Diseño y desarrollo de un nuevo algoritmo basado en la naturaleza para la resolución del Problema del Vendedor Ambulante.

Iñigo García De las Cuevas
Máster Universitario en Bioinformática y Bioestadística
Programación para la bioinformática

Brian Jiménez García

2 de Enero de 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño y desarrollo de un nuevo algoritmo basado en la naturaleza para la resolución del Problema del Vendedor Ambulante.*
Nombre del autor:	*Iñigo García De las Cuevas*
Nombre del consultor/a:	*Brian Jiménez García*
Nombre del PRA:	*María Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	*01/2019*
Titulación:	*Máster en Bioinformática y Bioestadística*
Área del Trabajo Final:	*Programación para la bioinformática*
Idioma del trabajo:	*Castellano*
Palabras clave	*Optimización combinatoria, Problema del Vendedor Ambulante, algoritmo basado en la naturaleza, Enjambre de Partículas, Algoritmo Genético, configuración, Problema del Vendedor Ambulante.*

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

El Problema del Vendedor Ambulante, es un problema de optimización combinatoria que trata de encontrar una ruta que pase por una serie de puntos de manera que ésta la menor distancia posible. La resolución de este problema se complica exponencialmente al aumentar el número de puntos por los que la ruta debe pasar.

Para la resolución de este problema se utilizan métodos computacionales que realizan cálculos para el avance de la búsqueda de la ruta óptima, algunos de los cuales se basan en procesos biológicos por la capacidad de solución a problemas complejos que brindan.

Uno de estos métodos es el llamado Optimización por Enjambre de Partículas que modela conductas sociales como el movimiento de un enjambre de abejas o un banco de peces. Otro de los métodos es conocido como Algoritmo Genético, el cual basa la generación de soluciones en operadores como la recombinación o mutación de cromosomas.

Z.E.R.G. es un algoritmo ideado para la resolución del Problema del Vendedor Ambulante, combinando elementos de los dos métodos mencionados. El algoritmo cuenta con 9 configuraciones distintas que caracterizan el desempeño de su búsqueda de la soluciones.

En el presente trabajo, se presenta el diseño y la implementación del algoritmo, además de un análisis de los resultados generados para Problemas

del Vendedor Ambulante de variada dificultad.

Abstract (in English, 250 words or less):

The Travelling Salesman Problem is a combinatorial optimization problem that aims to find the path that goes through a series of points, with the least possible distance. The resolution of this problem grows exponentially with the number of points that the path must go through.

For the resolution of this problem computational methods are used in order to advance in the search of the optimal solution, some of which are based in biological processes because of the capacity to solve complex problems that they bring.

One of these methods is the one called Particle Swarm Optimization which models social conducts such as the movement of a bee swarm or a fish school. Another of the methods is known as Genetic Algorithm, which bases the generation of solutions in operators like the crossover or mutation of chromosomes.

Z.E.R.G. is an algorithm devised for the resolution of the Travelling Salesman Problem, combining elements of the two mentioned methods. The algorithm has different configurations that characterize the performance of its solution search.

In this assignment, the design and development of the algorithm is presented, in addition to an analysis of the results generated for varying difficulty Travelling Salesman Problems.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

La optimización de problemas de alta complejidad no es un tema reciente, aunque con los últimos desarrollos en las tecnologías de la información y computación, recibe creciente atención. Para la optimización de dichos problemas se utilizan técnicas de programación que muchos conocerán con el nombre de Inteligencia Artificial, de la cual nace el aprendizaje automático o Machine Learning.

El aprendizaje automático se basa en la transcripción de problemas con un alto número de factores condicionantes a un programa informático, de manera que el propio programa sea capaz de aprender las decisiones o resultados esperados, basados en los valores de los factores recibidos. Se podría entender como el proceso de aprendizaje de un bebé, el cual durante su aprendizaje para caminar cae repetidas veces al suelo, pero vuelve a levantarse, sabiendo que los movimientos y fuerzas realizados previamente resultaron en una caída, o dicho en términos más generales, en una recompensa negativa. Este mismo principio puede aplicarse a los algoritmos para dar lugar a una inteligencia artificial.

Inicialmente, los altos tiempos de computación debidos a la tecnología contemporánea hicieron que la inteligencia artificial como la conocemos hoy no pudiera alcanzar su máximo potencial. Su utilización podía encontrarse en las líneas de fabricación de coches, donde cada etapa de fabricación registraba diferentes parámetros de calidad de ensamblaje. Estos parámetros se tenían en cuenta en las subsiguientes etapas para ser corregidos o desechar el coche por la acumulación de errores en cada etapa aunque éstos no fueran decisivos en cada una de ellas. Esta tecnología sigue aplicándose a día de hoy[1].

Bien son conocidos algunos ejemplos de inteligencia artificial como los almacenes llenos de robots que traen y llevan paquetes a los diferentes operarios sin llegar a chocarse entre sí[2], o también los no tan novedosos sistemas expertos para la gestión del tráfico aéreo o la entrada y salida de trenes a grandes estaciones. También puede aplicarse a la búsqueda de rutas óptimas a la hora de planear viajes complejos que es de lo que tratará este proyecto.

Algunos de estos algoritmos para la búsqueda de soluciones óptimas están basados en fenómenos de la naturaleza como el comportamiento de animales o la evolución biológica y su base genética. Estos algoritmos imitan procedimientos pulidos por la evolución, para plantear una serie de

pasos que permiten generar una solución para estos problemas de alta complejidad.

Uno de estos tipos de algoritmos son los algoritmos de enjambre, los cuales imitan el comportamiento de los enjambres de insectos. En un enjambre cada insecto actúa de manera individual, aunque de alguna manera éstos parecen coordinarse entre sí ya que actúan en conjunto para un objetivo común como puede ser el de conseguir comida o defenderse de atacantes.[5] Otro ejemplo de este tipo de algoritmos son los Algoritmos Genéticos, basados en la capacidad del ADN de dos progenitores de recombinarse entre sí para formar nuevos cromosomas.

Estas propiedades mencionadas pueden utilizarse a la hora de implementar algoritmos de búsqueda de soluciones, de manera que un grupo de soluciones pueden interactuar entre sí para evolucionar hacia una solución de mayor calidad o más cercana a la solución óptima al problema en cuestión.

Aun con la infinidad de posibilidades que la inteligencia artificial brinda, cabe la posibilidad de que la solución propuesta por uno de sus algoritmos no sea la óptima, bien sea por limitaciones del propio algoritmo o por propia complejidad de los fenómenos a predecir. Sin embargo, muchas veces las soluciones propuestas permiten un acercamiento hacia la solución óptima o incluso puede considerarse como una solución lo suficientemente buena dependiendo de la criticidad del problema a resolver o del tiempo de respuesta deseado.

1.2 Objetivos del Trabajo

Este proyecto se basa en el diseño e implementación de un algoritmo basado en la naturaleza, más concretamente un algoritmo de enjambre, para la predicción de la ruta óptima entre una lista de ciudades más comúnmente conocido como el *Travelling Salesman Problem* (TSP) o Problema del Vendedor Ambulante.

El proyecto consistirá, fundamentalmente, en el desarrollo del algoritmo capaz de devolver la ruta más corta para un grafo de distancias, por ejemplo una lista de ciudades y coordenadas introducidas, en el lenguaje de programación Python. Además, el software representará gráficamente el progreso temporal de la búsqueda realizada por el algoritmo.

Para ello, se plantean objetivos específicos de cara al desarrollo propios del ciclo de vida del desarrollo software, los cuales se traducen a tareas específicas y testeo del algoritmo, con diferentes grupos de ciudades para realizar pruebas de diferente complejidad.

1.3 Objetivos Específicos

1. Diseño del banco de pruebas

    a. Elección de grupos de ciudades que formarán grupos de diferente complejidad.

    b. Implementación de dichos grupos, con la localización de cada ciudad perteneciente a cada grupo.

    c. Uso de otros problemas de referencia.

2. Desarrollo del algoritmo

    a. Identificación del algoritmo basado en la naturaleza y justificación.

    b. Implementación del algoritmo base para la generación de soluciones a problemas multivariables discretos.

    c. Implementación de distintos tipos de crossover y mutación de partículas sobre el algoritmo básico, dando lugar a una serie de configuraciones para un algoritmo híbrido genético-enjambre.

3. Estudio del rendimiento de las diferentes configuraciones.

    a. Evaluación del desempeño de los algoritmos sobre los diferentes grupos de ciudades que conforman el banco de pruebas.

    b. Estudio de la configuración óptima para cada grupo de pruebas.

4. Representación de resultados

    a. Elección de método de visualización de la progresión de búsqueda de soluciones.

    b. Implementación de dicha representación elegida.

1.4 Recursos

- La librería será escrita en el lenguaje de programación Python, apoyándose en su versión 3.6, utilizando la distribución Anaconda de 64 bits para Windows.

- El desarrollo del algoritmo ha sido gestionado mediante la ayuda de un repositorio en *GitHub*[21].

- La lista de ciudades y distancias reales entre sí serán introducidas en forma de listas con coordenadas para cada ciudad. Las distancias entre sí serán calculadas a posteriori partiendo de dichas coordenadas. Para la obtención de estos datos se utilizará la página *Getty Thesaurus of Geographic Names*®[3] u otras fuentes[6].

- La representación gráfica de la evolución de la búsqueda de la solución óptima para el problema se apoyará en las librerías open source *networkx*[22] y *matplotlib*[23].

- El desarrollo, implementación y testeo se llevará a cabo en un PC ASUS R500A:K55A con un procesador de 8 núcleos de 2.4 GHz, 8GB de memoria RAM y Windows 10 Home como sistema operativo.

1.5 Fases del proyecto

- Fase 0. Búsqueda bibliográfica y recopilación de documentación para el desarrollo del proyecto.

- Fase 1. Inicio de la redacción de la memoria, investigación del estado del arte de los algoritmos de enjambre y preparación del entorno de programación.

- Fase 2. Recopilación de datos de ciudades y preparación de bancos de pruebas.

- Fase 3. Implementación del algoritmo de enjambre base.

- Fase 4. Implementación de distintos tipos de crossover y mutación de partículas. Implementación de la representación gráfica de la evolución de la búsqueda.

- Fase 5. Evaluación de rendimiento de las diferentes configuraciones del algoritmo con respecto a bancos de pruebas de creciente complejidad.

1.6 Planificación con hitos y temporalización.

Etapa 0 [19 Septiembre – 8 Octubre]

Tareas	Estado	Septiembre 2018												Octubre 2018							
		19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6	7	8
Etapa 0: Investigación y revisión bibliográfica	**Completado**																				
Recopilación documentación inteligencia artificial, machine learning, optimización	Completado																				
Revisión de algoritmos basados en la naturaleza	Completado																				
Estado del arte de algoritmos de enjambre aplicados a TSP	Completado																				
Definición de contenidos del TFM	Completado																				

Ilustración 1. Diagrama de Gant de la Etapa 0

En la etapa inicial se plantea el proyecto desde cero. Durante los primeros días se realiza una búsqueda bibliográfica exhaustiva acerca de los temas de inteligencia artificial, aprendizaje automático y algoritmos basados en la naturaleza. También se realiza una revisión sobre el estado de los algoritmos de enjambre aplicados al problema de este trabajo.

Etapa 1 [4 Octubre– 15 Octubre]

Tareas	Estado	Octubre 2018										
		4	5	6	7	8	9	10	11	12	13	14	15
Etapa 1: Planificación y definición del trabajo	**Completado**												
Definición de objetivos generales y específicos	Completado												
Definición del plan de trabajo y hitos	Completado												

Ilustración 2. Diagrama de Gant de la Etapa 1

En esta etapa se define más claramente los contenidos del trabajo, los objetivos generales y específicos. Además, se definen las fechas límite para los diversos entregables.

Etapa 2 [16 Octubre – 22 Noviembre]

Ilustración 3. Diagrama de Gant de la Etapa 2

En esta fase se rediseñó la planificación del trabajo de forma que la implementación y el testeo del algoritmo fueran secuenciales en lugar de paralelos, más adecuadas para el ritmo de trabajo y disponibilidad del autor.

Durante este periodo se diseñó el banco de pruebas, las funciones auxiliares para la lectura de datos y generación de resultados, y las diferentes configuraciones del algoritmo. La implementación de cada unidad del algoritmo requería de una posterior verificación de su correcto funcionamiento.

Etapa 3 [23 Noviembre – 21 Diciembre]

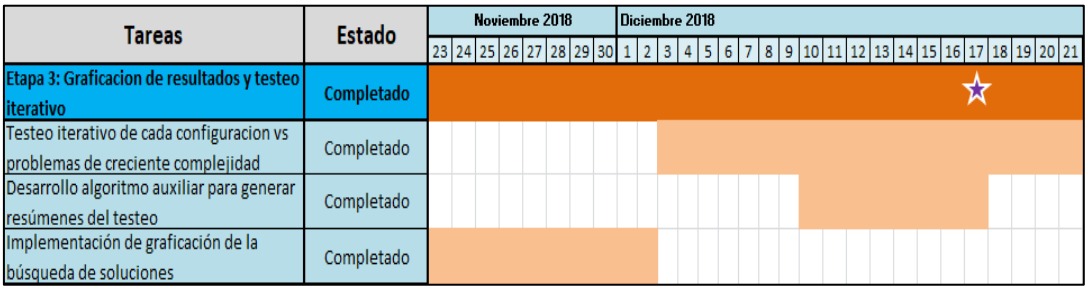


Ilustración 4. Diagrama de Gant de la Etapa 3

Se adaptó el algoritmo para su llamada iterativa y con el fin de reducir su impacto en la memoria, para agilizar el proceso de cálculo y evitar la ralentización de su ejecución durante el avance del mismo. También, se

implementó la generación una imagen que contiene el gráfico final representativo de la ruta óptima generada al final de cada búsqueda.

Esta creación y guardado de gráficas provocó un retraso en la ejecución del algoritmo para todos los problemas disponibles, extendiendo el plazo de la etapa hasta el 21 de diciembre. Se generaron alrededor de 3GB de resultados. A su vez, la gran cantidad de *log files* generados requirió el desarrollo de un programa auxiliar para escanear y resumir los resultados generados para la facilitación de su posterior análisis.

Etapa 4 [12 Diciembre – 2 Enero]

Tareas	Estado	Diciembre 2018										Enero	
		22	23	24	25	26	27	28	29	30	31	1	2
Etapa 4: Evaluación de resultados del testeo iterativo y redacción de la memoria	**Completado**												
Evaluación del los resultados para las diferentes configuraciones del algoritmo	Completado												
Redacción de la memoria	Completado												
2º Testeo iterativo de cada configuracion vs problemas de creciente complejidad	Completado												
Desarrollo de modificaciones al algoritmo auxiliar para generar resúmenes del testeo	Completado												
Documentación para el uso algoritmo	No Completado												

Ilustración 5. Diagrama de Gant de la Etapa 4

La concentración de los esfuerzos en la redacción de la memoria hizo que la evaluación de los resultados se aplazara hasta el 28 de diciembre. Al examinar los resultados, se observaron errores derivados de la incorrecta implementación del algoritmo.

Acto seguido, se iniciaron nuevas tareas para la corrección y mitigación de los errores en los resultados del primer testeo iterativo que se llevaron a cabo entre el 28 y 30 de diciembre.

La inicialmente planeada documentación para el uso del algoritmo no fue escrita por el cambio de los objetivos del propio proyecto durante el desarrollo del mismo.

Etapa 5 [28 Diciembre – 10 Enero]

Tareas	Estado	Diciembre 2018				Enero 2019							
		28	29	30	31	1	2	3	4	5	6	7	8	9	10
Etapa 5: Cierre del trabajo, documentación, presentación y defensa	Completado						★								★
Revisión y cierre de la memoria del trabajo	No Completado														
Presentación para la defensa	En curso														

Ilustración 6. Diagrama de Gant de la Etapa 5

En esta etapa estaba previsto realizar la revisión de la memoria pero esto no fue posible por el inesperado retraso en la etapa anterior. En el periodo del 3 al 10 de Enero se preparará y realizará la presentación del trabajo de final de máster.

1.7 Resultados del proyecto

1. Estado del arte de algoritmos inspirados en la naturaleza y algoritmos de enjambre y su utilización para la resolución del Problema del Vendedor Ambulante.

2. Algoritmo híbrido combinando características de Algoritmos Genéticos y de enjambre. Documentación para su uso en la resolución de problemas de optimización combinatoria. Representación gráfica de la búsqueda para la presentación de la evolución de la búsqueda y obtención de resultados.

3. Memoria final.

4. Diferentes PECs entregadas:

    a. **PEC0**. Definición del trabajo, objetivos a desarrollar y bibliografía de apoyo.

    b. **PEC1**. Planificación y planteamiento del trabajo, especificación de objetivos.

    c. **PEC2**. Informe de seguimiento. Algoritmo base implementado y primeros resultados obtenidos. Identificación de riesgos y posibles modificaciones en el desarrollo de la librería.

    d. **PEC3**. Informe de seguimiento. Algoritmo totalmente implementado y listo para su evaluación relativa a la configuración

elegida. Evaluación del rendimiento con respecto a problemas de variable complejidad.

1.8 Descripción de capítulos de la memoria

2. *Travelling Salesman Problem* (TSP)

En este capítulo se recoge el fundamento del problema que se quiere resolver mediante la utilización del algoritmo desarrollado.

3. Algoritmos inspirados en la naturaleza

En este capítulo se recoge el fundamento teórico de los algoritmos basados en la naturaleza y un resumen del estado del arte de los mismos.

4. Optimización por Enjambre de Partículas (PSO) y estado del arte para su utilización en la resolución del TSP

Profundización en uno de los tipos de los algoritmos basados en la naturaleza descrito en el anterior capítulo, los *Particle Swarm* o algoritmos de enjambre. Se describe la historia de los algoritmos de enjambre, su evolución a lo largo del tiempo y sus aplicaciones al problema en cuestión.

