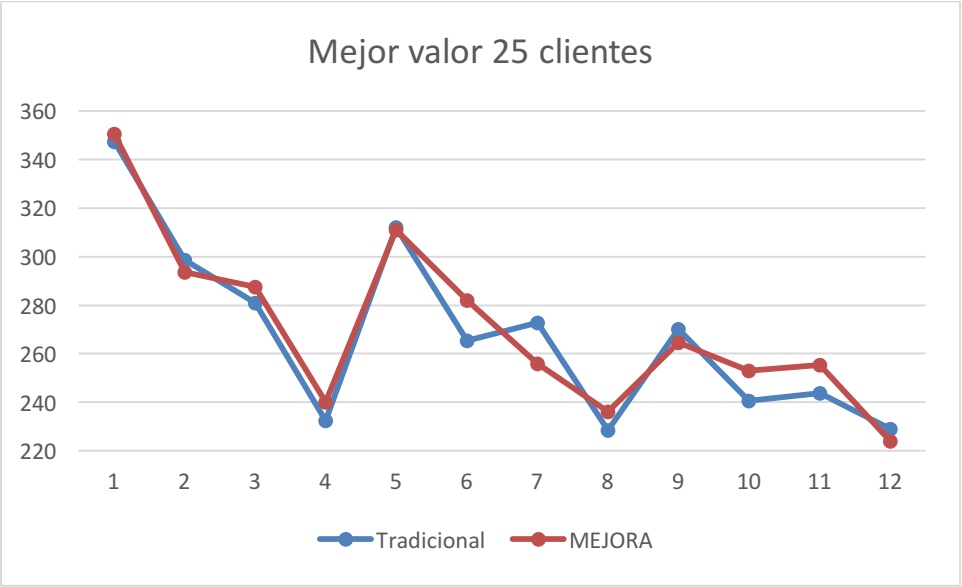


Figura 25. Mejor valor 25 clientes. CS tradicional vs mejora.

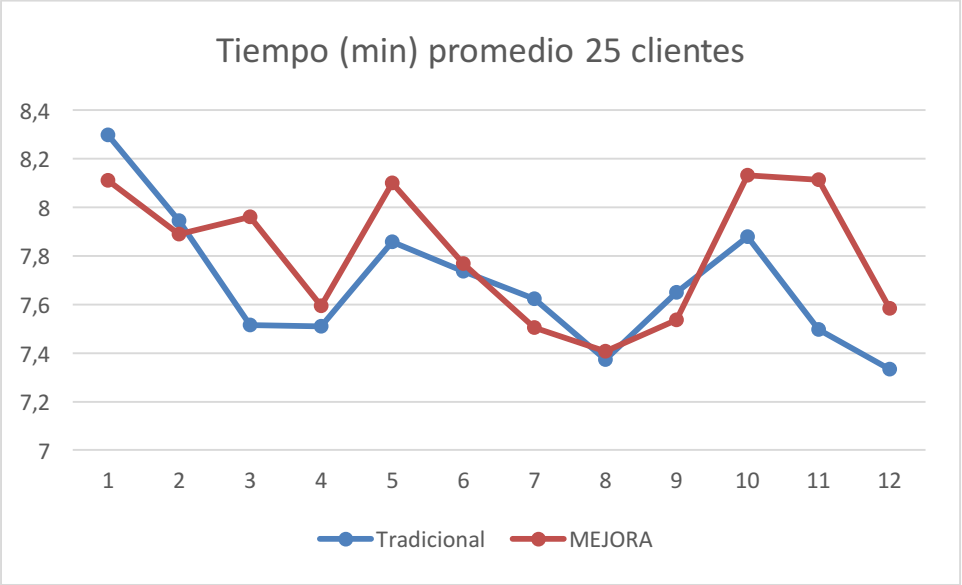


Figura 26. Tiempo medio de ejecución 25 clientes. CS tradicional vs mejora.

Para la batería 1 (25 clientes) no se aprecian mejoras significativas e incluso se obtienen mayores tiempos de ejecución en 8 de los 12 escenarios implementados.

