



Universidad de Valladolid



**ESCUELA DE INGENIERÍAS
INDUSTRIALES**

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

GRADO EN INGENIERÍA EN ORGANIZACIÓN INDUSTRIAL

**Estudio y aplicación de métodos de
optimización por colonia de hormigas en
industria**

Autora:

Tomé Martín, Verónica

Tutora:

**Jiménez Gómez, María Isabel
Dpto. CMEIM EGI CGF IM IPF**

Valladolid, Septiembre de 2020.

RESUMEN

En este Trabajo Fin de Grado se propone la utilización de métodos metaheurísticos para la optimización de soluciones, concretamente aplicando algoritmos basados en colonia de hormigas, en el ámbito de la logística industrial.

En primer lugar, se presentan los distintos métodos de optimización existentes, para después escoger para este trabajo, el método a aplicar a dos casos concretos de la logística industrial. El transporte externo entre factorías y la gestión interna de los almacenes. En ambos casos se busca la optimización, bien de tiempos de transporte, o bien de rutas en los almacenes.

Para cada uno de los casos se realiza el correspondiente desarrollo, para obtener los resultados necesarios y su optimización. A continuación, se realiza un análisis de los resultados obtenidos para cada uno de dichos casos. Finalmente, se presenta un análisis conjunto y las conclusiones derivadas del uso de algoritmos colonia de hormigas aplicados a la logística industrial.

PALABRAS CLAVE

Algoritmo colonia de hormigas (ACO), Logística, Métodos metaheurísticos, Transporte. Vehículo de guiado automático (AGV).

ABSTRACT

In this Final Degree Project, the use of metaheuristic methods for the optimization of solutions is proposed, specifically applying ant colony-based algorithms in the field of industrial logistics.

Firstly, the different existing optimization methods are presented, and then the method to be applied to two specific cases of industrial logistics is chosen for this work. Those specific cases are external transport between factories and internal management of warehouses. In both cases, the aim is to optimize transport times or routes in the warehouses.

For each one of the cases, the corresponding development is carried out to obtain the necessary results and their optimization. The results obtained in each case are then analyzed. Finally, a global analysis and the conclusions derived from the use of ant colony algorithms applied to industrial logistics are presented.

KEYWORDS

Ant Colony Optimization (ACO), Automated guided vehicle (AGV), Logistics, Metaheuristic methods, Transport.

Índice de contenido

RESUMEN	3
PALABRAS CLAVE	3
ABSTRACT	3
KEYWORDS	3
1 INTRODUCCIÓN	15
1.1 CONTEXTO	15
1.2 JUSTIFICACIÓN DEL TFG	16
1.3 OBJETIVOS	17
1.3.1 Objetivo principal	17
1.3.2 Objetivos complementarios	17
1.4 MEDIOS SOFTWARE EMPLEADOS	17
1.4.1 MATLAB	17
1.4.2 EXCEL	18
1.4.3 AUTOCAD	18
1.5 ESTRUCTURA DE LA MEMORIA	18
2 MARCO TEÓRICO	21
2.1 MÉTODOS Y ALGORITMOS DE OPTIMIZACIÓN	21
2.1.1 Métodos exactos	21
2.1.2 Reglas de despacho	24
2.1.3 Algoritmos heurísticos	25
2.1.4 Métodos Metaheurísticos	26
2.2 MÉTODO DE OPTIMIZACION POR COLONIA DE HORMIGAS (ACO)	36
2.2.1 Estructura básica del algoritmo de colonia de hormigas	39
2.2.2 Optimización basada en colonia de hormigas en dos etapas	42
2.3 DISTRIBUCIÓN EN PLANTA	42
3 DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE TRANSPORTE	45
3.1 INTRODUCCIÓN AL PROBLEMA	45
3.2 PROGRAMACIÓN DEL ALGORITMO	46
3.2.1 Programación de ACO para camiones	46

3.2.2	Entrada de datos al programa principal	51
3.3	CASO BÁSICO	54
3.3.1	Datos del experimento.....	54
3.3.2	Método exacto.....	55
3.3.3	Método ACO.....	56
3.3.4	Resultados y conclusiones del experimento	57
3.4	CASO COMPLEJO	58
3.4.1	Datos del caso complejo	58
3.4.2	Desarrollo del caso	59
4	DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES.....	63
4.1	AGV (Automated Guided Vehicles- Vehículos de guiado automático) ...	63
Tipos de guiado de AGVs	64	
4.2	PROGRAMACION DEL ALGORITMO.....	67
4.2.1	Programación de ACO para AGVs.....	67
4.2.2	Programa para la entrada de datos.....	72
4.3	CASO BÁSICO	81
4.3.1	Objetivos del caso básico	81
4.3.2	Desarrollo del caso básico	81
4.3	CASO COMPLEJO	91
4.3.1	Objetivos del caso complejo.....	91
4.3.2	Desarrollo del caso complejo	91
5	ANÁLISIS DE LOS RESULTADOS	97
5.1	ANÁLISIS DEL ALGORITMO LOGÍSTICA DE TRANSPORTE	97
5.1.1	Análisis del caso simple.....	97
5.1.2	Análisis del caso complejo	98
5.1.3	Modificación de las variables.....	101
5.1.4	Conclusiones obtenidas	101
5.2.	ANÁLISIS DEL ALGORITMO PARA LOGÍSTICA DE ALMACENES.....	102
5.2.1	Modificación del número de hormigas	102
5.2.2	Modificación del número de iteraciones	103
5.2.3	Análisis del paso por puntos obligatorios.....	103
5.2.4	Conclusiones del análisis	104

5.2.5 Análisis del caso complejo	104
5.3 ANÁLISIS CONJUNTO	104
5.3.1 Principales diferencias entre los algoritmos	104
5.3.2 Modificación de las variables.....	106
5.3.3 Conclusiones del análisis global	107
5.4. RECOMENDACIONES PARA LOS ALGORITMOS ACO	107
5.4.1 Ventajas	107
5.4.2 Inconvenientes	108
5.4.3 Aplicaciones.....	108
6 ESTUDIO ECONÓMICO.....	111
6.1 DESCRIPCIÓN DE LAS ETAPAS DE DESARROLLO.....	111
6.2 CÁLCULO DE LOS COSTES	113
6.2.1 Costes de personal	113
6.2.2 Costes de amortización	115
6.2.3 Costes generales.....	116
6.3 COSTES TOTALES.....	116
6.4. CÁLCULO DEL PRECIO DE VENTA DEL PROYECTO	117
7 CONCLUSIONES Y LÍNEAS FUTURAS.....	119
7.1 CONCLUSIONES	119
7.2 LÍNEAS FUTURAS	119
8 BIBLIOGRAFÍA	121

Índice de figuras

Figura 1 Clasificación redes neuronales (Acosta-Buitrago, 2000).....	31
Figura 2 Las hormigas en el punto de decisión (Dorigo, 1996)	37
Figura 3 Elección de forma aleatoria entre los caminos inferior y superior (Dorigo, 1996).....	37
Figura 4 Efectos de la cantidad de hormigas que circulan por cada camino (Dorigo, 1996).....	38
Figura 5 Mayor acumulación de feromonas en el camino más corto. (Dorigo, 1996).....	38
Figura 6 Tipos de distribución en planta por producto (Machuca, 1995)	43
Figura 7 Distribución en planta por proceso (Fuente: Elaboración propia)	43
Figura 8 Distribución en planta por células de trabajo (Fuente: Elaboración propia).....	44
Figura 9 Esquema del programa ACO para el caso de transporte (Fuente: Elaboración propia).....	48
Figura 10 Ejemplo situación de diferencias en tiempos (Fuente: Elaboración propia).....	52
Figura 11 Hoja "RESULTADOS" fichero Excel (Fuente: Elaboración propia)	53
Figura 12 Mapa con las ciudades seleccionadas para el caso simple (fuente: Elaboración propia).....	55
Figura 13 Esquema de combinaciones posibles (Fuente: Elaboración propia)	55
Figura 14 Número de iteraciones y resultado. (Fuente: Elaboración propia).....	57
Figura 15 Variables "colony.queen.tour" y "colony.queen.fitness" (Fuente: Elaboración propia).....	57
Figura 16 Mapa con las localidades seleccionadas (Fuente: Elaboración propia)	59
Figura 17 Mejores datos obtenidos (Fuente: Elaboración propia)	60
Figura 18 Ruta obtenida en la mejor solución (Fuente: Elaboración propia).....	61
Figura 19 Accesorios para AGVs (Kivnon, 2020 y Cizon, 2020)	64
Figura 20 Representación filoguiado (ATRIA INNOVATION, 2020).....	65
Figura 21 Figura AGV por guiado magnético (Cizon, 2020).....	65
Figura 22 Representación guiado láser (ATRIA INNOVATION, 2020).....	66
Figura 23 Esquema del programa ACO para el caso de almacenes (Fuente: Elaboración propia).....	69
Figura 24 Ejemplo de la hoja "Mapa" (Fuente: Elaboración propia).....	75
Figura 25 Ejemplo de datos en la hoja "Auxiliar" (Fuente: Elaboración propia)	75
Figura 26 Fragmento de código (Fuente: Elaboración propia)	77
Figura 27 Ejemplo de la hoja "Datos" tras pulsar "Puntos Posibles" (Fuente: Elaboración propia).....	77

Figura 28 Ejemplo de la hoja "Datos" en excel "Datos_ACO" (Fuente: Elaboración propia).....	78
Figura 29 AGV que puede rotar sobre sí mismo (Fuente: Elaboración propia)	79
Figura 30 AGV que requiere más espacio que sus propias dimensiones para girar (Fuente: Elaboración propia)	79
Figura 31 Representación de la cuadrícula y el radio de maniobra del AGV (Fuente: Elaboración propia)	80
Figura 32 Cuadrado de referencia, destacado en azul (Fuente: Elaboración propia).....	80
Figura 33 Implantación caso básico (almacén simple) (Fuente: Elaboración propia).....	82
Figura 34 Puntos obligatorios para el caso básico (Fuente: Elaboración propia)	83
Figura 35 Hoja "Mapa" del Excel "Datos_ACO" para el caso básico (Fuente: Elaboración propia).....	84
Figura 36 "Datos" del Excel "Datos_ACO" para el caso básico (Fuente: Elaboración propia).....	85
Figura 37 Introducción de datos de Excel en función "createGraph (Fuente: Elaboración propia).....	85
Figura 38 Introducción de datos de Excel en función "createColony" (Fuente: Elaboración propia).....	86
Figura 39 Introducción de datos de Excel en función "drawGraph" (Fuente: Elaboración propia).....	86
Figura 40 Introducción de datos de Excel en función "drawBestTour" (Fuente: Elaboración propia).....	86
Figura 41 Parámetros de número de hormigas y de iteraciones en algoritmo ACO (Fuente: Elaboración propia)	87
Figura 42 Ventana de comandos de MatLab durante la ejecución del algoritmo (Fuente: Elaboraciónn propia).....	87
Figura 43 Gráfica obtenida por la función "drawGraph" (Fuente: Elaboración propia).....	88
Figura 44 Gráfica obtenida por la función "drawBestTour" en la primera iteración (Fuente: Elaboración propia)	89
Figura 45 (a) y (b) Gráficas obtenidas por la función "drawBestTour" en iteraciones intermedias (Fuente: Elaboración propia)	89
Figura 46 Última gráfica obtenida por la función "drawBestTour" (Fuente: Elaboración propia).....	90
Figura 47 Valores obtenidos en la variable "colony.queen.tour" (Fuente: Elaboración propia).....	90
Figura 48 Implantación caso complejo (Fuente: Elaboración propia).....	92
Figura 49 Hoja "Mapa" del excel "Datos_ACO" caso complejo (Fuente: Elaboración propia).....	93
Figura 50 Hoja "Datos" del Excel "Datos_ACO" caso complejo	94

Figura 51 Mensajes obtenidos en la ventana de comandos de MatLab (Fuente: Elaboración propia)	95
Figura 52 Gráfico de todos los puntos y uniones del caso complejo (Fuente: Elaboración propia)	95
Figura 53 Gráfico cajas y bigotes de los resultados obtenidos para colony.queen.fitness (Fuente: Elaboración propia)	99
Figura 54 Esquema correspondiente al ejemplo de reparto (Fuente: Elaboración propia)	108
Figura 55 Gráfico de costes totales del proyecto (Fuente: Elaboración propia)	117

Índice de tablas

Tabla 1 Tiempo de trayecto entre localidades (Fuente: Elaboración propia) ..	52
Tabla 2 Datos caso básico camiones (Fuente: Elaboración propia)	54
Tabla 3 Resultados caso básico (Fuente: Elaboración Propia).....	56
Tabla 4 Datos caso complejo rutado de camiones (Fuente: Elaboración propia)	58
Tabla 5 Resultados obtenidos (Fuente: Elaboración propia).....	60
Tabla 6 Leyenda codificación hoja "MAPA" en "Datos_ ACO". (Fuente: Elaboración propia).....	74
Tabla 7 Resultados colony.queen.fitness (Fuente: Elaboración propia).....	98
Tabla 8 Diagrama de Gantt, etapas del proyecto (Fuente: Elaboración propia)	112
Tabla 9 Horas efectivas trabajadas (Fuente: Elaboración propia)	113
Tabla 10 Horas dedicadas a cada etapa (Fuente: Elaboración propia).....	114
Tabla 11 Salarios en euros (Fuente: Elaboración propia).....	114
Tabla 12 Costes por etapas en euros (Fuente: Elaboración propia).....	115
Tabla 13 Coste y amortización de los equipos en euros (Fuente: Elaboración propia).....	115
Tabla 14 Costes generales en euros (Fuente: Elaboración propia)	116
Tabla 15 Coste total del proyecto en euros (Fuente: Elaboración propia)....	116
Tabla 16 Precio de venta en euros (Fuente: Elaboración propia)	117

1 INTRODUCCIÓN

1.1 CONTEXTO

Dentro del ámbito industrial un aspecto muy relevante es todo lo relacionado con la logística, tanto a nivel interno en el ámbito de la gestión de los almacenes, como a nivel externo, relacionado con la distribución o recogida de mercancías en distintos puntos o ubicaciones.

La complejidad de este tipo de gestiones logísticas reside en intentar optimizar al máximo los recorridos y también los tiempos, puesto que se trata de operaciones que no proporcionan valor al producto, sino que únicamente aumentan los costes. Sin embargo, es necesario y relevante, ya que en la mayoría de los procesos industriales existe un suministro de las materias primas y transporte de los productos en curso y terminados. Todos estos materiales, tanto las materias primas como los productos en curso o los productos terminados, generalmente, necesitan una serie de almacenes donde ser almacenados en los tiempos intermedios del proceso, o una vez ha terminado el procesado y antes de ser distribuidos. Todas estas operaciones se engloban en el ámbito de la logística industrial.

Existe una gran variedad de métodos de optimización, de diversos tipos, cuyo objetivo es proporcionar soluciones a la problemática en este ámbito. Estos métodos se pueden clasificar en exactos, reglas de despacho, métodos heurísticos y metaheurísticos. Dentro de estos últimos se encuentran los basados en la naturaleza. Estos métodos utilizan algoritmos fundamentados en comportamientos observados en distintos entornos, como pueden ser los comportamientos eficientes observados y estudiados en algunas especies.

En este Trabajo Fin de Grado se plantea utilizar uno de estos métodos, concretamente la optimización por medio de colonia de hormigas, aplicado a la logística industrial.

Se plantean dos situaciones diferentes, un primer caso aplicado a la logística de transporte exterior a las factorías, en la cual se gestiona la ruta que deba seguir un vehículo por carretera, el cual debe, partiendo de un lugar pasar por varias localidades y regresar al punto de partida. Esto se puede aplicar a situaciones de reparto o recogida de productos. Para este caso se optimizará la solución en función al tiempo de recorrido, ya que una menor distancia no implica menor tiempo debido a la velocidad, determinada por el tipo de vías. Además, en estos casos resulta interesante disminuir el tiempo que se tarda en hacer la ruta y, por tanto, disminuir los costes asociados al personal y medios destinados a ello.

En segundo lugar, se plantea un caso aplicado a la logística interna en las factorías, relativa a los almacenes. En dicho caso, se trata de optimizar el

recorrido a realizar por un vehículo de guiado automático (AGV) para pasar por distintos puntos de un almacén y llegar a un punto final indicado. Esto se plantea para situaciones en las que un AGV deba recoger distintos elementos distribuidos por un almacén y llevarlos a un punto donde se dejarán estos elementos. Para esta situación se optimizará la solución en función de la distancia, ya que la velocidad será aproximadamente constante, únicamente variando en los puntos donde deba parar. Puesto que la velocidad es constante, el tiempo está directamente relacionado con la distancia. Además, esta velocidad vendrá marcada por la normativa de este tipo de dispositivos, por las características del vehículo utilizado y por la normativa de seguridad. Por tanto, en estos casos resulta interesante disminuir la distancia, ya que al hacer esto, se disminuye el recorrido a realizar y, por tanto, los costes asociados al desplazamiento, tanto en tiempo de uso como en tiempo y coste que supone la carga de baterías.

1.2 JUSTIFICACIÓN DEL TFG

El uso y aplicación de los métodos de optimización, es una práctica extendida en el ámbito de investigación científico-técnica, pero este Trabajo Fin de Grado se centra en la implementación del algoritmo colonia de hormigas, puesto que no es uno de los más utilizados a día de hoy, pero presenta un gran potencial.

Pese a encontrarse diversos estudios basados en colonia de hormigas, éstos no se aplican en demasiados casos a la logística, campo en el que es de gran importancia la optimización de los recursos, por su limitación y el coste que éstos suponen. Por ejemplo, en el transporte la gestión logística no aporta valor a los productos, pero sí que es necesaria en gran parte de los procesos y siempre supone unos costes importantes. La manera de poder minimizar estos costes es optimizar los procesos relacionados, y es ahí donde se estimó oportuno el aplicar este tipo de método de optimización.

Por ello, en este Trabajo Fin de Grado se plantea el utilizar métodos de optimización por colonia de hormigas en el ámbito de la logística industrial, tanto para el caso del transporte, como para la gestión interna de los almacenes industriales, que son los dos casos fundamentales en dicho ámbito. De hecho, este Trabajo Fin de Grado parte de estos dos casos, como inicio de esta línea de investigación, buscando soluciones que permitan responder a las necesidades existentes en la logística industrial, puesto que se considera que puede ser un entorno interesante en el que utilizar este tipo de algoritmos.

1.3 OBJETIVOS

1.3.1 Objetivo principal

El principal objetivo en este trabajo es la aplicación de un algoritmo colonia de hormigas, lo cual es un método metaheurístico de resolución de problemas, al ámbito industrial.

En concreto, se aplica este método a dos casos relacionados con el transporte, la obtención de rutas para camiones en tareas de reparto o recogida, y la obtención de rutados para vehículos autónomos AGVs dentro de la distribución en planta.

Además, se realiza un análisis de cada caso, así como un análisis conjunto de todos los resultados obtenidos, para obtener conclusiones acerca de la utilidad de algoritmos colonia de hormigas para estos casos.

1.3.2 Objetivos complementarios

Además del objetivo principal, este Trabajo Fin de Grado también plantea los siguientes objetivos complementarios:

- Conocer los distintos tipos de métodos de optimización.
- Estudiar detalladamente el método de optimización por colonia de hormigas.
- Comprender y utilizar la algorítmica de colonia de hormigas.
- Diseñar y desarrollar el algoritmo colonia de hormigas utilizando el software Matlab.
- Obtención de los resultados y análisis pormenorizado de cada caso, y también análisis conjunto, para realizar recomendaciones.
- Realizar un estudio económico del proyecto.

1.4 MEDIOS SOFTWARE EMPLEADOS

A continuación, se indican las principales herramientas software utilizadas para la realización de este trabajo.

1.4.1 MATLAB

MatLab es un programa diseñado para la computación técnica. El nombre proviene de Matrix Laboratory puesto que el dato básico que gestiona es una matriz (array). MatLab se puede utilizar para computación matemática, simulación y modelado, análisis y procesamiento de datos, representación y visualización de gráficos y desarrollo de algoritmos.

Se empleará en este trabajo para ejecutar el algoritmo ACO (Algoritmo de Colonia de Hormigas), generar las gráficas pertinentes y en base a ello, poder analizar los resultados obtenidos.

1.4.2 EXCEL

Excel es un programa informático que forma parte del conjunto ofimática “Office” desarrollada por “Microsoft”. Excel está incluido en la categoría de programas informáticos denominada *hojas de cálculo*, las cuales fueron desarrolladas para simular las hojas de trabajo contable y automatizar de esta forma este tipo de tareas. Estos programas permiten trabajar con datos numéricos, de forma que se pueden almacenar y realizar cálculos aritméticos básicos, además de utilizar funciones matemáticas más complejas o funciones estadísticas. También permite analizar fácilmente distintos tipos de datos por medio de gráficos y tablas dinámicas.

Se empleará en este trabajo como entrada de datos al Software Matlab funcionando como un puente o pasarela entre el usuario y dicho software, de tal manera que se puedan generar los puntos que se han de introducir en Matlab ayudándose de un plano, generado en nuestro caso en Autocad.

1.4.3 AUTOCAD

AutoCAD es un software de diseño asistido por ordenador (Computer Aided Design) desarrollado por la compañía Autodesk. Permite tanto el diseño en 2D como en 3D. Este software trabaja utilizando imágenes de tipo vectorial, aunque también permite importar archivos de otros tipos.

AutoCAD tiene como objetivo principal el diseño de planos, ofreciendo para ello una gran librería de recursos como tipo de línea, colores, o texturas entre otros. Además, tiene un sistema de capas que permite organizar adecuadamente el diseño y facilitan el trabajo.

En dicho software, que puede ser reemplazado por cualquier otro que permita el desarrollo de implantaciones de naves industriales o lugares de tránsito de vehículos autónomos, se generará la implantación que se utilizará posteriormente para generar los distintos tipos de puntos que se utilizarán como datos en el algoritmo. En éste se indican el espacio sobre el que los AGVs pueden o no pueden desplazarse, representando los distintos elementos a través de los cuales se debe desplazar el AGV y las zonas de interacción importantes como las zonas de carga de baterías de dicho elemento.

1.5 ESTRUCTURA DE LA MEMORIA

A continuación, presento la estructura que sigue la memoria de este Trabajo Fin de Grado.

En el capítulo 2 se presentan los distintos métodos de optimización, partiendo de los métodos exactos y las reglas de despacho hasta llegar a los métodos heurísticos y metaheurísticos. A continuación, se expone más detalladamente el método de optimización por colonia de hormigas, el cual es el que se utilizará

en este proyecto y finalmente se ofrece un apartado en el que se exponen el tipo de distribuciones en planta, lo cual será relevante para una parte de este proyecto.

En el capítulo 3 se expone uno de los algoritmos creados en este trabajo, el cual consiste en utilizar un algoritmo ACO para la obtención de rutas de camiones. En este capítulo se indican los programas realizados para este propósito y dos casos prácticos, uno más sencillo para exponer detalladamente el funcionamiento y otro caso de mayor complejidad, el cual se asemeja a un caso real.

El capítulo 4 presenta en este caso cómo se utiliza un algoritmo ACO para la obtención de rutas de AGVs. La estructura del capítulo es similar al anterior, exponiendo la realización de los programas necesarios y dos casos, uno simple en el que se puede indicar detalladamente el funcionamiento y otro más complejo asemejándose a un caso real.

En el capítulo 5 se muestra un análisis de los casos realizados en los capítulos 3 y 4. Consta de cuatro grandes apartados. El primero de ellos contiene un análisis de los resultados obtenidos en el algoritmo creado para rutas de camiones, correspondiente al capítulo 3. En el segundo se encuentra un análisis similar para el caso de los AGVs, expuesto en el capítulo 4. A continuación aparece un análisis conjunto donde se presentan las diferencias existentes entre ambos algoritmos y las conclusiones generales comunes a todos los casos. Finalmente, se propone un apartado de recomendaciones para los algoritmos ACO basándose en los datos obtenidos en este trabajo.

El capítulo 6 ofrece un estudio económico correspondiente a los costes de realización del proyecto y del precio de venta estimado de dicho proyecto.

El capítulo 7 expone las conclusiones obtenidas tras la realización de este Trabajo Fin de Grado, y también las líneas futuras de dicho trabajo.

Finalmente, el capítulo 8 está dedicado a toda la bibliografía consultada y empleada en este Trabajo Fin de Grado.

2 MARCO TEÓRICO

En este capítulo se presentan y describen todos los conceptos que se van a emplear en el desarrollo del TFG.

2.1 MÉTODOS Y ALGORITMOS DE OPTIMIZACIÓN

La optimización o búsqueda de la solución óptima a un problema consiste en obtener la mejor solución posible dentro de todas las soluciones factibles para un determinado problema. Generalmente la solución óptima es aquella donde la función objetivo alcanza su valor mínimo o máximo, dependiendo del objetivo que se desea obtener. Esta solución indica el valor que deben tomar las variables de decisión, las cuales están sujetas a restricciones.

Existen problemas de optimización sencillos de resolver, donde es posible llegar a la solución óptima con un costo computacional razonable. Sin embargo, existen otros problemas más complicados donde no es posible encontrar la solución en un tiempo de cálculo razonable. En estos últimos casos se utilizan métodos heurísticos, en los cuales se llega a una solución suficientemente buena con mucho menor costo computacional. En estos casos es tan importante la solución como que el tiempo de cálculo de ésta sea razonable.

Para afrontar problemas sencillos se pueden utilizar los métodos exactos donde se llega a la solución óptima. A continuación, se muestran los más importantes.