5. Módulos del algoritmo

Descripción detenida de la implementación de las diferentes unidades del algoritmo.

6. Diseño y limitaciones del algoritmo

Fundamento operativo del algoritmo y su uso intencionado así como sus restricciones.

7. Resultados del algoritmo

En este capítulo se muestran los resultados de diversas ejecuciones del algoritmo, con diferentes configuraciones, respecto a problemas de diversa complejidad.

8. Conclusiones y futuro desarrollo

En este capítulo final se recogen las conclusiones y reflexiones del autor tras el diseño del algoritmo y su testeo frente a diferentes bancos de pruebas.

9. Glosario

Listado de términos y acrónimos utilizados en la memoria.

10. Referencias

Referencias a la bibliografía y recursos utilizados durante la redacción de la memoria.

2. *Travelling Salesman Problem* (TSP)

El *Travelling Salesman Problem*, o traducido como el *Problema del Vendedor Ambulante*, es un problema clásico de optimización. El problema fue formulado por primera vez en el año 1930 y es uno de los problemas más estudiados en optimización combinatoria, siendo considerado el problema por excelencia de la disciplina.

El problema trata de lo siguiente:

"Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta que pasa por cada una de las ciudades una sola vez y finaliza en la ciudad de origen?"

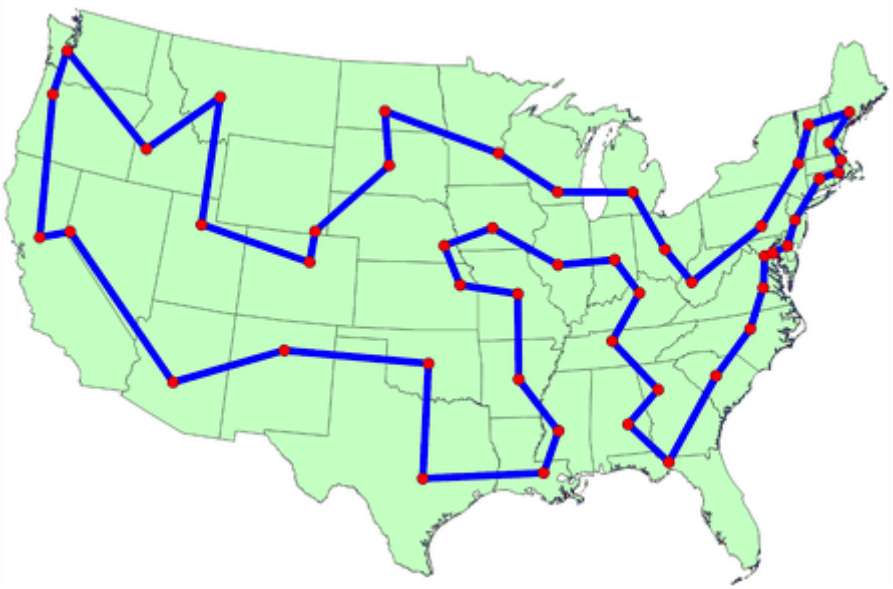


Ilustración 7. Ejemplo de ruta más corta entre una lista de ciudades[4]

Dado un grafo G = (N, E), donde N es el conjunto de nodos/ciudades y E es el conjunto de trayectorias que unen todas las ciudades entre sí, con un coste asociado a cada trayectoria entre dos ciudades, el problema trata de encontrar el ciclo entre todos los nodos con el menor coste asociado.

La longitud de una ruta se consigue sumando los costes de todas las trayectorias de una ruta. En el caso de ser los costes únicamente las distancias entre nodos, el problema es calificado como simétrico dado que el coste de un nodo *i* a un nodo *j* idéntico al coste de *j* a *i*. Dado que únicamente se evalúa la distancia de ruta, todos los problemas estudiados en el presente trabajo son de carácter simétrico.

Aunque pueda parecer un problema de fácil solución, el tiempo de computación necesario para su resolución aumenta exponencialmente a medida que

aumenta el número de ciudades a visitar. Además, otros factores pueden ser introducidos en el problema para dotarlo de mayor complejidad como por ejemplo establecer límites temporales, costes de viaje, recompensa de visitar cada ciudad etc.

Como manera de combatir esta complejidad surgen algoritmos para la búsqueda de soluciones, capaces de realizar cálculos a una altísima velocidad que ayudan en la búsqueda de una solución óptima, o al menos cercana a la óptima. Algunos de estos algoritmos se encuentran inspirados en la naturaleza y los expondremos en el siguiente capítulo.

3. Algoritmos inspirados en la naturaleza

El comportamiento social de los animales es algo que ha fascinado a quienes lo estudian desde hace muchos años. ¿Quién coordina un enjambre de abejas? ¿Cómo puede un banco de peces moverse en armonía? En definitiva, ¿Quién se encarga de dar órdenes, prever el futuro, elaborar planes y mantener la estabilidad?

Cada insecto de una colonia parece actuar de forma independiente, y sin embargo, la colonia en conjunto trabaja de manera organizada. Algunos tipos de hormigas forman cadenas y estructuras con sus propios cuerpos para alcanzar lugares que de forma individual no podrían. Por otro lado, un enjambre de abejas puede coordinarse ante el ataque de un invasor para proteger a la reina y mantener así la fuente de generación de nuevos individuos.

Al mismo tiempo, no todos los individuos de una colonia de insectos realizan las mismas tareas, sino que cada individuo realiza aquellas tareas para las cuales está especializado, ya sea por su morfología, edad o incluso prioridad de las tareas a realizar de la propia colonia. Todo esto se basa en la evolución de las especies, proceso por el cual unas características son más deseables frente a otras ante un entorno cambiante.

Muchos de los aspectos de las actividades colectivas de los insectos surgen de la autoorganización de los propios insectos. Las teorías de la autoorganización se originan para dar explicación a la aparición de patrones macroscópicos resultado de procesos en el nivel microscópico, aplicándose a los insectos para explicar la complejidad de su comportamiento colectivo surgido de las interacciones entre individuos con comportamientos más simples.

La autoorganización se basa en cuatro principios[5]:

1. **Retroalimentación positiva**.

   Si la acción de un individuo del sistema resulta exitosa en su objetivo, ésta debe ser comunicada al resto de individuos. Cuantos más individuos se dirijan hacia una misma solución, más individuos serán atraídos hacia la misma a su vez.

2. **Retroalimentación negativa**.

   La parte contraria a la retroalimentación positiva. Ayuda a estabilizar el comportamiento colectivo de manera que el colectivo no se comporte como un individuo. En el ejemplo de la búsqueda de alimentos la retroalimentación negativa podría manifestarse como el agotamiento de una de las fuentes de alimento.

3. **Interacción**.

   Los individuos del colectivo deben utilizar tanto sus resultados como los de sus vecinos para lograr el objetivo común. En esta propiedad se basa la complejidad de las soluciones proporcionadas por el sistema. Las interacciones entre los individuos pueden ser tanto directas como indirectas.

4. **Amplificación de fluctuaciones**.

   La aleatoriedad del sistema es clave para el descubrimiento de nuevas soluciones y evitar la temprana convergencia del sistema a una solución.

De esta manera se puede entender una colonia de insectos como un sistema descentralizado, el cual se autoorganiza para la resolución de problemas. Sin embargo, el traslado de estos sistemas a un programa informático no resulta tarea fácil ya que requiere un profundo conocimiento tanto del comportamiento de cada individuo como de las interacciones, tanto entre individuos como con el entorno de los mismos, necesarias para la generación del comportamiento global.

Por su flexibilidad para adaptarse estos procesos biológicos han sido adaptados a algoritmos con la esperanza de proporcionar soluciones a problemas que de otra manera no se podrían resolver debido a su altísima complejidad como por ejemplo diseñar sistemas de vuelo, gestionar el tráfico de las vías de metro de una ciudad con millones de habitantes o encontrar la ruta más corta que recorra una lista de 200 ciudades.

Los algoritmos basados en la naturaleza tienen un esquema común: se parte desde una solución base a la que se introducen una serie de cambios de forma iterativa, los cuales deben respetar unas limitaciones (*constraints*). Los resultados de estos cambios se evalúan de acuerdo con un criterio de bondad (*fitness*) como forma de determinar si el cambio va en la dirección correcta para la solución del problema.

Fase 1: Definiciones del problema. Se define el supuesto inicial y las limitaciones o *constraints* propias del problema.

Fase 2: Inicialización. De acuerdo con el supuesto definido en el primer paso, se genera un punto de partida para la búsqueda de la solución a nuestro problema.

Fase 3: Iteración. Desde el punto de inicio, se proponen cambios. Si estos cambios respetan las limitaciones y además devuelven una solución mejor que la inicial generada en el paso 2, se integran en la que será la solución final devuelta por el algoritmo.

Fase 4: Solución final. Si la solución obtenida en alguna de las iteraciones de la fase 3 cumple con los criterios que definen la solución como “suficientemente buena”, o si se llega al número máximo establecido de iteraciones, el algoritmo devuelve la solución para nuestro problema. Esta solución podría ser

descartada a posteriori en caso de no cumplir con los requisitos para el problema en cuestión. A continuación se realiza un repaso a varios algoritmos inspirados en la naturaleza.

ANT SYSTEM

Para la resolución de un TSP, cada hormiga de un *Ant System* (AS) recorre las ciudades de la lista de una en una hasta completar la lista entera, desplazándose de forma probabilística de una ciudad a otra. La elección de la próxima ciudad a visitar depende de:

1. Si la ciudad ya ha sido visitada no será considerada como candidata para la próxima visita.

2. El inverso de la distancia a una ciudad, también llamado visibilidad. Representa la deseabilidad de elección de la ciudad a raíz de su proximidad a la ciudad actual.

3. El rastro de otras hormigas o deseabilidad aprendida. Si otras muchas hormigas han elegido una ciudad como siguiente punto a visitar, lo más seguro es que sea la opción correcta para la generación de la ruta más corta. El rastro de feromonas, al contrario que la distancia, representa una información global dentro del AS y además cambia a lo largo del tiempo.

La probabilidad de que una hormiga elija una ciudad para su próxima visita se puede calcular con la siguiente fórmula[5]:

$$p_{ij}^{k}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l \epsilon J_i^k}\left[\tau_{il}(t)\right]^{\alpha}\left[\eta_{il}\right]^{\beta}}$$

Los parámetros α y β controlan la importancia que tienen la visibilidad ($\tau_{ij}(t)$) y el rastro de otras hormigas (η_{ij}) en la toma de la decisión. En el caso de que el parámetro α fuera nulo, la ciudad elegida seria aquella previamente elegida por el mayor número de hormigas. Por el contrario, si β fuera nulo, la ciudad elegida siempre sería la más cercana a la ciudad actual.

Tras completar un tour por la lista de ciudades, cada hormiga dejará un rastro por su camino, cuya intensidad será relativa a la calidad de su tour. Este rastro debe degradarse a lo largo del tiempo para evitar el refuerzo de elecciones aleatorias iniciales que a su vez causarían que la solución final no fuera la óptima.

Otra opción para la mejora de rendimiento de este modelo sería la introducción de hormigas “elitistas” que recorran la mejor solución encontrada hasta el momento, reduciendo el tiempo de computación necesario para la generación de la solución final.

ANT COLONY SYSTEM

El modelo de *Ant Colony System* (ACS) fue formulado por primera vez para la mejora de rendimiento de un AS, basándose dicha mejora en cuatro modificaciones:

1. **Diferente elección de ciudad a visitar.**

   Dependiendo del valor tomado por una variable aleatoria uniformemente distribuida, la siguiente ciudad a visitar será elegida aleatoriamente o por el contrario será aquella ciudad entre las más cercanas cuyo rastro sea también de cierta intensidad. El valor límite que determina un tipo u otro de elección puede ser alterado a lo largo de la búsqueda para favorecer la exploración o la explotación dependiendo del objetivo en el momento actual de la búsqueda.

2. **Diferente actualización del rastro de hormigas.**

   Al contrario que en un AS, en el cual todas las hormigas pueden reforzar su rastro al finalizar una iteración, en un ACS solo aquella hormiga con la mejor ruta puede reforzar su rastro de manera global. Esto provoca que la búsqueda local de otras hormigas se realice alrededor de las elecciones realizadas por la hormiga con la solución proporcionada. Para que el sistema no solo obedezca a la mejor solución proporcionada en la iteración previa, surgen actualizaciones de rastro locales.

3. **Actualizaciones locales del rastro de otras hormigas para favorecer la exploración.**

   La intensidad del rastro entre dos ciudades disminuye tras ser recorrido su camino por una hormiga. De esta manera, aquellos caminos entre ciudades más frecuentemente visitados disminuirán la intensidad de su rastro, favoreciendo así la exploración de otros caminos que podrían resultar en una mejor solución.

4. **Restricción de la próxima ciudad a visitar.**

   En lugar de considerar todas las ciudades para la próxima visita, únicamente un número predeterminado de las ciudades más cercanas serán consideradas. Una vez una ciudad ya ha sido visitada, quedará fuera de consideración para la próxima iteración de la hormiga. Una vez todas las ciudades de la lista restringida hayan sido visitadas al menos una vez, el resto de ciudades serán consideradas, siendo elegidas de manera secuencial aquellas con menor distancia a la ciudad actual.

Como posibles adiciones a este modelo también pueden considerarse las siguientes modificaciones:

- Aumentar el número de hormigas que pueden actualizar su rastro de manera global al finalizar una iteración.

- Reducir el rastro de las peores soluciones.

- Utilizar otro procedimiento para la búsqueda local.

ALGORITMOS GENÉTICOS

Estos algoritmos están inspirados en la evolución biológica y su base genético-molecular. Partiendo de una población inicial de soluciones, aquellas de mayor calidad se recombinan entre sí para dar lugar a un nuevo grupo de soluciones.

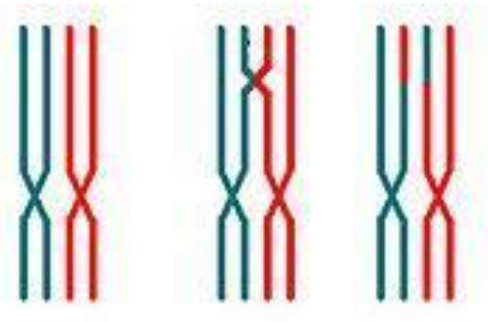


Ilustración 8. Representación gráfica de la recombinación genética[8]

También, para la exploración de nuevas soluciones que no estuvieran consideradas en la población inicial, hay otro elemento de cambio en las soluciones llamado mutación.

Estos algoritmos cuentan con fases:

1. **Generación de la población inicial.** Se genera, de forma aleatoria o no, una serie de soluciones al problema.

2. **Determinación de efectividad de cada solución.** Es preciso evaluar la calidad de las soluciones generadas para la correcta evolución de la búsqueda.

3. **Selección de soluciones progenitoras.** Aquellas soluciones cuya efectividad sea mayor de cara a la resolución del problema serán seleccionadas para dar lugar a la siguiente generación de soluciones.

4. **Generación de soluciones hijas.** Las soluciones progenitoras se recombinan entre sí, generando a su vez nuevas soluciones al problema.

5. **Mutación de soluciones.** Las soluciones generadas pueden sufrir alteraciones aleatorias (mutaciones) de manera que las soluciones generadas no dependan únicamente de aquellas generadas en la población inicial.

Esta generación de nuevas soluciones puede aplicar se forma iterativa y al final de cada iteración, tanto las soluciones progenitoras como las nuevas soluciones generadas se ven sometidas a una función que evalúa su desempeño o efectividad en la resolución del problema en cuestión.

Aquellas soluciones cuya efectividad sea menor serán eliminadas de cara a la siguiente iteración de recombinaciones y mutaciones. De esta manera, se consigue que el conjunto de soluciones converja hacia mejores soluciones.

Existen múltiples maneras de realizar las recombinaciones y las mutaciones de una solución candidata. A continuación se listan algunas de ellas (En negrita se encuentran aquellos operadores que han sido implementados en el algoritmo del presente trabajo)[9]:

Operadores de recombinación (*crossover*).

- ***Order1***
- ***Cycle***
- *Edge Recombination*
- ***PMX***
- *Order Multiple*
- *Direct Insertion*

Operadores de mutación.

- ***Inversion***
- *Random Slide*
- *Insertion*
- *Single Swap*
- ***Random Swap***
- *Scramb*

4. Optimización por Enjambre de Partículas (PSO) y estado del arte de su utilización en la resolución del TSP

El *Particle Swarm Optimization* (PSO), u Optimización de Enjambre de Partículas, es un algoritmo presentado por primera vez en 1995[12] basado en el comportamiento social observado en distintos tipos de animales que se mueven en conjunto como puede ser un banco de peces o un enjambre de abejas. Este tipo de algoritmo será el que desarrollaremos para la resolución del TSP.

El PSO ha ganado popularidad en las últimas dos décadas gracias a su robustez y eficiencia en la resolución de problemas. En este algoritmo, cada partícula representa una posible solución del problema y dichas partículas se desplazan por el espacio de soluciones buscando una solución óptima del problema.

En un enjambre de un algoritmo PSO, cada partícula:

- Tiene una posición (x) y una velocidad (V).
- Conoce su posición actual (x_t) y el valor de coste asociado a la misma ($f(x_t)$).
- Conoce su mejor posición hasta el momento (p_{best}) y su valor de coste asociado ($f(p_{best})$).
- Conoce sus vecinos, sus mejores posiciones (g_{best}) y valores de coste ($f(g_{best})$).