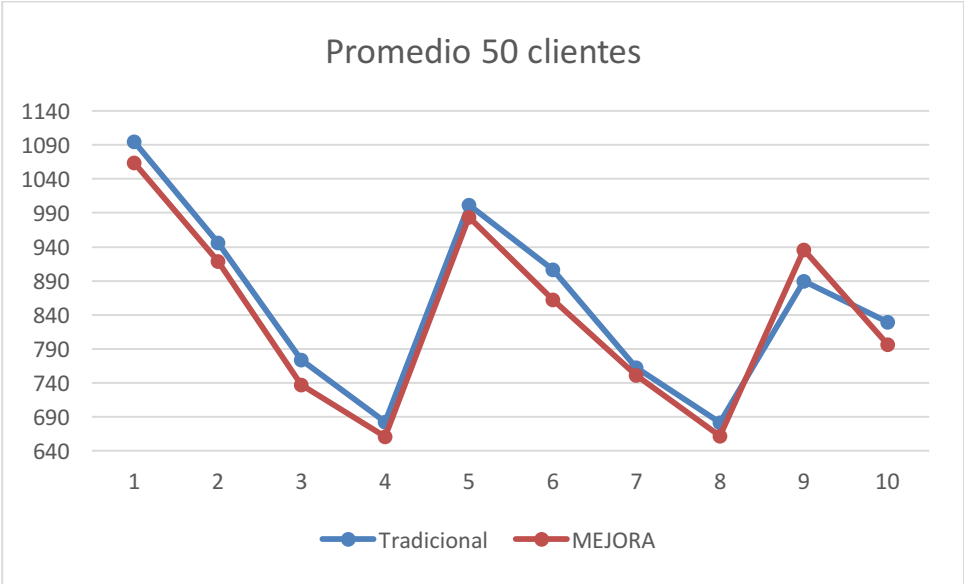


Figura 27. Promedio 50 clientes. CS tradicional vs mejora.

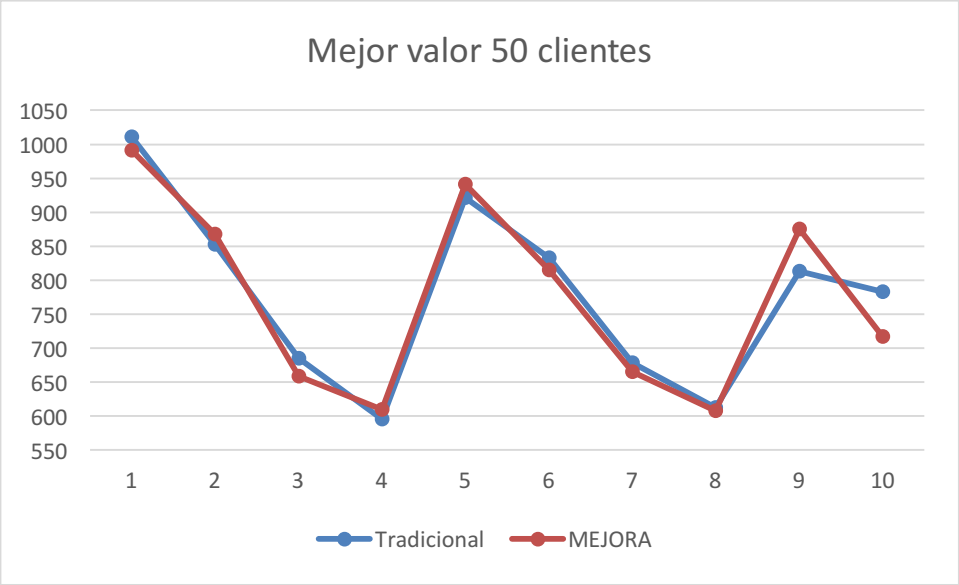


Figura 28. Mejor valor 50 clientes. CS tradicional vs mejora.

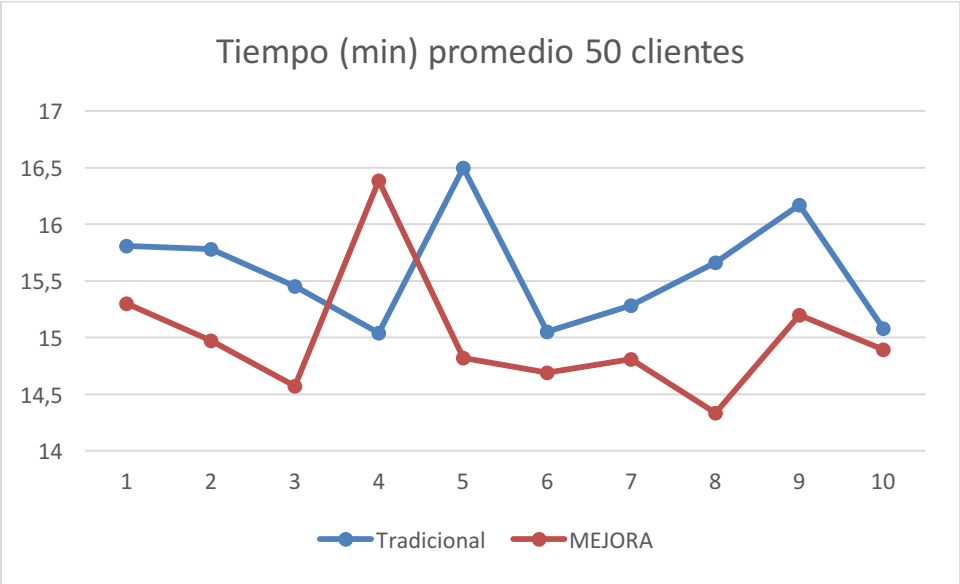


Figura 29. Tiempo medio de ejecución 50 clientes. CS tradicional vs mejora.