Para problemas de mayor complejidad en ocasiones no es posible llegar a la solución óptima por métodos exactos por lo que se pasan a utilizar métodos heurísticos y, más tarde, metaheurísticos. Estos, pese a que no en todas las ocasiones llegan a la solución óptima, sí que proporcionan una solución suficientemente buena en un tiempo de procesamiento adecuado. Tanto los métodos heurísticos como los metaheurísticos se exponen a continuación mostrando los más importantes.

2.1.1 Métodos exactos

En esta categoría se engloban los algoritmos que aseguran, en caso de que exista, la solución óptima al problema que se ha planteado. Tienen una base fundamentada en teoremas matemáticos.

Los métodos exactos presentan desventajas que impiden su uso en gran parte de los problemas de optimización como son el gran coste computacional y los elevados tiempos de cálculo necesarios para llevarlos a cabo. Sin embargo, para los casos en los cuales es posible utilizar métodos exactos resulta

adecuado su uso ya que son los únicos que proporcionan en todas las ocasiones la solución óptima y no una aproximación de ésta.

Algunos métodos exactos son:

Búsqueda exhaustiva (exhaustive Search): este método se puede aplicar a gran cantidad de problemas. Consiste en generar y evaluar todas las soluciones posibles dentro del espacio de búsqueda. A pesar de ser un método muy robusto tiene la desventaja de consumir una gran cantidad de tiempo de cálculo.

Se trata de una técnica simple y eficiente para problemas pequeños. Se considera que es útil para problemas tipo P, refiriéndose a la clase de complejidad que tiene un problema de decisión donde el tiempo de cómputo aumenta de manera polinomial, dentro de esta categoría se engloban la mayoría de los problemas naturales, funciones simples y algoritmos de programación lineal.

Desde el punto de vista práctico este método resulta útil para problemas muy pequeños, pero tiene una viabilidad dependiente de los recursos disponibles para la ejecución de los algoritmos. Esto nos lleva a la conclusión de que no resulta viable cuando los problemas comienzan a aumentar de tamaño, ya que, aun siendo problemas pequeños, pueden formar un espacio de búsqueda de la solución óptima con millones de soluciones factibles.

Programación lineal (Integer Programming): Es una de las herramientas más utilizadas, principalmente después de 1947 cuando George Dantzig desarrolló el método símplex para la solución de problemas lineales en la función objetivo y en sus restricciones. Se trata de una técnica robusta y flexible la cual se puede utilizar para una gran variedad de problemas.

La hipótesis principal de este método es que todas las funciones deben ser combinaciones lineales de las variables de decisión del problema. La función objetivo es la que se desea maximizar o minimizar. Las restricciones delimitan el espacio de búsqueda, por lo que son condiciones que se deben cumplir con los valores de las variables de optimización.

En los modelos de programación lineal las variables son continuas. Cuando estas variables son, además, enteras o binarias, el problema pasa a ser de Programación Lineal Entera Mixta.

Estos modelos, aunque llegarían a la solución óptima, para la mayoría de los problemas no resultan adecuados por el tiempo de cálculo necesario, ya que es demasiado elevado para comprobar todas las iteraciones necesarias.

Divide y vencerás (Divide and Conquer): Este método está basado en el algoritmo de Euclides, aunque la descripción de este algoritmo en ordenadores apareció en un artículo de John Mauchy en 1946. En este método se plantea que un problema complejo se puede dividir en varios sub-problemas más sencillos de resolver. Tras las resoluciones de los diferentes sub-problemas se

unen las distintas soluciones obtenidas para construir la solución óptima del problema global.

Esta metodología se considera eficiente cuando el esfuerzo y el tiempo requerido para la división del problema inicial, el cálculo de las soluciones de los sub-problemas y la unión de las soluciones para obtener la solución al problema inicial es menor que los necesarios para calcular la solución del problema inicial por otro método. Además, cabe destacar que la solución obtenida no es siempre la óptima y que puede darse el caso de que no sea una solución factible.

Programación dinámica (Dynamic Programming): Se trata de una metodología adecuada para problemas en los que se deben tomar decisiones de manera secuencial. Este método consiste en encontrar la solución global para un problema complejo partiendo de las soluciones secuenciales resultas, mediante un procedimiento recursivo. Para resolver un problema utilizando programación dinámica debe cumplir las siguientes características:

- Descomponerse en una secuencia de decisiones que se tomarán en diferentes etapas.
- Cada etapa debe tener un número finito de posibles estados.
- La decisión tomada en una etapa lleva a un estado de la siguiente etapa.
- La mejor decisión de una etapa es independiente de las decisiones tomadas en las anteriores.
- Se debe poder definir bien el costo de pasar de un estado a otro a través de las etapas, esta función de costos debe ser recursiva.

La programación dinámica puede ser de dos tipos, hacia adelante o hacia atrás. En esta última se comienza a partir del objetivo deseado y se hace el análisis hacia atrás en las etapas evaluando los costos. En la práctica, este método es difícil de aplicar y además no siempre se llega a la solución óptima, aunque siempre es a una solución factible.

Ramificación y poda (Branch and Bound): esta metodología consiste en dividir el espacio factible y buscar las soluciones únicamente donde se sabe que se puede encontrar la solución óptima, desechando de esta forma los espacios de soluciones factibles que no mejoran la solución actual. Esta división se realiza de manera particular truncando el espacio de soluciones posibles a medida que se construye la solución. Los conjuntos de soluciones factibles se acotan por medio de reglas de dominancia, calculando de forma matemática la información suficiente para, sin calcular las soluciones completas, saber si éstas son peores que la mejor solución actual y, en ese caso, descartar toda esa rama de soluciones factibles.

(Morillo, Moreno y Díaz, 2013)

2.1.2 Reglas de despacho

Las denominadas reglas de despacho no se consideran un algoritmo de optimización, sino un método simple que permite definir las prioridades a la hora de seleccionar una solución para el problema que se está considerando. Aunque estas reglas aparecieron antes que los siguientes métodos que veremos, heurísticos y metheurísticos se suelen utilizar en estos métodos.

A continuación, se muestran algunas de las reglas de despacho más comunes:

FIFO (First In First Out): En este caso el orden de salida es el mismo que el orden de llegada, esto significa que lo primero que llega es lo primero en salir. Habitualmente se utiliza en el caso del procesamiento de elementos perecederos.

LIFO (Last In First Out): En este caso lo último en llegar es lo primero en salir. No es muy habitual, pero se puede utilizar para procesar elementos de gran volumen cuando hay mucha rotación.

SPT (Sort Process Time): Consiste en ordenar los elementos de menor a mayor tiempo de procesamiento.

LPT (Longest Process Time): Se trata del caso contrario a SPT, ordenando de mayor a menor tiempo de procesamiento.

EDD (Earliest Due Date): En esta regla se ordenan los trabajos en orden de prioridad, esto es, en orden según sea la fecha de entrega más próxima. El primer trabajo que debe ser entregado será el primero de la lista.

ECT (Earliest Completion Time): En este caso se ordenan los trabajos según la fecha de finalización, realizando en primer lugar los trabajos que se terminen antes.

Holgura Mínima: Se consideran los tiempos restantes hasta la finalización del trabajo, ordenando en primer lugar los que tengan menor holgura. De esta forma se programan en primer lugar los trabajos con mayores probabilidades de retrasarse.

SIRO (Service In Random Orden): En este caso no se toma ningún criterio de prioridad para seleccionar los elementos, sino que se toma un orden aleatorio.

(Guerra, 2015)

2.1.3 Algoritmos heurísticos

En casos de problemas más complejos no es posible utilizar los métodos exactos, ya que para calcular la solución óptima necesitan tiempos de cálculo demasiado elevados, llegando en algunos casos a resultar inviable. Para estos casos se utilizan métodos heurísticos, a partir de los cuales se obtienen soluciones suficientemente buenas (aunque no sea la mejor solución) en tiempos de cálculo razonables. En estos casos es tan importante que la solución sea buena como la rapidez de cálculo de esta.

Una definición de métodos heurísticos es:

“Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución.” (Diaz, 1996)

Algunos de los casos en los que se utilizan métodos heurísticos son para:

- Problemas para los cuales no se conoce ningún método de resolución exacto
- Problemas en los cuales el método exacto es muy costoso computacionalmente
- Problemas en los que se introducen condiciones de difícil modelación, ya que los métodos heurísticos tienen mayor flexibilidad
- Casos donde el método heurístico se utiliza como herramienta para un procedimiento global donde se garantizará la solución óptima. Puede ser porque se utilice el método heurístico para obtener una buena solución inicial de partida o porque este método participe como un paso intermedio de un procedimiento.

Los algoritmos heurísticos se pueden clasificar de diferentes maneras, a continuación, se muestra una clasificación de los más relevantes.

Métodos de descomposición: Se descompone el problema inicial en subproblemas que son más sencillos de resolver.

Métodos inductivos: Consisten en generalizar versiones más sencillas al caso completo; se pueden aplicar al caso completo las características o propiedades que se han identificado en los casos más sencillos.

Métodos de Reducción: Se trata de identificar las propiedades que se cumplen generalmente en las buenas soluciones e incluirlas como restricciones para el problema; de esta forma se limita el espacio de soluciones, simplificando así el problema. En este método existe el riesgo de dejar fuera del espacio de soluciones la solución óptima.

Métodos Constructivos: Consisten en construir paso a paso la solución al problema seleccionando la mejor solución en cada iteración.

Métodos de Búsqueda Local: En estos métodos se parte de una solución inicial y se va mejorando progresivamente con las diferentes iteraciones. Para ello, en cada paso se realiza un movimiento de una solución a otra con un valor mejor, terminando el método cuando no aparece ninguna solución que mejore la última que se ha obtenido. Puesto que las soluciones son evoluciones de la solución inicial, puede ocurrir que no se llegue a la solución óptima porque ésta no llegue a ser una evolución de la solución que se ha tomado inicialmente.

(Martí, 2003)

2.1.4 Métodos Metaheurísticos

Se trata de algoritmos de aproximación con los que se puede obtener una buena solución (aunque no la óptima) en poco tiempo. La ventaja de estos métodos es la gran flexibilidad de la que disponen, por lo que se pueden utilizar para una gran variedad de problemas. La principal desventaja de este tipo de métodos es que la solución obtenida no será la óptima, aunque sí que será una buena solución. Una definición de estos métodos es la dada por Osman y Kelly:

“Los metaheurísticos son métodos aproximados diseñados para resolver problemas de optimización combinatoria, en los que los heurísticos clásicos no son efectivos. Los metaheurísticos proporcionan un marco general para crear nuevos algoritmos híbridos, combinando diferentes conceptos derivados de la inteligencia artificial, la evolución biológica y los mecanismos estadísticos.”
(Osman y Kelly, 1996)

A continuación se presentan los algoritmos metaheurísticos más importantes.

2.1.4.1 Algoritmos GRASP

Los algoritmos GRASP (Greedy Randomize Adaptative Search Procedure) consisten en un procedimiento de búsqueda voraz aleatorizado y adaptativo. Se trata de una técnica que se puede considerar como un algoritmo multi-arranque. Este método fue propuesto por Tomas Feo y Mauricio Resende en 1989.

Estos algoritmos se basan en dos ideas:

- En primer lugar, se basan en que en gran cantidad de problemas de optimización combinatoria se pueden determinar soluciones incorporando iterativamente a una de ellas un elemento compatible con los ya existentes. La incorporación de estos elementos puede ser por medio de reglas con mayor o menor complejidad o incluso aleatoriamente. Si la elección del elemento es al azar es posible obtener una gran variedad de soluciones con valores muy dispersos; por otro

lado, si se selecciona el elemento por medio de una regla razonable la solución obtenida probablemente será satisfactoria, aunque no será la óptima. Una opción intermedia a los dos casos anteriores es elegir un elemento a incorporar al azar, pero únicamente de entre una serie de elementos inicialmente prometedores, de forma que se obtengan una variedad de soluciones, pero no tan dispersas como en el caso de seleccionar aleatoriamente entre todos los elementos compatibles.

- En segundo lugar, se aplica un procedimiento de optimización local a la solución construida. Inicialmente los candidatos a ser seleccionados para formar el conjunto de soluciones de las cuales se elegirán al azar se forma por un cierto número de elementos compatibles con los que ya forman parte de la lista de candidatos y aquellos cuyo indicador no varía en un porcentaje especificado del que se corresponde al primer elemento de la lista.

Algunas limitaciones de este método son que las soluciones que resultan de aplicar un GRASP tradicional suelen pertenecer a un subconjunto estricto de soluciones factibles, ya que las limitaciones impuestas para la selección de candidatos a menudo no permiten explorar todo el conjunto de soluciones posibles. La limitación de los candidatos también puede provocar que de dos elementos con el mismo valor de indicador únicamente uno de éstos sea candidato. La última limitación del método es el hecho de que todos los elementos candidatos tengan la misma probabilidad.

(Bautista et al., 1999)

2.1.4.2 Recocido simulado (simulated annealing)

El recocido simulado es una técnica de búsqueda aleatoria dirigida, cuya primera propuesta fue de la mano de Kirkpatrick, Gelatt y Vecchi (1983). Ésta se basa en una analogía al comportamiento de los metales en el proceso de enfriamiento.

El recocido es un proceso por el cual los metales adquieren una estructura estable o de mínima energía. Esto se consigue calentando la sustancia hasta altas temperaturas que permiten el movimiento de sus partículas y luego enfriando lentamente, de forma que las partículas adoptan la configuración más estable posible para dicha temperatura. El proceso pasa por diversos estados de equilibrio a medida que se enfría el material, hasta llegar a la temperatura ambiente, en el caso de realizarse un cambio brusco de temperatura el material obtenido tendría estructuras cristalinas defectuosas o alcanzaría un estado con estructuras localmente óptimas, denominado metaestable.

La propuesta de Kirkpatrick consiste en una simulación Metrópolis Montecarlo para determinar la configuración de energía mínima de un sistema.

La simulación Montecarlo realiza una serie de exploraciones de un espacio de soluciones, y tabula los resultados obtenidos. A partir de estos resultados se puede obtener información acerca del espacio de soluciones del problema. El espacio de soluciones es explorado por medio de movimientos aleatorios, considerándose cualquier punto valido para obtener información de la configuración de este espacio de soluciones.

El método Metrópolis es una modificación de la simulación Montecarlo. La diferencia es la manera de generar las configuraciones que se analizarán. Éstas se irán generando a partir de una ligera modificación de la anterior.

El método de Kirkpatrick, el cual es una simulación Metrópolis Montecarlo, consiste en elevar la temperatura de un sistema permitiendo una evolución rápida de éste hacia una configuración estable. Tras esto se disminuye la temperatura lentamente de manera progresiva, lo que permite al sistema evolucionar hacia configuraciones de mínima energía. Este proceso es conocido como “annealing” (recocido), conociéndose el método como “Simulated Annealing” (Recocido Simulado).

Este método comienza por medio de una simulación Metrópolis a altas temperaturas, en este caso el criterio de aceptación de la simulación permitirá aceptar un alto porcentaje de configuraciones en las cuales existan incrementos de energía del sistema. Tras una serie de simulaciones la temperatura es disminuida y se vuelve a realizar la simulación. Este procedimiento se va realizando hasta llegar a una temperatura mínima definida previamente. A medida que la temperatura va disminuyendo también disminuye la probabilidad de aceptar configuraciones que impliquen un incremento de la energía del sistema. De esta manera, a altas temperaturas el proceso es guiado por las características del sistema que más afectan a su estado energético, y a medida que la temperatura disminuye predominan en la evolución los detalles de la configuración por delante de dichas características.

Gracias al criterio de aceptación de configuraciones de mayor energía, proporcionado por Metrópolis, este método tiene una capacidad de salir de óptimos locales muy elevada lo cual resulta interesante para casos en los cuales existan varios óptimos locales, de forma que no se quede anclado en uno de ellos, sino que pueda explorar más allá del primer óptimo local que encuentre.

(Vázquez, 1995)

2.1.4.3 Búsqueda tabú (*tabu search*)

Este método fue introducido por Fred Glover en 1986 en el artículo “Future Paths for Integer Programming and Links to Artificial Intelligence” para la revista *Computers and Operations Research*. Más tarde, todos los principios fundamentales fueron agrupados en el libro “Tabu Search” en 1997, de dicho autor.

La búsqueda tabú es un procedimiento metaheurístico diferenciado por el uso de memoria adaptativa y con la característica de tener diferentes estrategias especiales de resolución de problemas. Las estructuras de memoria de este método funcionan a partir de referencia a cuatro dimensiones, que son la propiedad de ser reciente, calidad, frecuencia e influencia. Estas son fijadas frente a los antecedentes de estructuras lógicas y conectividad.

La búsqueda tabú va más allá del criterio de finalización en un óptimo local, diferenciándose así de los métodos anteriores de forma que el objetivo de este método es la obtención de la solución óptima al problema planteado.

Para comenzar este algoritmo es necesario dar una solución inicial obtenida habitualmente tras aplicar un método heurístico. A partir de aquí, este método utiliza un procedimiento de búsqueda local, pero, además, para explorar regiones del espacio de soluciones que el procedimiento de búsqueda local no considera, la búsqueda tabú modifica la estructura búsqueda para cada solución a medida que avanza el algoritmo. Las soluciones admitidas son determinadas mediante el uso de estructuras de memoria, progresando la búsqueda moviéndose iterativamente de una solución a otra.

La estructura de memoria utilizada para determinar las soluciones permitidas, denominada lista tabú, es una memoria de corto plazo donde se almacenan las soluciones visitadas recientemente (en un número de iteraciones especificado), de forma que la búsqueda excluye estas soluciones en el periodo de tiempo que aparecen en la lista tabú. Esto permite que el algoritmo explore nuevas soluciones sin perder las que se encuentran memorizadas en la lista tabú, pero sin repetirlas. Es posible incluir en la lista tabú, además, soluciones que tengan ciertos atributos por lo que no sean deseables, de forma que no se sigan buscando soluciones entorno a ellas.

Por último, hay que indicar que para que no ocurra que una solución buena sea desechada por estar incluida en la lista tabú, aparecen los criterios de aspiración, los cuales pueden modificar el estado tabú o incluir alguna solución de las que están excluidas del conjunto de soluciones permitidas. Uno de los criterios de aspiración más comunes es admitir una solución cuando es mejor que la mejor solución conocida.

(Melián y Glover, 2007)

2.1.4.4 Redes neuronales (*artificial neural networks*)

Las redes neuronales artificiales (RNA), ANN en inglés, consisten en simular los procesos que tienen lugar en las redes neuronales naturales, por lo que pretenden lograr sistemas capaces de aprender de la experiencia y de realizar predicciones. Estos sistemas se basan en una densa interconexión de pequeños procesadores denominados neuronas.

Las redes neuronales como método de optimización comienzan a desarrollarse en la década de 1940 por medio de los estudios de McCulloch y Pitts. En 1958 Frank Rosenblatt desarrolla un de los tipos más conocidos de redes neuronales, el denominado Perceptron. Tras esto, en el año 1969 se detuvieron las investigaciones en este tema, cuando Minsky y Papert demostraron la incapacidad de resolver problemas tipo XOR por el método Perceptron. Con la aparición de nuevos tipos de redes neuronales con capacidad para la optimización no lineal se retomaron los estudios en este ámbito.

Una neurona artificial es una representación matemática del funcionamiento de las neuronas naturales. De manera simplificada se pueden comparar con un amplificador operacional en el cual se reciben señales de entrada, se realiza una comprobación por medio de una función, y en función del resultado se produce una señal de salida o no.

Las redes neuronales artificiales generalmente están formadas por varias neuronas interconectadas, salvo en algunas excepciones como es la red Perceptron, la cual está formada únicamente por una neurona. Habitualmente las neuronas están colocadas formando capas, aunque la disposición y la forma de conexión varía entre los distintos tipos de redes.

La topología consta, en primer lugar, de la denominada capa de entrada, que no es una capa de neuronas, sino los puntos en los cuales se recogen las señales que se pasarán a la red. Esta capa únicamente recoge los datos de entrada, no procesa la información.

A continuación, se encuentran las capas ocultas, las cuales toman la información de la capa de entrada y la procesan. Dependiendo del tipo de red varían tanto el número de capas ocultas que existen, como el número de neuronas que se encuentran en cada capa, además del modo de conexión de éstas.

Por último, se encuentra la capa de salida, la cual recibe la información de la última capa oculta y se la transmite al medio.

Una vez es configurada la topología de la red, ésta debe someterse a un entrenamiento, lo cual permitirá proporcionar salidas convenientes a las entradas presentadas. Este entrenamiento es diferente para cada tipo de red.

Un ejemplo de entrenamiento de redes es el modelo supervisado, en el cual, para adaptar el comportamiento de la red al problema planteado en primer lugar, se aplican una serie de entradas en las que se conoce el resultado esperado. Se comparan las salidas obtenidas de la red con los resultados conocidos, y se van adaptando los distintos parámetros en función del error cometido. Este proceso se repite hasta que el error se reduce según un valor determinado, momento en el cual se considera que está entrenada la red.

Existen diferentes clasificaciones para las redes neuronales dependiendo de criterios como el tipo de valores de entrada, la forma de aprendizaje, las funciones de activación o la topología. Una de estas clasificaciones es la proporcionada por (Acosta-Buitrago, 2000), en la cual se toma como criterio principal el tipo de entrada y en segundo lugar el tipo de aprendizaje, pudiendo ser supervisado o no supervisado, como se puede observar en la Figura 1:

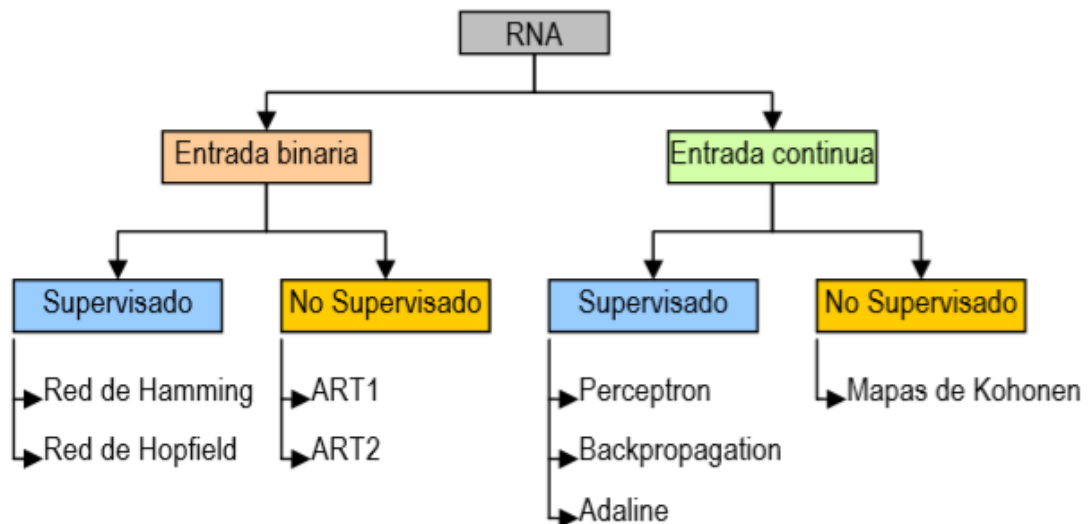


Figura 1 Clasificación redes neuronales (Acosta-Buitrago, 2000)

En la clasificación anterior se muestran las RNA más extendidas en utilización, pero existen también otros tipos de redes neuronales.

(Diego, 2006)

2.1.4.5 Búsqueda local iterativa (*iterated local search*)

Este método se propuso en 2001 por Ibaraki, Kubo, Uno y Yagiura. En éste se comienza con una solución inicial, la cual puede ser obtenida utilizando un método heurístico o de manera aleatoria. Una vez es obtenida esta solución inicial se busca dentro de los valores cercanos a ella un mínimo local. Tras esto se aplican tres operadores de forma iterativa, la perturbación que al modificar la solución permite alejarla del mínimo local obtenido, la búsqueda que analiza

la nueva zona para encontrar un nuevo mínimo local y el criterio de aceptación, por medio del cual se decide cual será la siguiente solución para aplicar el primer operador (la perturbación), si al nuevo mínimo obtenido o al que teníamos del paso anterior. A continuación, se continúa con la siguiente iteración aplicando los distintos operadores. Se trata de un método sencillo, pero con gran eficacia. (Yepes, 2002)

2.1.4.6 Computación evolutiva (evolutionary computing)

Este método plantea los problemas de búsqueda y optimización bajo un enfoque evolutivo, basándose a grandes rasgos en la teoría de la evolución de las especies de Darwin (1859), implementándose con los llamados algoritmos evolutivos. De manera general, estos algoritmos realizan una búsqueda basándose en la evolución de una estructura y seleccionan las más adecuadas en cada iteración.

Los algoritmos evolutivos se pueden clasificar en tres categorías:

- Algoritmos genéticos
- Estrategias evolutivas
- Programación evolutiva

Estos tres métodos tienen un esquema común (debido a que se basan en las mismas ideas de partida), aunque se desarrollaron de manera independiente.

Parten de un conjunto de soluciones generadas aleatoriamente denominado población inicial. Esta población se somete a transformaciones que darán lugar a una nueva población. Se valora la capacidad de estas soluciones a adaptarse al medio mediante una función de transformación, denominada función objetivo. Como resultado de la valoración de los distintos individuos, e incluyendo un factor de aleatoriedad, se favorece a los mejores individuos en un proceso de selección para formar la próxima generación. Este proceso se repite y, tras cierto número de iteraciones, la población ha evolucionado hacia una mejor adaptación.