La nueva posición de una partícula y su nueva velocidad de movimiento se calculan de acuerdo a las siguientes fórmulas:

$$\boldsymbol{x_{t+1} = x_t + v_t}$$

$$\boldsymbol{V_{t+1} = \omega * V_t + c_1 * (p_{best} - x_t) + c_2 * (g_{best} - x_t)}$$

La nueva velocidad de una partícula depende de tres componentes:

a) Continuar su propio camino. **(Componente de impulso)**

b) Tendencia a volver a una mejor posición previa. **(Componente cognitiva)**

c) Dirigirse hacia una mejor solución encontrada por sus partículas vecinas. **(Componente social)**

La individualidad (ω, c_1) y la sociabilidad (c_2) de las partículas deben ser equilibradas[16] para realizar una búsqueda efectiva. Inicialmente, los movimientos individuales de cada partícula (a) son preferibles ya que aseguran la exploración de nuevas soluciones. A medida que la búsqueda evoluciona, las componentes cognitivas (b) y sociales (c) toman mayor importancia para que las partículas del enjambre se centren en regiones del espacio de búsqueda que contienen las mejores soluciones encontradas.

Las partículas asociadas a cada partícula (vecinos) pueden elegirse desde un principio, o pueden estar basadas en la distancia a las mismas[17]. La elección de los vecinos en función de la distancia requiere la computación de las distancias entre partículas para cada iteración, con su consecuente impacto en el rendimiento temporal del algoritmo.

El modelo PSO estándar implica que las partículas son actualizadas de manera sincrónica[14], lo que se traduce en que la posición y velocidad de cada partícula se calcula únicamente con la información de la anterior generación. Para lograr esto, el modelo PSO estándar utiliza 2 enjambres que representan la generación progenitora y la sucesora.

Por otro lado, un modelo que permite actualizar una partícula en cualquier momento[15] solo utiliza 1 enjambre. Por ello, las modificaciones realizadas hasta el momento en la presente iteración se tienen en cuenta a la hora de actualizar cada partícula. El modelo desarrollado en el presente trabajo adopta esta estrategia de actualización.

En el caso del TSP, un problema cuyas soluciones son discretas, cada partícula del enjambre contiene una secuencia que representa el orden de visita de las ciudades del problema. Para un problema de 10 ciudades, un ejemplo de enjambre de 4 partículas sería el siguiente:

Partícula 1: [1 2 3 4 5 6 7 8 9 10] **Partícula 3: [1 3 5 7 9 2 4 6 8 10]**
Partícula 2: [2 4 5 7 8 9 3 1 6 10] **Partícula 4: [2 1 4 3 6 5 8 7 10 9]**

Las partículas del enjambre pueden modificar elementos de su propia secuencia o intercambiar segmentos con las secuencias de otras partículas para dar lugar a nuevas soluciones que podrían representar una ruta con menor distancia. Cada partícula debe tener otras partículas asociadas para realizar estos intercambios de información, aunque no deben estar todas conectadas entre sí para evitar la convergencia de todo el enjambre a una misma solución.

En caso de que todas las partículas estuvieran conectadas entre sí para el intercambio de información, todas las partículas intercambiarían segmentos con la mejor de todo el enjambre por lo que inevitablemente todo el enjambre convergería rápidamente hacia la región que contiene dicha solución ya que las nuevas soluciones generadas contendrían segmentos de la primera mejor solución.

Este intercambio de información entre partículas se repite de forma iterativa hasta alcanzar alguna de las condiciones para el final de la búsqueda: obtención de una solución lo suficientemente buena, alcanzar el número máximo de iteraciones establecido o detectar que la mejor solución entre todas las del enjambre no ha cambiado durante un número determinado de iteraciones.

Para asegurar la exploración de diferentes soluciones por parte del modelo desarrollado en el presente trabajo, se han implementado operaciones de mutación que modifican aleatoriamente la secuencia contenida en cada partícula.

Los algoritmos PSO han sido exhaustivamente estudiados durante los últimos años y desde la aparición del modelo estándar, una gran cantidad de nuevos modelos han sido presentados. A continuación se realiza un repaso a algunos de estos modelos propuestos para la resolución del TSP[18]:

Una de las vertientes del tradicional algoritmo PSO para asegurar la diversidad de la búsqueda surge de añadir memoria a cada partícula del enjambre[13], de manera que cada partícula tiene una lista de soluciones alternativas que adoptar en lugar de su solución óptima local. La necesidad de mantener una lista por cada partícula impacta en la memoria necesaria para su ejecución, además de la necesidad de establecer la probabilidad con la que una partícula escogerá una de las soluciones alternativas de su lista.

Como ya comentábamos en el capítulo 2, el TSP puede adquirir mayor complejidad al añadir variables limitadoras como combustible disponible o recompensa al visitar cada ciudad. Como manera de lidiar con estas condiciones, se presentó un algoritmo de enjambre con una estrategia de intercambio de información multinivel[11], en la que cada variable limitadora es tratada como un objetivo del algoritmo al igual que la distancia a recorrer.

Una de las variantes del TSP, es el *Probabilistic Travelling Salesman Problem* en el cual dentro de una lista de ciudades, las ciudades a visitar son aleatorias para cada instancia del problema. Uno de los modelos propuestos para la resolución de este problema[19] se basa en realizar búsquedas multi-enjambre y combina características del PSO, *Expanding Neighborhood Search* (ENS) y *Greedy Randomized Adaptative Search Procedure* (GRASP)[20].

Otras propuestas de algoritmo PSO se basan en combinar el modelo tradicional con otros dando lugar a llamados modelos híbridos como por ejemplo PSO - Genetic Algorithm - Fast Local Search[18].

5. Módulos del algoritmo

El algoritmo consta de los siguientes módulos:

- ***main.py***

  Módulo principal de más alto nivel, que llama a su vez al resto de módulos para la ejecución de la búsqueda. En este módulo se realiza la generación del *log file*, el enjambre de soluciones, el reemplazo de partículas del enjambre durante la evolución de la búsqueda, la representación gráfica de la mejor ruta de cada iteración y la medición del tiempo de búsqueda.

  Dentro de este módulo también encontramos la condición para la decisión de si una búsqueda está estancada: cuando la mejor solución del enjambre no ha cambiado durante más del 15% de iteraciones máximas.

  Los parámetros de entrada de la función principal que condicionan su desempeño a la hora de realizar la búsqueda son: tamaño del enjambre, número de partículas asociadas a cada partícula, número de iteraciones máximas y operadores de recombinación y mutación.

- ***FuncionesInicializar.py***

  Módulo encargado de la selección del archivo de texto a leer, de su lectura y del cálculo de distancias entre las ciudades contenidas en el archivo de texto. El archivo de texto a leer debe ser introducido con su nombre completo (p. ej. “berlin52.txt”) y estar contenido en la carpeta *data*. Los archivos de texto deben contener en cada línea, separadas por espacios entre sí, el nombre de la ciudad, su latitud y su longitud, en valor numérico con signo.

  Como caso especial, los archivos con las capitales autonómicas y mundiales pueden ser leídos introduciendo “autonomicas” o “mundiales” en la llamada al algoritmo. Estos archivos, a diferencia de los otros, contienen los campos de cada línea separados por tabuladores en lugar de espacios. Adicionalmente, las coordenadas de las capitales mundiales en lugar de signos se encuentran anotadas con puntos cardinales al final de los valores numéricos (N, E = + / S, W = -).

- ***funcionesLog.py***

  Únicamente contiene una función dedicada a escribir la cabecera de los *log files*. En la cabecera se especifican las condiciones de la búsqueda: número de ciudades, tamaño del conjunto/enjambre de soluciones, número máximo de iteraciones para la búsqueda y la configuración de los operadores de recombinación y mutación.

- ***GenParticula.py***

  Módulo encargado de la generación de soluciones/partículas. Para ello, se le introduce una secuencia de índices representativa del orden de ciudades de la ruta, las distancias entre todas las ciudades y los índices de las partículas del enjambre que serán asociadas a la partícula a generar. A partir de la secuencia de índices y las distancias entre ciudades se calcula la distancia resultante de la ruta.

- ***Graficar.py***

  Módulo encargado de la representación gráfica de la mejor ruta obtenida en cada iteración y el guardado de las figuras generadas en cada búsqueda realizada. Las funciones para la generación de gráficos de este módulo incluyen funciones de las librerías open source *networkx*[22] y *matplotlib*[23].

- ***Evolucion.py***

  Este módulo contiene las funciones encargadas de la selección de las operaciones de recombinación y mutación utilizadas para la generación de nuevas soluciones.

- ***Crossovers.py***

  Contiene las funciones encargadas de realizar las operaciones de recombinación de secuencias. Los operadores implementados son:

  a) *Order1*

  Un segmento aleatorio de una de las dos secuencias progenitoras se elige como punto de partida para la creación de la nueva secuencia. Esta secuencia toma su misma posición dentro de la nueva. Los elementos contenidos en esta secuencia son descartados de la segunda secuencia progenitora.

  El resto de posiciones libres de la nueva secuencia son rellenadas por elementos de la segunda secuencia, en el orden en el que aparecen en la misma. A continuación se plantea un ejemplo ilustrativo:

  1. Secuencias iniciales:

     Progenitor 1: (3 1 7 5 4 2 8 9 6)
     Progenitor 2: (1 2 3 4 5 6 7 8 9)

  2. Operación inicial:

     Progenitor 1: (3 1 7 5 4 2 8 9 6)
     Progenitor 2: (1 ~~2~~ 3 ~~4 5~~ 6 7 ~~8~~ 9)

     Nueva secuencia: (- - - 5 4 2 8 - -)

3. Relleno final:

Progenitor 1: (3 1 7 5 4 2 8 9 6)
Progenitor 2: (1 ~~2~~ 3 ~~4 5~~ 6 7 ~~8~~ 9)

Nueva secuencia: (1 3 6 5 4 2 8 7 9)

b) *Cycle*

Primero se deben identificar los llamados ciclos entre las secuencias progenitoras. A continuación se muestra un ejemplo ilustrativo para la identificación de ciclos[9]:

Primer ciclo.

Progenitor 1: (8 4 7 3 6 2 5 1 9 0)
Progenitor 2: (0 1 2 3 4 5 6 7 8 9)

Comenzamos desde el primer elemento de la primera secuencia progenitora, de valor 8. En su misma posición pero en la segunda secuencia progenitora se encuentra el valor 0, el cual seguidamente buscamos en la primera secuencia.

En la misma posición que el valor 0 tiene en la primera secuencia, encontramos el valor 9 en la segunda. Si repetimos la operación previa una vez más, el valor encontrado en la segunda secuencia es el número 8, el mismo que el valor de inicio, por lo que el primer ciclo finaliza.

Tras la finalización del primer ciclo, repetimos la operación con el siguiente elemento del primer progenitor que no haya sido incluido en ciclos anteriores. En el caso del ejemplo sería el número 4.

Segundo ciclo.

Progenitor 1: (- 4 7 3 6 2 5 1 - -)
Progenitor 2: (- 1 2 3 4 5 6 7 - -)

Último ciclo.

Progenitor 1: (- - - 3 - - - - - -)
Progenitor 2: (- - - 3 - - - - - -)

Los ciclos identificados son:

	Progenitor #1	Progenitor #2
Ciclo 1	**[8 - - - - - - - 9 0]**	**[0 - - - - - - - 8 9]**
Ciclo 2	**[- 4 7 – 6 2 5 1 - -]**	**[- 1 2 – 4 5 6 7 - -]**
Ciclo 3	**[- - - 3 - - - - - -]**	**[- - - 3 - - - - - -]**

Tabla 1. Ciclos identificados entre las dos secuencias ejemplo

Estos ciclos, concatenados de forma alterna según sus progenitores, dan lugar a las nuevas secuencias:

Nueva Secuencia 1: [8 1 2 3 4 5 6 7 9 0]

Nueva Secuencia 2: [0 4 7 3 6 2 5 1 8 9]

Aunque se generen dos nuevas secuencias con este operador, sólo aquella con los ciclos impares de la primera secuencia y los pares de la segunda se evalúa en el algoritmo de este trabajo.

c) *Partially Mapped Crossover* (PMX)[9]

La operación recombinación PMX puede ser entendida como una combinación de las dos operaciones anteriormente descritas. La operación inicial es la misma que la del operador *Order1*, aunque los elementos del segmento inicial no son directamente eliminados de la otra secuencia progenitora.

Progenitor 1: (8 4 7 3 6 2 5 1 9 0)
Progenitor 2: (0 1 2 3 4 5 6 7 8 9)

Nueva secuencia: (- - - 3 6 2 5 - -)

La siguiente operación tiene cierta semejanza con la identificación de ciclos. Comenzamos con el elemento de la segunda secuencia, dentro de las posiciones cubiertas por el segmento inicial, que no está entre los elementos de dicho segmento. En nuestro ejemplo sería el número 4.

Progenitor 1: (8 4 7 3 6 2 5 1 9 0)
Progenitor 2: (0 1 2 3 4 5 6 7 8 9)

El elemento que ocupa su misma posición en la primera secuencia sería el número 6. Sin embargo, la posición 6 está incluida dentro de las posiciones que ocupa el segmento elegido de la primera secuencia por lo que la operación deberá ser repetida, hasta encontrar un elemento que no esté incluido en dichas posiciones.

Progenitor 1: (8 4 7 3 6 2 5 1 9 0)
Progenitor 2: (0 1 2 3 4 5 6 7 8 9)

Nueva secuencia: (- - 4 3 6 2 5 1 - -)

La misma operación se repite para el número 7 de la segunda secuencia.

Progenitor 1: (8 4 7 3 6 2 5 1 9 0)
Progenitor 2: (0 1 2 3 4 5 6 7 8 9)

Nueva secuencia: (- 7 4 3 6 2 5 1 - -)

Las posiciones restantes pasan a ser rellenadas por el resto de elementos de la segunda secuencia aún no incluidos en la nueva, en las mismas posiciones que en su secuencia original.

Nueva secuencia: (- 7 4 3 6 2 5 1 - -)
Progenitor 2: (0 ~~1 2 3 4 5 6 7~~ 8 9)

Nueva secuencia: (0 7 4 3 6 2 5 1 8 9)

Los números que identifican las distintas operaciones son 1 (*Order1*), 2 (*Cycle*) y 3 (*PMX*). En el caso de introducir cualquier otro número, se realizará la operación por defecto *Order1*.

- **Crossovers_test.py**

  Contiene las funciones de recombinación modificadas para la realización de pruebas para la verificación del correcto funcionamiento de las operaciones de recombinación.

- **Mutations.py**

  En este módulo están implementadas las funciones para el cambio aleatorio de soluciones (mutaciones). Las operaciones de mutación tienen un 15% de probabilidades de ocurrir y son opcionales. Los dos operadores de mutación implementados son:

  a) *RandomSwap*

  Dos segmentos aleatorios de la ruta intercambian su posición.

  (8 4 7 3 6 2 5 1 9 0) → (8 1 9 3 6 2 5 4 7 0)

  b) *Inversion*

  Se invierte el orden de los elementos contenidos en un segmento aleatorio de la ruta.

  (8 4 7 3 6 2 5 1 9 0) → (8 4 7 5 2 6 3 1 9 0)

  Los números que identifican las distintas operaciones son 1 (*Inversion*) y 2 (*RandomSwap*). En caso de introducir cualquier otro número, no se realizará ninguna operación.

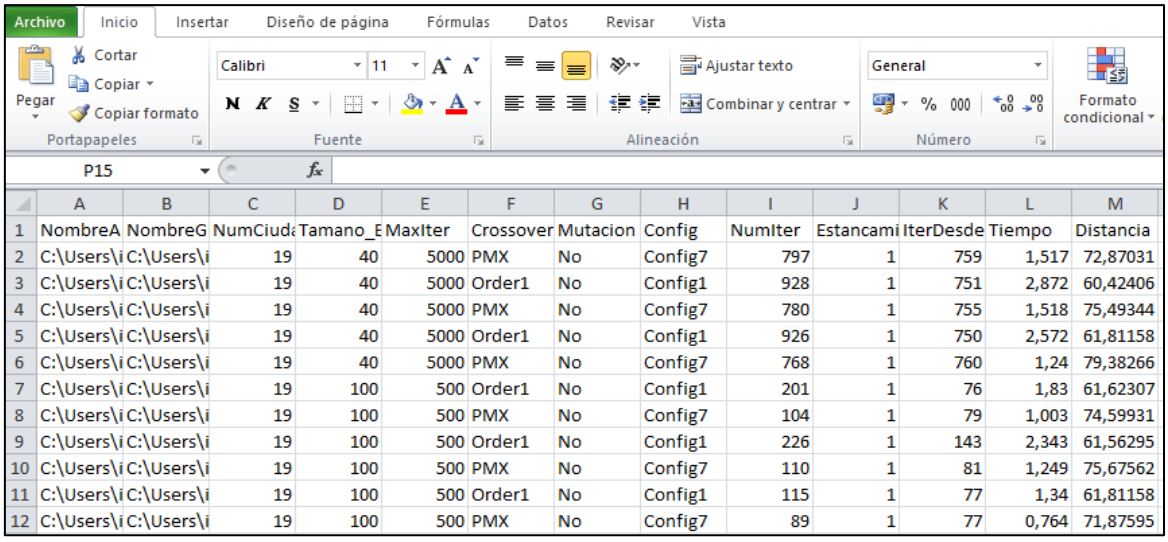
- **Mutations_test.py**

  Este módulo contiene las funciones implementadas dentro de *Mutations.py,* modificadas para la verificación de su correcto funcionamiento.

- **ResumenData.py**

  Este módulo contiene un programa auxiliar, inicialmente no planeado, para la extracción de datos de los *log files* generados durante la ejecución del algoritmo con diferentes parámetros de entrada. Todos los datos se vuelcan en una hoja Excel para facilitar su posterior análisis.