En la batería 2 (50 clientes) se mejoran levemente los resultados promedios y se obtiene menores tiempos de ejecución respecto al algoritmo tradicional. En cuanto a los mejores valores, se observa que se obtienen resultados similares en todos los escenarios.

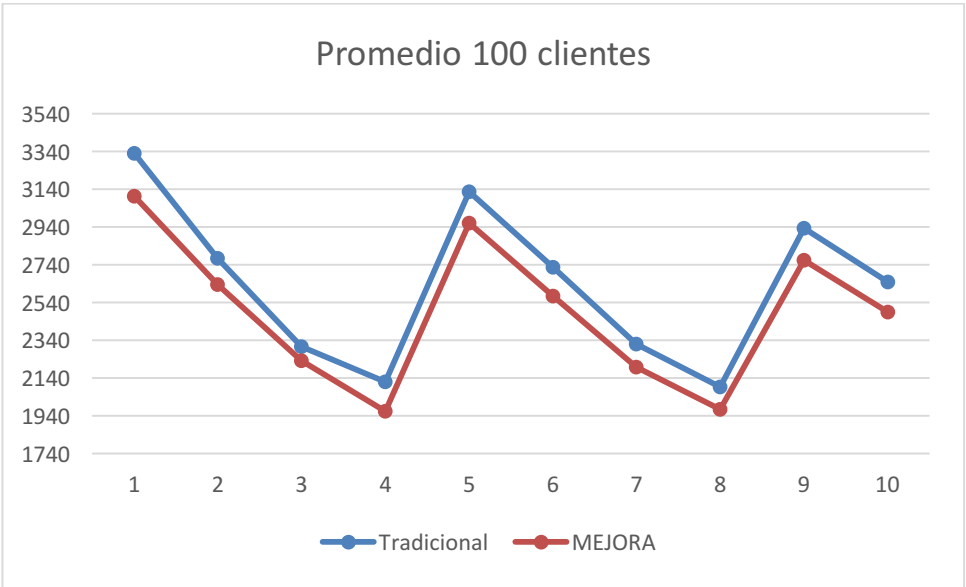


Figura 30. Promedio 100 clientes. CS tradicional vs mejora.

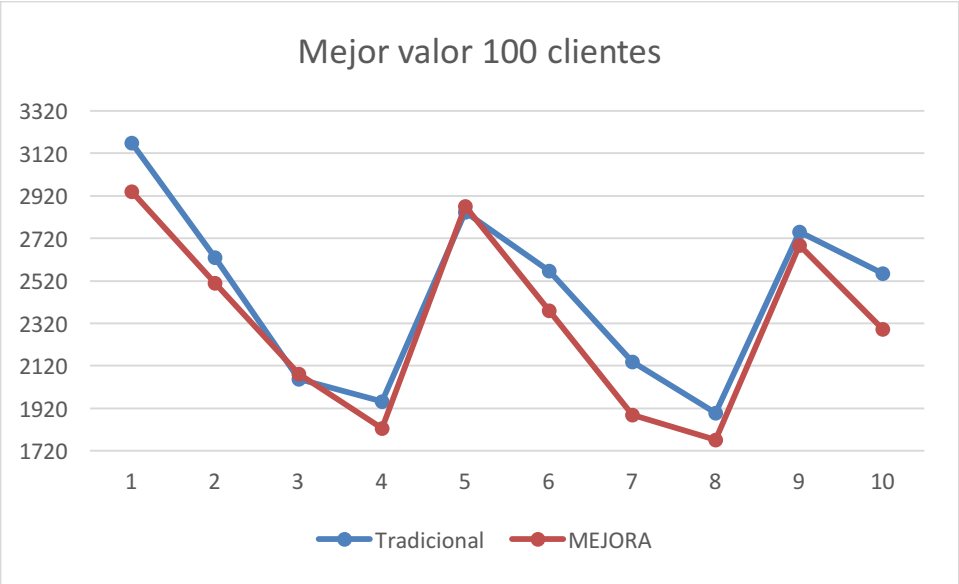


Figura 31. Mejor valor 100 clientes. CS tradicional vs mejora.