Estas características son comunes a todos los algoritmos evolutivos. La diferencia entre estos métodos consiste en las diferentes alteraciones que se realizan para dar lugar a los nuevos individuos y en los métodos utilizados para evaluar las soluciones obtenidas y seleccionar los individuos que pasarán a formar parte de la siguiente generación.

Estos algoritmos se escogen en función del problema que se quiere resolver, pues cada uno de ellos es más adecuado para un tipo de problema.

A continuación, se indican las particularidades de los diferentes algoritmos evolutivos.

(Diego, 2006)

Algoritmos genéticos

Los algoritmos genéticos, originalmente propuestos por Jonh Holland (1975), son unos de los algoritmos evolutivos más extendidos, lo que es debido a las siguientes características:

- Se trata de algoritmos flexibles que se pueden adaptar a gran cantidad de problemas de diferentes ámbitos, además de permitir combinarse con otras técnicas (denominado hibridación). Son muy versátiles, permitiendo su uso sin demasiado conocimiento específico del problema en cuestión.
- Poseen una gran base teórica, y recogen perfectamente las ideas del enfoque evolutivo en el cual se basan.
- Es posible implantarlos en ordenadores sin grandes capacidades de cálculo y aun así se obtienen buenos resultados.

Estos algoritmos se denominan de búsqueda ciega, ya que únicamente disponen de la información proporcionada por medio de la función objetivo utilizada.

El método comienza como se ha descrito en el caso general para algoritmos evolutivos, tomando un conjunto de soluciones aleatorias. Estas soluciones están codificadas (generalmente en binario) por medio de una cadena de longitud finita, de forma que se corresponde con un punto del dominio del problema. Tras evaluar las soluciones, se lleva a cabo el proceso de selección de los individuos a partir de los cuales se formará la siguiente generación. Este proceso puede ser de diversos tipos, desde seleccionar a los individuos con mejores valoraciones, a seleccionarlos por uno de los diferentes métodos existentes de muestreo estocástico.

A continuación, se produce la transformación de los individuos seleccionados para formar la siguiente generación. Existen diferentes métodos a partir de los denominados operadores genéticos, algunos de los más utilizados son la selección, el cruce y la mutación, los cuales tienen gran cantidad de variantes. A partir de esos operadores se obtienen individuos nuevos que heredan las características de sus predecesores, los cuales formarán la población en la nueva generación.

A partir de aquí, el proceso se repite hasta que se cumpla una de las siguientes condiciones: algún individuo alcanza un valor prefijado de nivel de adaptación, el algoritmo converge o se llega al número de iteraciones máximas fijado. (Diego, 2006)

Estrategias evolutivas

Las estrategias evolutivas fueron desarrolladas por Ingo Rechenberg (1973) para resolver problemas de optimización paramétrica. Hay varios tipos:

- Estrategias evolutivas simples, (de dos miembros) hacen evolucionar únicamente a un individuo, utilizando la mutación como operador genético para las transformaciones. Al utilizar un único individuo no existe la fase de selección y el único criterio de reemplazo consiste en si se sustituye al progenitor o no, dependiendo de la valoración obtenida.
- Estrategias evolutivas múltiples, que aparecen para solucionar un problema frecuente en las estrategias evolutivas simples, las cuales tendían a converger rápidamente en óptimos locales, finalizando ahí el algoritmo. Introducen la búsqueda múltiple y la discriminación de los descendientes, incluyendo los operadores genéticos selección y cruce. En este método el criterio de reemplazo es determinista, seleccionando siempre a los individuos con mejores resultados.

Las principales diferencias entre estas estrategias y los algoritmos genéticos son:

- Las estrategias evolutivas necesitan mayor conocimiento específico del problema a resolver.
- Los algoritmos genéticos realizan una búsqueda global, mientras que por el contrario, las estrategias evolutivas tienen mejores resultados en búsquedas localizadas.
- En las estrategias evolutivas el operador genético más importante es la mutación y en los algoritmos genéticos lo es el cruce.
- En el caso del reemplazo también existen diferencias, puesto que en las estrategias evolutivas es determinista, y en los algoritmos genéticos utilizan métodos estocásticos.
- Los algoritmos genéticos son adecuados para su utilización en problemas de optimización de atributos, mientras que las estrategias evolutivas son más adecuadas para la optimización paramétrica.

(Diego, 2006)

Programación evolutiva

La programación evolutiva fue inicialmente propuesta por Lawrence J. Fogel (1960), con el objetivo de dotar a las máquinas de la capacidad de conocer su entorno, prever los cambios y reaccionar de manera adecuada. Fogel planteó un problema codificando el entorno como una secuencia de símbolos de un alfabeto finito. El algoritmo realizaba las operaciones necesarias para anticipar el próximo símbolo en aparecer en el entorno. Esto se llevaba a cabo por medio de autómatas finitos, que constan de un conjunto de reglas de transición y estados, de forma que cuando reciben una entrada, pueden cambiar o no de estado, y producen una salida.

El procedimiento es el siguiente:

- En primer lugar, se crea una población inicial de autómatas aleatoriamente y se le proporciona una cadena de símbolos que codifican las variaciones conocidas del entorno. Respecto a cada símbolo de entrada las máquinas proporcionan un símbolo de salida que se compara con el siguiente símbolo de la cadena, valorando de esta manera a los individuos dependiendo de la proximidad de la realidad y la predicción.
- A partir de mutación aleatoria se crean nuevos autómatas y se evalúan de la misma manera que sus progenitores.
- A continuación, se crea la nueva población, seleccionando a los autómatas con mejores resultados, tanto de los iniciales como de los creados por mutaciones.
- Se realizan los pasos de crear mutaciones y seleccionar los autómatas, mejorando así las poblaciones, hasta que aparece un nuevo símbolo en la cadena del entorno, en este momento se reinicia el proceso desde el principio.

Aunque se trate de computación evolutiva, este método difiere bastante de los presentados anteriormente. En este caso la codificación no asocia cada cadena con un individuo, sino que representa un conjunto de individuos de características similares, lo que se puede considerar una especie. Además, en cuanto a operadores genéticos no existe la posibilidad de cruce, la cual proporciona individuos inviables, sino que únicamente se realiza el proceso de adaptación por medio de mutaciones.

(Diego, 2006)

2.1.4.7 Colonia de hormigas (ant colony optimization, ACO)

El método denominado Ant System (AS), desarrollado por Dorigo en 1992, se considera el primer algoritmo perteneciente a la metaheurística de colonia de hormigas (ACO). Este primer algoritmo fue aplicado al problema del agente viajero.

Las metaheurísticas basadas en colonia de hormigas se desarrollaron a partir de la observación de las capacidades de las hormigas, insectos que pese a ser prácticamente ciegos logran realizar muy buenos recorridos para buscar alimento. Se ha visto que las hormigas se comunican a través de una sustancia química denominada feromona, la cual depositan a medida que realizan los distintos recorridos hasta la fuente de comida y de nuevo al hormiguero. A medida que más hormigas van y vuelven a la fuente de comida se intensifican las feromonas de los caminos más cortos, mientras que los caminos más largos son tomados por menos hormigas y las feromonas se evaporan con el tiempo. A pesar de los caminos de feromonas, en todo momento existe la probabilidad

de que algunas hormigas seleccionen caminos nuevos de manera aleatoria, lo que permite explorar nuevas soluciones. Con el paso del tiempo finalmente solo se utilizan los mejores o el mejor camino. El algoritmo colonia de hormigas está basado en este comportamiento natural de las hormigas. (Robles, 2010)

Existen diferentes variantes de los algoritmos colonia de hormigas; además, este método se puede combinar con otros, dando lugar a métodos mixtos, donde es común por ejemplo la combinación de colonia de hormigas con método de búsqueda local.

En el apartado siguiente se verá con mayor profundidad la algorítmica de colonia de hormigas.

2.2 MÉTODO DE OPTIMIZACION POR COLONIA DE HORMIGAS (ACO)

En este apartado se desarrolla la algorítmica de colonia de hormigas que se ha mencionado en el apartado anterior.

La teoría de optimización por colonia de hormigas aparece en 1992 como herramienta para solucionar problemas complejos de optimización, de la mano de Marco Dorigo. Se trata de un método heurístico, por lo que se trata de un algoritmo que trata de buscar una solución lo suficientemente buena a un problema complejo en un tiempo de cálculo razonable.

Como se ha indicado en el apartado anterior, esta teoría está basada en el comportamiento de las hormigas en la naturaleza cuando buscan alimento. Las hormigas inicialmente buscan la comida de manera aleatoria, y cuando una de ellas encuentra alimento, regresa al hormiguero dejando un rastro de una sustancia química (feromona) para que otras hormigas puedan seguirla, imitando su trayecto. Con el paso del tiempo, cada vez más hormigas se desplazan hacia el alimento, dejando reflejado el camino seguido, y de esta forma, en el camino más corto y frecuentado, la cantidad de feromonas se intensifica, llevando finalmente a que todas las hormigas vayan por el camino más corto.

Mostramos este proceso de seleccionar el camino más corto con un ejemplo ilustrativo en la Figura 2:

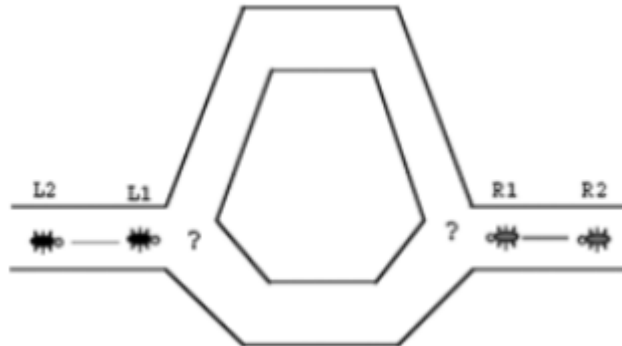


Figura 2 Las hormigas en el punto de decisión (Dorigo, 1996)

Inicialmente no hay presencia de feromonas, por lo que las hormigas irán aleatoriamente por cualquiera de los caminos. En el caso sencillo que utilizaremos como ejemplo, en promedio, la mitad de las hormigas irán por el camino superior y la mitad irá por el inferior, tanto para las que se desplazan desde la izquierda (denominadas L, left) como las que se desplazan desde la derecha (denominadas R, right).

Suponiendo que todas las hormigas se desplazan a la misma velocidad, podemos ver el avance de éstas en las Figuras 3 y 4. En estas imágenes vemos que el número de feromonas sería proporcional al número de líneas punteadas (rastro que han ido dejando las hormigas a su paso).

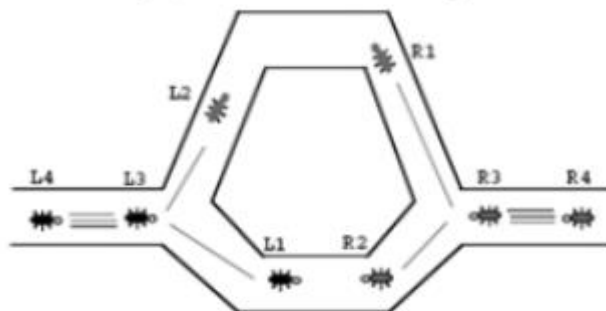


Figura 3 Elección de forma aleatoria entre los caminos inferior y superior (Dorigo, 1996)

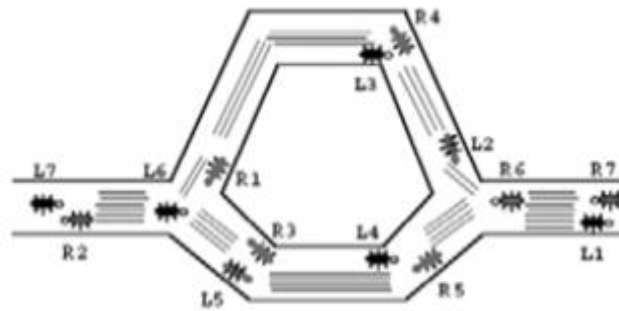


Figura 4 Efectos de la cantidad de hormigas que circulan por cada camino (Dorigo, 1996)

Como el camino superior es más largo, para el mismo periodo de tiempo, por el camino inferior habrán circulado más hormigas, aumentando así la cantidad de feromonas en este camino frente al más largo, lo que se puede ver en la Figura 5.

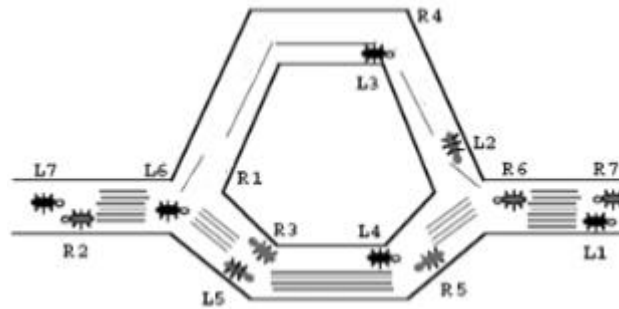


Figura 5 Mayor acumulación de feromonas en el camino más corto. (Dorigo, 1996)

Al cabo de un tiempo, la diferencia de cantidad de feromonas entre los dos caminos es lo suficientemente significativa como para influir en la decisión que tomarán las nuevas hormigas que realizan el camino, escogiendo el camino corto, por ser en el que perciben mayor cantidad de feromonas.

Además, según pasa el tiempo, puesto que las feromonas se van evaporando, únicamente quedará el rastro del camino más corto.

Los algoritmos de optimización basados en el método colonia de hormigas se basan en el comportamiento descrito anteriormente, y lo simulan computacionalmente trabajando de forma conjunta y comunicándose a partir de rastros de feromonas. Este tipo de programas son constructivos ya que, con cada iteración, cada una de las hormigas construye una solución al problema propuesto a través de un grafo. Cada una de las aristas representa las diferentes opciones que se pueden tomar, y se les asocia la siguiente información:

- Información heurística: preferencia heurística que tienen las hormigas para tomar un camino u otro. Esta información no es modificada durante la ejecución del algoritmo.
- Información de rastros de feromonas artificiales: éstas representan los caminos que han tomado otras hormigas en iteraciones anteriores, de forma que se puede medir a partir de esta información la deseabilidad aprendida a partir del movimiento, imitando las feromonas reales de las hormigas. Esta información se va modificando a medida que se ejecuta el algoritmo en función de las soluciones encontradas.

Las hormigas artificiales son las que construyen las diferentes soluciones computacionales simples basándose en la información anterior.

Este método tiene las propiedades siguientes:

- Encuentra soluciones válidas con menor costo.
- Almacena la información de todos los caminos recorridos en una memoria, por lo que se pueden construir soluciones válidas y reconstruir el camino seguido.
- Posee un estado inicial con una secuencia unitaria y unas condiciones de parada asociadas.
- Empiezan con un estado inicial moviéndose siguiendo estados válidos y construyendo de esta manera la solución de manera incremental.

(Robles, 2010)

2.2.1 Estructura básica del algoritmo de colonia de hormigas

Durante un periodo de tiempo, en el algoritmo se crean una serie de generaciones distintas de hormigas que recorrerán de manera concurrente los diferentes nodos que forman el problema, de forma que cada una de estas hormigas construye una solución al problema.

Para la selección del camino en cada uno de los nodos, las hormigas tienen en cuenta las feromonas existentes de iteraciones anteriores y un valor heurístico correspondiente al problema. Dependiendo de los algoritmos, en algunos casos además de agregar feromonas a los caminos seleccionados, se pueden disminuir la cantidad de feromonas de un camino, esto puede ser durante el recorrido para no repetir el mismo camino con varias hormigas o al finalizar la iteración para evitar que se tenga en cuenta malas soluciones únicamente por las feromonas depositadas en ellas.

Tras cada iteración se produce una evaporación de feromonas, lo que favorece a que las hormigas exploren regiones nuevas, evitando de esta forma que la

búsqueda se estanque. Por último, existen varias acciones finales que solo se incluyen en algunos algoritmos ACO pero que no son necesarias, , entre estas opciones se encuentran la de depositar feromonas extra en el mejor recorrido de la iteración, colocar feromonas de forma aleatoria (denominado comportamiento genético), o mejorar los recorridos obtenidos utilizando búsquedas locales. Esto se realiza para modificar el algoritmo colonia de hormigas básico. (Soifer, 2015)

A continuación, se describen algunos modelos de optimización basados en colonia de hormigas.

2.2.1.1 El sistema de hormigas (SH)

Fue desarrollado por Dorigo, Maniezzo y Colorni en 1991, siendo el primer algoritmo de ACO. Presenta tres variantes SH-densidad SH-cantidad y SH-ciclo, diferenciadas por el modo de actualizar los rastros de feromonas. En SH-densidad se deposita la feromona de manera constante a medida que se construye la solución, en SH-cantidad depende de la deseabilidad heurística de la transición la cantidad de feromona depositada, finalmente en SH-ciclo la deposición de las feromonas se lleva a cabo una vez se ha completado la solución. Esta última es la que mejores resultados ofrece, por lo que es el sistema de hormigas más conocido.

El SH se caracteriza porque la actualización de las feromonas se realiza una vez que todas las hormigas han terminado sus soluciones. Y esto se realiza de la siguiente manera: en primer lugar, se reducen por medio de un factor constante todos los rastros de feromonas existentes, simulando la evaporación, y tras esto, cada hormiga deposita una cantidad de feromona en función de la calidad de su solución.

(Alonso et al., 2003)

2.2.1.2 El sistema de Colonia de Hormigas (SCH)

El Sistema de Colonia de Hormigas se trata de uno de los primeros sucesores del sistema de hormigas, en el cual se introducen tres importantes modificaciones:

- El SCH utiliza una regla de transición distinta, denominada regla proporcional pseudo-aleatoria, por medio de la cual se selecciona el siguiente nodo por parte de cada hormiga. Esta regla permite tanto la exploración de nuevas conexiones aleatorias como utilizar la información heurística y de rastros de feromonas existentes para tomar la mejor opción.
- Por otra parte, la actualización de las feromonas se realiza fuera de la línea de los rastros, considerando únicamente la hormiga que generó la

mejor solución global. La actualización se realiza tras evaporar todas las feromonas existentes.

- Finalmente, son aplicadas una actualización en línea paso a paso por parte de las hormigas de los rastros de feromonas, lo cual favorece la generación de distintas soluciones.

(Alonso et al., 2003)

2.2.1.3 El sistema de Hormigas Max-Min (SHMM)

Este sistema fue desarrollado por Stützle y Hoos en 1996 (SHMM), siendo éste una extensión del sistema de hormigas SH de las que mejores rendimientos proporcionan. Amplía el SH en los aspectos siguientes:

- Se actualizan los rastros de feromonas fuera de línea, como ocurre en SCH.
- Una vez que todas las hormigas han construido su solución cada rastro de feromonas se evapora en función de una constante.
- Tras esto se depositan las feromonas de forma que la mejor hormiga es la que añade su solución, pudiendo ser ésta la mejor solución de la iteración o la mejor solución global.

De manera experimental se ha visto que los resultados que presentan mejores rendimientos se obtienen cuando se incrementa de manera gradual la frecuencia de seleccionar la mejor solución global para actualizar la feromona.

Se limitan a un rango los valores posibles para los rastros de feromonas, de tal forma, que disminuye la probabilidad de estancamiento del algoritmo, puesto que se le da a cada conexión una probabilidad de ser escogida. Los valores máximo y mínimo del intervalo son determinados por medio de heurísticas.

Finalmente, para incrementar la exploración de soluciones nuevas, en este método se utilizan las re-inicializaciones de los rastros de feromonas. Estos rastros se inicializan a una estimación del máximo permitido para un rastro, en lugar de inicializarlos a una cantidad pequeña. Esto permite la diversificación en el algoritmo.

(Alonso et al., 2003)

2.2.1.4 Sistema de hormigas con ordenación

El sistema de hormigas con ordenación (SH_{ordenación}) es otra extensión del sistema SH, que fue propuesta en 1997 por Bullnheimer, Hartl y Strauss. En este método se incorpora la idea de ordenar a las hormigas para la actualización de la feromona.

En este sistema, las hormigas se ordenan según la calidad de sus soluciones, de mejor a peor. La cantidad de feromonas depositadas dependen del orden de la hormiga y, por tanto, de la calidad de la solución. Además, se deposita

una cantidad adicional de feromona por la solución proporcionada por la mejor hormiga global.

(Alonso et al., 2003)

2.2.2 Optimización basada en colonia de hormigas en dos etapas

Se trata de una variante de exploración para algoritmos de la metaheurística ACO basada en el principio divide y vencerás. En esta alternativa se divide en dos fases el principio de exploración, en la primera únicamente un subconjunto de la colonia es utilizado para explorar parte del problema. Tras esto, las mejores soluciones encontradas en esta primera búsqueda forman los estados de partida, de los cuales las hormigas restantes inician su recorrido para completar las soluciones. Gracias a este mecanismo se mejora la eficiencia de los algoritmos ACO.

Éste es un ejemplo de un algoritmo ACO combinado con un método exacto, con lo que es posible mejorar el algoritmo. Al igual que en este caso existen gran cantidad de variantes del algoritmo colonia de hormigas combinado con distintos métodos, es interesante conocer que existen estas variantes.

2.3 DISTRIBUCIÓN EN PLANTA

La distribución en planta de las naves y talleres en la industria juega un papel importante a la hora de diseñar, tanto las distintas instalaciones como el modo de producción y transporte por ellas. En función de cómo sea esta distribución se tendrán distintos tipos de cadenas de producción o de transporte de los elementos. En este apartado se van a presentar los tipos más comunes de distribución en planta.

La distribución en planta se puede definir como el proceso por el cual se determina la mejor disposición de los elementos disponibles, de forma que formen un sistema productivo por el cual alcanzar los objetivos fijados de manera eficiente y correcta.

Existen diversos tipos de distribuciones, los cuales se pueden dividir en dos grandes grupos, en función de si están orientadas al proceso o al producto, aunque también pueden ser una combinación de las dos, dando lugar a distribuciones híbridas.

En lo referente a las distribuciones por producto, generalmente se dan cuando la producción está organizada de forma continua o repetitiva, como puede ser el caso de las cadenas de montaje. En estos casos, cada operación se coloca tan cerca como sea posible de su predecesora, habitualmente a lo largo de una línea en el orden en el cual se deben utilizar, de forma que el producto es el

que recorre esta línea pasando por las distintas operaciones. Este flujo puede tener diversas formas, dependiendo de la que se adapte mejor a la situación. En la Figura 6 se muestran las más habituales.

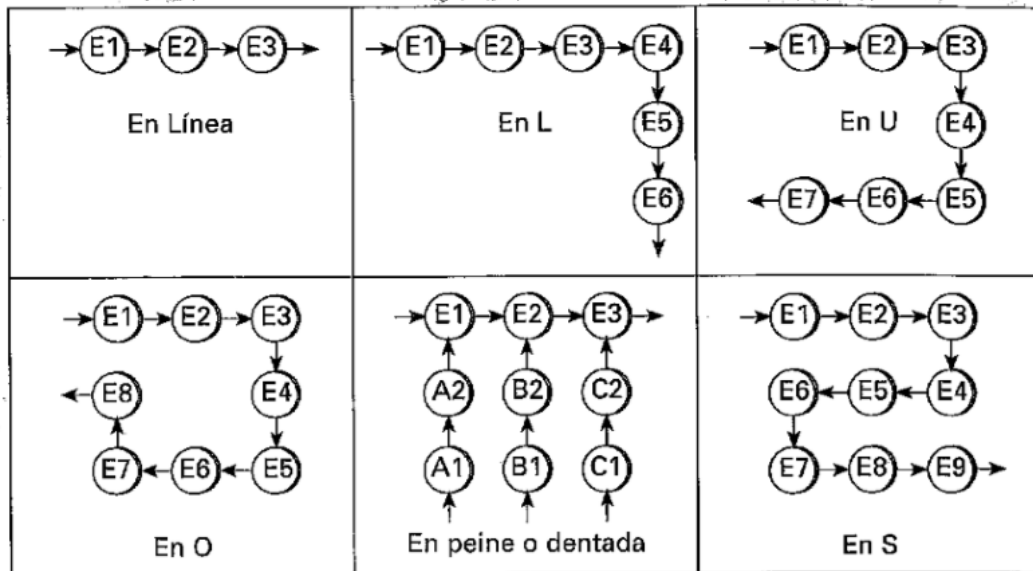


Figura 6 Tipos de distribución en planta por producto (Machuca, 1995)

Las distribuciones en planta por proceso se utilizan generalmente cuando la producción se organiza por lotes. En estos casos se agrupan el personal y los equipos que realizan una misma función, por lo que se pueden llamar distribuciones por talleres o por funciones. Además, los ítems se desplazan de un área a otra en función de los procesos que tengan que seguir, de forma que esta secuencia no es igual para todos. Un ejemplo de este tipo de distribuciones se puede ver en la Figura 7.