  Además de los datos crudos contenidos en los *log files*, también se calcula, como posible parámetro indicativo de la calidad de la búsqueda, cuantas iteraciones antes del final de la búsqueda se produce la última mejoría del mejor resultado global del enjambre.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	NombreA	NombreG	NumCiud	Tamano_E	MaxIter	Crossover	Mutacion	Config	NumIter	Estancami	IterDesde	Tiempo	Distancia
2	C:\Users\i	C:\Users\i	19	40	5000	PMX	No	Config7	797	1	759	1,517	72,87031
3	C:\Users\i	C:\Users\i	19	40	5000	Order1	No	Config1	928	1	751	2,872	60,42406
4	C:\Users\i	C:\Users\i	19	40	5000	PMX	No	Config7	780	1	755	1,518	75,49344
5	C:\Users\i	C:\Users\i	19	40	5000	Order1	No	Config1	926	1	750	2,572	61,81158
6	C:\Users\i	C:\Users\i	19	40	5000	PMX	No	Config7	768	1	760	1,24	79,38266
7	C:\Users\i	C:\Users\i	19	100	500	Order1	No	Config1	201	1	76	1,83	61,62307
8	C:\Users\i	C:\Users\i	19	100	500	PMX	No	Config7	104	1	79	1,003	74,59931
9	C:\Users\i	C:\Users\i	19	100	500	Order1	No	Config1	226	1	143	2,343	61,56295
10	C:\Users\i	C:\Users\i	19	100	500	PMX	No	Config7	110	1	81	1,249	75,67562
11	C:\Users\i	C:\Users\i	19	100	500	Order1	No	Config1	115	1	77	1,34	61,81158
12	C:\Users\i	C:\Users\i	19	100	500	PMX	No	Config7	89	1	77	0,764	71,87595

Ilustración 9. Ejemplo de archivo resumen

Cada línea representa una búsqueda realizada. Las columnas del archivo generado son:

1. **NombreArchivo:** Ruta del *log file* del cual se han extraído el resto de parámetros.

2. **NombreGrafo:** Ruta de la imagen correspondiente a la mejor ruta contenida en el *log file*.

3. **NumCiudades**: Número de ciudades del problema.

4. **Tamano_Enjambre**: Tamaño del enjambre introducido en la llamada al algoritmo.

5. **MaxIter**: Número máximo de iteraciones establecido para la búsqueda.

6. **Crossover**: Operación de recombinación elegida.

7. **Mutacion**: Operación de mutación elegida.

8. **Config:** Configuración de operadores de recombinación y mutación elegidos.

9. **NumIter**: Número de iteraciones al finalizar la búsqueda.

10. **Estancamiento**: Indicador de si la búsqueda ha finalizado por estancamiento o no (1=Sí / 0=No).

11. **IterDesdeUltMejora**: Número de iteraciones desde la última mejora hasta el final de la búsqueda. Este dato no se encuentra registrado de forma explícita en los *log files* sino que es calculado durante la generación de los archivos resumen.

12. **Tiempo**: Tiempo desde el inicio hasta el final de la búsqueda.

13. **Distancia**: Distancia de la mejor ruta obtenida durante la búsqueda.

14. **NumVecinos**: Número de partículas asociadas a cada partícula del enjambre, con las cuales realizará operaciones de recombinación y mutación.

6. Diseño y limitaciones del algoritmo

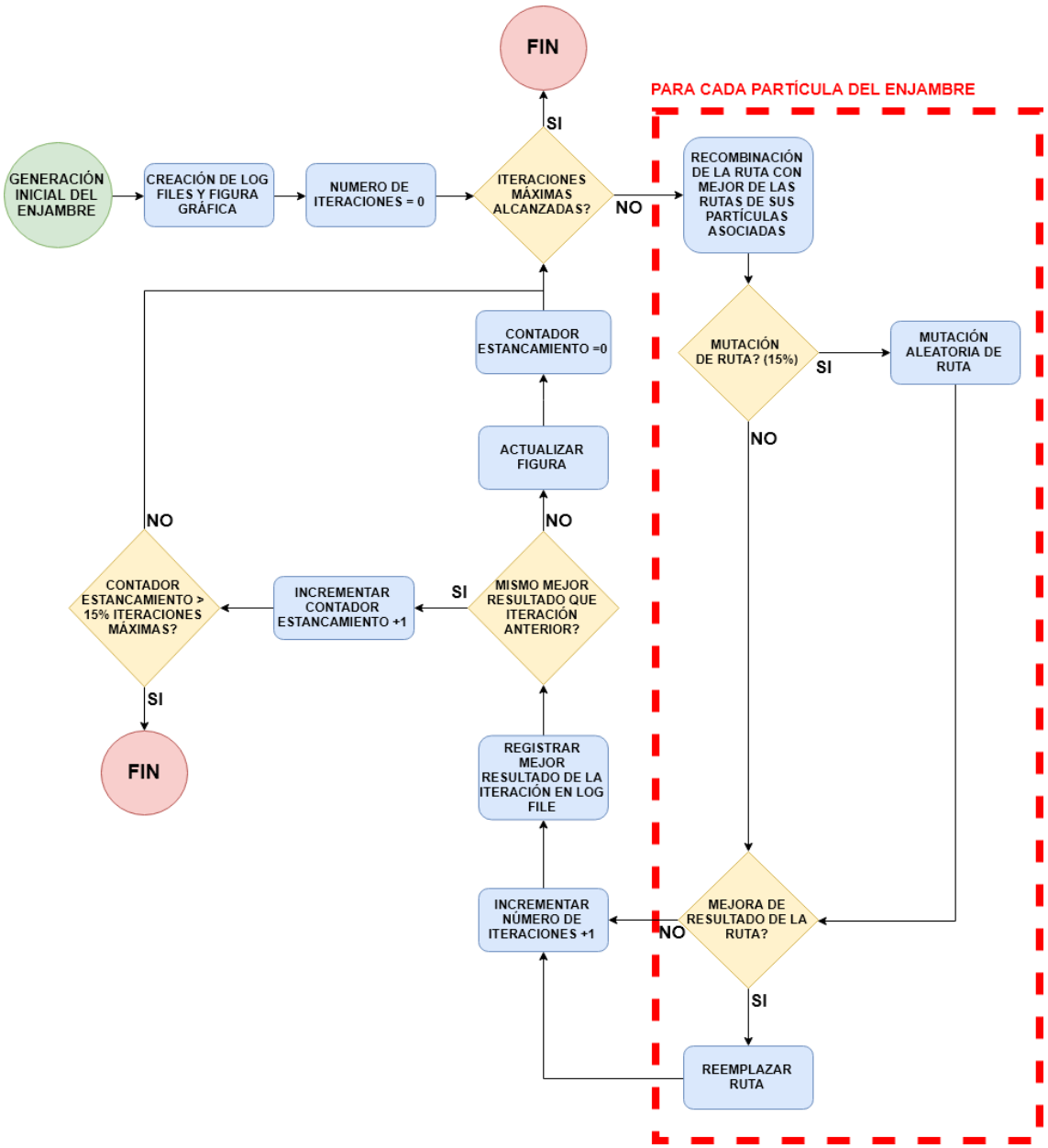


Ilustración 10. Diagrama de flujo del algoritmo

El algoritmo desarrollado ejecuta una búsqueda de la ruta más corta divida en las siguientes fases:

1. Lectura de un archivo de texto que contiene la lista de ciudades con sus coordenadas, separadas por espacios. Se calculan las distancias entre todas las ciudades.

2. Generación de un *log file*, que podrá ser analizado a posteriori, para el registro del avance de la búsqueda. El nombre del archivo responde al

nombre del problema analizado, seguido de la fecha y hora de su creación. El archivo contiene los parámetros de entrada de la búsqueda y registra la mejor solución generada en cada iteración de la búsqueda. Al finalizar la búsqueda registra la mejor solución obtenida, su distancia y el tiempo utilizado hasta la finalización de la búsqueda.

3. Creación de una figura gráfica que muestra la evolución de la búsqueda de la solución óptima. La solución se representa en forma de grafo, representando los nodos las ciudades leídas del archivo de texto, unidas entre sí por segmentos que corresponden a la mejor solución obtenida en cada iteración. La figura únicamente se actualizará en el caso de que tras una iteración la mejor solución global haya evolucionado favorablemente.

4. Generación inicial aleatoria de un enjambre de partículas. Cada partícula tiene otras partículas asociadas con las que realizar operaciones de recombinación para la generación de nuevas soluciones.

5. Generación de nuevas soluciones. En cada iteración, cada una de las partículas del enjambre elige aquella de sus partículas asociadas cuya secuencia tenga menor distancia. Posteriormente las secuencias de ambas partículas se recombinan entre sí para dar lugar a una nueva secuencia. La nueva secuencia generada tiene una posibilidad del 15% de sufrir mutaciones aleatorias.

6. En caso de que la nueva secuencia generada en el punto anterior tenga una menor distancia de ruta que la secuencia de la partícula evaluada, la secuencia de dicha partícula es reemplazada por la nueva.

7. La búsqueda finaliza cuando se llega al número máximo de iteraciones o cuando la mejor solución no ha evolucionado tras un 15% del máximo de iteraciones.

8. Al finalizar, el archivo de texto y la figura, generados en el punto 2 y 3, son guardados en un directorio para su posterior análisis. Ambos archivos comparten el mismo nombre y se difieren por su extensión (*.log* y *.png*).

Los parámetros de entrada al algoritmo que son determinantes para su desempeño son:

- **Tamaño de enjambre:** Número de partículas que compone un enjambre.
- **Número de vecinos:** Número de partículas que cada partícula tiene asociada.
- **Operador de recombinación.**
- **Operador de mutación.**

Además de estos parámetros, también se introducen otros aunque éstos no afectan al desempeño del algoritmo durante la búsqueda: figura/ventana donde se graficarán a tiempo real los avances en la búsqueda, problema a resolver y número máximo de iteraciones.

En total se han implementado 9 diferentes configuraciones para las posibles combinaciones de operadores de recombinación y mutación. La siguiente tabla asigna un número a cada configuración:

		RECOMBINACIÓN		
		Order1	Cycle	PMX
MUTACION	N/A	Config1	Config4	Config7
	Inversion	Config2	Config5	Config8
	RandomSwap	Config3	Config6	Config9

Tabla 2. Tabla de configuraciones del algoritmo

El algoritmo cuenta con las siguientes limitaciones:

- Una de las operaciones de recombinación de secuencias (*Cycle*) del algoritmo genera dos nuevas secuencias. Sin embargo, solo una de las dos secuencias es seleccionada.

- La probabilidad de una secuencia nueva generada de sufrir una mutación está fijada en 15%.

- El número de iteraciones para evaluar el estancamiento de la búsqueda está fijado en un 15% de las iteraciones máximas.

- El formato de los archivos aceptados como entrada al algoritmo está limitado a archivos de texto con tres columnas, separadas por espacios, que indican el nombre de la ciudad, su latitud y su longitud en ese orden. La latitud y la longitud de los puntos a recorrer de la ruta debe estar en formato de número decimal con signo.

- Si las posiciones elegidas aleatoriamente para realizar las operaciones de recombinación y mutación coinciden o se encuentran en la primera o última posición de la secuencia, no se realizará operación alguna sino que se devolverá la misma secuencia original introducida para la operación.

- No hay opción para elegir los recursos computacionales dedicados al proceso de búsqueda.

- El algoritmo no se encuentra adaptado para ejecutarse directamente desde la línea de comando. Es necesario escribir un script de llamada al algoritmo.

```
import main
import Graficar

# Input de datos
archivos = ["autonomicas", "mundiales","berlin52.txt","ch130.txt","ch150.txt","PoblacionesSpa.txt"]
# Creamos la figura
figura = Graficar.crear_figura()
# Tamnos de enjambres
tamanos = [40,100,250,500]
# Numero maximo de iteraciones
iteraciones = [100,200,500,1000]
# Numero de vecinos entre particulas
numerodevecinos = [3,5,10] # Numero de vecinos para cada particula
# Operadores de crossover
operadores_c = [1,2,3]
# Operadores de mutacion
operadores_m = [0,1,2]

for archivodatos in archivos:
    for tamano in tamanos:
        for itermax in iteraciones:
            for numvecinos in numerodevecinos:
                for operador_cross in operadores_c:
                    for operador_mut in operadores_m:
                        for i in range(0,10):
                            main.ZERG(figura,archivodatos,tamano,itermax,numvecinos,operador_cross,operador_mut)
```

Ilustración 11. Ejemplo de script de llamada al algoritmo

Los argumentos de entrada pueden ser declarados de manera que múltiples combinaciones de los mismos se cubran durante una sola ejecución. Adicionalmente, al ser declarada la figura una sola vez y ésta ser introducida como argumento de entrada en la llamada al algoritmo se reduce el impacto en memoria por parte de la generación de figuras.

- Una vez escrito el script de llamada, es necesario colocarlo en el directorio en el que se encuentra el algoritmo desarrollado para poder ejecutarlo desde la consola de Python.

  Para ello, se utilizaron los siguientes comandos en la consola de Python:

  import os
  os.chdir("C:\\Users\\igarc\\ZERG")
  exec(open("test.py").read())

  Tanto el directorio donde se encuentra el algoritmo ("C:\\Users\\igarc\\ZERG") como el nombre del script de llamada al algoritmo ("test.py") están sujetos a posibles cambios.

  Al ejecutar el algoritmo de esta manera es posible observar la evolución de la mejor ruta encontrada por el algoritmo para cada instancia del problema.

- La siguiente estructura de carpetas dentro del directorio del algoritmo es necesaria para su correcto funcionamiento.

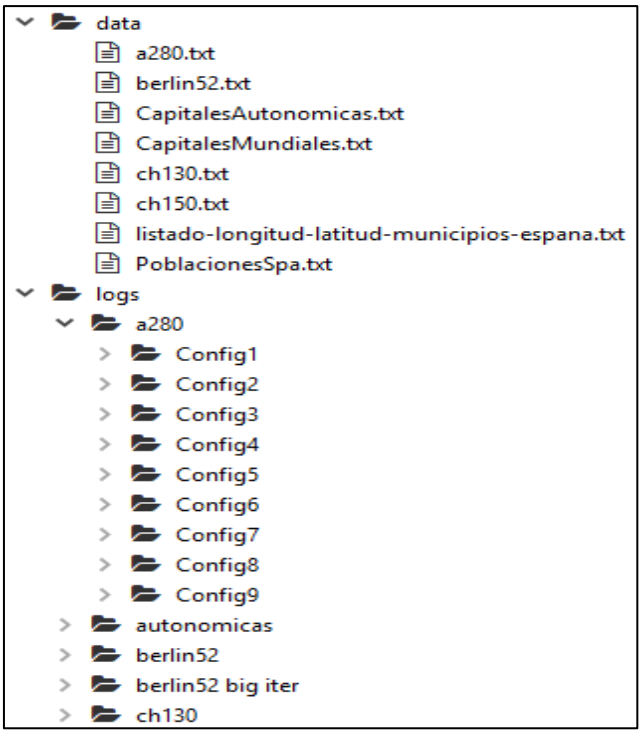


Ilustración 12. Estructura de carpetas necesaria para el algoritmo

Los archivos con los datos de las ciudades deben colocarse dentro una carpeta llamada *data*. Dentro de la carpeta llamada *logs*, debe haber una carpeta con el nombre de cada problema con 9 subcarpetas, una para cada configuración, donde se guardarán los *log files* y las imágenes generadas para cada configuración.

- El programa auxiliar para la generación de los resúmenes de datos está estrictamente adaptado a la estructura de los *log files* generados.

Durante el análisis de los resultados del algoritmo generados durante la etapa 4 del proyecto, pudieron observarse los siguientes errores en el código:

1. El guardado de los resultados en las subcarpetas para las configuraciones 7, 8 y 9 no se realizó exitosamente. Esto se debe a un error en la línea 56 del archivo *main.py* en la que la comparación del número del operador de recombinación no se compara correctamente al número. Esto provocó que los resultados para las configuraciones 7, 8 y 9 se guardaran en las subcarpetas de las configuraciones 1, 2 y 3.

   El error ha sido corregido en el archivo *main.py* del repositorio y la correcta clasificación de los resultados pudo solventarse mediante una adaptación del programa de resumen de datos *ResumenData.py*.

2. En el problema con menor número de ciudades, cuando la búsqueda convergía de manera extremadamente rápida, un mismo *log file* registraba dos búsquedas distintas. Esto se debe a que el nombre de los archivos

contiene una marca temporal con las horas, minutos y segundos del momento de su creación. Cuando una búsqueda finalizaba y la siguiente comenzaba en un mismo segundo, el mismo nombre era dado para el *log file* de ambas búsquedas.

Como solución, únicamente utilizada para el problema con menor número de ciudades, se implementó un tiempo de espera de 1 segundo tras cada búsqueda con la finalidad de asegurar la generación de un *log file* por cada búsqueda.

3. En los *log files* y en los archivos resumen, el operador de mutación *RandomSwap* está registrado bajo el nombre *SingleSwap*. Esto se debe a el operador de mutación inicialmente implementado fue *SingleSwap* (intercambio de un elemento de la secuencia por otro) aunque posteriormente fue modificado a *RandomSwap* (intercambio de segmentos de la secuencia).

   El error en el registro del operador de mutación en los *log files* fue corregido en la línea 44 del archivo *funcionesLog.py*. El correcto nombramiento del operador en los archivos resumen pudo ser solventado a posteriori mediante una modificación del programa *ResumenData.py*.

4. De manera similar al anterior punto, el operador de recombinación *Order1*, se registraba erróneamente en los *log files* como *Oder1*. El error fue corregido en la línea 37 de *funcionesLog.py* y el correcto nombramiento para el análisis de los resultados fue solventado mediante una modificación del programa auxiliar *ResumenData.py*.

5. El número de partículas vecinas para cada partícula no fue registrado en los *log files* generados para el análisis, por lo que uno de los parámetros de entrada es desconocido para cada instancia de las búsquedas. El número de partículas vecinas asociadas para cada búsqueda puede ser 3, 5 o 10.

6. Los archivos *main.py* y *ResumenData.py* fueron modificados en los últimos días de la redacción de la memoria para incluir el número de partículas vecinas evaluadas en cada búsqueda.