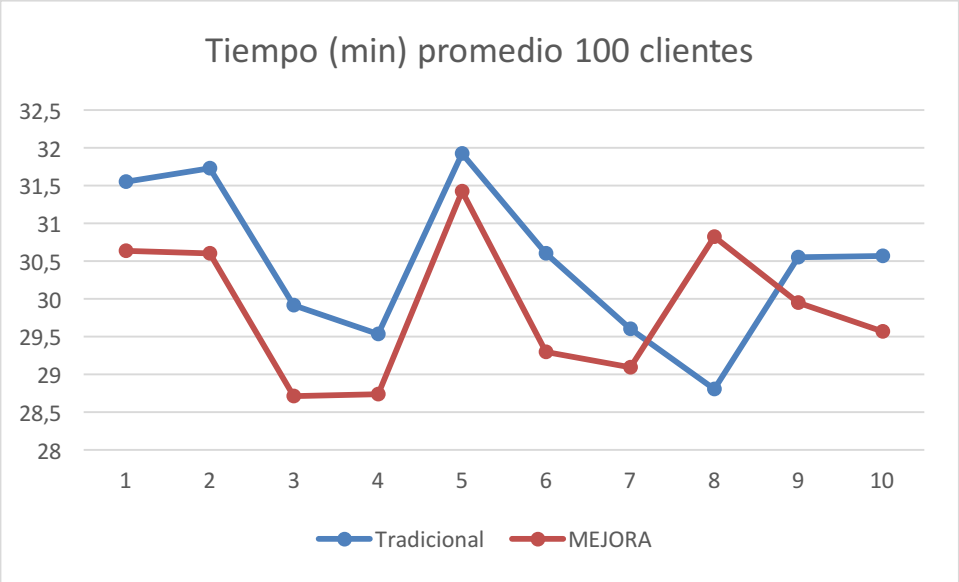


Figura 32. Tiempo medio de ejecución 100 clientes. CS tradicional vs mejora.

Finalmente, en la batería 3 (100 clientes) es donde se obtienen mejores resultados. Mejorando notablemente los resultados promedios y los mejores valores obtenidos, así como la disminuyendo el tiempo de ejecución en todos los escenarios (a excepción de uno).

De forma general se observa que las mejoras son más significativas a mayor número de clientes. Se aprecia que los resultados promedios mejoran en la mayoría de los escenarios, pero tan solo se aprecia una mejora en los mejores valores obtenidos en la batería 3.

7 CONCLUSIONES

Mediante este capítulo se destaca los puntos más importantes del conjunto del trabajo realizado, sobre la resolución de un problema de rutado de vehículos destinados al *e-commerce* mediante el algoritmo *Cuckoo Search*. Una vez obtenidos y habiendo analizado los resultados tras la resolución de la batería de problemas, se extraen una serie de conclusiones para determinar si se ha cumplido con los objetivos establecidos al inicio del trabajo.

A continuación, se expone un breve resumen con los pasos que se han seguido para la realización de este trabajo:

- En primer lugar, se realiza una pequeña introducción sobre la importancia del diseño de rutas de vehículos, concretamente las destinadas al comercio electrónico, y se realiza una pequeña introducción sobre este tipo de comercio, a fin de establecer el escenario desde el que se parte.
- A continuación, se realiza un breve estado del arte sobre el problema VRP y, otro, sobre el algoritmo *Cuckoo Search*, implementado para la resolución del problema.
- Posteriormente, se han explicado detalladamente las características del problema a resolver, EC-VRP, especificando todas aquellas consideraciones que se deben tener en cuenta y todas las particularidades que presenta frente al VRPTW clásico. Además, también se detalla las características propias de la CS, a fin de explicar el funcionamiento del mismo y como adaptarlo para llevar a cabo la resolución.
- Finalmente, se aplica el algorirmo CS tradicional a una batería de problemas escogida. Posteriormente se implementa una posible mejora al CS tradicional, comparándose los resultados obtenidos entre ambos.

Una vez se conocen los resultados y comprobados los objetivos establecidos, se puede concluir que el desarrollo global del trabajo obtiene resultados aceptables conforme a estos objetivos. De forma general se pueden destacar las siguientes conclusiones:

- Este trabajo demuestra la eficiencia de la Cuckoo Search para la resolución de problemas de rutado de vehículos y, concretamente, el EC-VRP. De esta forma, se considera que proporciona rutas de transporte bien diseñadas, reduciendo costes.
- Los resultados obtenidos demuestran que en muchas ocasiones se mejoran a resultados obtenidos mediante meta-heurísticas tradicionales, como la Búsqueda Tabú o el Recocido Simulado.
- En cuanto a los tiempos computacionales se pueden considerar aceptables. Aunque sean mayores que los de las meta-heurísticas tradicionales, se considera que están dentro de un tiempo polinómico y, por lo tanto, son válidos.