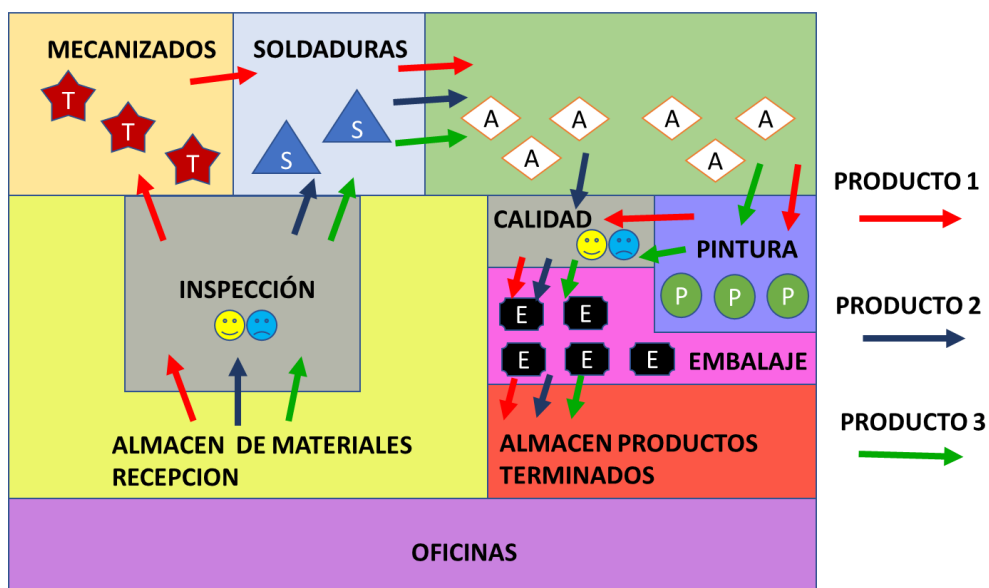


Figura 7 Distribución en planta por proceso (Fuente: Elaboración propia)

En el ámbito de las distribuciones en planta híbridas destacan las de tipo célula de trabajo, en las cuales se agrupa una serie de máquinas y trabajadores que elaboran una sucesión de operaciones sobre unos determinados ítems o familias de éstos. A continuación, se muestra un esquema de este tipo de procesos en la figura 8.

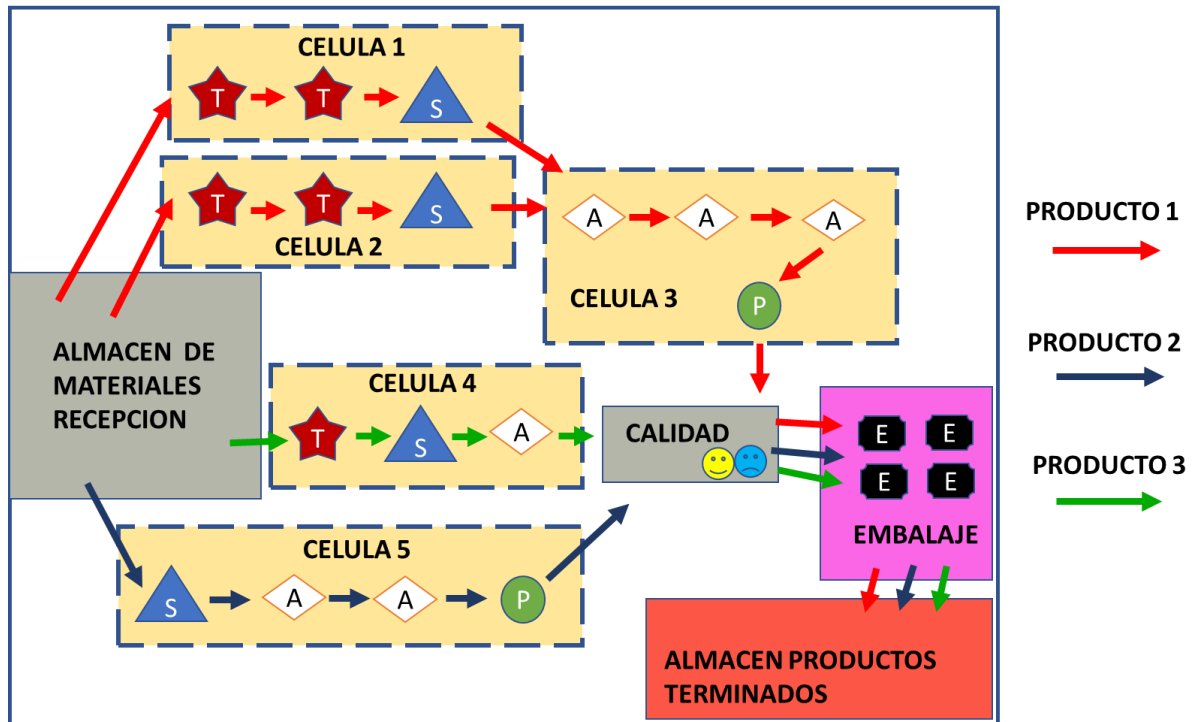


Figura 8 Distribución en planta por células de trabajo (Fuente: Elaboración propia)

Se han descrito los tipos más comunes de distribución en planta, aunque existen también otras opciones. Por ejemplo, para la distribución de almacenes es importante aprovechar al máximo la superficie (incluyendo el almacenamiento en altura) sin penalizar el acceso a los distintos elementos. (Machuca, 1995)

La aplicación del caso práctico realizado en este trabajo está enfocado principalmente a distribuciones por proceso o a otras complejas como es el caso de almacenes, y no para las distribuciones en planta por producto, ya que la propia línea define el recorrido a seguir de los elementos.

3 DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE TRANSPORTE

En este capítulo se presenta una de las aplicaciones de los algoritmos de colonia de hormigas, en este caso, para el diseño de rutas para las flotas de camiones. Este diseño se basará en un algoritmo colonia de hormigas donde tendremos las localidades por las que debe pasar el vehículo, y una localidad que será el origen y destino, de forma que se obtenga una ruta cerrada pasando por todas las localidades intermedias.

Esta aplicación está pensada para diseñar las rutas que deba seguir un camión, por ejemplo, para reparto o recogida de elementos por distintas localidades, desde una nave industrial que sea el punto de partida y también el de llegada.

3.1 INTRODUCCIÓN AL PROBLEMA

En este capítulo se va a desarrollar el diseño y desarrollo de un algoritmo de optimización por colonia de hormigas aplicado a un caso de logística de transporte. El objetivo es obtener la mejor solución, en este caso la mejor ruta posible, para ir desde una nave industrial ubicada en una localidad a una serie de localidades y volver al punto de partida.

El objetivo es obtener la ruta de menor duración, por lo que los datos que se proporcionarán serán el tiempo que se tarda en ir de cada una de las localidades a todas las demás. Estos datos se toman en tiempo ya que al ser transporte por carretera la velocidad no es constante en todo momento sino depende del tipo de vía y de las normativas de tráfico. Tampoco se toma la opción de tomar como dato la distancia puesto que puede ocurrir que dos localidades estén unidas por una distancia menor, debido al tipo de vía que las una el tiempo sea mayor al que se pueda tardar entre dos localidades más alejadas, pero con mejores comunicaciones, ya que en este segundo caso la velocidad será mayor. Por ello se considera que lo más adecuado es tomar los datos en tiempo independientemente de cual sea la distancia o la velocidad. Esto se debe a que el objetivo en este caso es minimizar el tiempo que se tarda en recorrer la ruta.

Más adelante se presentan, en primer lugar, la programación del algoritmo utilizado y, tras esto un caso sencillo para explicar detalladamente el funcionamiento. Finalmente, en este apartado se encuentra un caso más complejo, el cual se asemeja a un caso real. Los datos utilizados para cada uno de los casos se especifican en los distintos casos. Los resultados obtenidos en este capítulo se analizarán en el capítulo 5 de este trabajo.

3.2 PROGRAMACIÓN DEL ALGORITMO

Según lo expuesto anteriormente, en este capítulo, se presenta el desarrollo e implementación de la programación de los algoritmos. El desarrollo realizado en este TFG se ha basado en una serie de programas empleando MATLAB y Excel.

Para crear el orden que debe seguir la ruta se ha creado un programa en Matlab con un algoritmo basado en la colonia de hormigas, el cual se detallará más adelante; y una hoja de cálculo Excel, que se utilizará tanto para introducir los datos necesarios, como para presentar los resultados obtenidos de manera clara.

3.2.1 Programación de ACO para camiones

En este apartado se va a presentar la programación utilizando Matlab de un algoritmo ACO (Ant Colony Optimization, optimización por colonia de hormigas) para resolver un problema concreto de creación de rutas para camiones, aunque se puede extrapolar a necesidades similares.

3.2.1.1 Introducción al algoritmo ACO para camiones

Se utiliza un programa desarrollado en Matlab, que proporciona el orden de la ruta a seguir, y cuyo criterio es tardar el menor tiempo posible en pasar por todas las localidades indicadas. Como datos se proporciona una tabla con el tiempo en minutos que se tarda en ir de cada una de las localidades a todas las demás. Esto se presenta con mayor detalle en el apartado relativo a la introducción de datos.

3.2.1.2 Objetivos del programa

El programa que se desarrolla debe cumplir los siguientes requisitos:

- Crear un método de entrada de datos sencillo y eficiente, lo cual se consigue con el apoyo de un fichero Excel.
- El punto de origen también será el de destino final, por lo que la ruta será circular.
- Es necesario que la ruta pase por todos los puntos (dichos puntos equivalen a las localidades).

3.2.1.3 Descripción del código

Para la programación del algoritmo se parte de un algoritmo colonia de hormigas (ACO) que proporciona la mejor ruta cerrada, en la cual el origen y el destino es el mismo punto, de una serie de puntos proporcionados por medio de coordenadas cartesianas (x, y). (Seyedali, 2020).

Este algoritmo está formado por una función principal denominada “ACO” desde la cual se va llamando a distintas funciones que realizan pequeñas partes del algoritmo, esto permite tener un código más simple en la función principal.

Los datos de los puntos por los que debe pasar el elemento, cuyo camino óptimo queremos obtener, se proporcionan a partir de una función denominada “createGraph”. En ella se definen dos vectores, uno denominado X en el que se indican las coordenadas en el eje x de los puntos indicados y otro vector llamado Y donde se dan las coordenadas en el eje y, de forma que cada punto está definido por $X(n), Y(n)$, siendo n el punto indicado.

En el algoritmo del que partimos se obtienen rutas cerradas pasando una vez por cada uno de los puntos dados. Además, permite representar gráficamente los distintos caminos posibles, y el mejor recorrido realizado hasta el momento.

A continuación, se presenta el algoritmo diseñado y desarrollado para este trabajo. Partiendo del algoritmo anterior, hemos modificado las funciones necesarias para introducir nuestros datos, los cuales no están en coordenadas cartesianas sino en una única variable de tiempo, y para obtener los resultados necesarios para este caso de rutas de camiones.

Las características de este caso son las siguientes:

- El punto de origen y final es el mismo, dando como resultado una ruta cerrada.
- Los datos no son introducidos por medio de coordenadas cartesianas, sino que se indican el tiempo, en minutos, que se tarda en ir de cada uno de los puntos a todos los demás.

En la figura 9, se muestra un esquema indicando el orden que sigue el programa y donde intervienen las distintas funciones que serán explicadas más adelante.

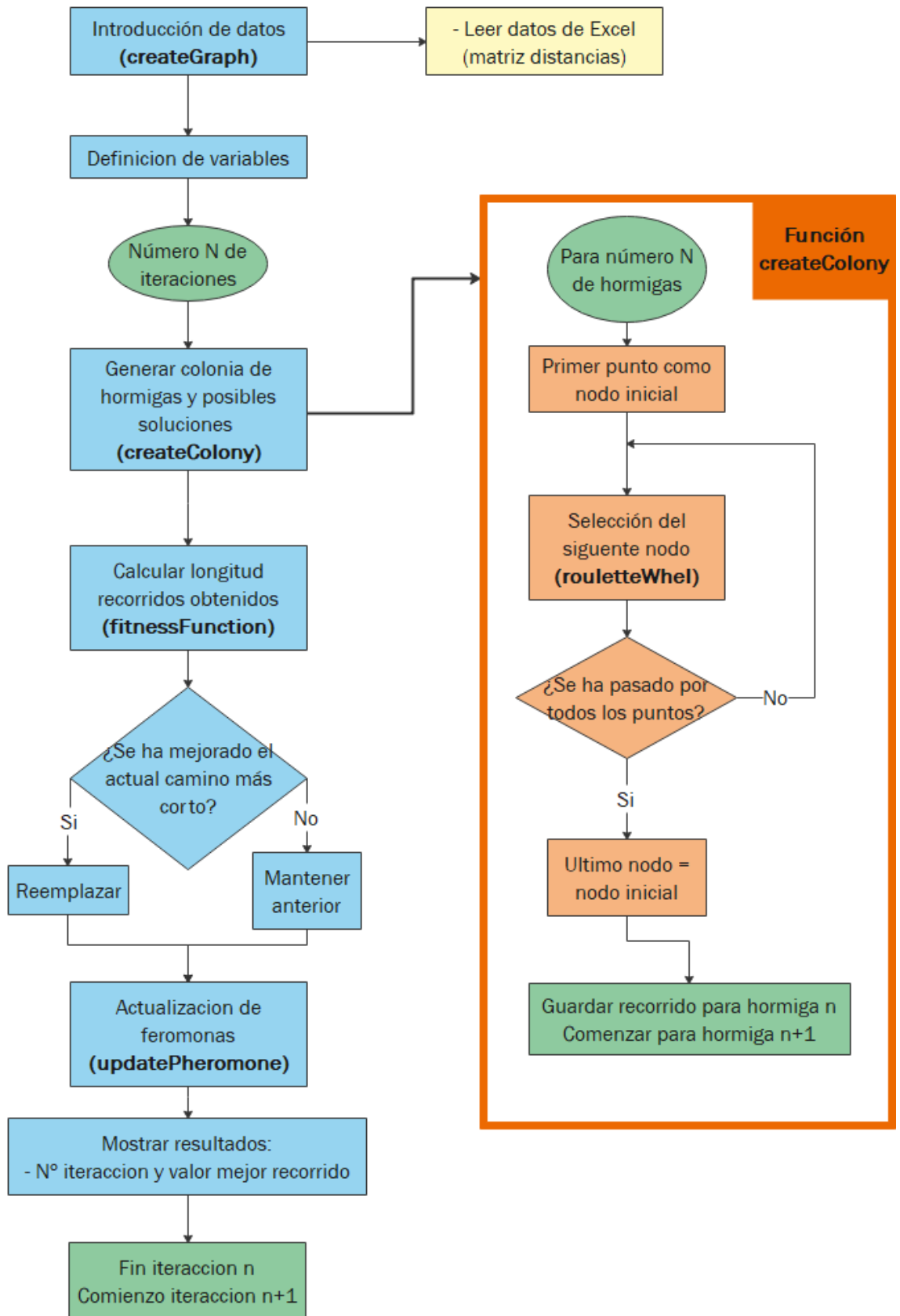


Figura 9 Esquema del programa ACO para el caso de transporte (Fuente: Elaboración propia)

A continuación, se explica cada una de las funciones que componen el código del algoritmo ACO.

La función principal, que contiene el denominado “*main*” del programa es denominada “ACO”. En esta función lo primero que tenemos son las instrucciones para borrado de valores previos antes de la ejecución, de forma que no se comentan errores por tener alguna variable residual de ejecuciones anteriores.

A continuación, se llama a la primera función en ejecutarse, denominada “*createGraph*”.

En “*createGraph*” se proporcionan los datos necesarios para la ejecución del algoritmo. En primer lugar, se introduce un vector en el cual se indica el número de nodos (localidades en nuestro caso) existentes. A continuación, se indican los datos correspondientes al tiempo que se tarda en ir de cada uno de los puntos a todos los demás, guardando estos datos en la matriz *graph.edges*. Todos estos datos se leen por medio de la función *xlsread*.

Volviendo a la función “ACO”, tenemos la definición de los distintos parámetros iniciales que utiliza el algoritmo. Entre otros aquí es donde se definen el número máximo de iteraciones y el número de hormigas para las que se ejecutará el algoritmo. Esta elección se basa en lo que se recomienda en la bibliografía para este tipo de algoritmos y, una vez seleccionados estos valores por medio del ensayo-error, ejecutando el programa varias veces para comprobar que estos valores sean adecuados.

A partir de este momento comienza la parte principal del programa. En primer lugar, se definen las variables *bestFitness* y *bestTour*, las cuales contendrán la mejor solución; siendo la primera el tiempo de recorrido más corto y la segunda un vector del orden de los puntos por los que pasa. A partir de este punto se abre un bucle *for* donde se encuentra el resto de la función ACO. Este bucle se ejecuta el número máximo de iteraciones definido. Dentro de este bucle *for* se ejecutan una serie de funciones, la primera de ellas es “*createColony*”.

La función “*createColony*” es una de las más importantes de este programa, ya que es en la que se generan las rutas posibles. Está compuesta por dos bucles *for*. El primero que aparece se ejecuta el número de hormigas definido, de forma que se obtienen tantas rutas como hormigas. A continuación, antes de entrar en el siguiente bucle, se indica que el nodo inicial es el punto 1, a partir de este momento se ejecuta el siguiente *for* un número de veces igual al número de nodos menos 1. En este bucle en primer lugar se eliminan de los posibles nodos siguientes los que ya han sido utilizados. Con los nodos restantes creamos un vector “P” pasando los nodos a valores entre 0 y 1 para obtener de manera aleatoria el siguiente nodo utilizando la función “*rouletteWheel*” que se describe a continuación. Al terminar este primer bucle

se indica que el último punto es igual al primero, y finalmente se cierra el primer *for*. En todo este proceso se han guardado todas las opciones de ruta que se han generado en una variable que será utilizada más adelante.

La función “*rouletteWheel*” mencionada, genera un número aleatorio entre 0 y 1 y busca dentro de los nodos posibles el que tenga el valor más cercano e inferior, utilizando los valores del vector “*P*” descrito anteriormente. Esta función devuelve el siguiente nodo a la función “*createColony*”.

Tras ejecutar la función “*createColony*” se vuelve a la función principal “*ACO*” donde se calcula la aptitud de las hormigas ejecutando la función “*fitnessFunction*”.

En esta función “*fitnessFunction*” se calcula la longitud del camino de cada una de las hormigas de la iteración. Ya en “*ACO*” se utilizan estos datos para comprobar si alguno de estos recorridos es menor al mejor recorrido obtenido hasta el momento, y si es así éste pasará a ser el nuevo mejor recorrido.

Esto se hace de la siguiente manera:

- Primero se busca el valor mínimo entre todas las hormigas y se guarda el índice al que corresponde
- Tras esto se comprueba si es menor al valor denominado “*bestFitness*” que es el menor valor obtenido hasta el momento, si es menor se asigna este valor como el nuevo “*bestFitness*” y el recorrido correspondiente se guarda como “*bestTour*”
- Después de esto se asigna a “*colony.queen.tour*” el mejor recorrido y a “*colony.queen.fitness*” el mejor valor. Estas variables serán las que contengan la mejor solución obtenida con el algoritmo colonia de hormigas al finalizar su ejecución.

El siguiente paso es actualizar el valor de “*tau*”, que es la variable utilizada para indicar el valor utilizado para la evaporación de feromonas, por medio de la función “*updatePheromone*”. La evaporación va borrando progresivamente las feromonas existentes en cada uno de los caminos, de forma que si no se vuelven a utilizar en varias iteraciones se borran completamente.

Por último, se muestran los resultados obtenidos en la ejecución del programa, mostrando en la ventana de comandos de MatLab el texto “*Iteration #*” seguido del número de iteración y “*Shortest length =*” seguido del valor de “*colony.queen.fitness*”. Este mensaje indica el número de la iteración que acaba de finalizar y el valor de “*colony.queen.fitness*” que contiene la suma del recorrido más corto obtenido hasta el momento, el cual no tiene por qué ser necesariamente de la última iteración realizada.

Como se ha indicado, en la parte principal de la función “ACO” se ejecuta el número de iteraciones definido. A medida que se ejecuta cada iteración, aparece en la pantalla de comandos el número de iteración que acaba de finalizar, seguida de la menor longitud obtenida hasta el momento, lo cual nos permite saber en cada instante el avance de la ejecución del programa.

3.2.2 Entrada de datos al programa principal

Se utilizará una hoja de datos Excel en la que es sencillo introducir los datos de cada uno de los casos, para que luego, estos datos sean pasados al programa principal (en Matlab) expuesto en el apartado anterior.

Para la lectura de datos de un fichero Excel, en primer lugar, es necesario tener guardado el archivo en la misma ubicación en la que se encuentran los archivos del programa de Matlab que se estén utilizando para evitar errores, e indicar desde MatLab la ubicación del archivo.

A continuación, para asignar valores a un vector o una matriz, se utiliza la función “`xlsread('nombre del Excel', 'rango de datos')`” por ejemplo `xlsread('datos','B2:B25')`, de la cual se obtendrán los datos del archivo Excel denominado “datos” que están en la columna B desde la fila 2 a la fila 25.

De esta forma, para modificar los datos y realizar los distintos casos, en la programación en Matlab únicamente será necesario cambiar la información del fichero que se desea leer (nombre y rango de datos), en lugar de introducir los datos en la programación, lo cual resulta más tedioso y menos eficiente.

3.2.2.1 Hoja de cálculo de datos

La hoja de cálculo que utilizaremos en este trabajo se denominará “Datos_camiones.xlsx” la cual está formada por dos hojas.

En la primera hoja, llamada “DATOS” es la que se utilizará para introducir los datos necesarios al programa principal. Contiene una tabla formada por dos partes. La primera fila corresponde a un índice asignando a las localidades para las cuales se aplicará el algoritmo. A partir de la segunda fila de la tabla, lo cual corresponde a la segunda parte de la misma, aparecen los datos de tiempo que se tarda en recorrer cada tramo. En la tabla 1 tenemos un ejemplo de esta tabla.

DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE TRANSPORTE

INDICE	1	2	3	4	5	6	7	8
DESTINO ORIGEN	inicio (Zaratán)	Arroyo de la encomienda	Villanubla	Santovenia de Pisuerga	Laguna de Duero	Cabezón de Pisuerga	Simancas	Cistérniga
inicio (Zaratán)	0	9	13	19	14	18	10	18
Arroyo de la encomienda	11	0	19	20	9	17	7	14
Villanubla	10	16	0	23	21	22	17	25
Santovenia de Pisuerga	16	17	20	0	17	9	18	16
Laguna de Duero	15	11	23	20	0	17	15	13
Cabezón de Pisuerga	17	18	22	11	17	0	20	16
Simancas	10	5	17	19	11	19	0	16
Cistérniga	18	14	25	18	13	15	17	0

Tabla 1 Tiempo de trayecto entre localidades (Fuente: Elaboración propia)

En dicha tabla se muestran las 8 localidades próximas a Valladolid por las que debe transcurrir el trayecto, junto con el tiempo que se tarda en ir de cada una de ellas a las demás, tomando la primera columna como origen y la primera fila como destino. Como se puede ver en la diagonal de la tabla aparecen ceros ya que en estos casos el origen y el destino coinciden.

Además, la primera fila presenta un índice, que indica el número que se asigna a cada localidad, ya que el programa trabaja con este índice, en lugar de los nombres de las localidades.

Estos datos han sido obtenidos utilizando la aplicación Google.maps, tomando el tiempo que se tarda en ir en transporte por carretera, el cual se ha indicado en minutos. Se puede ver que en algunos casos es distinto el tiempo que se tarda en ir de una localidad a otra y en volver esto puede deberse a que no se utiliza exactamente el mismo recorrido por cuestiones relativas a los distintos trazados de las vías. A continuación, se muestra una situación que puede generar estas diferencias en la figura 10.

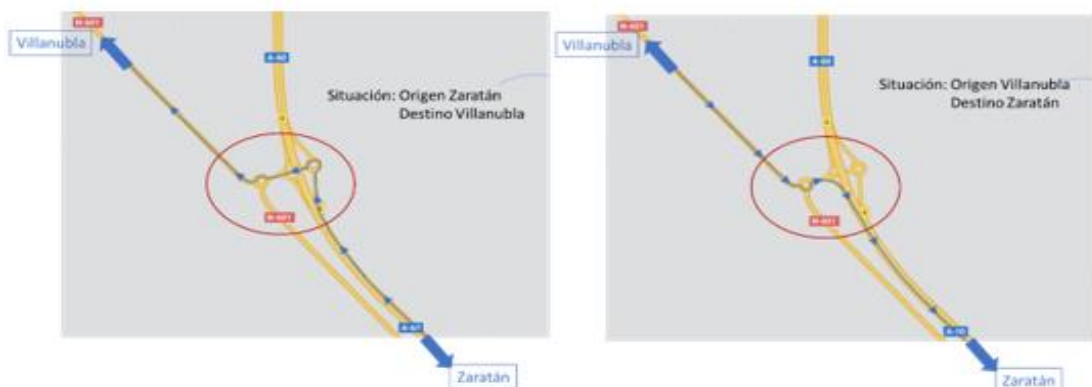


Figura 10 Ejemplo situación de diferencias en tiempos (Fuente: Elaboración propia)

En esta situación, la cual únicamente es un ejemplo de las causas de pueden producir las diferencias en los tiempos a la ida y a la vuelta de la misma localidad, se puede ver que en el primer caso tras tomar la salida se debe ir hacia una rotonda la cual permite cruzar sobre la autovía por la que veníamos A-60 y de esta forma llegar a la otra rotonda que permite la incorporación a la nacional N-601. En el segundo caso, al venir en sentido contrario, esta operación es más simple, ya que únicamente se tiene que tomar una rotonda y, tras esto, la incorporación a la autovía A-60 es inmediata. Esto es un pequeño ejemplo de casos que pueden generar diferencias de tiempos, pero puede haber muchos más casos debido a la diversidad en los tipos de vías y a las normativas de tráfico, como sentidos obligatorios, vías de un único sentido, límites de velocidad...