7. El registro de la configuración dentro de los *log files*, también tenía un error a la hora de registrar el operador de mutación, más concretamente en la línea 43 del archivo *funcionesLog.py*. Este error provocó lo siguiente:

   A. El operador de mutación *RandomSwap* quedó registrado como si no hubiera operador de mutación para las configuraciones con operadores de recombinación *Order1* y *PMX*.

   B. Los *log files* para la configuración 4, con el operador de recombinación *Cycle*, indicaban que el operador de mutación era *RandomSwap* cuando en realidad no se estaba utilizando ningún operador de mutación.

Afortunadamente, combinando la información acerca del operador de recombinación, la localización de los *log files* dentro de las subcarpetas y el error número 1 de esta lista, se pudo solventar este último error mediante la modificación del programa auxiliar *ResumenData.py*. La lógica para la correcta reclasificación de las configuraciones se explica en el siguiente diagrama:

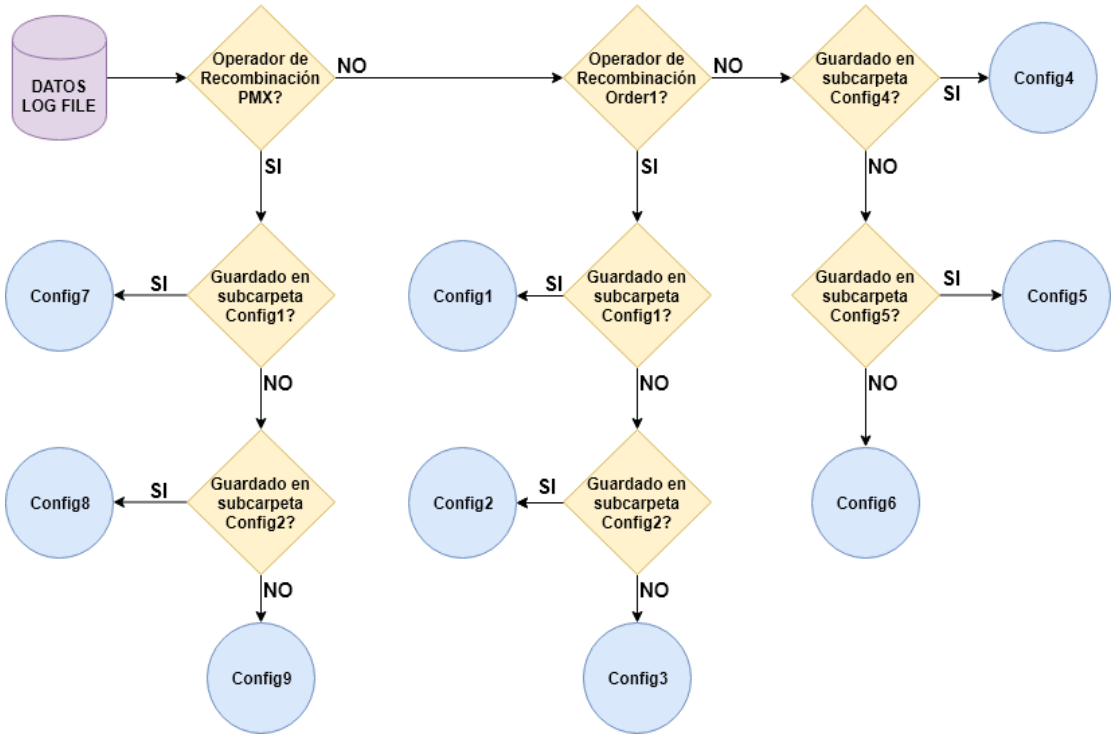


Ilustración 13. Solución al erróneo registro del operador de mutación

Seguidamente de haber corregido los errores indicados, se volvió a realizar el testeo iterativo para algunos problemas, de manera que se pudiera realizar el análisis completo, aunque con menor rango de parámetros y repeticiones por experimento para poder satisfacer la fecha de entrega del proyecto.

Los rangos de parámetros estudiados fueron:

A) <u>Experimentos sin número de partículas asociadas registrado:</u>

- **Número máximo de iteraciones**: [100, 200, 500, 1000, 2000, 5000]
- **Tamaño de enjambres**: [40, 100, 250, 500]
- **Número de partículas asociadas**: [3, 5, 10]
- **Número de repeticiones**: 10

B) <u>Experimentos con número de partículas asociadas registrado:</u>

- **Número máximo de iteraciones**: [100, 200, 500, 1000]
- **Tamaño de enjambres**: [40, 100, 250]
- **Número de partículas asociadas**: [3, 5, 10]
- **Número de repeticiones**: 3

7. Resultados del algoritmo

Los resultados de 3 problemas fueron analizados: las capitales autonómicas españolas (19 ciudades), los problemas de referencia Berlin52 (52 ciudades) y ch130 (130 ciudades). El análisis de los archivos resumen fue realizado con la ayuda del software estadístico R (versión 3.3.3).

7.1 Capitales Autonómicas (19 ciudades)

Para este problema contamos con datos de 4212 instancias de búsqueda, teniendo 972 de estas instancias el número de partículas asociadas registrado. Entre los parámetros de entrada al modelo encontramos:

MaxIter: 100, 200, 500, 1000, 2000, 5000
Tamaño de enjambre: 40, 100, 250, 500
Número de partículas asociadas: 3, 5, 10

Evaluamos las distribuciones y las medidas estadísticas de las distancias obtenidas para las diferentes configuraciones.

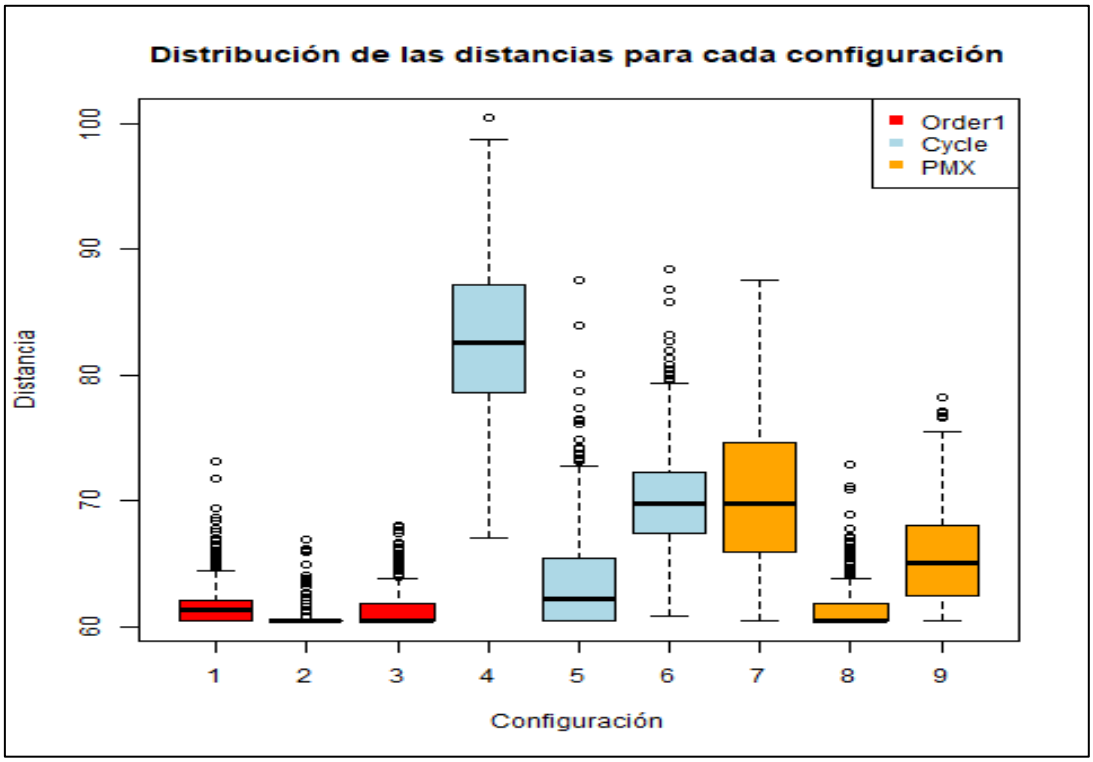


Ilustración 14. [Autonomicas19] Boxplot de distancias según configuración

	Config1	Config2	Config3	Config4	Config5	Config6	Config7	Config8	Config9
MIN	60.42	60.42	60.42	67.03	60.42	60.80	60.42	60.42	60.42
MAX	73.14	66.95	68.05	100.43	87.53	88.43	87.60	72.90	78.25
MEDIA	61.76	60.70	61.36	83.10	63.73	70.15	70.64	61.35	65.57
DESV. EST.	1.75	0.86	1.55	6.40	3.91	4.28	5.99	1.76	3.81

Tabla 3. [Autonomicas19] Medidas estadísticas de la distancia por configuración

La distancia mínima encontrada por 7 de las 9 configuraciones posibles es de 60.42. La representación gráfica de la ruta con dicha distancia es la siguiente:

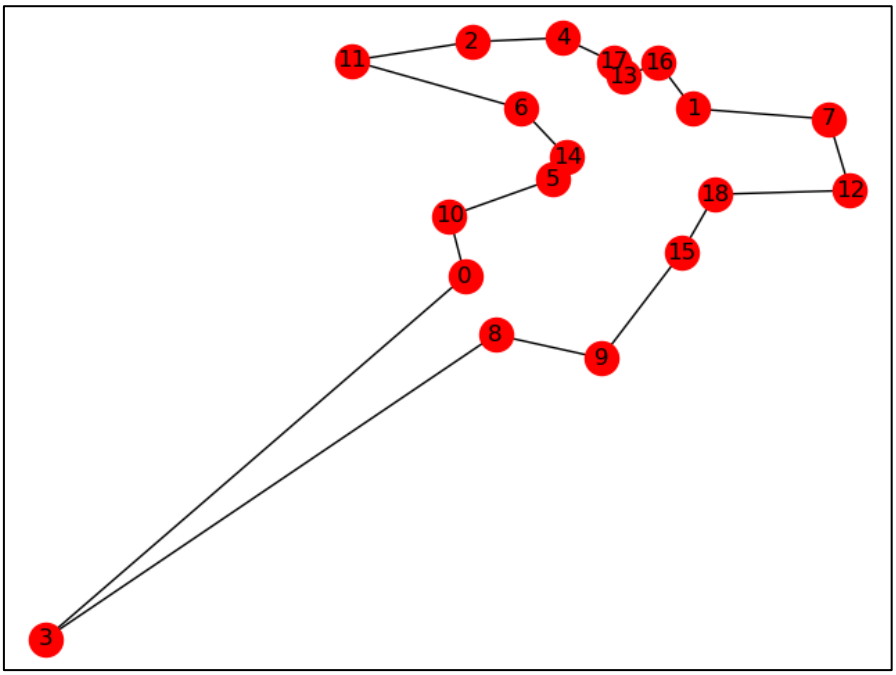


Ilustración 15. [Autonomicas19] Gráfica de la mejor ruta

Observando la ilustración 14 y los resultados de la Tabla 3, las configuraciones con el operador de recombinación *Order1* (1, 2 y 3) y la número 8 (*PMX* + *Inversion*) son las que mejores resultados han devuelto. Por otro lado, ninguna de las configuraciones con el operador de recombinación *Cycle* ha devuelto resultados de gran calidad.

También se puede observar que las configuraciones con el operador de mutación *Inversion* han devuelto resultados de mejor calidad que aquellas configuraciones sin operador o con el operador *RandomSwap*. En especial destaca la baja dispersión de las distancias devueltas por la configuración 2.

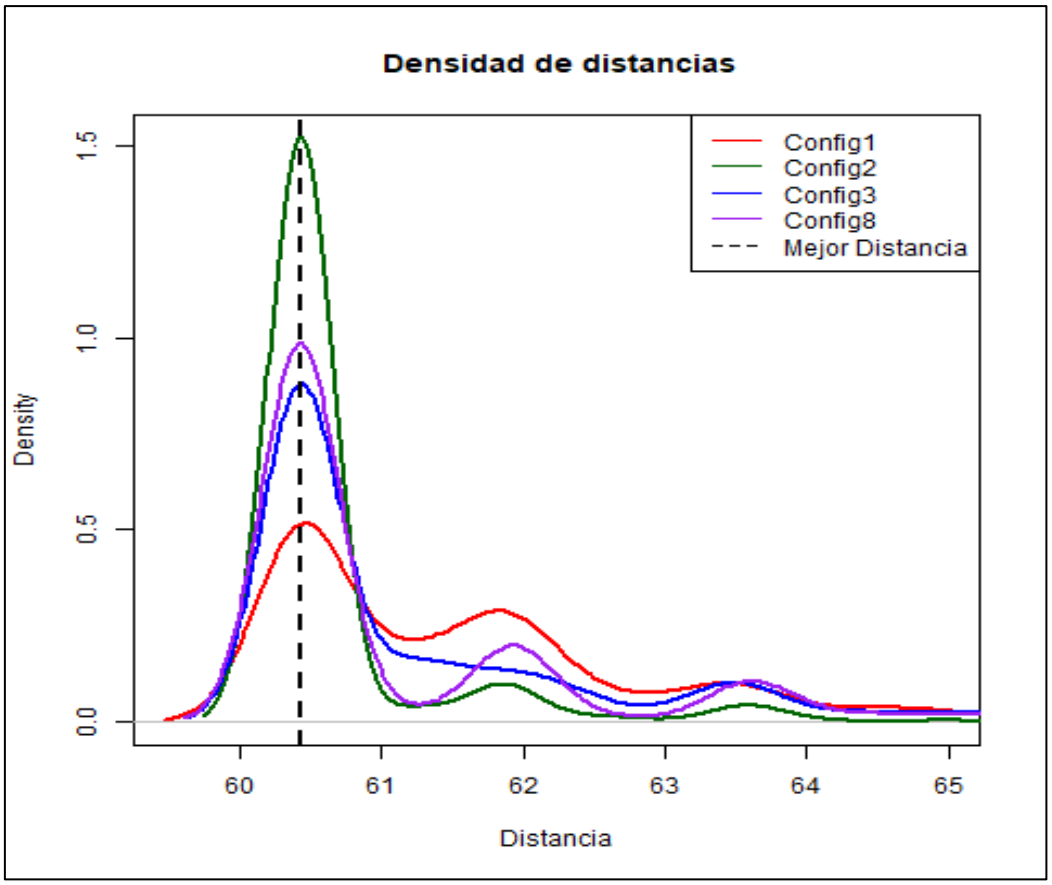


Ilustración 16. [Autonomicas19] Función de densidad de probabilidad de las distancias devueltas para las Configuraciones 1, 2, 3 y 8

En la ilustración 16, queda en evidencia que configuración más robusta para el cálculo de la mejor distancia de la ruta más corta entre las 19 ciudades del problema es la Configuración 2 (Recombinación *Order1* + Mutación *Inversion*).

A continuación, evaluamos de manera similar a la distancia, el tiempo necesitado por cada configuración de las 4 mejores configuraciones elegidas:

	Config1	Config2	Config3	Config8
MIN	0.64	0.72	0.70	0.59
MAX	12.88	18.20	24.77	27.87
MEDIA	3.53	3.89	4.08	4.51
DESV. EST.	2.75	3.23	3.50	4.15

Tabla 4. [Autonomicas19] Medidas estadísticas del tiempo para las Configuraciones 1, 2, 3 y 8

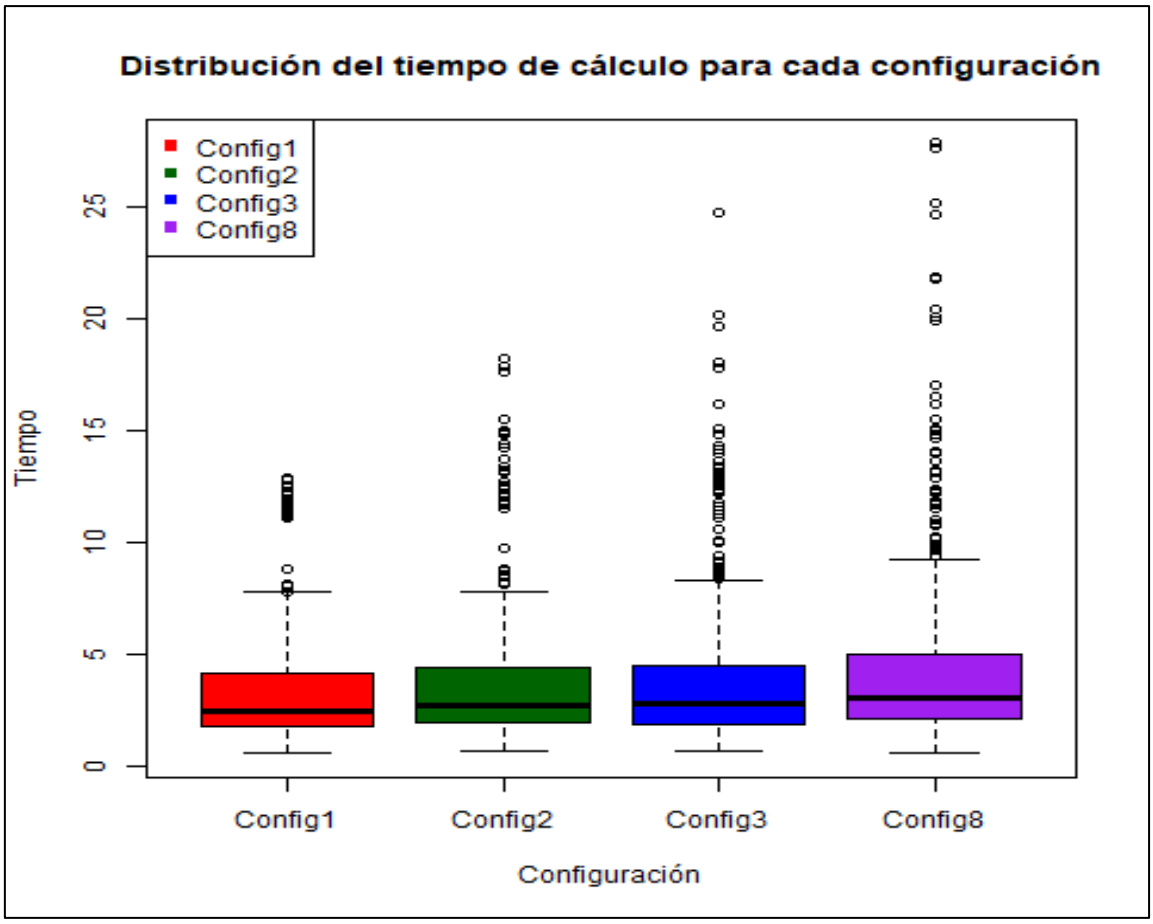


Ilustración 17. [Autonomicas19] Boxplot de los tiempos de cálculo para las Configuraciones 1, 2, 3 y 8

No se observan grandes diferencias entre los tiempos requeridos por las cuatro configuraciones (Ilustración 17 y Tabla 4). El hecho de que la media de tiempo de cálculo de la configuración 1 sea menor que la del resto, responde al propio hecho de que dicha configuración no cuenta con operador de mutación, lo cual inevitablemente conduce a un mayor tiempo de cálculo. También, la variabilidad del tiempo de cálculo es mayor para las configuraciones con operador de mutación ya que las operaciones de mutación se producen aleatoriamente en un 15% de los casos.