A la vista de los resultados y comparandolos con los obtenidos mediante otros algortimos, se observa que se obtienen resultados promedios aceptables (Figura 13, Figura 16 y Figura 19). Llegando incluso a superar el mejor resultado obtenido mediante la Búsqueda Tabú, el Algortimo Genético y el Recocido Simulado, en alguno de los escenarios.

En cuanto a la modificación sobre el algoritmo CS tradicional, a la vista de los resultados se

concluye que es efectiva. Y, por lo tanto, se obtienen mejores resultados, siendo más destacables cuando la batería cuenta con un mayor número de clientes. Además, de forma general, disminuye el tiempo promedio de ejecución.

Finalmente, tras haber analizado los resultados, se puede afirmar que la *Búsqueda Cuco* o *Cuckoo Search* (CS) es un algoritmo válido para hallar soluciones del problema de rutado de vehículos asociado al modelo de negocio e-commerce (EC-VRP), obteniéndose una mejor organización en cuanto al reparto de mercacías que se traduce en una disminución de costes..

REFERENCIAS

Alssager, M., Othman, Z.A., 2013. Cuckoo search algorithm for capacitated vehicle routing problem. *Energy*, 60(1), pp.99–108.

Braysy, O. & Gendreau, M., 2005. Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science*, 39(1), pp.104–118.

Bräysy, O. & Gendreau, M., 2005. Vehicle Routing Problem with Time Windows, Part II: Metaheuristics. *Column Generation*, (October 2016), pp.67–98.

Civicioglu, P. & Besdok, E., 2013. *A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms*,

Cordone, R. & Calvo, R.W., 2001. A Heuristic for the Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 7(2), pp.107–129.

Gandomi, A.H., Yang, X.S. & Alavi, A.H., 2013. Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. *Engineering with Computers*, 29(1), pp.17–35.

Ghiani, G. et al., 2003. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151(1), pp.1–11.

Pillac, V. et al., 2013. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1), pp.1–11. Available at: http://dx.doi.org/10.1016/j.ejor.2012.08.015.

Pisinger, D. & Ropke, S., 2007. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34(8), pp.2403–2435.

Prins, C., 2004. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, 31(12), pp.1985–2002.

Solomon, M.M., 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35(2), pp.254–265. Available at: http://links.jstor.org/sici?sici=0030-364X(198703/04)35:2<254:AFTVRA>2.0.CO;2-E.

Taillard, E. et al., 1997. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31(October 2016), pp.170–186.

Teymourian, E. et al., 2016. Enhanced intelligent water drops and cuckoo search algorithms for solving the capacitated vehicle routing problem. *Information Sciences*, 334–335, pp.354–378. Available at: http://www.sciencedirect.com/science/article/pii/S0020025515008749.

Walton, S. et al., 2011. Modified cuckoo search: A new gradient free optimisation algorithm. *Chaos, Solitons and Fractals*, 44(9), pp.710–718. Available at: http://dx.doi.org/10.1016/j.chaos.2011.06.004.

Yang, X.-S. & Deb, S., 2010. Engineering Optimisation by Cuckoo Search. *Int. J. Mathematical Modelling and Numerical Optimisation*, 1(4), pp.330–343. Available at: http://arxiv.org/abs/1005.2908.

Yang, X. & He, X., 2016. *Nature-Inspired Computation in Engineering*, Available at: http://link.springer.com/10.1007/978-3-319-30235-5.

Yang, X.S. & Deb, S., 2014. Cuckoo search: Recent advances and applications. *Neural Computing and Applications*, 24(1), pp.169–174.

Yang, X.S. & Deb, S., 2009. Cuckoo search via L??vy flights. In *2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings*. pp. 210–214. Available at: http://arxiv.org/abs/1003.1594.

Yang, X.S. & Deb, S., 2013. Multiobjective cuckoo search for design optimization. *Computers and Operations Research*, 40(6), pp.1616–1624. Available at: http://dx.doi.org/10.1016/j.cor.2011.09.026.

Yildiz, A.R., 2013. Cuckoo search algorithm for the selection of optimal machining parameters in milling operations. *International Journal of Advanced Manufacturing Technology*, 64(1–4), pp.55–61.