La segunda hoja de este archivo excel, denominada "RESULTADOS" no es necesaria realmente sino que se propone utilizarla para pasar de los resultados obtenidos del programa en Matlab, el cual proporciona un vector numérico con el orden a seguir, a una tabla con el orden a seguir indicado por el nombre de las localidades que queda mucho más claro para utilizar los datos después que el vector proporcionado por el programa principal. En la figura 11 se puede ver un ejemplo de esta hoja, que consta de dos partes, una primera tabla a la derecha donde se indican las localidades y su número asignado, y otra tabla a la izquierda con el orden obtenido. La columna F, en la cual aparecen los nombres de las localidades se rellena automáticamente utilizando la función *BUSCARV* de excel.

	A	B	C	D	E	F	G
1							
2		Indice	Localidades		Resultados ACO Matlab	Ruta obtenida (nombre localidades)	
3		1	inicio (Zaratán)		1	inicio (Zaratán)	
4		2	Arroyo de la encomienda		8	Cistérniga	
5		3	Villanubla		5	Laguna de Duero	
6		4	Santovenia de Pisuerga		2	Arroyo de la encomienda	
7		5	Laguna de Duero		7	Simancas	
8		6	Cabezón de Pisuerga		3	Villanubla	
9		7	Simancas		4	Santovenia de Pisuerga	
10		8	Cistérniga		6	Cabezón de Pisuerga	
11					1	inicio (Zaratán)	
12							

Figura 11 Hoja "RESULTADOS" fichero Excel (Fuente: Elaboración propia)

Como se puede ver, la ruta comienza y termina en la misma localidad, la cual corresponde al índice 1, ya que en esta posición se indica la localidad que se utilizará como origen.

3.3 CASO BÁSICO

En este apartado se va a presentar un caso sencillo, o experimento, que muestra la ruta de vehículos únicamente con 4 puntos, de forma que se puedan hacer todas las combinaciones posibles y obtener los resultados de cada una de ellas para, posteriormente comparar los resultados obtenidos con la solución proporcionada por el algoritmo ACO. Por otra parte, se va a ejecutar el programa descrito anteriormente, para comprobar que los resultados obtenidos son válidos.

3.3.1 Datos del experimento

Para este caso se van a tomar 4 ciudades, Valladolid, Zamora, Palencia y León, seleccionando Valladolid como origen de la ruta, ciudad a la que también se debe volver al final de ésta.

A continuación, se muestra la tabla 2 con los datos de tiempo que se tarda en ir desde cada una de las ciudades a las demás. Estos datos se han obtenido utilizando la aplicación “Google.maps” ya que para este experimento es suficiente precisión contar con los datos aproximados de dicha fuente.

INDICE	1	2	3	4
DESTINO ORIGEN	inicio (Valladolid)	Zamora	Palencia	León
inicio (Valladolid)	0	67	37	102
Zamora	69	0	95	92
Palencia	36	89	0	85
León	102	92	88	0

Tabla 2 Datos caso básico camiones (Fuente: Elaboración propia)

En la figura 12, se muestra una imagen del mapa destacando las cuatro localidades seleccionadas.



Figura 12 Mapa con las ciudades seleccionadas para el caso simple (fuente: Elaboración propia)

3.3.2 Método exacto

Para este apartado se han comprobado los resultados que se obtienen al calcular el tiempo que se tarda en cada una de las combinaciones posibles, tomando en todos los casos Valladolid (ciudad 1) como origen y como punto final para cerrar la ruta. En el siguiente esquema, mostrado en la figura 13, se muestran las posibles combinaciones.

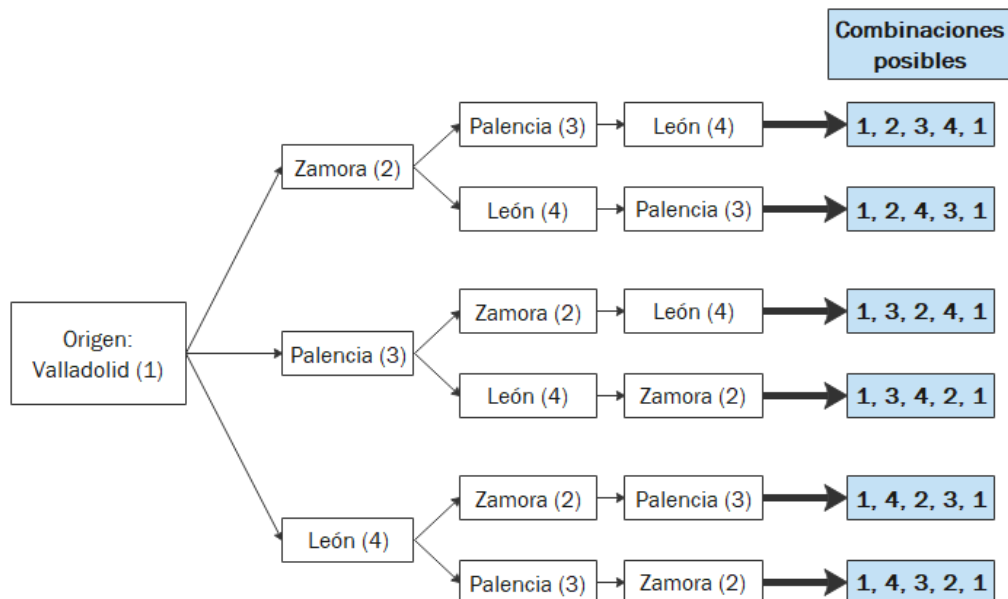


Figura 13 Esquema de combinaciones posibles (Fuente: Elaboración propia)

Como se puede ver en el esquema anterior, existen 6 combinaciones posibles para realizar una ruta cerrada desde Valladolid que pase por las otras 3 ciudades.

A continuación, se muestran los resultados de cada una de las iteraciones en la tabla 3, en la cual se ha resaltado en la que se ha hallado la mejor solución.

Combinación	Resultado (min)
1, 2, 3, 4, 1	349
1, 2, 4, 3, 1	285
1, 3, 2, 4, 1	320
1, 3, 4, 2, 1	283
1, 4, 2, 3, 1	327
1, 4, 3, 2, 1	348

Tabla 3 Resultados caso básico (Fuente: Elaboración Propia)

Como se puede observar, se obtiene como solución óptima para este caso la combinación 1, 3, 4, 2, 1; la cual corresponde a la ruta Valladolid, Palencia, León, Zamora, Valladolid. Para esta combinación se obtiene un tiempo de 283 minutos.

3.3.3 Método ACO

En este apartado se muestra la utilización del método colonia de hormigas por medio del programa creado en Matlab para resolver el problema anterior, y así comprobar si el resultado es el mismo que se ha obtenido en el caso anterior.

Para un problema tan sencillo no es necesario aplicar un método metaheurístico, ya que éstos están orientados para situaciones en las cuales el número de combinaciones posibles es tan elevado que no es posible comprobar todas las combinaciones una a una.

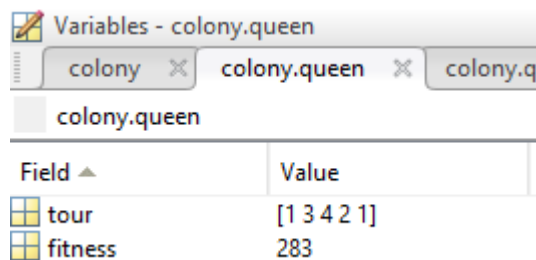
En primer lugar, se modifican las líneas correspondientes a la entrada de datos en el programa de Matlab, lo que se hace en la función "createGraph".

Para este caso, al ser un caso simple, se utilizan únicamente 4 hormigas y 15 iteraciones, puesto que solo hay 6 combinaciones posibles. En casos complejos, como el que se presentará más adelante, estos valores serán más elevados, puesto que las combinaciones posibles serán muchas más.

A continuación, se muestran los resultados obtenidos tras ejecutar el programa, que se pueden ver en las figuras 14 y 15.


```
Command Window
Iteration #1 Shortest length = 285
Iteration #2 Shortest length = 283
Iteration #3 Shortest length = 283
Iteration #4 Shortest length = 283
Iteration #5 Shortest length = 283
Iteration #6 Shortest length = 283
Iteration #7 Shortest length = 283
Iteration #8 Shortest length = 283
Iteration #9 Shortest length = 283
Iteration #10 Shortest length = 283
Iteration #11 Shortest length = 283
Iteration #12 Shortest length = 283
Iteration #13 Shortest length = 283
Iteration #14 Shortest length = 283
Iteration #15 Shortest length = 283
fx >>
```

Figura 14 Número de iteraciones y resultado. (Fuente: Elaboración propia)



Field	Value
tour	[1 3 4 2 1]
fitness	283

Figura 15 Variables "colony.queen.tour" y "colony.queen.fitness" (Fuente: Elaboración propia)

3.3.4 Resultados y conclusiones del experimento

Como se puede comprobar, los resultados obtenidos son los mismos en los dos casos, de lo que se deduce que el algoritmo creado proporciona resultados correctos y válidos.

Además, cabe indicar que el algoritmo se ha ejecutado 6 veces, con distintos valores de hormigas (3, 4, 5,10) y distintas iteraciones (10, 15, 20,100) y los resultados obtenidos, han sido en todos los casos los mismos, aunque no en todas las ocasiones llega al óptimo en la misma iteración, sino que esto ha sido diferente en todos los casos. Esto se debe en parte a la propia aleatoriedad del algoritmo y en parte al número de hormigas seleccionado. Se puede relacionar con el número de hormigas utilizado ya que cuantas más hormigas se utilizan más posibles combinaciones comprueba en cada iteración. Esto además coincide con el echo de que a medida que se aumentaba el valor del número de hormigas la iteración en la que llegaba a la solución óptima disminuía.

3.4 CASO COMPLEJO

En este caso se va a utilizar el programa descrito en este capítulo para realizar un caso más complejo, que podría ser un caso real, ya que el número de combinaciones posibles es elevado, por lo cual, es interesante utilizar un método metaheurístico.

3.4.1 Datos del caso complejo

En este apartado se va a utilizar el algoritmo para obtener la ruta a seguir entre 10 localidades próximas a Valladolid, situando el origen de la ruta en Zaratán.

A continuación, se muestra la tabla 4 indicando el tiempo, en minutos, que se tarda en ir de cada una de las localidades a todas las demás. Estos datos se han obtenido utilizando la aplicación “Google.maps”.

INDICE	1	2	3	4	5	6	7	8	9	10
DESTINO ORIGEN	inicio (Zaratán)	Arroyo de la encomienda	Villanubla	Santovenia de Pisuegra	Laguna de Duero	Cabezón de Pisuegra	Simancas	Cistérniga	Herrera de Duero	Fuensaldaña
inicio (Zaratán)	0	9	13	19	14	18	10	18	20	13
Arroyo de la encomienda	11	0	19	20	9	17	7	14	15	15
Villanubla	10	16	0	23	21	22	17	25	26	8
Santovenia de Pisuegra	16	17	20	0	17	9	18	16	21	15
Laguna de Duero	15	11	23	20	0	17	15	13	12	18
Cabezón de Pisuegra	17	18	22	11	17	0	20	16	21	16
Simancas	10	5	17	19	11	19	0	16	17	13
Cistérniga	18	14	25	18	13	15	17	0	16	19
Herrera de Duero	19	16	27	23	14	20	19	15	0	23
Fuensaldaña	12	13	10	17	18	17	14	20	23	0

Tabla 4 Datos caso complejo rutado de camiones (Fuente: Elaboración propia)

A continuación, se muestra un mapa de la zona, señalando las localidades seleccionadas en la figura 16.



Figura 16 Mapa con las localidades seleccionadas (Fuente: Elaboración propia)

En este experimento las combinaciones posibles ascienden a 3.628.800, por lo que, en este caso, no resulta abordable calcular todas las posibilidades. A medida que se aumenta el número de localidades, el número de combinaciones aumenta significativamente. Debido a esto, para casos reales con gran cantidad de localidades se plantea utilizar el algoritmo ACO descrito en este capítulo.

3.4.2 Desarrollo del caso

Al igual que en el apartado anterior, para ejecutar el programa en Matlab es necesario adaptar el código para la introducción de los datos desde el fichero Excel, para indicar el nombre del fichero y el rango de datos.

El algoritmo se ha ejecutado múltiples veces modificando las variables número de hormigas y número de iteraciones, para comprobar cómo se comporta este programa cuando se modifican las variables principales.

A continuación, se muestra en la figura 16 un ejemplo de cómo proporciona Matlab los datos, correspondiente a mejor resultado obtenido, y en la tabla 5 todos los resultados obtenidos. Se ha ejecutado el algoritmo 10 veces para cada uno de los casos.

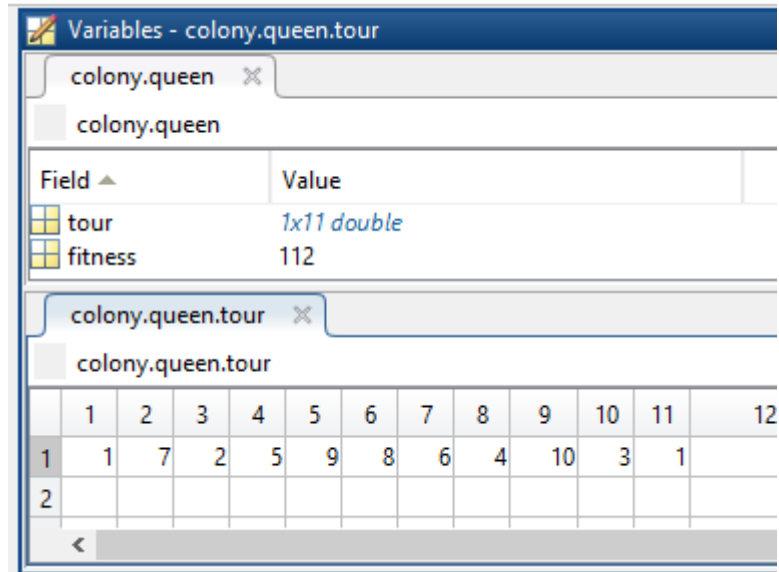


Figura 17 Mejores datos obtenidos (Fuente: Elaboración propia)

Nº hormigas	Nº iteraciones	colony.queen.fitness en cada ejecución (min)										Mejor resultado (min)	Peor resultado (min)
		1	2	3	4	5	6	7	8	9	10		
5	20	131	130	128	127	131	129	127	128	125	127	125	131
5	100	120	125	119	115	123	124	122	119	125	121	115	125
5	300	122	120	112	117	120	121	121	121	117	114	112	122
20	20	115	117	123	128	117	127	122	124	123	130	115	130
20	100	120	115	114	115	117	117	121	112	115	122	112	122
20	300	114	115	117	115	112	115	115	117	115	114	112	117
40	400	112	114	112	115	114	115	112	114	115	112	112	115

Tabla 5 Resultados obtenidos (Fuente: Elaboración propia)

En la tabla se muestra únicamente el resultado obtenido para la variable *colony.queen.fitness*, lo que corresponde al tiempo que se tarda en realizar la mejor ruta obtenida. Se muestran estos datos, que son los resultados relevantes obtenidos a analizar con más detalle en el Capítulo 5.

Como se puede observar, a medida que aumentan tanto el número de hormigas como el número de iteraciones, va mejorando el resultado obtenido por el algoritmo. Sin embargo, debido a la aleatoriedad de los caminos, no se obtienen los mismos resultados cada vez que se ejecuta el algoritmo. Por ejemplo, en algunas ocasiones pese a tener más iteraciones que la ejecución anterior, se ha obtenido un resultado peor, aunque sigue siendo favorable

Además, cabe mencionar que no en todas las ocasiones se llega al mejor resultado en la misma iteración, sino que esto varía cada vez que se ejecuta el

programa, aunque tenga los mismos valores en las variables, lo cual es debido a la aleatoriedad del algoritmo.

Finalmente, se muestra la mejor solución obtenida y el orden de las localidades correspondientes. Para ello se utiliza el fichero Excel descrito, el cual se muestra en la figura 18.

	A	B	C	D	E	F	G
1							
2		DATOS			Resultados ACO Matlab	Ruta obtenida (nombre localidades)	
3		Índice	Localidades		1	inicio (Zaratán)	
4		1	inicio (Zaratán)		7	Simancas	
5		2	Arroyo de la encomienda		2	Arroyo de la encomienda	
6		3	Villanubla		5	Laguna de Duero	
7		4	Santovenia de Pisuerga		9	Herrera de Duero	
8		5	Laguna de Duero		8	Cistérniga	
9		6	Cabezón de Pisuerga		6	Cabezón de Pisuerga	
10		7	Simancas		4	Santovenia de Pisuerga	
11		8	Cistérniga		10	Fuensaldaña	
12		9	Herrera de Duero		3	Villanubla	
13		10	Fuensaldaña		1	inicio (Zaratán)	
14							
15							

Figura 18 Ruta obtenida en la mejor solución (Fuente: Elaboración propia)

4 DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES

En este capítulo se va a exponer una de las aplicaciones del algoritmo colonia de hormigas para el diseño de rutado de AGV. Este diseño se basará en un algoritmo colonia de hormigas donde tendremos los puntos por los que debe pasar el AGV de manera obligatoria para realizar sus funciones y los puntos por los que es posible pasar, eliminando todos los puntos que tengan obstáculos por los que sea imposible circular. Esta aplicación está pensada para diseñar los distintos recorridos tanto en la fase de diseño de una nueva instalación como en fases de explotación, en las cuales, dependiendo de las funciones asignadas deba realizar recorridos diferentes.

En primer lugar, se va a exponer brevemente qué es un AGV y los distintos tipos de AGV que existen,

4.1 AGV (Automated Guided Vehicles- Vehículos de guiado automático)

Un AGV es un vehículo de guiado automático. Este tipo de elementos es cada vez más utilizado en la industria, pues permite automatizar flujos de manera poco invasiva. Otros métodos de automatización de flujos son los tapices o transportadores, pero este tipo de elementos generalmente son muy voluminosos e impiden que se pueda pasar por esta superficie con cualquier tipo de vehículo, lo que generalmente dificulta el acceso principalmente para tareas de mantenimiento.

A continuación, se van a presentar brevemente los diferentes tipos de AGV que existen, siguiendo varios criterios.

Los AGVs se pueden clasificar según el modo de transportar la carga, pudiéndose distinguir dos tipos:

- La carga se coloca directamente sobre el AGV, ésta debe posicionarse centrada y equilibrada en la máquina.
- Utilizando una mesa de arrastre, de forma que la carga se coloca sobre este dispositivo de manera centrada y equilibrada y el AGV se sitúa bajo la mesa, se fija a ésta por medio de uno o varios pines y desplaza la mesa.

Utilizar una mesa de arrastre o el propio AGV para transportar la carga dependerá de las necesidades del proceso y de las características de dicha carga.

Los AGVs también se pueden clasificar en función de las diferentes formas y tamaños que tienen para poder adaptarse a las diferentes utilidades, por lo que en función de las necesidades es necesario seleccionar el tipo que más se adapte a estas. También

es posible incluir accesorios en estos dispositivos AGV como mesas de elevación, mesas de rodillos, etc. En la figura 19 se pueden ver dos ejemplos.

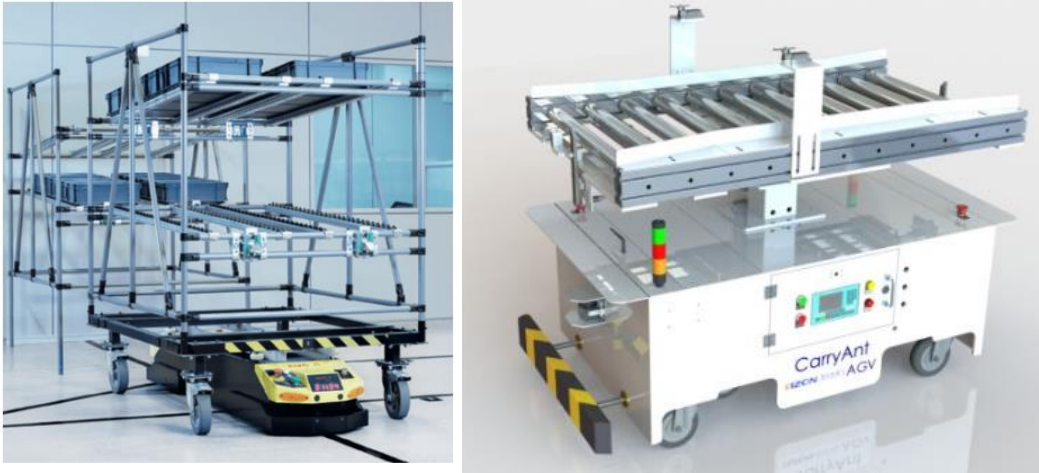


Figura 19 Accesorios para AGVs (Kivnon, 2020 y Cizon, 2020)

Otra de las características de las que disponen los vehículos AGV son los equipos de comunicación que pueden incluir para transmitir información o comunicarse con otros elementos del sistema.

Otra de las clasificaciones más importantes de los AGV es en función del tipo de guiado, pues dependiendo de la utilidad, del ambiente en el que se utilicen, o del tipo de recorrido que deba realizar, existen varios tipos, éstos se verán con más detalle en el siguiente apartado.

(ATRIA INNOVATIO, 2020)

Tipos de guiado de AGVs

Según el tipo de guiado existen diversas opciones, a continuación, se indican los métodos de guiado más comunes para este tipo de vehículos

Filoguiado

En este caso el AGV se desplaza siguiendo un hilo conductor que se ha instalado bajo el suelo. Pese a ser el sistema más utilizado inicialmente, es el que menos flexibilidad permite, ya que las rutas se limitan a las rutas creadas con los hilos instalados. En la figura 20 se puede ver un esquema.

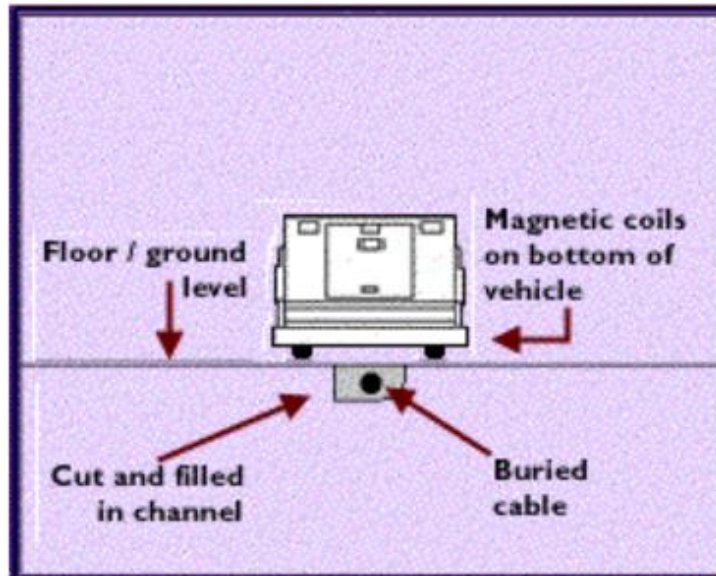


Figura 20 Representación filoguiado (ATRIA INNOVATION, 2020)

Guiado magnético

Este tipo de guiado consiste en que el AGV sigue un camino marcado en el suelo por medio de una línea magnética. En este caso, para modificar la trayectoria es necesario modificar estas bandas colocadas en el suelo. En la figura 21 se puede ver un ejemplo.



Figura 21 Figura AGV por guiado magnético (Cizon, 2020)

Optoguiado

El AGV se desplaza guiado por medio de una tira de espejo catadióptrico detectado por medio de fotocélulas instaladas en el AGV. En este caso, la modificación de rutas se realiza instalando tiras de espejo por el techo a nuevas zonas a las que se desea llegar y modificando los movimientos en el AGV.

Visión artificial

Mediante un sistema de visión artificial 360° el vehículo obtiene, procesa y analiza el entorno, interpolando su localización mediante la identificación de marcas visuales.

Guiado laser

En este caso, el AGV conoce su posición en el mapa de la instalación que tiene en la memoria, por medio de barridos que realiza con una unidad láser giratoria, de forma que identifica la mayor parte de reflectores posibles. Se sitúan espejos catadióptricos en posiciones estratégicas de la instalación como puntos de referencia. La ventaja de este método es la facilidad de cambiar la ruta, pues no es necesario modificar los espejos de la instalación sino únicamente modificar el software. En la figura 22 se puede ver una representación esquemática.

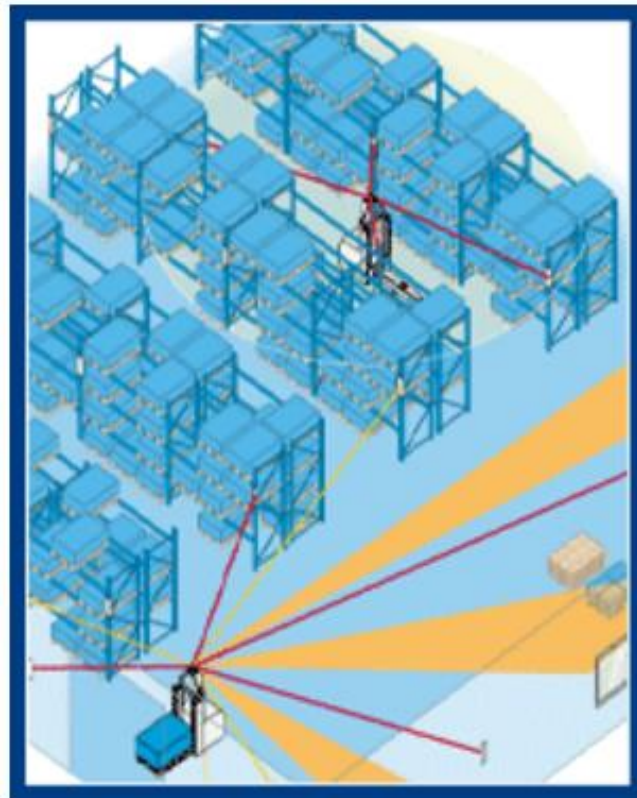


Figura 22 Representación guiado láser (ATRIA INNOVATION, 2020)

Navegación por mapeo

Está basado en la tecnología SLAM (Simultaneous Localization And Mapping). El AGV circula de manera totalmente libre y sin elementos de orientación sobre el suelo. El sistema de mapeo de navegación alcanza la capacidad de construir un mapa de forma incremental del entorno, y utilizar ese mapa para determinar la ubicación exacta del AGV.