Tras este análisis, profundizamos en los datos para la configuración que mejores resultados devuelve: la Configuración 2.

	40	100	250	500
% MEJOR	67.46	85.71	97.62	100

Tabla 5. [Autonomicas19] Porcentaje de búsquedas con la Configuración 2 que alcanzan la distancia mínima, dividido por tamaño de enjambre

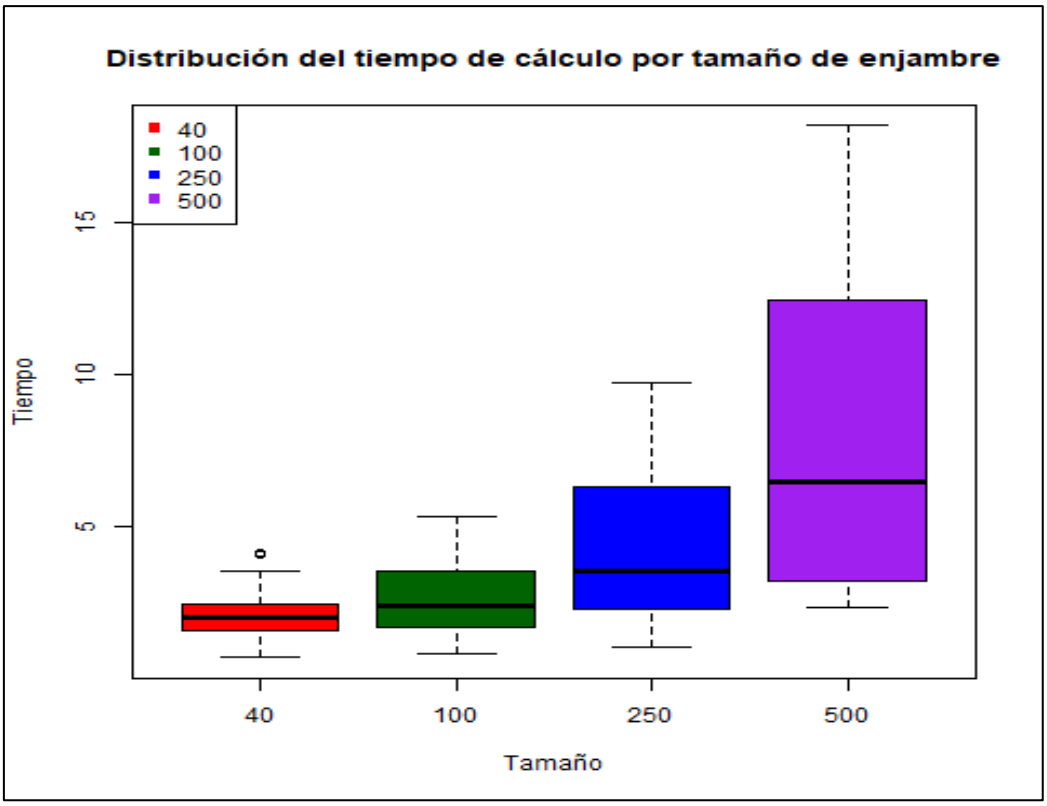


Ilustración 18. [Autonomicas19] Boxplot tiempo de cálculo según tamaño de enjambre para la Configuración 2

Aunque las búsquedas con un tamaño de enjambre de 500 partículas tengan un 100% de éxito a la hora de encontrar la ruta más corta para este problema (Tabla 5), el de 250 partículas tiene también un porcentaje muy alto de búsquedas que encuentran la mejor ruta para este problema (97.62%). Esto unido al hecho de que el tiempo de cálculo aumenta exponencialmente con el tamaño de enjambre (Ilustración 18) inclina a optar por un tamaño de enjambre de 250. Si el objetivo fuera conseguir una respuesta más rápida, aun conservando un alto porcentaje de soluciones óptimas, podría elegirse un tamaño entre 100 y 250.

	100	200	500	1000	2000	5000
Búsquedas Estancadas (%)	100	100	100	100	100	100
Búsquedas Sin Estancar (%)	0	0	0	0	0	0

Tabla 6. [Autonomicas19] Porcentaje de búsquedas estancadas de la Configuración 2 según número máximo de iteraciones establecido para enjambres 250

Dado que todas las búsquedas del enjambre de 250 partículas finalizan por condición de estancamiento (Tabla 6), un límite de 100 iteraciones resulta

suficiente para la generación de su mejor ruta. Establecer límites más altos solo perjudicaría el rendimiento temporal del algoritmo.

Finalmente, analizamos la relación entre el número de partículas asociadas y el número de iteraciones de las búsquedas realizadas con la Configuración 2, con un tamaño de enjambre de 250 partículas.

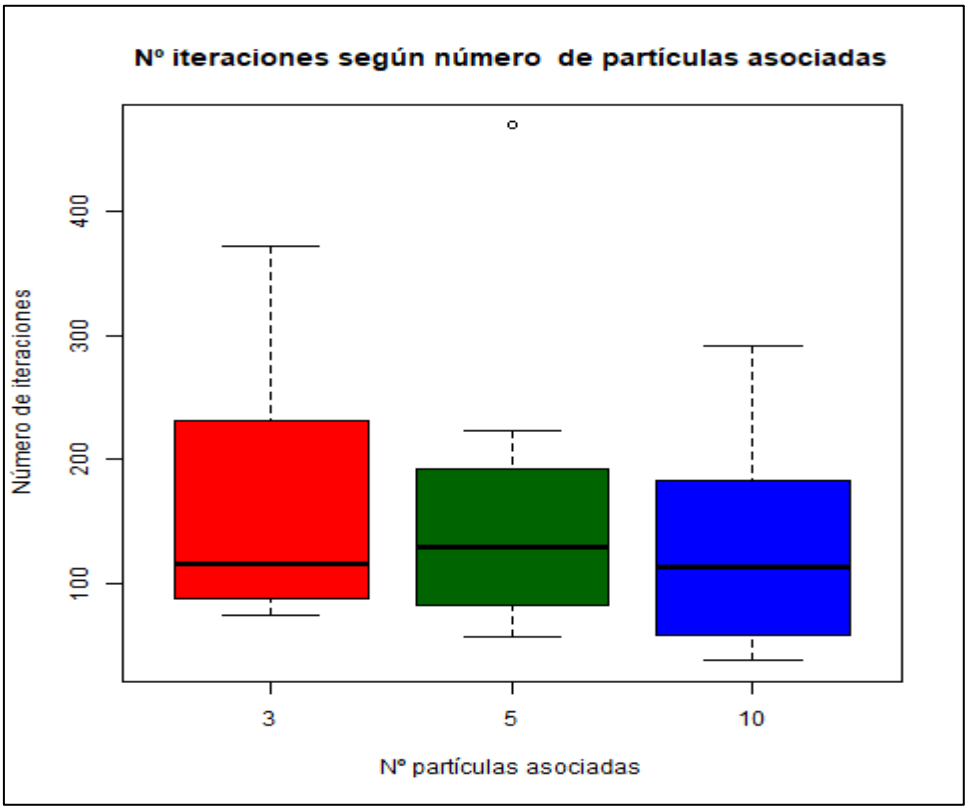


Ilustración 19. [Autonomicas19] Número iteraciones según número de partículas asociadas para enjambres de 250 partículas de la Configuración 2

El número de iteraciones de las búsquedas con 5 partículas asociadas tiene menor dispersión. Aun así, no se aprecian diferencias significativas entre los 3 diferentes grupos, quizás motivado por el tamaño del propio enjambre con respecto a la dimensión del problema en cuestión (19 ciudades).

7.2 Berlin52 (52 ciudades)

Para este problema contamos con datos de 8178 instancias de búsqueda, teniendo 972 de estas instancias el número de partículas asociadas registrado. Entre los parámetros de entrada al modelo encontramos:

MaxIter: 100, 200, 500, 1000, 2000, 5000
Tamaño de enjambre: 40, 100, 250, 500
Número de partículas asociadas: 3, 5, 10

Evaluamos las distribuciones y las medidas estadísticas de las distancias obtenidas para las diferentes configuraciones:

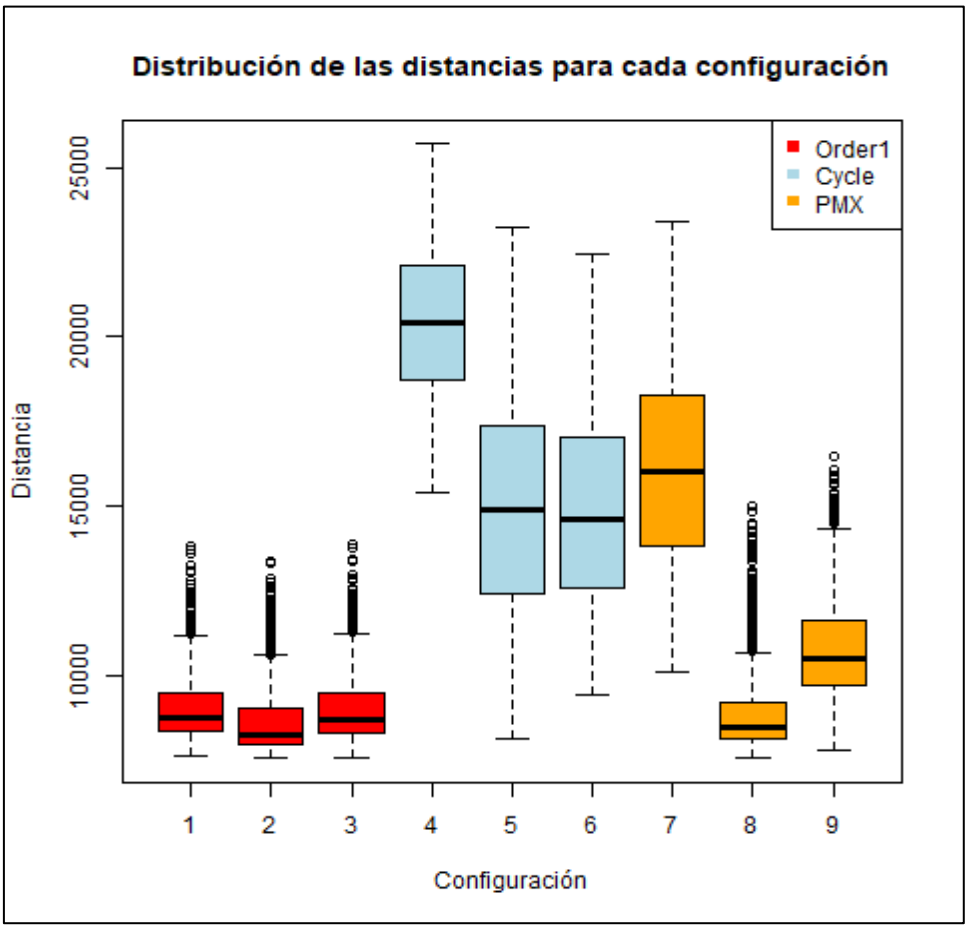


Ilustración 20. [Berlin52] Boxplot de distancias según configuración

	Config1	Config2	Config3	Config4	Config5	Config6	Config7	Config8	Config9
MIN	7597	7544	7566	15385	8108	9437	10116	7544	7806
MAX	13811	13377	13858	25700	23214	22477	23410	14984	16464
MEDIA	9054	8694	9035	20418	14936	14887	16086	8977	10812
DESV. EST.	1036	1148	1087	2139	3310	2865	2729	1368	1509

Tabla 8. [Berlin52] Medidas estadísticas de la distancia por configuración

De manera similar al anterior problema, las configuraciones 1, 2, 3 y 8 parecen devolver las mejores soluciones al problema y las búsquedas con las configuraciones con el operador de recombinación *Cycle* las peores. La ruta con menor distancia ha sido encontrada por búsquedas de las configuraciones 2 y 8, aunque ésta vez no han sido encontradas con tanta frecuencia como en el anterior problema.

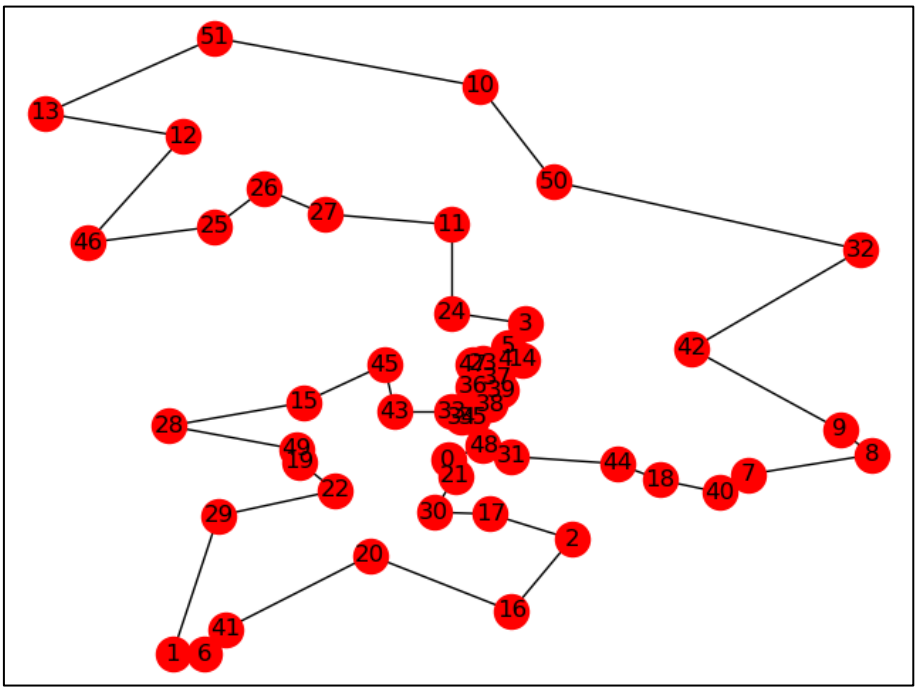


Ilustración 21. [Berlin52] Gráfica de la mejor ruta

Dado el tamaño de los nodos representados en la gráfica de la ilustración 21, no es posible determinar si la ruta es la más óptima posible ya que en el grupo central de ciudades no es posible observar las trayectorias exactas entre las mismas.