ANEXO

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Fri Dec  2 10:37:34 2016

@author: Carlos Uribe Astolfi
"""

import scipy.io
import math
import matplotlib.pyplot as plt
from ecvrptwlib import *
from ecvrptwlib import Route
import numpy as np
from random import randint
import random
from time import time
import os

#%%
# *****Funciones auxiliares*****
def distanciruta(route):
    huevo=[]
    for i in range(len(route)):
        dni=route[i][0]
        if dni!=0:
            execution=route[i][1]
            if execution==2:
                cliente=3*dni
            elif execution==1:
                cliente=(3*dni)-1
            else:
                cliente=(3*dni)-2
            huevo.append(cliente)
    null,distance=interpretaResultado(huevo)
    return distance

def interpretaResultado(egg): #funcion de EVAULACION
    ruta=[]
    t=0
    t_p=0
    fitness=0
    ruta.append(0)
    distancia=0
    for i in range(len(egg)):
        t_todepot=d[interprete(egg[i]),interprete(0)]

        if i==0: t_p=t+d[interprete(0),interprete(egg[i])]
        else: t_p=t+d[interprete(egg[i-1]),interprete(egg[i])]

        if (t_p+t_servicio+t_todepot)<HT: #TW Depot
            t_p,flag=checkTW(t_p,egg[i])
            if flag==True:
                t=t_p+t_servicio
                distancia+=d[interprete(ruta[-1]),interprete(egg[i])]
                ruta.append(egg[i])
```

```
            elif flag==False:
                distancia+=d[interprete(egg[i-1]),interprete(0)]
                ruta.append(0)
                distancia+=d[interprete(ruta[-1]),interprete(egg[i])]
                ruta.append(egg[i])
                #t=0
                t_p=d[interprete(0),interprete(egg[i])]
                t,flag=checkTW(t_p,egg[i])
                t+=t_servicio

        else:
            t+=d[interprete(egg[i-1]),interprete(0)]
            fitness+=t
            distancia+=d[interprete(egg[i-1]),interprete(0)]
            ruta.append(0)
            distancia+=d[interprete(ruta[-1]),interprete(egg[i])]
            ruta.append(egg[i])
            t_p=d[interprete(0),interprete(egg[i])]
            t,flag=checkTW(t_p,egg[i])
            t+=t_servicio

        if i==len(egg)-1:
            t+=d[interprete(egg[i]),interprete(0)]
            fitness+=t
            distancia+=d[interprete(egg[i]),interprete(0)]

    return ruta, distancia

def modoTester(ruta):
    routes=[]
    route=[]
    for i in range(len(ruta)):
        if i==0: route.append(interprete(0))
        elif i>0:
            if ruta[i]==0:
                route.append(interprete(0))
                routes.append(route)
                route=[]
                route.append(interprete(0))
            else:
                route.append(interprete(ruta[i]))

        if i==len(ruta)-1:
            route.append(interprete(0))
            routes.append(route)


    return routes


def checkTW(time,cliente):
    flag=True
    custome=interprete(cliente)
    early=customer[custome[0]].execution[custome[1]].tw[0]
    late=customer[custome[0]].execution[custome[1]].tw[1]
    if early<=time and time<=late:
        t_p=time
    else:
        if time<early: t_p=early
        if time>late:
            flag=False
            t_p=0
```

```
    return t_p,flag


def interprete(cliente): #traduce un elemento del vector [CLIENTE], al
formato LIB
    if cliente==0:
        clientes=0,0
    else:
        dni=int(math.ceil(cliente/float(3)))
        if cliente==3*dni:
            execution=2
        elif cliente==(3*dni)-1:
            execution=1
        else:
            execution=0
        clientes= dni, execution

    return clientes

def levyFlight(u):
    return math.pow(u,-1.0/3.0)

def randF():
    return random.uniform(0.0001,0.9999)

def swap(sequence,i,j):
    temp = sequence[i]
    sequence[i]=sequence[j]
    sequence[j]=temp

def twoOptMove(nest,a,c):
    nest = nest[0][:]

    aux=0
    minimo=99999999

    #### CHECK EXECUTION (elijo ejecución con menor distancia) ####

    cliente_a=interprete(nest[a])
    cliente_c=interprete(nest[c])
    for i in range(3):
        for j in range(3):
            nest_p=nest[:]
            nest_p.remove(nest_p[a])
            nest_p.insert(a,3*cliente_a[0]-i)
            nest_p.remove(nest_p[c])
            nest_p.insert(c,3*cliente_c[0]-j)
            swap(nest_p,a,c)
            null,aux=interpretaResultado(nest_p)
            if aux<minimo:
                minimo=aux
                ii=i
                jj=j
    nest.remove(nest[a])
    nest.insert(a,3*cliente_a[0]-ii)
    nest.remove(nest[c])
    nest.insert(c,3*cliente_c[0]-jj)
    swap(nest,a,c)
    fitness=minimo