Además de las citadas también existen sistemas de guiado que utilizan marcas visuales o dispositivos RFID (Radio Frequency Identification) Esto son unos dispositivos que se colocan generalmente en las intersecciones para facilitar la navegación.

Por último, podemos diferenciar los AGV en función del sentido de la marcha y la capacidad de rotación. Pueden ser unidireccionales, bidireccionales o con

movimiento 360°, tipo Quad (movimiento en todas direcciones). Esta característica es importante pues de ella dependerá el radio de maniobra, que es la capacidad que tiene un AGV para cambiar de dirección o sentido. Esto se verá con mayor detalle en los siguientes apartados.

(ATRIA INNOVATIO, 2020)

4.2 PROGRAMACION DEL ALGORITMO

Según lo presentado anteriormente, en este apartado se muestra la programación del algoritmo correspondiente.

Para crear los recorridos de los AGV por una distribución en planta, se utilizarán un programa en Matlab con el algoritmo ACO utilizado y una hoja de cálculo Excel por medio de la cual se definirán los datos de entrada. En este apartado se explica en detalle el funcionamiento de cada desarrollo realizado.

4.2.1 Programación de ACO para AGVs

A continuación, se presenta la programación de un algoritmo ACO para resolver un problema concreto, en este caso el guiado de AGVs.

4.2.1.1 Introducción

Se utiliza un programa desarrollado en Matlab por medio del cual se genera el orden de los puntos que debe pasar el AGV para recorrer todos los puntos obligatorios. En este programa se tiene un punto de origen, un punto de final y una serie de puntos obligatorios. Además, para unir estos puntos se cuenta con puntos posibles que se deben utilizar, para unir los puntos obligatorios y crear así el recorrido a realizar.

En este apartado se explica con detalle el programa en Matlab utilizado, el modo en el que se introducen los datos necesarios para su ejecución y los resultados que se obtienen.

4.2.1.2 Objetivo del programa

- El programa que se desarrolla debe cumplir los siguientes requisitos:
 - Crear un método de entrada de datos ágil y eficiente.
 - Establecer un origen y un destino para el elemento que debe circular (que en el algoritmo está representado por hormigas)
 - Establecer en la programación del algoritmo que los puntos por los que se pueda pasar puedan ser opcionales, u obligatorios (por ejemplo, por los que debe pasar el elemento para realizar una operación).
 - Permitir que el elemento que circule pueda repetir puntos opcionales, ya que en los talleres en industria es habitual tener calles sin salida en los cuales se debe salir por el mismo sitio por el que se entra, supone un problema ya que el elemento no podría regresar.

4.2.1.3 Descripción del Código.

Para la programación del algoritmo se parte del mismo algoritmo colonia de hormigas (ACO) que en el apartado anterior, el cual proporciona la mejor ruta cerrada, en la cual el origen y el destino es el mismo punto, de una serie de puntos proporcionados por medio de coordenadas cartesianas (x, y). (Mathworks, 2020).

Como se expuso en el capítulo anterior, este algoritmo está formado por una función principal (ACO) y varias funciones auxiliares. Esto permite tener un código más simple en la función principal, ya que desde ésta se llama a las funciones auxiliares que realizan distintas partes del algoritmo.

La entrada de datos de los puntos por los que debe pasar el elemento, cuyo camino óptimo queremos obtener, se realiza por medio de una función en la cual se definen dos vectores, definiendo los puntos por medio de coordenadas cartesianas (x,y).

En el algoritmo del que partimos se obtienen rutas cerradas pasando una vez por cada uno de los puntos dados. Además, permite representar gráficamente los distintos caminos posibles, y el mejor recorrido realizado hasta el momento.

A continuación, se presenta el algoritmo diseñado y desarrollado para este trabajo. Partiendo del algoritmo anterior, hemos modificado las funciones necesarias para obtener los resultados que se consideran necesarios para definir los rutados de los AGV. Además, se indican las características de este programa para poder obtener los resultados adecuados en el caso de definir rutados de AGVs.

Las características de nuestro caso son las siguientes:

- El tipo de recorridos parten de un punto de origen hacia un punto de final, pasando por una serie de puntos por los que es obligatorio pasar.
- Todos dichos puntos están contenidos en un conjunto mucho mayor, los cuales son los puntos posibles que se pueden utilizar al desarrollar el algoritmo, pero que, si no son necesarios, no formarán parte del camino final óptimo.

A continuación, en la figura 23, se muestra un esquema indicando el orden que sigue el programa y donde intervienen las distintas funciones.

DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES

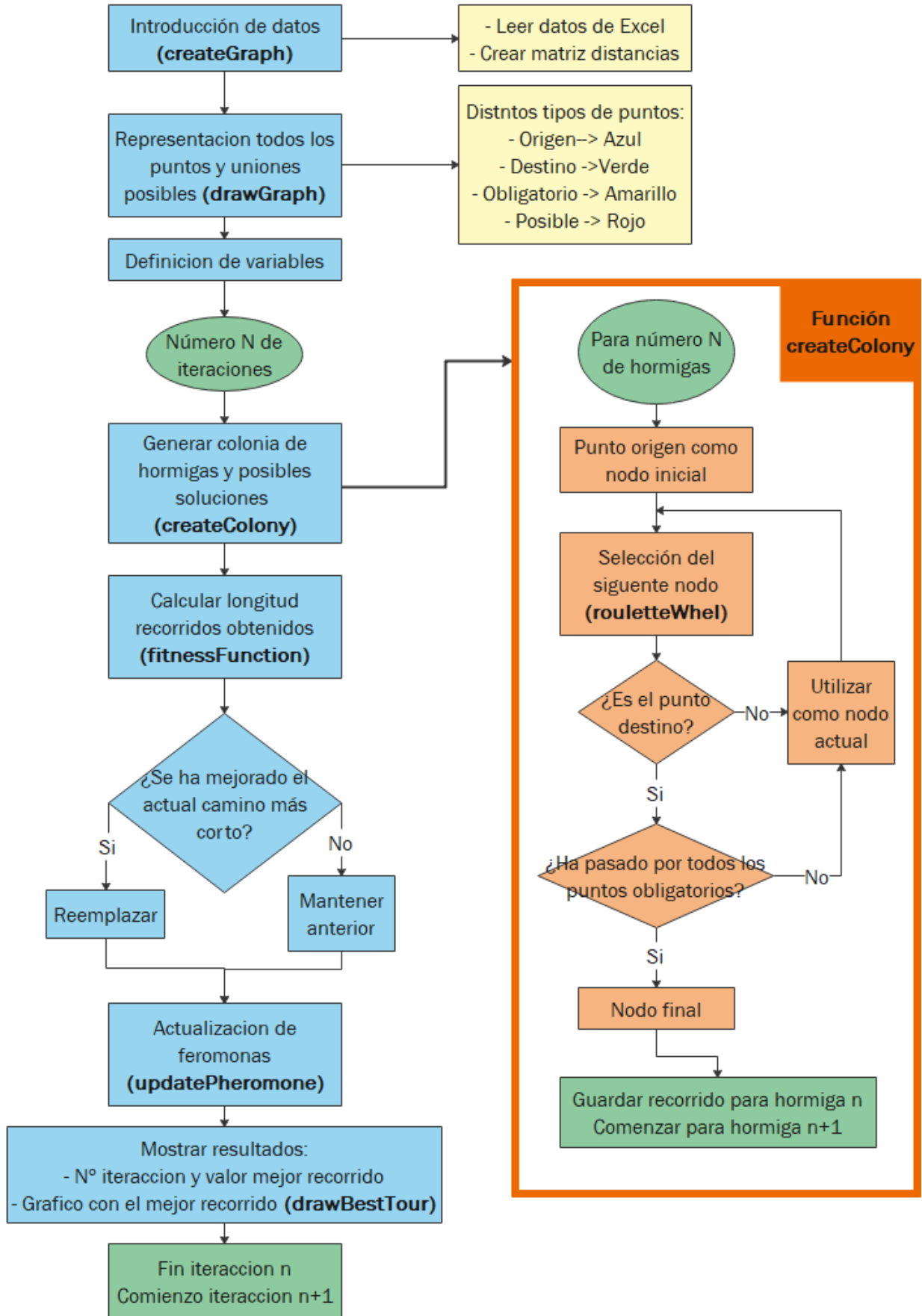


Figura 23 Esquema del programa ACO para el caso de almacenes (Fuente: Elaboración propia)

Puesto que el programa es similar, en método, al utilizado en el caso de los camiones del capítulo anterior, únicamente se explicarán detenidamente las funciones que se diferencian o que no aparecían en el caso anterior, omitiendo la explicación de las funciones que son iguales, puesto que ya fueron expuestas.

La función principal, que contiene el denominado “*main*” del programa es denominada “ACO”. En esta función lo primero que tenemos son las instrucciones para borrado de valores previos antes de la ejecución, de forma que no se comentan errores por tener alguna variable residual de ejecuciones anteriores. A continuación, se llama a la primera función en ejecutarse, denominada “createGraph”, esta función es diferente al caso anterior ya que los datos de entrada son distintos.

En “createGraph” en primer lugar se definen los puntos de los vectores X e Y, los cuales en el código original eran los puntos por los que debía pasar el recorrido y en nuestro algoritmo se trata de los puntos posibles por los que puede pasar, aunque únicamente es obligatorio pasar por los puntos inicial, final, y los que se definan como obligatorios más adelante. Estos vectores se obtienen leyendo los datos de un archivo Excel que se explica más adelante. En esta función también se almacenan, en la matriz *graph.edges*, la distancia existente entre todos los puntos, dos a dos.

Volviendo a la función “ACO”, a continuación, se llama a la función “drawGraph”, la cual no aparece en el caso anterior, para representar gráficamente los puntos definidos y todas las posibles uniones entre ellos. Esta función inicialmente representaba las uniones de cada uno de los puntos con todos los demás, de forma que un punto de un extremo se podía unir con los puntos de los extremos contrarios.

Para nuestro programa vamos a considerar que desde un punto únicamente se puede ir a los adyacentes. Para la parte de la representación de la gráfica, vamos a dibujar únicamente los puntos que estén a una distancia menor o igual a uno, esto se realiza para que el recorrido no pueda ir de un extremo a otro, puesto que al hacer esto podría pasar a través de obstáculos. Se restringe que únicamente se pueda mover a los puntos que están al lado para asegurar que el recorrido es posible. Esto se realizará también en otras funciones para seleccionar el siguiente nodo, ya que en esta parte exclusivamente se realiza la representación gráfica.

En la representación de todas las gráficas, en este TFG, los puntos importantes se destacan mediante un cambio de color, de forma que el punto de inicio pasa a ser azul, el punto de final pasa a ser verde y los puntos obligatorios amarillos, quedando los puntos posibles en color rojo.

Volviendo a la función principal, a continuación de lo ya expuesto, tenemos la definición de los distintos parámetros iniciales que utiliza el algoritmo.

A partir de este momento comienza la parte principal del programa. En primer lugar, se definen las variables *bestFitness* y *bestTour*, las cuales contendrán la mejor

solución; siendo la primera el tiempo de recorrido más corto y la segunda un vector del orden de los puntos por los que pasa. A partir de este punto se abre un bucle *for* donde se encuentra el resto de la función ACO. Este bucle se ejecuta el número máximo de iteraciones definido. Dentro de este bucle *for* se ejecutan una serie de funciones, la primera de ellas es “createColony”.

La función “createColony” es una de las más importantes de este programa. Puesto que es distinta al caso anterior y debido a su importancia en el programa, se expondrá detalladamente. En primer lugar, se definen algunas variables auxiliares que se utilizarán en esta función. Además, es aquí donde se lee desde Excel la información de los puntos por los que es obligatorio pasar. A continuación, se abre un bucle *for* que se ejecuta tantas veces como número de hormigas se han definido. Primero se asigna el primer punto como nodo uno, siendo éste el principio del recorrido. Tras esto tenemos otro bucle *for* que comienza en 2 y se ejecuta 1000xnúmero de nodos veces, aunque se le ha dado un valor muy elevado al que no llegará prácticamente nunca, puesto que en el momento que el algoritmo haya pasado por todos los puntos obligatorios se ejecutará una función *break* por medio de la cual se sale del bucle *for* y se continúa ejecutando el programa a partir de éste.

A continuación, lo primero que se hace es asignar el último nodo seleccionado como nodo actual, tras esto se realiza una búsqueda de los que pueden ser el próximo nodo, excluyendo de las posibilidades todos aquellos nodos que estén alejados del nodo actual con un valor superior a 1. Esto nos permite que desde un nodo únicamente se pueda ir a nodos adyacentes y no a cualquier punto. Tras esto, se utilizan las constantes para determinar las probabilidades de cada nodo y se eliminan de los nodos posibles tanto el nodo actual como los nodos excluidos en el paso anterior. Con los nodos que nos quedan disponibles creamos un vector “P” pasando los nodos a valores entre 0 y 1 y ejecutamos la función “rouletteWheel” (explicada en el caso anterior) para seleccionar el siguiente nodo, añadiendo este a la lista de nodos visitados en la última posición.

Siguiendo con la función “createColony”, a continuación, se actualiza una variable denominada *numNodo* en la cual se tiene el número de nodos por los que ha pasado el algoritmo. Tras esto, se evalúa si el que se ha seleccionado como siguiente nodo coincide con el nodo final. En caso negativo la función vuelve a ejecutar el bucle *for*. En el caso de que se corresponda con el último punto, entramos en un *if* para comprobar si se han cumplido todas las condiciones y en caso positivo, el camino termina aquí. En cuanto a las condiciones, en primer lugar, se comprueba si se ha pasado ya por todos los puntos obligatorios que se han extraído al principio de la función desde Excel. Para ello se utiliza un bucle *for* para comprobar si se ha pasado por cada uno de los puntos obligatorios, contando por cuantos puntos obligatorios ha pasado la función, independientemente del orden. Si se ha pasado por todos los puntos, y teniendo en cuenta que el nodo actual es el nodo final se utiliza un *break* para salir de los bucles abiertos, pasando a ejecutarse la siguiente hormiga.

Habitualmente todas las hormigas terminarán su recorrido con las condiciones descritas cumplidas. En caso negativo se ejecutará el código de buscar nodo hasta que se cumpla el número de veces definido en el bucle *for*. En caso de que esto ocurriese, el valor obtenido al sumar el recorrido será elevado, por lo que no sustituirá a la mejor hormiga obtenida hasta el momento.

Una vez ejecutada la función “createColony” se vuelve a la función principal “ACO”.

A partir de este punto, el programa es similar al caso anterior por lo que se expone brevemente. En primer lugar, se llama a la función “fitnessFunction” en la cual se calcula la longitud de todos los caminos obtenidos. Estos resultados se comparan con el guardado en la variable “bestFitness” y en caso de que alguno sea menor que este se almacena como el nuevo recorrido más corto, guardando también el recorrido correspondiente en la variable “bestTour”. Finalmente se asignan estos valores a “colony.queen.tour” y a “colony.queen.fitness”.

A continuación, se utiliza la función “updatePheromone” Finalmente, igual que en el caso anterior, se muestran los resultados obtenidos. mostrando en la ventana de comandos de MatLab el texto “Iteration #” seguido del número de iteración y “Shortest length = ” seguido del valor de “colony.queen.fitness”.

En este caso, también se muestra gráficamente el mejor camino obtenido hasta el momento, en el gráfico llamado Best tour (the queen). En éste, se muestra gráficamente el recorrido cuya suma da el menor valor obtenido hasta el momento. Esto se hace por medio de la función “drawBestTour” en la cual se representan en primer lugar todos los puntos posibles, y después se representan las uniones de dichos puntos correspondientes al mejor recorrido (*colony.queen.tour*). En esta gráfica, al igual que en el caso de “drawGraph”, para facilitar la comprensión del gráfico se van a destacar los puntos importantes cambiándolos de color, siguiendo el mismo código de colores que en el caso anterior.

Como se ha expuesto anteriormente, la parte principal de la función “ACO” y por tanto, del programa, se ejecuta el número de iteraciones definido por medio de un bucle *for*. A medida que se ejecuta cada una de las iteraciones, se van actualizando los gráficos, de feromonas y de mejor recorrido, además de indicarse en la ventana de comando el número de iteraciones y la mejor longitud obtenida hasta el momento.

4.2.2 Programa para la entrada de datos

Se trata de un programa para facilitar la entrada de datos al algoritmo en MatLab, creada a partir de la hoja de cálculo Excel.

4.2.2.1 Introducción al programa

Se ha creado un programa en Excel por medio del cual se puede pasar de una implantación dibujada en un CAD a puntos por coordenadas cartesianas (x,y) interpretables por la función ACO de MatLab.

Ello permite superponer una imagen en una cuadrícula con tamaño de celda unitaria de radio de maniobra del AGV donde podremos indicar los puntos de paso posibles y obligatorios por los que debe circular.

Se han creado para este TFG una serie de macros que mediante pulsadores embebidos en la hoja de cálculo vacían la hoja y crean los nuevos puntos en coordenadas que puedan ser exportadas al algoritmo en MatLab. Esto se expone con mayor detalle más adelante.

4.2.2.2 *Objetivos de este programa*

- Diseñar una herramienta en Excel que facilite la entrada de Datos a la aplicación en MatLab.
- Establecer la forma a través de la cual el usuario, que trabaje con la herramienta, pueda establecer los puntos por los cual el AGV puede y debe pasar.

4.2.2.3 *Descripción del programa*

Como se indicó en el capítulo anterior, para pasar datos de un archivo Excel a Matlab se utiliza la función `xlsread`. Por medio de esta función se pasan los valores de los vectores X e Y en la función `createGraph`. Se debe indicar el nombre del fichero Excel en el que tengamos los datos y las casillas que se deben leer. Para este propósito, se ha creado una hoja Excel llamada `"Datos_ACO"`, el cual utilizaremos durante todo el programa.

También se utiliza el mencionado fichero Excel para exportar los datos de los puntos obligatorios de paso, utilizando la función de leer desde una hoja de cálculo, la función `"createColony"`, como se ha indicado anteriormente.

Al utilizar este fichero Excel se facilita la modificación de datos en el programa de Matlab, ya que, en lugar de indicar los datos completos, únicamente sea necesario modificar las líneas del programa en las cuales se llama a la función `"xlsread"` indicando la nueva ubicación de los datos en el Excel del cual se van a leer. Para facilitar aún más esta tarea, siempre se utilizará el fichero de Excel `"Datos_ACO"` creado para este propósito, de forma que solamente se modifiquen ligeramente las casillas en las que se incluyen los datos.

A continuación, se presenta la hoja de cálculo en Excel `"Datos_ACO"` que se ha creado para este algoritmo.

Esta hoja de cálculo está formada por 3 hojas:

- La primera, llamada `"Datos"`, contiene la información que se leerá por MatLab, indicada anteriormente (puntos de origen, final, paso obligatorio y posibles), además de algunas funciones (activadas mediante botones) para proporcionar toda esta información en el formato deseado.

DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES

- La segunda hoja, denominada “Mapa”, se trata de una cuadrícula donde se indicarán los puntos posibles y los puntos obligatorios del recorrido, además del punto inicial y final. Esta información se trata por medio de unas macros que se explican más adelante.
- Por último, la hoja 3 llamada “Auxiliar” contiene una serie de datos necesarios para realizar las operaciones de la hoja anterior.

Para la obtención de los datos necesarios se comienza a modificar el fichero Excel en la hoja denominada “Mapa” indicando los distintos tipos de puntos que tenemos, codificando con un 3 el punto inicial, con un 4 el final, con un 2 los puntos por los que es obligatorio pasar y con un 1 los puntos opcionales por los que es posible pasar, pero no es necesario, dejando en blanco las demás celdas.

Esta hoja de cálculo está preparada para cambiar de color las celdas en base a las diferentes posibilidades, de forma que se ve más claramente la característica de cada uno de los puntos. Esto se ha hecho utilizando un formato condicional para todas las celdas de la hoja de cálculo con las diferentes posibilidades mencionadas, de forma que las celdas que contengan un 1 pasan a ser de color verde, las que tengan un 2 de color azul, la inicial amarilla y la final naranja.

A continuación, vemos un ejemplo de esta hoja figura 24 y un resumen de los códigos utilizados en la Tabla 6.

1	Punto posible, se puede pasar pero no es obligatorio
2	Punto obligatorio, es necesario pasar por él
3	Punto de inicio del recorrido
4	Punto final del recorrido

Tabla 6 Leyenda codificación hoja "MAPA" en "Datos_ACO". (Fuente: Elaboración propia).

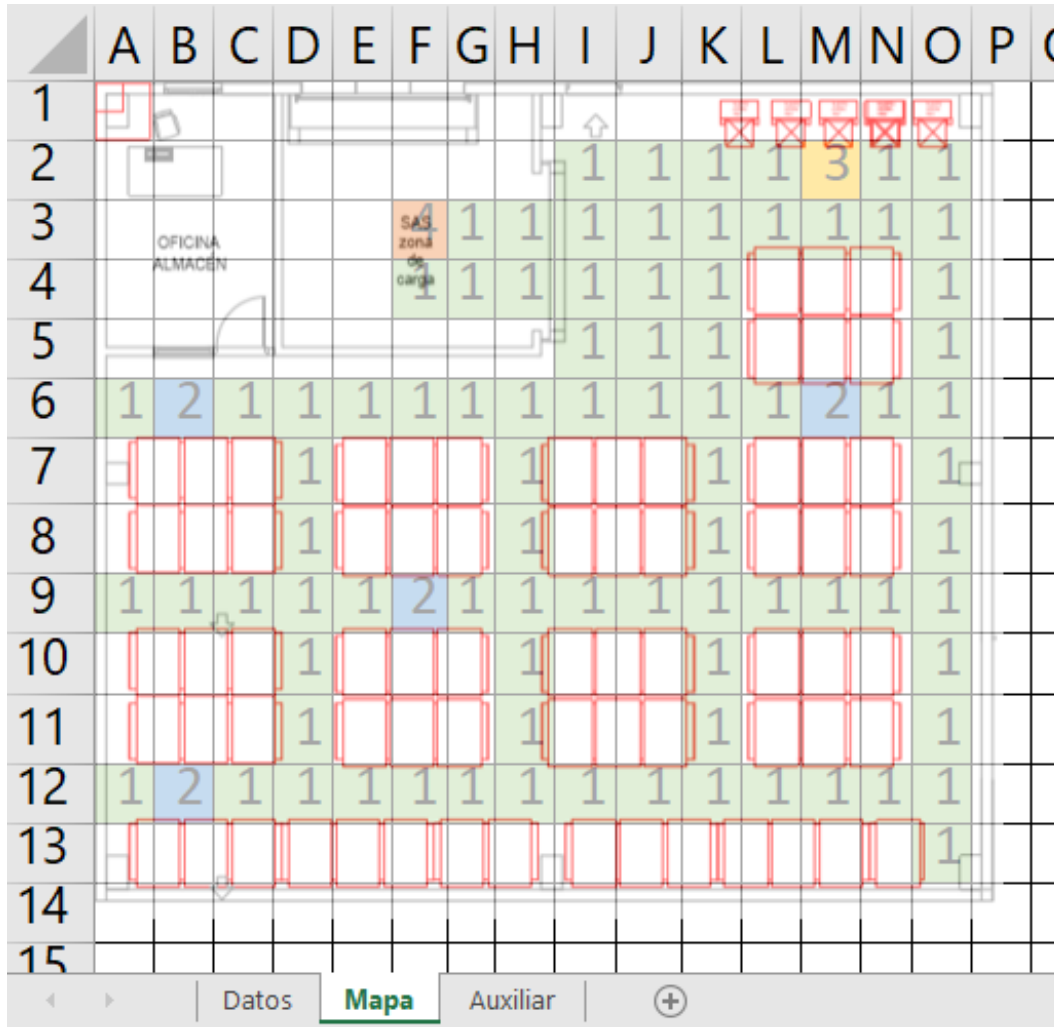


Figura 24 Ejemplo de la hoja "Mapa" (Fuente: Elaboración propia)

Además de rellenar estos datos es necesario indicar en la hoja "Auxiliar" el tamaño del "mapa", indicando longitud y anchura como vemos en la Figura 25. Esto es relevante pues son datos que se utilizarán más adelante para preparar los puntos.

	A	B	C
1			
2			
3	Longitud nave	10	
4	Ancho nave	10	
5			
6			

Figura 25 Ejemplo de datos en la hoja "Auxiliar" (Fuente: Elaboración propia)

Una vez se han rellenado los datos de estas dos hojas pasamos a la hoja principal, denominada "Datos" para transformar los puntos introducidos en "Mapa" al formato que se le pasará más tarde a MatLab. Para ello hay una serie de botones que ejecutan unas macros que se explican a continuación.

DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES

En primer lugar, se utiliza el botón “Borrar puntos” para eliminar los datos que pueden quedar de casos anteriores. Este botón llama a *borrar_puntos()* la cual al ejecutarse deja en blanco todas las casillas a partir de la segunda fila correspondientes a las columnas B,C,E,G y H, las cuales son las utilizadas en esta hoja para rellenar los datos, Se han seleccionado estas columnas para que estéticamente la hoja quede más limpia, dejando columnas en blanco para separar distintos conceptos.