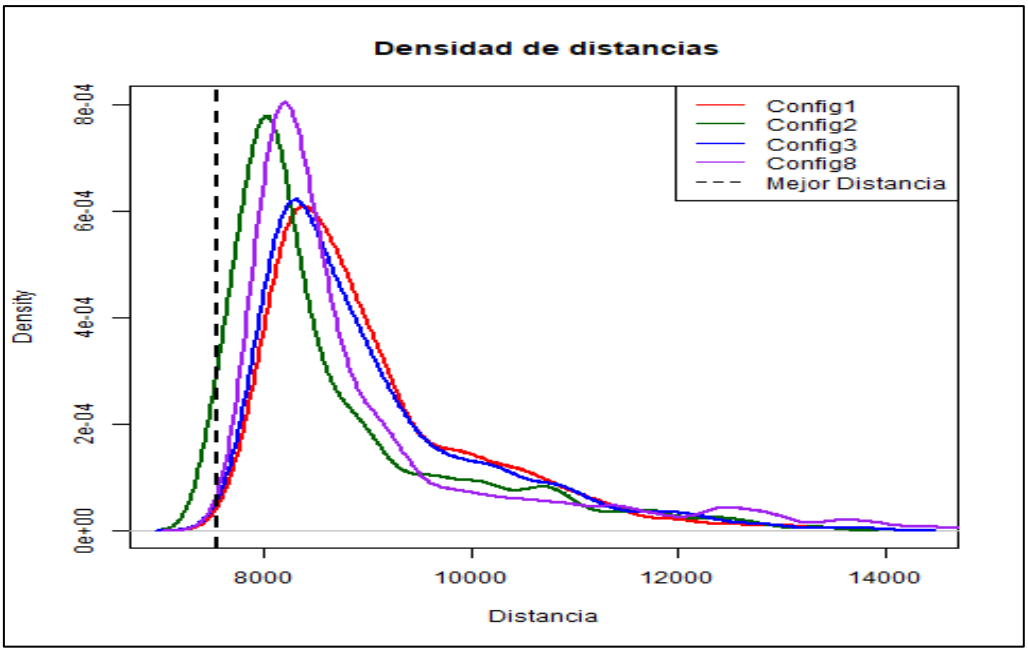


Ilustración 22. [Berlin52] Función de densidad de probabilidad de las distancias devueltas para las Configuraciones 1, 2, 3 y 8

En esta ocasión, las gráficas de las funciones de densidad no están centradas en el mejor resultado obtenido. Aunque la configuración 2 ha encontrado en mayor número de ocasiones la ruta con mejor distancia, en las distancias en el rango aproximado de 8.250 a 9.500 la configuración 8 tiene mayor éxito en sus búsquedas. Adicionalmente, graficamos la función de densidad acumulativa de las configuraciones 2 y 8:

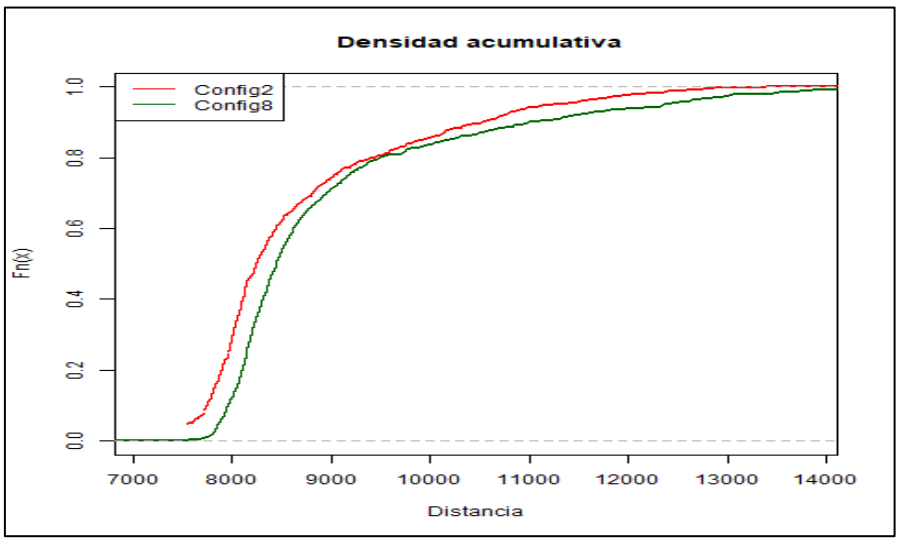


Ilustración 23. [Berlin52] Función de densidad acumulativa para las configuraciones 2 y 8

Aunque similares, la configuración 2 obtiene mejores resultados que la configuración 8. A continuación analizamos el tiempo de cálculo empleado por las configuraciones 2 y 8.

	Config2	Config8
MIN	3.34	2.64
MAX	74.85	95.70
MEDIA	17.75	16.36
DESV. EST.	12.28	10.64

Tabla 9. [Berlin52] Medidas estadísticas del tiempo de cálculo para las configuraciones 2 y 8

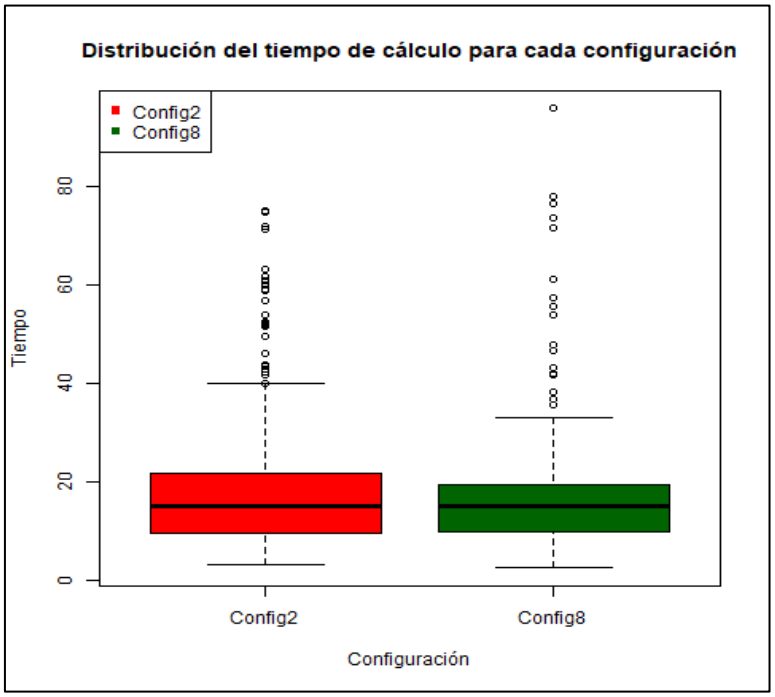


Ilustración 24. [Berlin52] Boxplot del tiempo de cálculo para las configuraciones 2 y 8

Aunque los tiempos por búsqueda de la Configuración 8 tienen menor dispersión que los de la Configuración 2, la media de la Configuración 2 sigue siendo menor. Esto, unido a la mayor acumulación de soluciones de mejor calidad (Ilustración 23) lleva a la decisión de elegir la Configuración 2 como la mejor de ambas.

	100	200	500	1000	2000	5000
Búsquedas Estancadas (%)	0	2.13	28.97	73.4	85.19	100
Búsquedas Sin Estancar (%)	100	97.87	71.03	26.6	14.81	0

Tabla 10. [Berlin52] Porcentaje de búsquedas estancadas de la Configuración 2 según número máximo de iteraciones establecido.

De acuerdo con los datos presentados en la Tabla 9, el número de iteraciones máximo permitido para la ejecución óptima del algoritmo estaría entre 1000 y 2000. A continuación analizamos la distribución de las distancias obtenidas según el tamaño de enjambre, para las búsquedas con iteraciones máximas mayores de 1000.

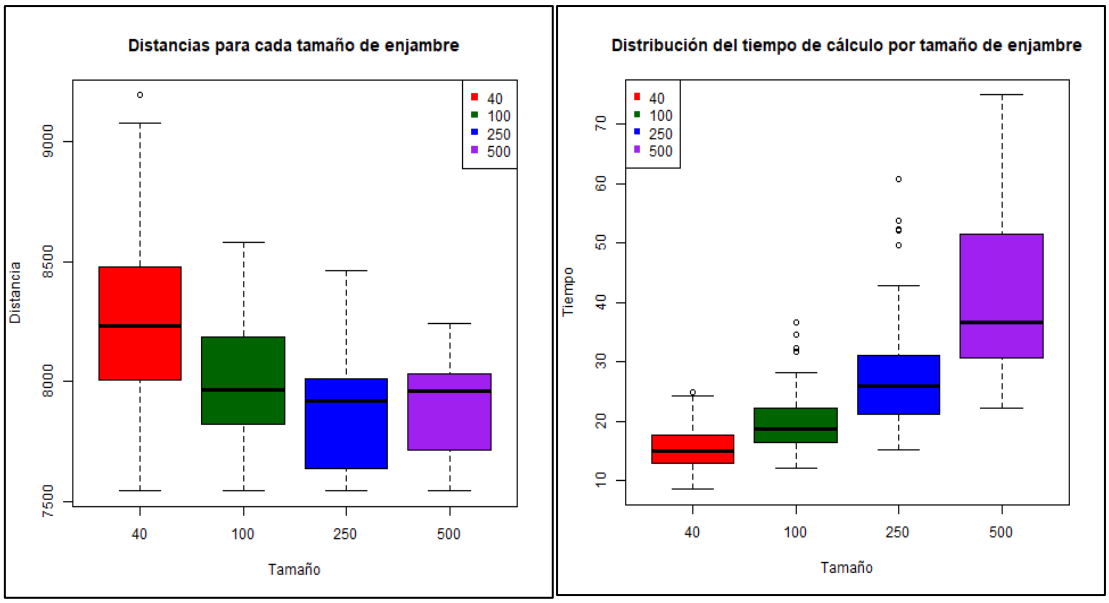


Ilustración 25. [Berlin52] Boxplot de las distancias y tiempos de cálculo según tamaños de enjambre para la Configuación 2, con máximos de iteraciones de 1000, 2000 y 5000

Aunque la distribución de las distancias para el tamaño de enjambre de 250 partículas de la Configuración 2 tiene una varianza mayor que la del tamaño de 500 partículas, la varianza de la distribución del tiempo de cálculo para el enjambre de 500 partículas es notablemente mayor.

En conclusión, el tamaño de enjambre más eficiente combinando la calidad de las soluciones y el tiempo de cálculo necesario es el de 250 partículas.

Por último, analizamos el impacto del número de partículas asociadas en el grado de estancamiento de las búsquedas para la configuración 1, 2, 3 y 8 con tamaño de enjambre de 250 partículas. Analizamos el número de partículas asociadas de las búsquedas no finalizadas en estancamiento para los límites de iteración de 500 y 1000.

	3	5	10
Búsquedas NO Estancadas (%)	54.16	29.16	16.68

Tabla 11. [Berlin52] Porcentaje de búsquedas (con las configuraciones 1, 2, 3 y 8) no estancadas según el número de partículas asociadas

Los datos presentados en la Tabla 11 señalan el número de partículas asociadas como un factor de peso para evitar el estancamiento prematuro de las búsquedas. El número de partículas asociadas que mejor parece funcionar para este problema parece ser el 3.

7.3 Ch130 (130 ciudades)

Para este problema contamos con datos de 5292 instancias de búsqueda, teniendo 972 de estas instancias el número de partículas asociadas registrado. Entre los parámetros de entrada al modelo encontramos:

MaxIter: 100, 200, 500, 1000
Tamaño de enjambre: 40, 100, 250, 500
Número de partículas asociadas: 3, 5, 10

Evaluamos las distribuciones y las medidas estadísticas de las distancias obtenidas para las diferentes configuraciones:

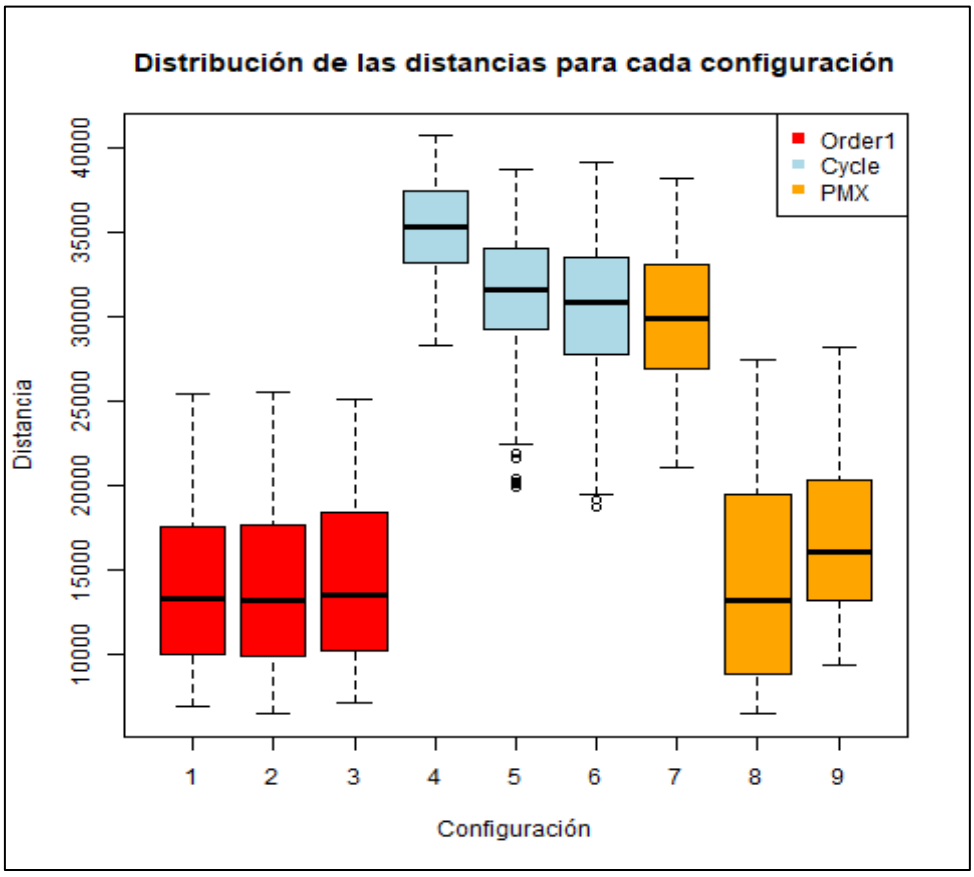


Ilustración 26. [Ch130] Boxplot de distancias según configuración

	Config1	Config2	Config3	Config4	Config5	Config6	Config7	Config8	Config9
MIN	6930	6470	7145	28256	19855	18739	21040	6498	9362
MAX	25474	25543	25081	40708	38683	39128	38232	27486	28178
MEDIA	13952	14021	14371	35345	31339	30434	30045	14386	16858
DESV. EST.	4692	4878	4822	2618	3560	3858	3805	6000	4491

Tabla 12. [Ch130] Medidas estadísticas de la distancia por configuración

Las configuraciones 1 y 2 parecen devolver las mejores soluciones al problema y las búsquedas con las configuraciones con el operador de recombinación *Cycle* continúan siendo las peores. Tanto la media como la dispersión de los resultados de la Configuración 1 son menores. La ruta con menor distancia ha sido encontrada por la Configuración 2.

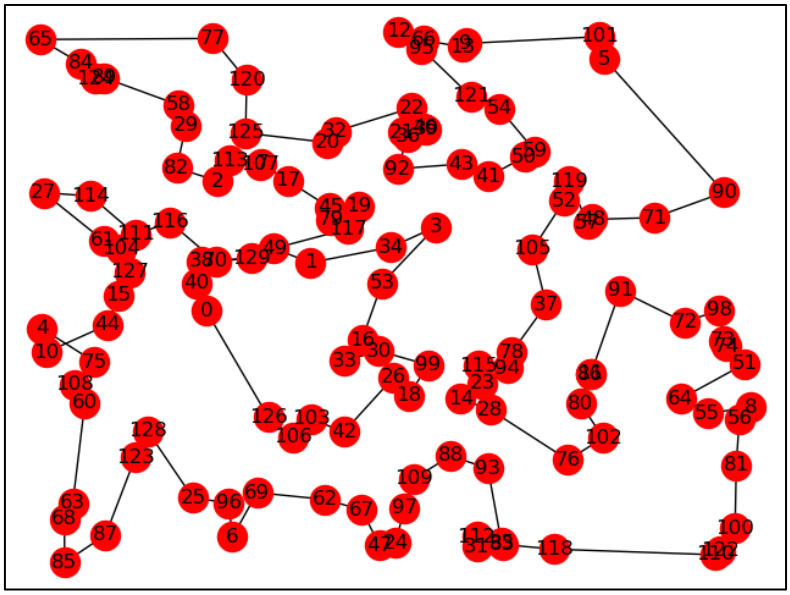


Ilustración 27. [Ch130] Gráfica de la mejor ruta

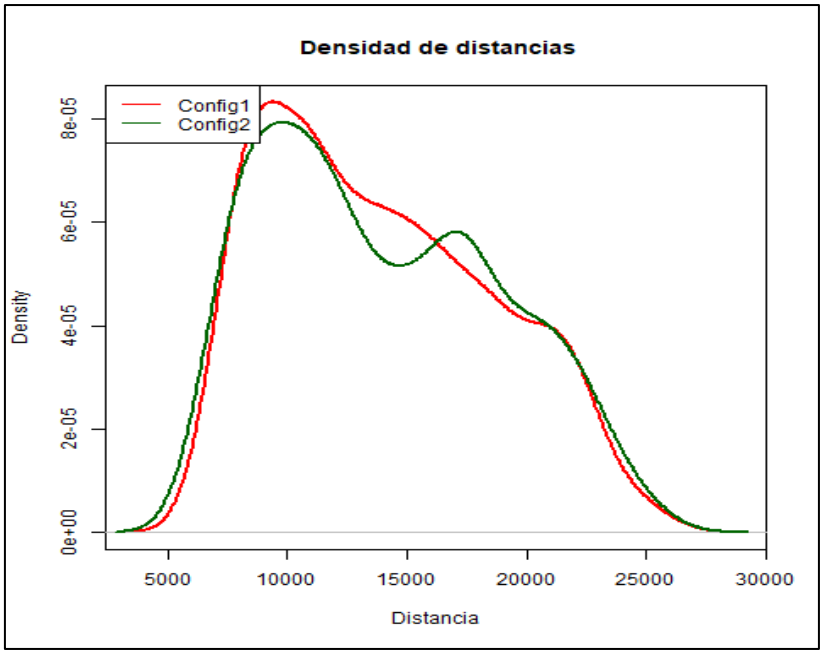


Ilustración 28. [Ch130] Función de densidad de probabilidad de las distancias devueltas para las configuraciones 1 y 2

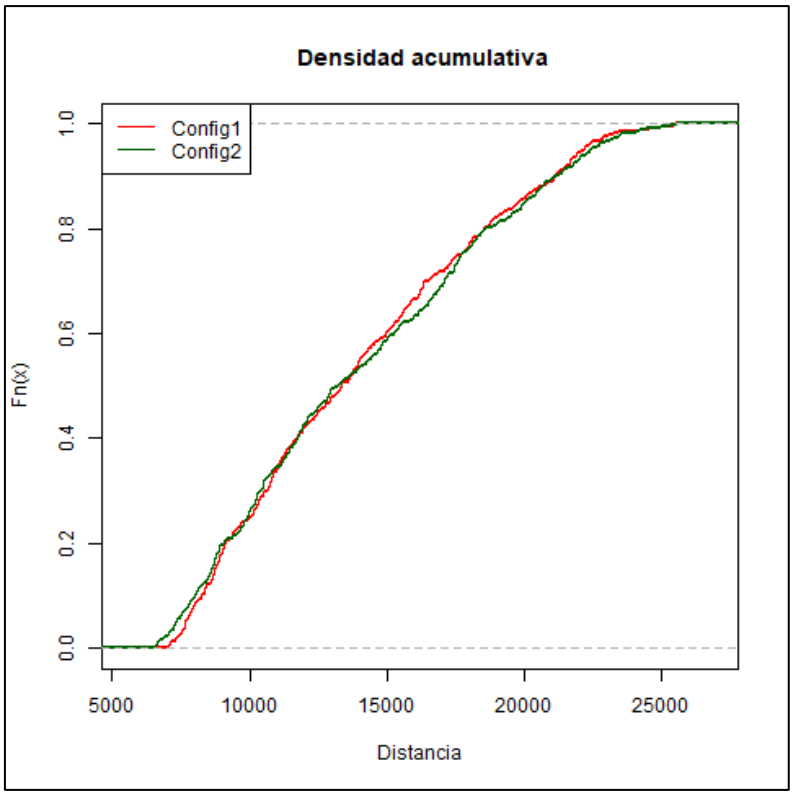


Ilustración 29. [Ch130] Función de densidad acumulativa para las configuraciones 1 y 2

Aunque la Configuración 2 ha encontrado la ruta con mejor distancia, no queda claro cuál de las dos devuelve resultados de manera más robusta. A continuación analizamos el tiempo de cálculo empleado por las configuraciones 1 y 2.