    ##################################################################
```

```
    return (nest,fitness)


def doubleBridgeMove(nest,a,b,c,d):
    nest = nest[0][:]

    #### CHECK EXECUTION (elijo ejecución con menor distancia) ####

    aux=0
    minimo=99999999
    cliente_a=interprete(nest[a]) #i
    cliente_b=interprete(nest[b]) #j
    cliente_c=interprete(nest[c]) #u
    cliente_d=interprete(nest[d]) #v
    for i in range(3):
        for j in range(3):
            for u in range(3):
                for v in range(3):
                    nest_p=nest[:]
                    nest_p.remove(nest_p[a])
                    nest_p.insert(a,3*cliente_a[0]-i)
                    nest_p.remove(nest_p[c])
                    nest_p.insert(c,3*cliente_c[0]-u)
                    nest_p.remove(nest_p[b])
                    nest_p.insert(b,3*cliente_b[0]-j)
                    nest_p.remove(nest_p[d])
                    nest_p.insert(d,3*cliente_d[0]-v)
                    swap(nest_p,a,c)
                    swap(nest_p,b,d)
                    null,aux=interpretaResultado(nest_p)
                    if aux<minimo:
                        minimo=aux
                        ii=i
                        jj=j
                        uu=u
                        vv=v
    nest.remove(nest[a])
    nest.insert(a,3*cliente_a[0]-ii)
    nest.remove(nest[c])
    nest.insert(c,3*cliente_c[0]-uu)
    nest.remove(nest[b])
    nest.insert(b,3*cliente_b[0]-jj)
    nest.remove(nest[d])
    nest.insert(d,3*cliente_d[0]-vv)
    swap(nest,a,c)
    swap(nest,b,d)
    fitness=minimo

    ###################################################################

    return (nest,fitness)

### Neighbourhood Structures
def shift(sequence,i,j):
    aux=sequence[i]
    sequence.remove(sequence[i])
    sequence.insert(j,aux)

def shift_1_0(nest,a,b):
    nest = nest[0][:]
```

```
    aux=0
    minimo=99999999 #Realizamos una busqueda local

    ##### CHECK EXECUTION (elijo ejecución con menor distancia) #####

    cliente=interprete(nest[a])
    for i in range(3):
        nest_p=nest[:]
        nest_p.remove(nest_p[a])
        nest_p.insert(a,3*cliente[0]-i)
        shift(nest_p,a,b)
        null,aux=interpretaResultado(nest_p)
        if aux<minimo:
                minimo=aux
                ii=i
    nest.remove(nest[a])
    nest.insert(a,3*cliente[0]-ii)
    shift(nest,a,b)
    fitness=minimo

    ############################################################

    return (nest,fitness)


#*****LIB*****LIB*****

#%% *****TESTER*****TESTER*****

### Definition of the probrem - Following Solomon Benchmarks
classProblem="r1"   # The unique class implemented is "r1"
testNumber=10        # From 1 to 12 (to class problem "r1")
sizeProblem=100   # Clientes (numCustomer)
numExecutions=3

for repetition in range(10):

    print "r1-",testNumber,"-",sizeProblem
    print 'Repeticion ',repetition+1

    ### Create the problem
    myproblem=Problem(classProblem,testNumber,sizeProblem,numExecutions)
    depot=myproblem.depot
    customer=myproblem.customer     # Customer 0 = Depot



        ### Calculate de distance between customers
    myproblem.distances()
    d=myproblem.d       #type diccionario, distancias

        # PLot the problem
    myproblem.plot()


        ########  PARAMETER  ########

    Nnest=25
    prob_a=0.25
```

```
    #ITERAMOS: marcamos como limite N° iteraciones o Tiempo (s)
    MAXiterations=35000
    TiempoMAX=0 #segundos
    mejora=True #True/False

        ###################################

    #*****TESTER*****TESTER*****


    #%%

        # Read the problem charasteristic

    depot=myproblem.depot
    customer=myproblem.customer # Customer 0 = Depot

    #d=myproblem.d   #myproblem.distances()

    HT=depot.tw[1]


        #####  INITIALIZE HOST NEST #####

    vect=[]
    nests=[]
    n=len(customer)-1 #customer 0 = depot
    initPath=range(1,(3*n)+1) #initPath=EGG   SOLO CUSTOMER

    t_limite=HT #customer[0].execution[0].tw[1] #=HT
    t_servicio=0

    for i in range(Nnest):
        random.shuffle(initPath)
        egg=[]
        tiempo=0
        penal=0
        cliente_servido=np.zeros(sizeProblem, dtype=bool) #comprueba si se ha
servido customer
        for ii in range(len(initPath)):

            if cliente_servido[int(math.ceil(initPath[ii]/float(3)))-
1]==False:

                egg.append(initPath[ii])
                cliente_servido[int(math.ceil(initPath[ii]/float(3)))-1]=True

        null,fitness=interpretaResultado(egg)

        nests.append((egg[:],fitness))