Tras esto se utilizan los otros dos botones que tenemos disponibles, en primer lugar, el denominado “Puntos posibles” y después de éste el llamado “Puntos obligatorios”. El pulsado de estos botones es importante hacerlo en este orden, puesto que el segundo utiliza los datos que se generan con el primero.

Al utilizar “Puntos posibles” se llama a la función creada denominada *vectores_XY()*, la cual pasa los puntos de la hoja “Mapa” a las coordenadas XY que más tarde serán leídas por MatLab. *Vectores_XY()* está formado por 3 partes diferenciadas, aunque tienen la misma estructura, Dicha estructura es la siguiente.

Se trata de dos bucles *for* anidados, lo cual es un método de recorrer una matriz. Estos bucles van desde el punto inicial, correspondiendo en este caso a la fila 1 y columna 1, hasta la longitud y la anchura que hemos indicado en la hoja “Auxiliar” definida anteriormente. Una vez que vamos recorriendo la matriz de puntos, comprobamos, en el primer caso si se trata del punto inicial, identificado por un 3. En el segundo caso si es un punto posible, incluyendo los obligatorios, por lo que se comprueba las casillas que tienen un 1 o un 2. Y en el último caso se busca el punto final, identificado con un 4. Cuando se cumple la condición que se está buscando en cada caso, siendo un 3 para el punto inicial un 4 para el punto final o un 1 o 2 para el caso de los puntos probables o posibles, se cumple la condición de un *if* por lo que se pasa a escribir las coordenadas X e Y del punto en cuestión en la hoja “Datos” en las columnas correspondientes, además de estas dos se escribe la columna índice al lado de estas.

En la figura 26, se muestra el primer bloque del código descrito. Como se ha indicado anteriormente, los otros dos bloques son similares a éste, excepto en las condiciones en el *if*.

```

Sub vectores_XY()

Dim i, j, k, ind As Integer

k = 2
ind = 1

'buscar punto de inicio, codificado como 3
For i = 1 To Hoja3.Cells(3, 2)
    For j = 1 To Hoja3.Cells(4, 2)

        If (Hoja2.Cells(i, j) = 3) Then

            Hoja1.Cells(k, 3) = i
            Hoja1.Cells(k, 4) = j
            Hoja1.Cells(k, 2) = ind

            k = k + 1
            ind = ind + 1

        End If
    Next
Next
    
```

Figura 26 Fragmento de código (Fuente: Elaboración propia)

Cuando pulsamos el botón “Puntos posibles” vemos que se rellenan las columnas coordenadas X, coordenadas Y e índice. Además de estas 3, también se rellena la columna auxiliar que utilizaremos para identificar los puntos obligatorios y una casilla en la que se indica el número total de puntos.

A continuación, se muestra un ejemplo de cómo queda el archivo en la figura 27

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
		aux	coordenada X	coordenada Y	Indice		X	Y	puntos obligatorios			longitud													
1		213	2	13	1				0			90													
2		29	2	9	2				0																
3		210	2	10	3				0																
4		211	2	11	4				0																
5		212	2	12	5				0																
6		214	2	14	6				0																
7		215	2	15	7				0																
8		37	3	7	8				0																
9		38	3	8	9				0																
0		39	3	9	10				0																
1		310	3	10	11				0																
2		311	3	11	12				0																
3		312	3	12	13				0																
4		313	3	13	14				0																
5		314	3	14	15				0																
6																									

Figura 27 Ejemplo de la hoja "Datos" tras pulsar "Puntos Posibles" (Fuente: Elaboración propia)

Una vez realizado todo lo anterior, se pasa al último paso, en el que se pulsa el botón de “Puntos obligatorios”, para crear el vector de los puntos por los que es necesario pasar, y que será leído por MatLab. Este botón ejecuta una macro similar al de “Puntos posibles”, donde se recorre la matriz en la cual se indican los distintos puntos. Cuando se encuentra con un punto identificado como obligatorio, indicado con un 2, escribe las coordenadas X e Y de manera similar al caso anterior en las casillas de las columnas F y G.

Una vez se obtienen los puntos obligatorios, se busca en qué posición se encuentran para pasar a MatLab el vector de los puntos obligatorios. Para crear este vector se utiliza una columna auxiliar en la que se colocan el valor de la coordenada X y de la coordenada Y seguidos, de forma que para X=2, Y=5 el valor de aux=25, a continuación se utiliza la función *BUSCARV* para buscar la posición que ocupan cada uno de los puntos obligatorios.

En la figura 28 se presenta un ejemplo de cómo queda la hoja “Datos”.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1		aux	coordenada X	coordenada Y	Indice		X	Y	puntos obligatorios			longitud												
2		14	1	4	1		3	10	14			49												
3		16	1	6	2		4	2	15															
4		17	1	7	3		6	8	26															
5		18	1	8	4		10	5	46															
6		21	2	1	5			-																
7		23	2	3	6			-																
8		24	2	4	7			-																
9		26	2	6	8			-																
10		28	2	8	9			-																

Figura 28 Ejemplo de la hoja "Datos" en excel "Datos_ACO" (Fuente: Elaboración propia)

En la Figura se puede observar en color azul las columnas y botón correspondientes a la parte de puntos posibles, en rosa las columnas y botón correspondientes a los puntos obligatorios y en amarillo los datos auxiliares y el botón de borrar.

4.2.2.4 Solapamiento plano con hoja Excel

En la hoja de cálculo se considera un mapeo de cuadros que representa la superficie a recorrer, en la cual se hace coincidir el tamaño de dicha cuadrícula con el tamaño del radio de maniobra del AGV que se vaya a utilizar, aproximándolo a un cuadrado, de forma que el AGV al pasar por los distintos cuadrados supere los límites. Con esto se consigue que si se realizan varias filas de cuadrados por un mismo pasillo varios AGVs podrían pasar simultáneamente siempre que no coincidan en la misma casilla, formando distintos carriles. Además de esta forma se asegura que el AGV en ningún caso va a chocar con algo siempre que no haya nada invadiendo los espacios destinados a casillas de paso de AGVs.

El radio de maniobra es una característica importante de los AGVs, la cual indica la capacidad que tienen para cambiar de dirección o sentido, o, lo que es lo mismo, su maniobrabilidad.

Existen AGVs que requieren mayor rango de distancia para girar, y otros que pueden rotar sobre sí mismos, reduciendo las dimensiones de la cuadrícula hasta las dimensiones físicas del AGV.

El radio de maniobra dependerá principalmente de las posibilidades de giro de las ruedas de los vehículos. En función del diseño de vehículo seleccionado se puede ver las diferentes características en las siguientes imágenes. En la figura 29 vemos un AGV rectangular que puede rotar sobre su eje, de forma que el radio de maniobra es igual a la longitud del lado mayor. En la figura 30 se puede ver un AGV que necesita mayor espacio para girar, puesto que requiere de hacer maniobras para cambiar de sentido, lo cual, hace que el radio de maniobra sea mayor respecto a las dimensiones del AGV.

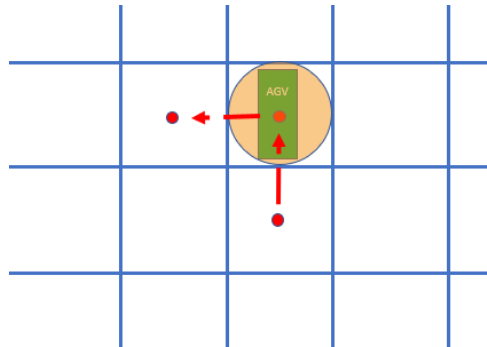


Figura 29 AGV que puede rotar sobre sí mismo (Fuente: Elaboración propia)

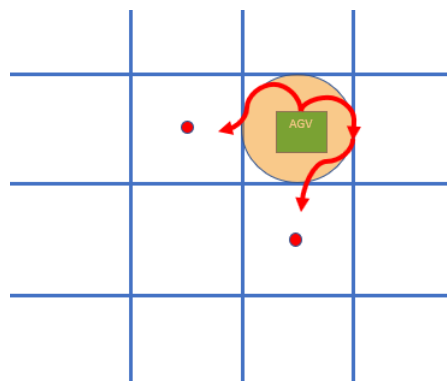


Figura 30 AGV que requiere más espacio que sus propias dimensiones para girar (Fuente: Elaboración propia)

Es posible que sea diferente la distancia necesaria para girar en los distintos sentidos, pero para garantizar que las dimensiones de la cuadrícula sigan siendo válidas una vez el AGV se ha desplazado se optará por la distancia de mayor longitud necesaria para definir el radio de la cuadrícula.

En función del Radio de la circunferencia inscrita en cada cuadro, mostrado en las imágenes anteriores, se definirá el cuadrado unitario del enrejado que será

empleado en la entrada de datos como un parámetro de configuración, esto se puede ver en la figura 31.

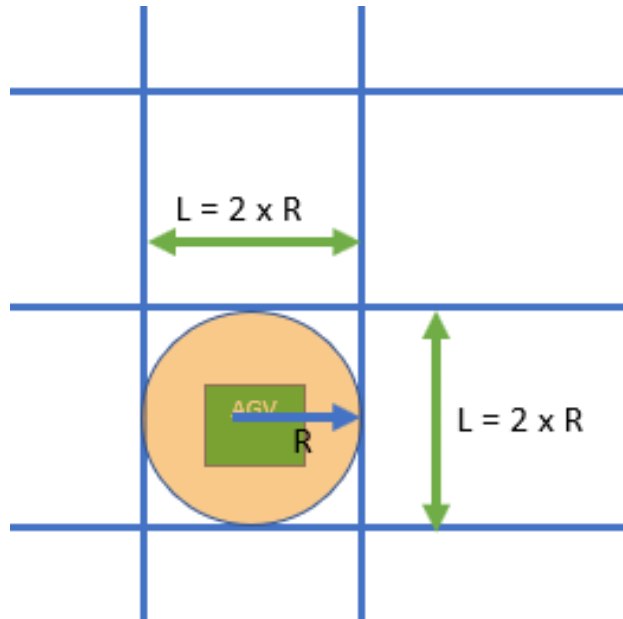


Figura 31 Representación de la cuadrícula y el radio de maniobra del AGV (Fuente: Elaboración propia)

Para facilitar la superposición de la imagen adecuándola a la rejilla se ha utilizado una función en visual basic (macro de Excel) para ajustar la cuadrícula de la hoja de forma que el largo y el ancho de todos los cuadrados sea el mismo. Además, para facilitar la colocación de la imagen se ha dibujado un cuadrado con las dimensiones que debe tener la cuadrícula en la esquina superior izquierda, de forma que se pueda ajustar la imagen haciendo coincidir este cuadrado con la celda A1 de la hoja “mapa”. A continuación, se muestra la figura 32 donde se puede ver esto.

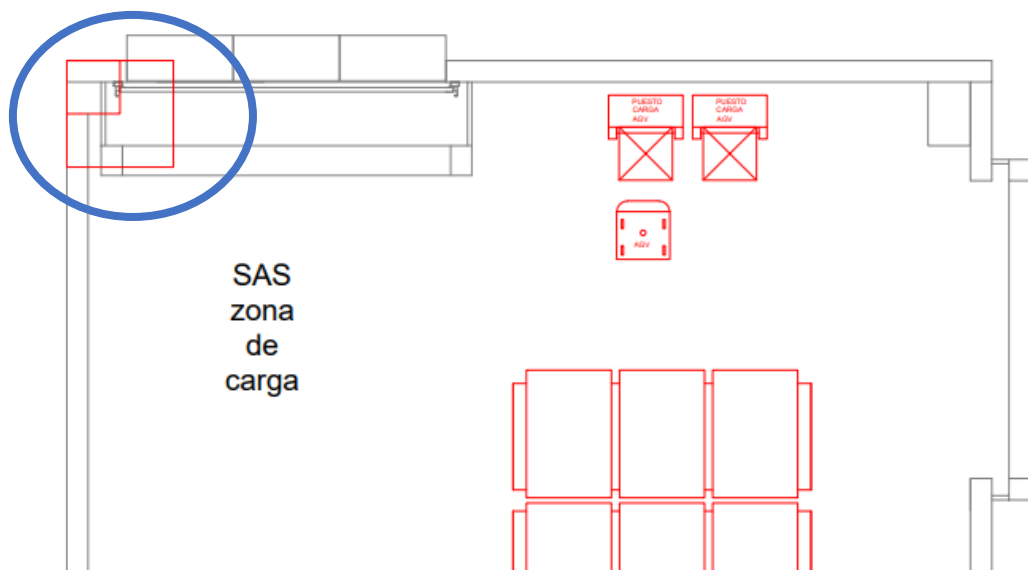


Figura 32 Cuadrado de referencia, destacado en azul (Fuente: Elaboración propia)

4.3 CASO BÁSICO

En este apartado presentamos el funcionamiento de las diversas partes del sistema, desde la creación de los datos que se necesitan, utilizando para ello la Hoja de cálculo “Datos_ACO” descrita anteriormente, como el código en MatLab, pudiendo observar los resultados obtenidos.

4.3.1 Objetivos del caso básico

Aplicación de del programa creado utilizando un algoritmo ACO para la definición de rutas en AGVs a un caso sencillo para conocer y adquirir experiencia en el funcionamiento de las distintas herramientas, y los pasos necesarios del algoritmo, tanto la parte de la adquisición de datos desde Excel como la ejecución en MatLab. Se presenta con un caso simple, puesto que el procedimiento de trabajo es el mismo para cualquier caso. Únicamente se diferenciarán por el tiempo de procesamiento necesario, por lo que se considera oportuno comenzar con un caso sencillo.

4.3.2 Desarrollo del caso básico

En principio se selecciona el caso concreto en el que se aplicará el algoritmo para seleccionar el mejor recorrido, indicando el origen y destino del elemento que se desplaza, (en este caso un AGV), y los puntos en los cuales tiene que realizar paradas. En este caso, se considera un almacén pequeño, en el cual el punto de origen es la zona de carga de los AGVs, el destino es la zona de carga de camiones, donde se descargarán los objetos recogidos, y los puntos de paradas intermedias las zonas del almacén donde debe recoger los diversos objetos que se llevarán a la zona de carga de camiones.

Por ello, en primer lugar, tenemos que tener la implantación de la zona en la que se desplazarán los AGVs. En este trabajo, para el trazado de dicha superficie, se utilizará el software AutoCAD. En la Figura 33 se puede ver la distribución que se ha diseñado para este caso.

Como se puede observar, se trata de un almacén pequeño, en el que tenemos en la parte superior izquierda la zona de carga de los AGVs, que será el punto inicial del recorrido. En la parte superior derecha se puede ver una zona de carga de camiones, la cual será el destino final del recorrido. Estas dos zonas están muy próximas por lo que al terminar el recorrido el AGV vuelve al punto inicial, donde tiene las estaciones para carga de baterías.

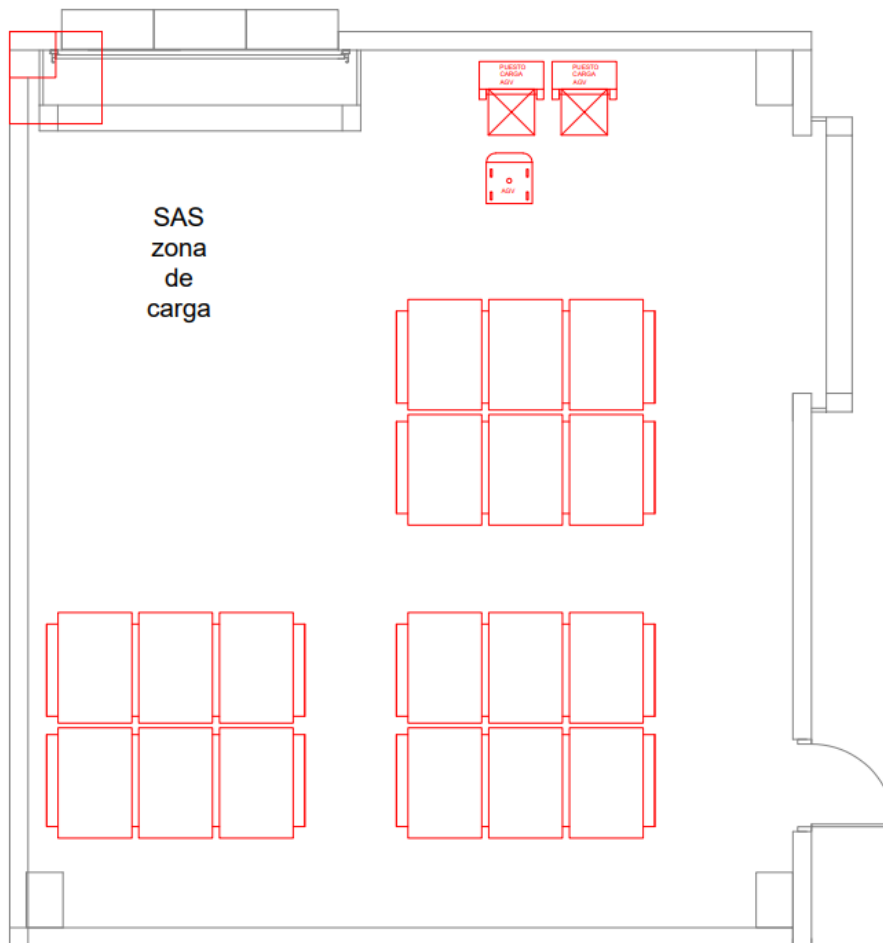


Figura 33 Implantación caso básico (almacén simple) (Fuente: Elaboración propia)

El resto de la distribución, toda la parte inferior, está ocupada por estanterías y pasillos de acceso a éstas. Estas estanterías serán los puntos por donde tiene que pasar el AGV antes de llegar al destino final. Se seleccionarán una serie de puntos en los cuales tendrá que parar para recoger algún objeto. En la siguiente Figura tenemos la propuesta de puntos de carga para este caso, aunque existen múltiples opciones.

En dicha figura se puede ver que aparecen dos cuadrados marcados en la esquina superior izquierda, los cuales como se han indicado en el apartado anterior, sirven de referencia para ajustar la cuadrícula de la hoja "mapa" en el Excel "Datos_ACO".

A continuación, en la figura 34, se pueden ver marcados los puntos por los que debe pasar el AGV, los cuales corresponderán a puntos obligatorios para el algoritmo ACO.

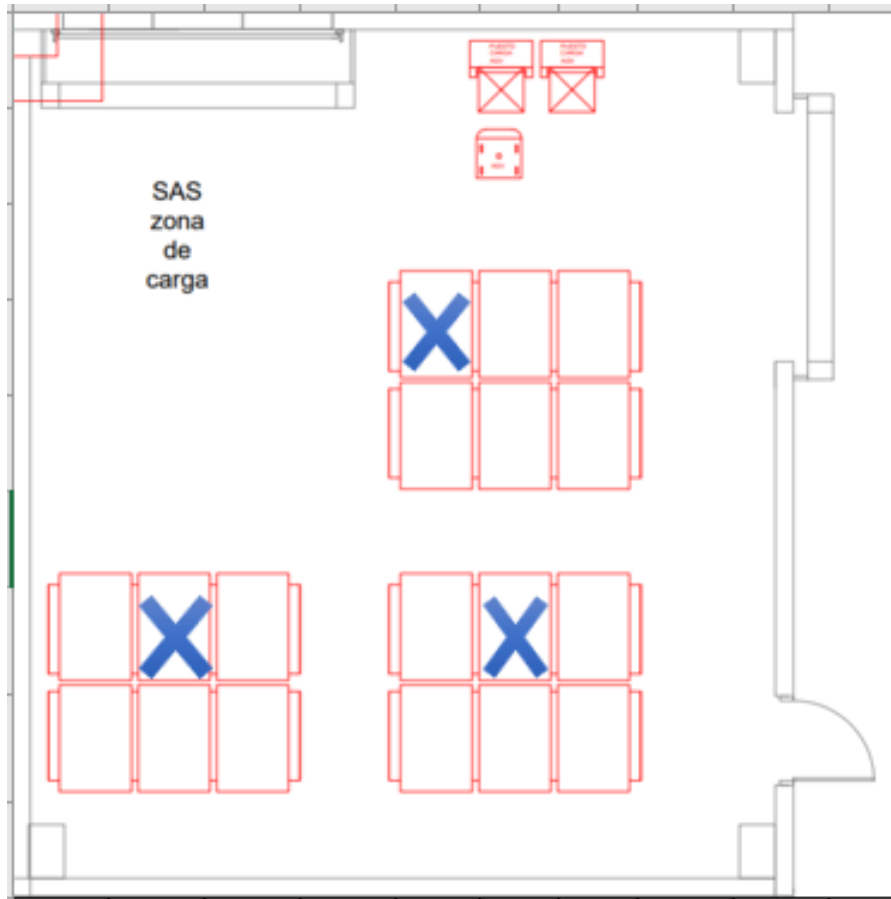


Figura 34 Puntos obligatorios para el caso básico (Fuente: Elaboración propia)

El siguiente paso es utilizarlo para definir la matriz de puntos posibles y puntos obligatorios, para ello se utiliza el fichero Excel "Datos_ACO" que se ha descrito anteriormente. Se siguen los pasos que se indican a continuación para obtener los distintos tipos de puntos que se pasarán más tarde al algoritmo en MatLab.

En la figura 35 se puede ver cómo queda la hoja "Mapa" tras realizar los ajustes necesarios. Se utiliza un radio de maniobra de 1 m de diámetro, aunque también se han hecho pruebas con un radio de 0.5 m de diámetro. Finalmente se considera el segundo puesto que para realizar el ejemplo básico no se considera necesario tener tantos puntos, ya que la cantidad de puntos es proporcional al tamaño del radio de maniobra.

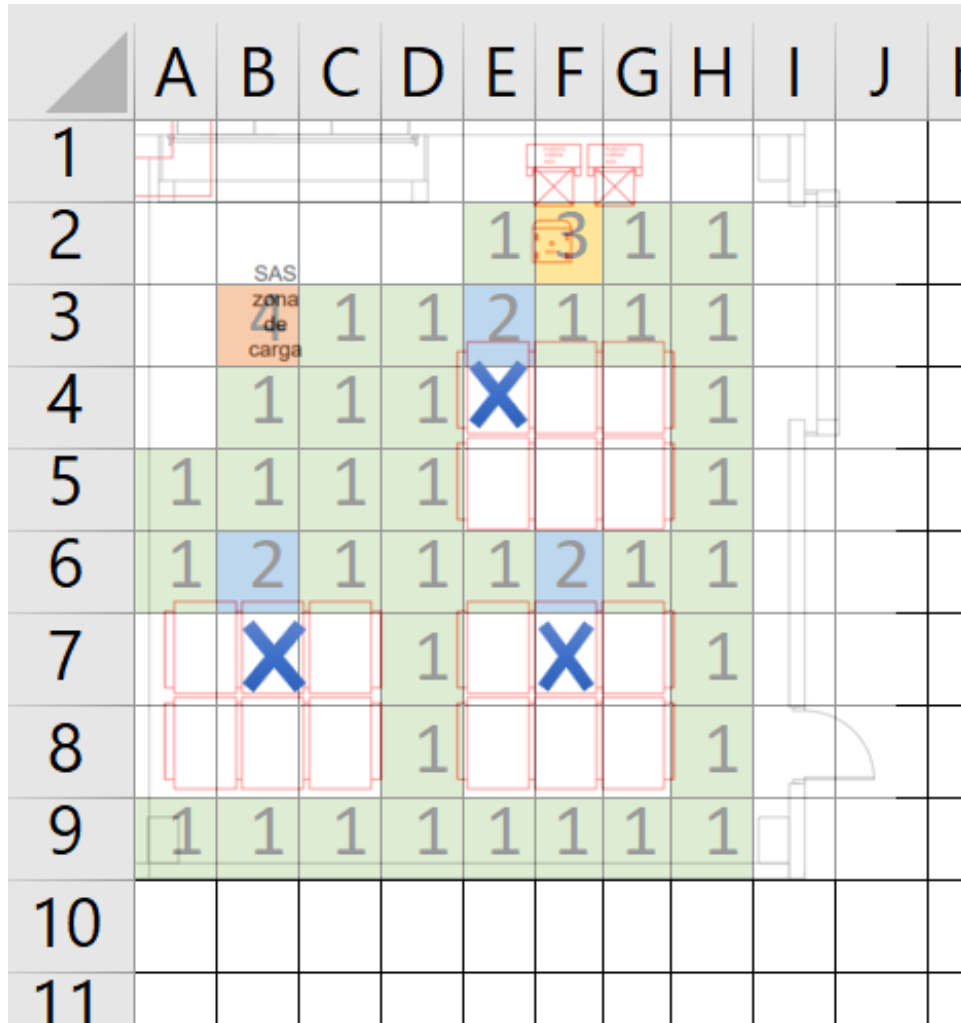


Figura 35 Hoja "Mapa" del Excel "Datos_ACO" para el caso básico (Fuente: Elaboración propia)

En la figura podemos ver cómo se ha completado la cuadrícula que ocupa la implantación, indicando 1 en las zonas libres por las que se puede desplazar el AGV y dejando en blanco los espacios que están ocupados por cualquier tipo de obstáculos como son las estanterías. De esta forma, se tienen todos los puntos posibles por los cuales se va a poder mover el AGV. Además, se indican una casilla con un 3, inicio del recorrido, que está situada en la zona de carga de AGVs y otra con un 4 indicando el final del recorrido, donde se realizará la descarga de los elementos recogidos en el recorrido. Por último, se indica con un 2 las casillas en las que es obligatorio pasar, donde se hará una parada para recoger los elementos a transportar. Una vez se han completado todos los valores tenemos el mapa completo.