	Config1	Config2
MIN	7.86	7.35
MAX	154.30	160.96
MEDIA	41.83	42.13
DESV. EST.	33.20	34.83

Tabla 13. [Ch130] Medidas estadísticas del tiempo de cálculo para las configuraciones 1 y 2

El tiempo de búsqueda utilizado por la Configuración 2, al igual que las distancias devueltas, tiene media y varianza mayores que la Configuración 1. Queda en evidencia que la Configuración 1 es la mejor de ambas.

	100	200	500	1000
Búsquedas Estancadas (%)	0	0	1.33	23.94
Búsquedas Sin Estancar (%)	100	100	98.67	76.06

Tabla 14. [Ch130] Porcentaje de búsquedas estancadas de la Configuración 1 según número máximo de iteraciones establecido

Como ya podíamos suponer de los datos del problema Berlin52, establecer el número de iteraciones máximo permitido por debajo de 1000 limita la posibilidad del algoritmo de encontrar soluciones de alta calidad. Analizamos la distribución de las distancias obtenidas según el tamaño de enjambre, para las búsquedas con iteraciones máximas de 1000.

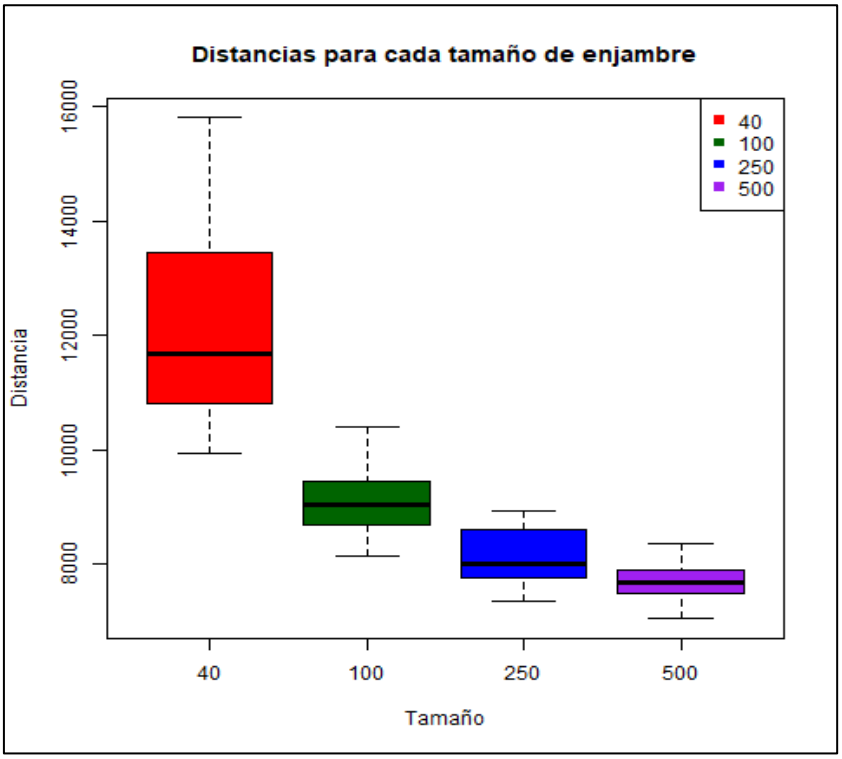


Ilustración 30. [Ch130] Boxplot de las distancias y tiempos de cálculo según tamaños de enjambre para la Configuación 1, con máximo de iteraciones establecido en 1000

De acuerdo al gráfico presentado en la Ilustración 30, los resultados para los enjambres de 500 partículas son los más robustos a la hora de obtener la mejor distancia. Esto es esperado ya que a más partículas, mayor número de soluciones son exploradas y mayor es la posibilidad de encontrar soluciones de alta calidad. Sin embargo, la diferencia con los datos de los enjambres de 250 partículas podría reducirse en caso de contar con datos para límites de iteración más altos y encontrarnos en un caso parecido al del problema Berlin52.

Dada la falta de datos con número de partículas asociadas para búsquedas con enjambres de 500 partículas de tamaño, se realiza el análisis de la relación entre número de partículas y estancamiento de búsquedas con datos de enjambres de 250 partículas. Como manera de incluir más datos en el análisis, también se tienen en consideración las configuraciones 2 y 3, además de la Configuración 1.

	3	5	10
Búsquedas Estancadas (%)	0	0	2.78

Tabla 15. [Ch130] Porcentaje de búsquedas con las configuraciones 1, 2 y 3 estancadas según el número de partículas asociadas

En los datos de la tabla 14, se puede observar que ninguna de las búsquedas con 3 o 5 partículas asociadas ha finalizado en estancamiento. Sin embargo, estos bajos porcentajes están condicionados por el pequeño número de búsquedas estancadas ya evidenciado en la Tabla 12.

8. Conclusiones y futuro desarrollo

Conclusiones del trabajo

- El algoritmo es capaz de encontrar la solución óptima para Problemas del Vendedor Ambulante.

- Las configuraciones con operadores de recombinación *Order1* son las que mejor resultados han devuelto de todas las configuraciones posibles.

- La inclusión de operadores de mutación en el proceso de búsqueda influye positivamente en la calidad de las soluciones devueltas.

- El operador de mutación *Inversion* parece tener un impacto más positivo en la calidad de las soluciones devueltas que el operador *RandomSwap*.

- Los malos resultados para las configuraciones con el operador de recombinación *Cycle*, indican un posible error en la implementación del mismo aun cuando los resultados de verificación fueron exitosos.

- El tamaño de los enjambres tiene un gran impacto en la calidad de las soluciones generadas, aunque para problemas de mayor dificultad el número de iteraciones máximo permitido parece ser el factor más limitante para la calidad de las soluciones.

- La implementación errónea inicial del registro de parámetros en los *log files* no ha permitido realizar un análisis en detalle de la influencia del número de partículas asociadas en las búsquedas.

- Los elevados tiempos de cálculo para los problemas de mayor dificultad apuntan a una implementación subóptima en cuanto a eficiencia en el uso de recursos computacionales.

- Se encuentran disponibles resultados para otros problemas de mayor complejidad generados durante la primera fase de testeo. Sin embargo, la elección de los rangos de parámetros evaluados ha sido inadecuada para la complejidad de dichos problemas, lo cual ha provocado la generación de resultados de baja efectividad para la evaluación del rendimiento algoritmo frente dichos problemas.

Logro de los objetivos

1. ***Diseño del banco de pruebas***
    a. ***Elección de grupos de ciudades que formarán grupos de diferente complejidad.***
    b. ***Implementación de dichos grupos, con la localización de cada ciudad perteneciente a cada grupo.***
    c. ***Uso de otros problemas de referencia.***

El banco de pruebas fue diseñado e implementado con éxito, aunque desafortunadamente no haya sido posible evaluar todos y cada uno de los grupos del mismo.

Para problemas con datos reales se recogieron e implementaron datos para capitales autonómicas españolas (19 ciudades), capitales mundiales (200 ciudades) y una lista de municipios españoles (8112 municipios).

Por otro lado, se recogieron datos para problemas de referencia mencionados en la literatura examinada durante la etapa inicial del proyecto. Los datos recogidos fueron para los problemas llamados Berlin52, Ch130, Ch150 y A280, con el número de ciudades de cada uno indicado en su propio nombre.

2. ***Desarrollo del algoritmo***
    a. ***Identificación del algoritmo basado en la naturaleza y justificación.***
    b. ***Implementación del algoritmo base para la generación de soluciones a problemas multivariables discretos.***
    c. ***Implementación de distintos tipos de crossover y mutación de partículas sobre el algoritmo básico, dando lugar a una serie de configuraciones para un algoritmo híbrido genético-enjambre.***

El algoritmo identificado para ser implementado fue elegido y justificado en base a la literatura leída durante la fase inicial del proyecto.

Aunque en los objetivos se especifica un algoritmo para la generación de soluciones a problemas multivariables, la implementación del algoritmo se ha llevado a cabo exclusiva y específicamente para la resolución del Problema del Vendedor ambulante.

Los operadores que dan lugar a las diferentes configuraciones fueron implementados, aunque la correctitud de implementación de algunos de ellos ha quedado en entredicho durante la evaluación de los resultados obtenidos.

3. ***Estudio del rendimiento de las diferentes configuraciones.***
    a. ***Evaluación del desempeño de los algoritmos sobre los diferentes grupos de ciudades que conforman el banco de pruebas.***
    b. ***Estudio de la configuración óptima para cada grupo de pruebas.***

Como ya se ha indicado, la evaluación de las diferentes configuraciones y desempeño del algoritmo solo ha sido posible para los problemas de menor

dificultad debido a la inadecuación de los parámetros de entrada evaluados para los problemas más complejos.

4. ***Representación de resultados***
    a. ***Elección de método de visualización de la progresión de búsqueda de soluciones.***
    b. ***Implementación de dicha representación elegida.***

La implementación del registro de resultados numéricos del algoritmo fue inicialmente defectuosa. La detección y corrección de sus fallos fueron realizadas en la etapa final del algoritmo, lo cual perjudicó a la hora de repetir los ensayos de testeo dada la proximidad de la fecha de entrega final.

La implementación de la representación gráfica de resultados fue realizada exitosamente, al igual que el guardado de dicha figura. Durante la ejecución de una búsqueda es posible observar el progreso.

Seguimiento de la planificación

El seguimiento de la planificación ha sido irregular debido a periodos de inactividad en el proyecto derivados de la situación personal del autor. Estos periodos de inactividad requirieron una reestructuración de los hitos temporales propuestos en un principio.

La metodología y objetivos propuestos inicialmente abarcaban un proyecto de una envergadura demasiado grande, los cuales fueron siendo acotados a medida que el proyecto avanzó. En un principio se pretendía implementar una librería y escribir la documentación para el uso de usuarios sin conocimientos informáticos. Más tarde, el proyecto se centró más en la implementación de un algoritmo con varias configuraciones y el análisis de los datos generados para cada configuración.

También, los hitos temporales y la planificación fueron actualizados en más de una ocasión por el surgimiento de nuevas tareas con el objetivo de corregir y mitigar errores durante detectados a lo largo del proyecto. Ejemplos de estas tareas son la corrección de errores del algoritmo en la etapa final, la necesidad de desarrollar programas auxiliares para la recopilación de información y modificaciones para la solución de errores arrastrados desde el algoritmo aún no corregidos en la primera fase de testeo.

Líneas de futuro desarrollo

- Repetir el testeo de los problemas de mayor complejidad con el objetivo de generar resultados más adecuados para su posterior análisis.

- Analizar los datos generados por las configuraciones con el operador de recombinación *Cycle* y extender la verificación de su funcionamiento para identificar posibles errores en su implementación y corregirlos.

- Flexibilizar el algoritmo:

    - Implementar funciones de lectura de mayor versatilidad que permitan leer archivos con coordenadas de diferentes formatos.

    - Añadir la posibilidad de establecer el porcentaje de iteraciones sin mejora para finalizar la búsqueda por estancamiento, actualmente fijado en 15%.

    - Añadir la posibilidad de establecer la probabilidad con la que se producen las operaciones de mutación, actualmente fijada en 15%.

    - Modificar el algoritmo de manera que pueda ser ejecutado mediante la línea de comandos.

    - Evaluar las dos secuencias generadas por el operador *Cycle* en lugar de solo una de las dos.

    - Añadir la posibilidad de elegir la representación gráfica en tiempo real de la evolución de la búsqueda.

- Añadir nuevas funcionalidades:

    - Implementar nuevos operadores de recombinación y mutación para dar lugar a nuevas configuraciones.

    - Idear e implementar nuevos factores indicativos del rendimiento de la búsqueda. Por ejemplo el número de intersecciones entre las trayectorias de una ruta eran notablemente reducidos o inexistentes en las rutas con mejores resultados.

    - Implementar una interfaz de usuario para la facilitación del uso del algoritmo para usuarios sin conocimientos informáticos.

9. Glosario

ADN: Ácido desoxirribonucleico. Ácido nucleico contenedor de información genética para el desarrollo y funcionamiento de organismos vivos.

Algoritmo: Conjunto ordenado de operaciones que permite calcular y hallar soluciones a problemas.

Anaconda: Distribución de los lenguajes R y Python utilizada en ciencia de datos y Machine Learning.

Banco de pruebas: Plataforma para la experimentación con el objetivo de comprobar de forma repetible el funcionamiento de algoritmos u otros elementos computacionales.

Cromosoma: Estructura organizada formada por proteínas y ADN que contiene información genética de los organismos vivos.

Crossover/Recombinación: Proceso por el cual los elementos de cromosomas intercambian secciones.

GitHub: Plataforma que permite alojar proyectos utilizando el control de versiones, utilizado normalmente para el registro de cambios en un proyecto de software colaborativo.

Librería: Conjunto de funcionalidades implementadas en un determinado lenguaje de programación con el objetivo de ofrecer una interfaz para su uso.

Mutación: Cambio en la secuencia u organización del ADN.

PEC: Prueba de Evaluación Continua.

Python: Lenguaje de programación de software con enfoque en que su sintaxis sea fácilmente legible.

Open Source: Se dice del software cuyo código fuente forma parte del dominio público y cuyo uso se considera libre, pudiendo modificarlo o redistribuirlo.

Optimización combinatoria: Rama de la disciplina de la optimización dentro de la ciencia de las matemáticas. Suele relacionarse con problemas de difícil resolución.

R: Lenguaje de programación enfocado al análisis estadístico de datos.

RAM: Random Access Memory. Memoria de trabajo en la cual se cargan las instrucciones a ejecutar por los procesadores de una computadora.

Sistema Operativo: Software de sistema que gestiona los recursos de hardware y software y permite la ejecución de aplicaciones software.

10. Referencias

[1] Rose, D. (2003) “Predicting Car Production using a Neural Network Technical Paper- Vetronics (Inhouse)”. Thesis, U.S. Army Tank Automotive Research, Development and Engineering Center (TARDEC)

[2] IEEE Spectrum. Warehouse Robots at Work

[3] The Getty Research Institute. Getty Thesaurus of Geographic Names® Online

[4] Solution to 48 States Traveling Salesman Problem

[5] E. Bonabeau, M. Dorigo, and G. Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York, NY, 1999.

[6] Business Intelligence fácil. Longitud y latitud de los municipios de España

[7] Z.E.R.G. PSO algorithm for TSP solving.

[8] Julián Pérez Porto y María Merino (2014). Definicion.de: Definición de recombinación (https://definicion.de/recombinacion/).

[9] Rubicite Interactive. Genetic Algorithms.

[10] Genetic Algorithms: A tutorial. SlidePlayer.

[11] Ray, T., Liew, K.M., “A swarm with an effective information sharing mechanism for unconstrained and constrained single objective optimization problems”, In: Kim, J.H. et al. (Eds.), Proceedings of the 2001 Congress on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, 2001, pp.75–80.

[12] J. Kennedy and R. C.Eberhart, “Particle Swarm Optimization”, IEEE International Conference on Neural Networks, 1995, pp.1942-1948.

[13] Hendtlass, T, “Preserving diversity in particle swarm optimization, Developments in Applied Artificial Intelligence”, Proceedings of the 16th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE 2003, Laughborough, UK, June 2003, In: Lecture Notes in Computer Science, Vol. 2718, 2003, pp.4104-4108.

[14] H. Xiaohui, S. Yuhui, R. Eberhart, “Recent Advances in Particle Swarm”, Proceedings of the IEEE Congress on Evolutionary Computation, pp. 90 - 97, 2004.

[15] L. Diosan and M. Oltean “Evolving the Structure of the Particle Swarm Optimization Algorithms”, EvoCOP 2006 Proceedings. (2006) 25-36.

[16] Pomeroy, P.: “An Introduction to Particle Swarm Optimization”. Electronic document available at www.adaptiveview.com/ipsop1.html

[17] Onwubulu, G.C., Clerc, M.: “Optimal Path for Automated Drilling Operations by a New Heuristic Approach Using Particle Swarm Optimization”. International Journal of Production Research Vol. 42, N. 3 (2004) 473-491.

[18] X. Yan, C. Zhang, W. Luo, W. Li, W. Chen and H. Liu. “Solve Traveling Salesman Problem Using Particle Swarm Optimization Algorithm” IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 6, No 2, November 2012 ISSN (Online): 1694-0814

[19] Yannis M, Magdalene M, “A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem”, Comput Oper Res 37(3), 2010, pp.432–442.

[20] Marinakis, Y., Migdalas, A., Pardalos, P.M.: Expanding neighborhood search GRASP for the probabilistic traveling salesman problem. Optim. Lett. 2, 351–361 (2008)

[21] https://github.com/igarciacuevas/Z.E.R.G.

[22] https://networkx.github.io/

[23] https://matplotlib.org/