        ## FITNESS: ordenamos por menor fitness

    nests.sort(key=lambda tup: tup[1])
    best_egg=nests[0]


    pa=int(prob_a*Nnest)
    pc=int(0.6*Nnest) #MEJORA

    #%%
        ###### Starting ITERATIONS ######
```

```
    t1=time()
    fit=[]
    vect_t=[]
    for t in range(MAXiterations):
    #while time()-t1<TiempoMAX:

        if mejora==True: cuckooNest = nests[randint(0,pc)] #MEJORA
        elif mejora==False: cuckooNest = nests[randint(0,Nnest-1)]

        m=len(cuckooNest[0])
        e=levyFlight(randF())
        if(e>2):
            cuckooNest = doubleBridgeMove(cuckooNest,randint(0,m-
1),randint(0,m-1),randint(0,m-1),randint(0,m-1))
        elif(e<1.5):
            cuckooNest=shift_1_0(cuckooNest,randint(0,m-1),randint(0,m-1))
        else:
            cuckooNest = twoOptMove(cuckooNest,randint(0,m-1),randint(0,m-1))

        randomNestIndex = randint(0,Nnest-1)

        if(nests[randomNestIndex][1]>cuckooNest[1]):
            nests[randomNestIndex] = cuckooNest

        for i in range(Nnest-pa,Nnest):
            m=len(nests[i][0])

            #### NEW ###
            ee=levyFlight(randF())
            if(ee<1.5):
                nests[i]=shift_1_0(cuckooNest,randint(0,m-1),randint(0,m-1))
            #elif(ee>2):
                #nests[i]=doubleBridgeMove(cuckooNest,randint(0,m-
1),randint(0,m-1),randint(0,m-1),randint(0,m-1))
            else:
                nests[i] = twoOptMove(cuckooNest,randint(0,m-1),randint(0,m-
1))

            ###
        nests.sort(key=lambda tup: tup[1])
        fit.append(nests[0][1])
        vect_t.append(time()-t1)


        ######################################################################
    tiempo_ejecucion=time()-t1

    #%% INTERPRETACION DE LA SOLUCION

    ruta,dist=interpretaResultado(nests[0][0])

    routes=modoTester(ruta)
    for i in range(len(routes)):
        print routes[i],distanciruta(routes[i])


    print 'Distancia Total= ', dist
    print 'N° vehiculos= ', ruta.count(0)


    ### GRAFICAR RUTAS ###
```

```
    #routes=[]
    xyroutes=[]

    #Calcular xyroutes
    for route in routes:
        xyroute=[]
        for customers in route:

xyroute=xyroute+[customer[customers[0]].execution[customers[1]].position]
        xyroutes=xyroutes+[xyroute]

    # Graficar
    for xyroute in xyroutes:
                xs = [x[0] for x in xyroute]
                ys = [y[1] for y in xyroute]
                plt.figure("customers")
                plt.plot(xs,ys)

    #Guardar Plot
    guardaren="/Users/CarlosUA/Documents/TFM/Resultados/graficas/r1-%s-
%s_%s.eps" %(str(testNumber), str(sizeProblem), str(repetition+1))
    plt.savefig(guardaren, format='eps', dpi=1000)

#plt.savefig('/Users/CarlosUA/Documents/TFM/Resultados/graficas/prueba.eps',
format='eps', dpi=1000)
    #os.system('say "Fin de ejecucion"')
    plt.clf()

    #Guarda datos
    guardarres="/Users/CarlosUA/Documents/TFM/Resultados/resultados/r1-%s-
%s_%s.txt" %(str(testNumber), str(sizeProblem), str(repetition+1))
    f = open(guardarres, 'w')
    f.write('***** PARAMETROS *****\n')
    f.write('Nidos='+str(Nnest)+'\n')
    f.write('Probabilidad abandono='+str(prob_a)+ '\n')
    f.write('Iteraciones='+str(MAXiterations)+ '\n')

    f.write('***** SOLUTION *****\n')
    f.write('Tiempo ejecucion (min)='+str(tiempo_ejecucion/60)+ '\n')
    f.write('Distancia Total='+str(dist)+ '\n')
    f.write('N° vehiculos='+ str(ruta.count(0))+'\n')
    for route in routes:
        f.write(str(route))
        f.write(str( distanciruta(route))+'\n')
    f.close()
```