En la hoja "Auxiliar" se completan los datos necesarios. Por último, se pasa a la hoja de "Datos" de este fichero, donde se utilizarán las macros creadas para obtener los listados de los distintos tipos de puntos. Para ello en primer lugar se eliminan los puntos de un caso anterior, y tras esto se obtienen los *puntos posibles* y *puntos obligatorios*, esto se realiza utilizando las funcionalidades proporcionadas por el fichero creado. Con esto se obtienen los puntos en el formato que más tarde se

pasará al algoritmo en MatLab. Esta hoja de cálculo final se puede ver en la figura 36.

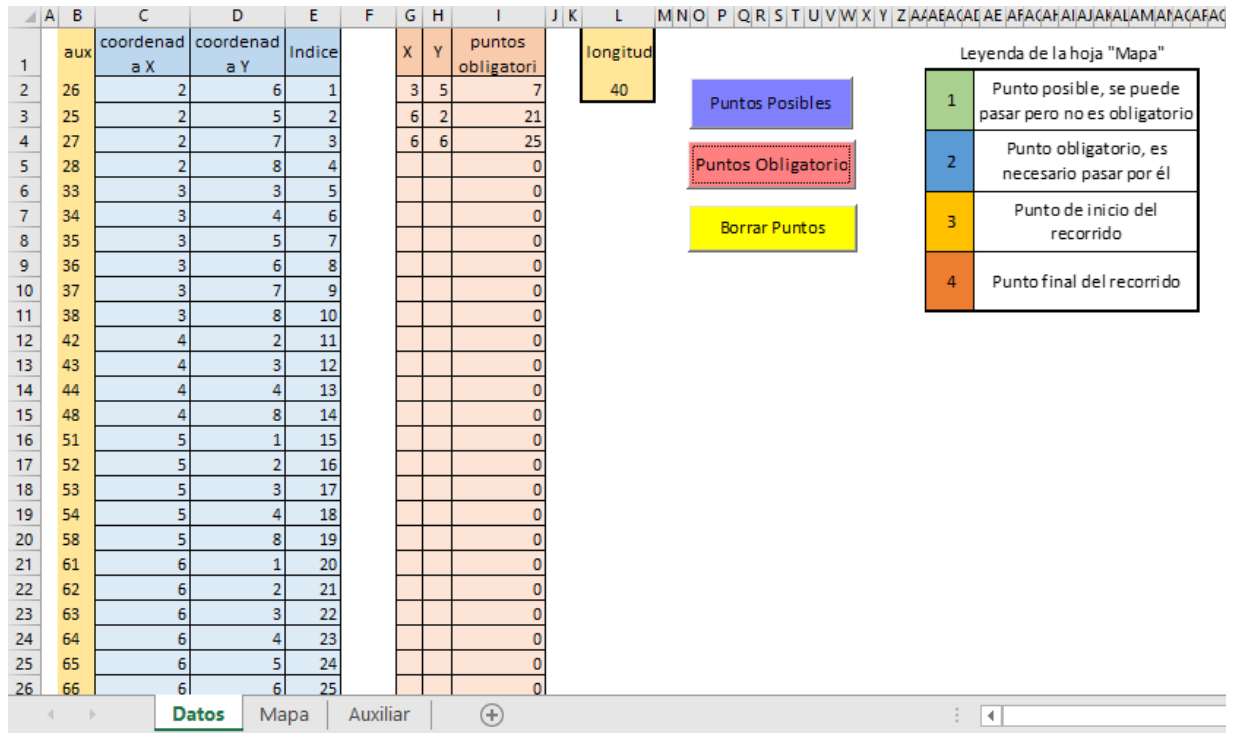


Figura 36 "Datos" del Excel "Datos_ACO" para el caso básico (Fuente: Elaboración propia)

El siguiente paso es la ejecución del algoritmo por medio de MatLab. Se comienza importando los datos obtenidos por medio del Excel "Datos_ACO", el cual se debe encontrar en la misma carpeta que los archivos de MatLab. Para la exportación de dichos datos se debe ir a la función "createGraph" en el editor de MatLab y modificar las líneas 7 y 8, las cuales contienen la función xlsread para tomar datos de un fichero Excel. Si no se ha modificado nada y se ha realizado todo como se indica en este capítulo únicamente es necesario modificar la última parte de la función, lo que corresponde a la longitud del vector de datos más uno. Es por este motivo por el que se muestra este valor en la hoja de "Datos" en el excel "Datos_ACO". Se puede ver en la figura 37.

```

3  function [ graph ] = createGraph_v1()
4  |
5  %DATOS: coordenadas de los puntos
6
7  x=xlsread('Datos_ACO.xlsx','C2:C41');
8  y=xlsread('Datos_ACO.xlsx','D2:D41');
9
    
```

Figura 37 Introducción de datos de Excel en función "createGraph (Fuente: Elaboración propia)

Con esto se han generado los vectores X e Y, los cuales contienen todos los puntos tanto obligatorios como opcionales.

Una vez realizado esto, se deben modificar las funciones donde se indican qué puntos de los introducidos corresponden a los puntos obligatorios. Esto es necesario modificarlo en 3 funciones de manera similar al caso anterior.

En la función “createColony” se modifica la línea 10, indicando las celdas del excel “Datos_ACO” en las cuales están indicados los puntos considerados obligatorios. Al igual que en el caso anterior únicamente es necesario modificar la parte final de la función `xlsread` como se ve en la siguiente figura 38.

```

2  function [ colony ] = createColony_v1( graph, colon:
3
4  % disp("empezar");
5  -  nodeNo = graph.n;
6  -  final.n = graph.n;
7  -  quitar=[];
8
9  %Indicar puntos obligatorios
10 -  obligatorios=xlsread('Datos_ACO.xlsx','I2:I4');
11

```

Figura 38 Introducción de datos de Excel en función "createColony" (Fuente: Elaboración propia)

En las funciones “drawGraph” y “drawBestTour” se modifican líneas similares a la indicada en la función “createColony. Como se puede ver en las siguientes figuras 39 y 40 estas líneas son la 5 en el primer caso y la 4 en el segundo.

```

2  function [ ] = drawGraph_v1( graph )
3  % To visualize the nodes and edges of the graph
4  -  hold on
5  -  obligatorios=xlsread('Datos_ACO.xlsx','I2:I4');
6

```

Figura 39 Introducción de datos de Excel en función "drawGraph" (Fuente: Elaboración propia)

```

2  function [ ] = drawBestTour_v1(colony , graph)
3
4  -  obligatorios=xlsread('Datos_ACO.xlsx','I2:I4');
5  -  queenTour = colony.queen.tour;

```

Figura 40 Introducción de datos de Excel en función "drawBestTour" (Fuente: Elaboración propia)

Si se quiere se pueden modificar los valores de hormigas utilizadas en el algoritmo y de iteraciones a realizar, esto se hace en la función principal “ACO” en las líneas 21 y 22 como se puede ver en la figura 41 Estos valores se elegirán en función de la complejidad del caso que se vaya a realizar, pudiendo ser posible probar con varias combinaciones de valores hasta obtener resultados coherentes. Para la ejecución de este caso se han dado unos parámetros de 500 iteraciones y 40 hormigas, los cuales son los aproximadamente los valores recomendados por las bibliografías consultadas.

```

18 %% ACO algorithm - Algoritmo ACO
19
20 %% Initial parameters of ACO - Parámetros iniciales de ACO
21 - maxIter = 500; % número máximo de iteraciones 500
22 - antNo =40; % número de hormigas 50
23

```

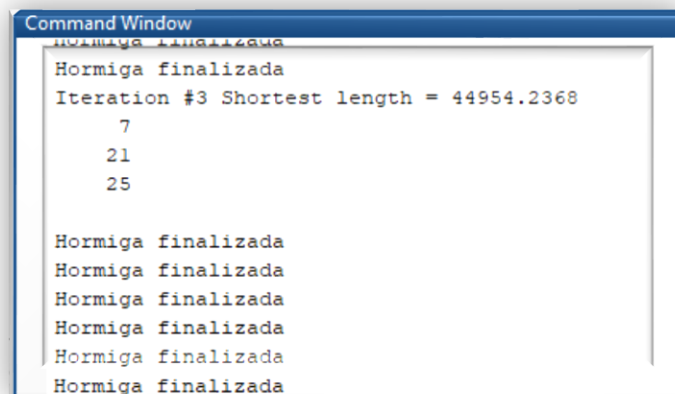
Figura 41 Parámetros de número de hormigas y de iteraciones en algoritmo ACO (Fuente: Elaboración propia)

Tras la importación de los datos, se debe ejecutar el algoritmo introduciendo la función `ACO_v1` en la ventana de comandos y se espera a que se ejecute, realizando las distintas iteraciones.

Al realizar la primera iteración ya se pueden visualizar los gráficos, y a medida que se van sucediendo las distintas iteraciones se puede ver cómo se van modificando dichos gráficos. En la ventana de comandos al finalizar cada iteración se muestra la longitud del mejor recorrido obtenido hasta el momento. Además, cada vez que una hormiga termina el recorrido aparece un mensaje indicando “hormiga finalizada”. Este mensaje se ha puesto porque el algoritmo tarda bastante tiempo en ejecutarse y para poder comprobar fácilmente que está funcionando y no se ha bloqueado en ningún punto del programa.

Esta información en las primeras iteraciones cambia prácticamente en cada iteración, pero a medida que se ejecuta el algoritmo, cada vez tarda más iteraciones en cambiar, puesto que solo se modifica si se mejora el último valor guardado como recorrido más corto, que es el óptimo.

A continuación, se muestra un ejemplo de cómo aparece la ventana de comando, figura 42, y de la sucesión de las gráficas al mejorar el recorrido obtenido. En la ventana de comando se ve que aparecen una serie de números después del mensaje de “Iteration #3 Shortest length = 44954.2368”. Estos números indican los puntos obligatorios. Este mensaje finalmente se ha eliminado de la programación, de forma que en los siguientes casos no aparece, pero se incluyó en un primer momento para comprobar que los datos se estaban tomando de manera correcta.



```

Command Window
Hormiga finalizada
Hormiga finalizada
Iteration #3 Shortest length = 44954.2368
    7
   21
   25

Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada

```

Figura 42 Ventana de comandos de MatLab durante la ejecución del algoritmo (Fuente: Elaboraciónn propia)

En la figura 43 se puede ver la gráfica obtenida por medio de la función “drawGraph”, en la cual se pueden ver todas las uniones posibles entre los puntos, y los puntos destacados como importantes, los cuales aparecen en color azul, verde y amarillo, correspondientes a los puntos origen, destino y puntos obligatorios respectivamente.

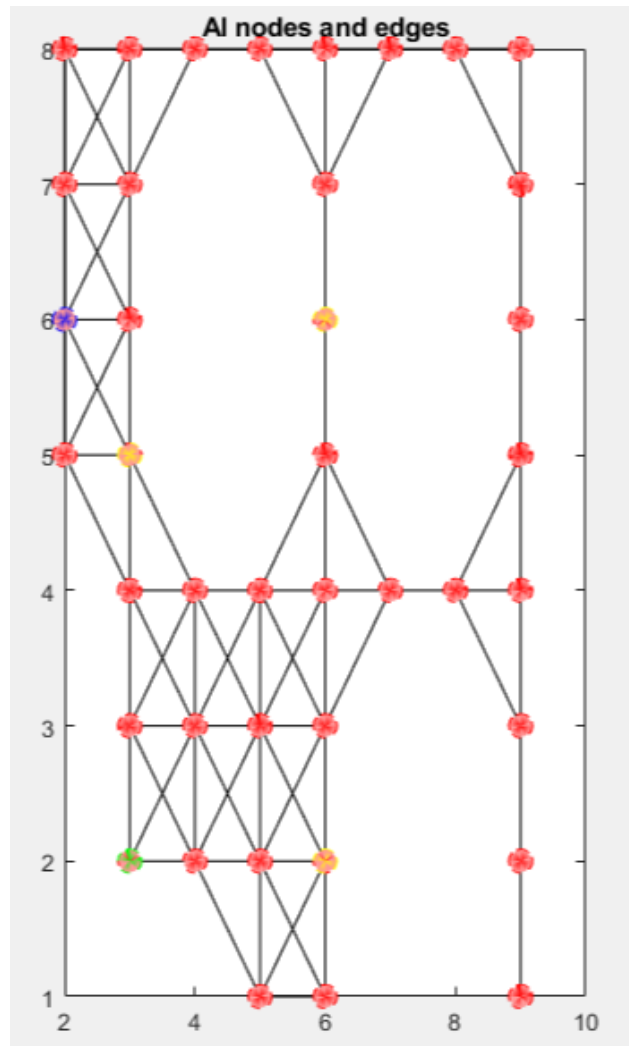


Figura 43 Gráfica obtenida por la función "drawGraph" (Fuente: Elaboración propia)

Finalmente, se muestran varios de los recorridos obtenidos en orden cronológico, siendo el último el obtenido como solución del algoritmo. No se muestran todos los recorridos ya que no se considera interesante mostrar tal cantidad de resultados obtenidos, sino una muestra para ver cómo evolucionan los caminos que se van obteniendo a medida que se ejecutan las iteraciones, para lograr la solución óptima.

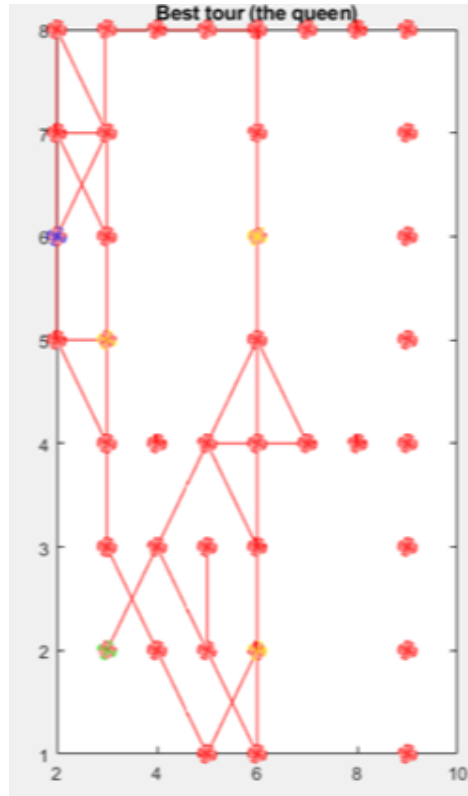


Figura 44 Gráfica obtenida por la función "drawBestTour" en la primera iteración (Fuente: Elaboración propia)

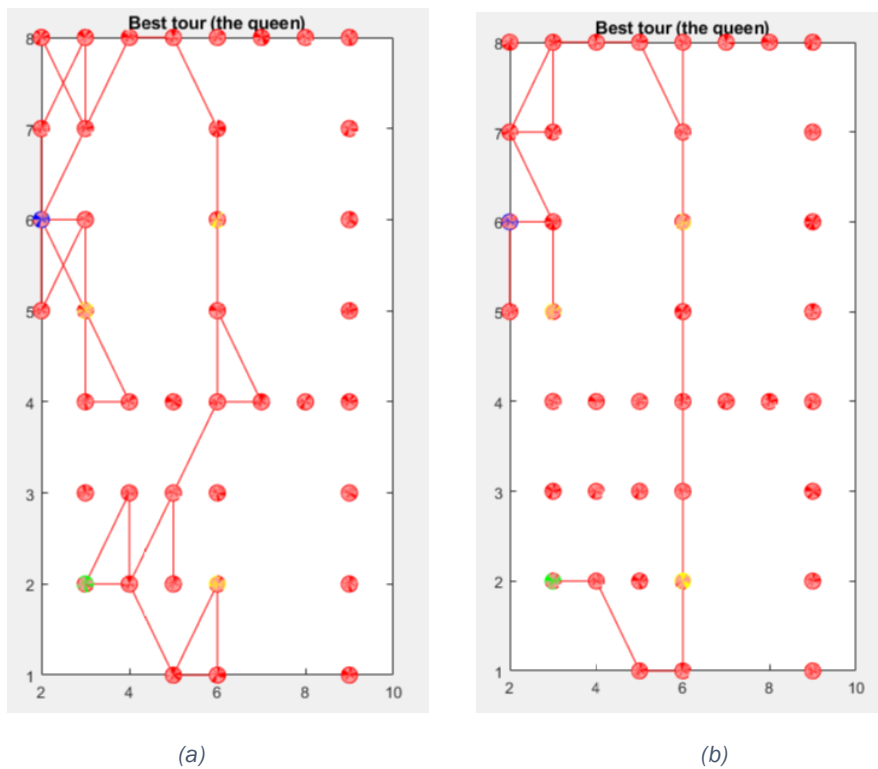


Figura 45 (a) y (b) Gráficas obtenidas por la función "drawBestTour" en iteraciones intermedias (Fuente: Elaboración propia)

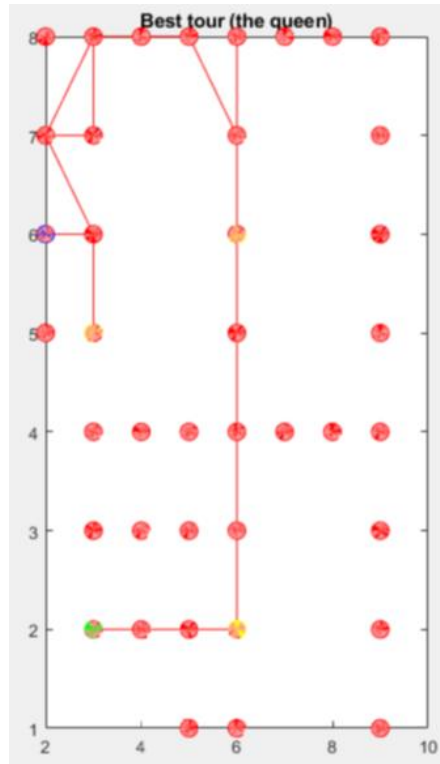


Figura 46 Última gráfica obtenida por la función "drawBestTour" (Fuente: Elaboración propia)

La información obtenida como solución del algoritmo, además de obtenerse gráficamente por medio de *Best Tour (the queen)*, como se ha mostrado en las imágenes anteriores; también se obtiene numéricamente. El obtenido como mejor camino se encuentra almacenado en la variable *colony.queen.tour*. Esta información se muestra a continuación, tanto en la figura 47 como más abajo, donde se pueden ver mejor los valores obtenidos.

	VARIABLE										SELECTION										EDIT										
colony.queen.tour																															
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	
2	1	2	1	8	7	8	1	8	3	10	9	3	9	3	10	14	19	26	27	26	25	24	23	22	21	16	11	16	11	40	
3																															

Figura 47 Valores obtenidos en la variable "colony.queen.tour" (Fuente: Elaboración propia)

COLONY.QUEEN.TOUR:

1	2	1	8	7	8	1	8	3	10	9
3	9	3	10	14	19	26	27	26	25	24
23	22	21	16	11	16	11	40			

Como se puede observar, la solución obtenida es una buena solución, pero aún es posible mejorarla, puesto que, en algunos casos, aunque gráficamente no se aprecie, el algoritmo en ocasiones pasa dos veces por los mismos puntos. Además, hay ocasiones en las que da algún rodeo para ir de un punto a otro que está al lado.

4.3 CASO COMPLEJO

En este apartado presentamos el desarrollo de un caso complejo, en el que se aplica algorítmica ACO expuesta en este capítulo.

En este caso, al igual que en el anterior, se estudiará la mejor ruta dentro de un almacén para recoger una serie de objetos y llevarlos a la zona de carga de camiones, como ejemplo de aplicación en industria.

4.3.1 Objetivos del caso complejo

En este caso se quiere observar el funcionamiento del algoritmo en un ejemplo más complejo que el anterior, aunque también próximo a una situación real en almacenes en naves industriales.

En este caso se aplica a un almacén de grandes dimensiones y con productos de diferente naturaleza, presentando un layout complejo en el cual no existe una linealidad por la que desplazarse, sino que las rutas dependen de los puntos a los que se deba ir a recoger o dejar objetos.

4.3.2 Desarrollo del caso complejo

El procedimiento de trabajo es similar al caso anterior, por lo que no se requiere tanto detalle ya expuesto anteriormente.

Al igual que en el apartado anterior la implantación a la que se aplicará el algoritmo será un almacén donde el punto inicial será la zona de carga de baterías de AGVs, donde se encontrarán estacionados hasta tener una orden. El punto final en el que terminarán su recorrido antes de volver a la zona de carga es una zona destinada a la carga de camiones, donde los AGVs descargarán los materiales que lleven. Además, el AGV debe pasar por una serie de puntos intermedios correspondientes a distintas zonas de almacén, frente a las estanterías donde deban recoger los elementos que tienen que llevar a la zona de carga.

Una vez descargan los AGVs, vuelven a la zona de carga de baterías, la cual se encuentra cerca del punto final, por lo que este camino de vuelta desde el punto final al inicial no se considera para el algoritmo ACO, ya que es un camino inmediato.

A continuación, se muestra el layout escogido para este caso en la figura 48, diseñado y representado también con AutoCAD.

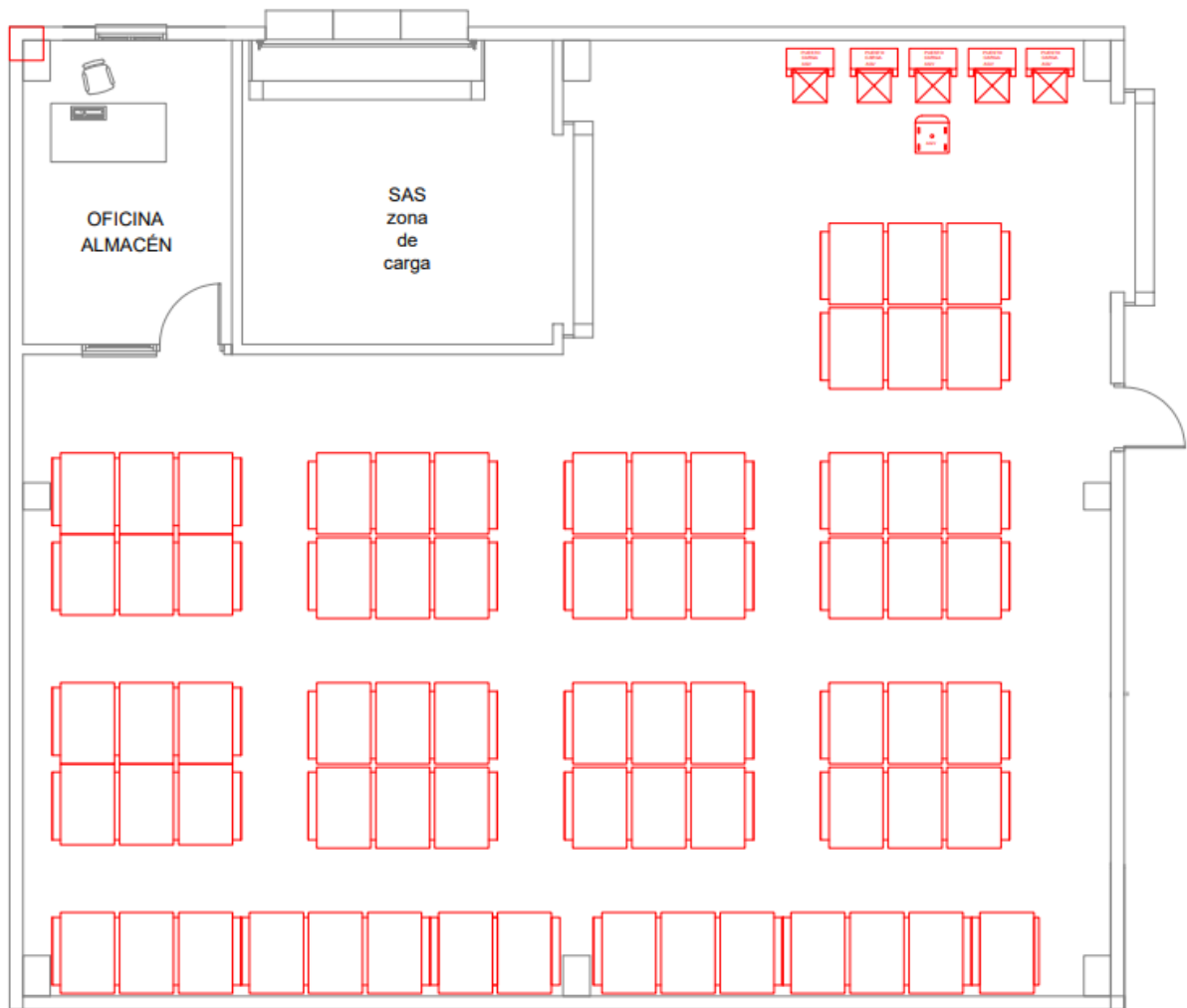


Figura 48 Implantación caso complejo (Fuente: Elaboración propia)

Al igual que en el otro caso, una vez tenemos la implantación concreta y los puntos que se consideren obligatorios (puntos para carga del AGV) utilizaremos esta implantación en el Excel "Datos_ACO".

Este procedimiento es igual al presentado antes, por lo que se indicará brevemente mostrando imágenes del resultado en Excel.

En primer lugar, se coloca la implantación en la hoja "Mapa" y se completa siguiendo el mismo código numérico que se ha explicado anteriormente. Una vez se ha rellenado la cuadrícula correspondiente al plano del almacén se obtiene el siguiente resultado, que se puede ver en la figura 49.

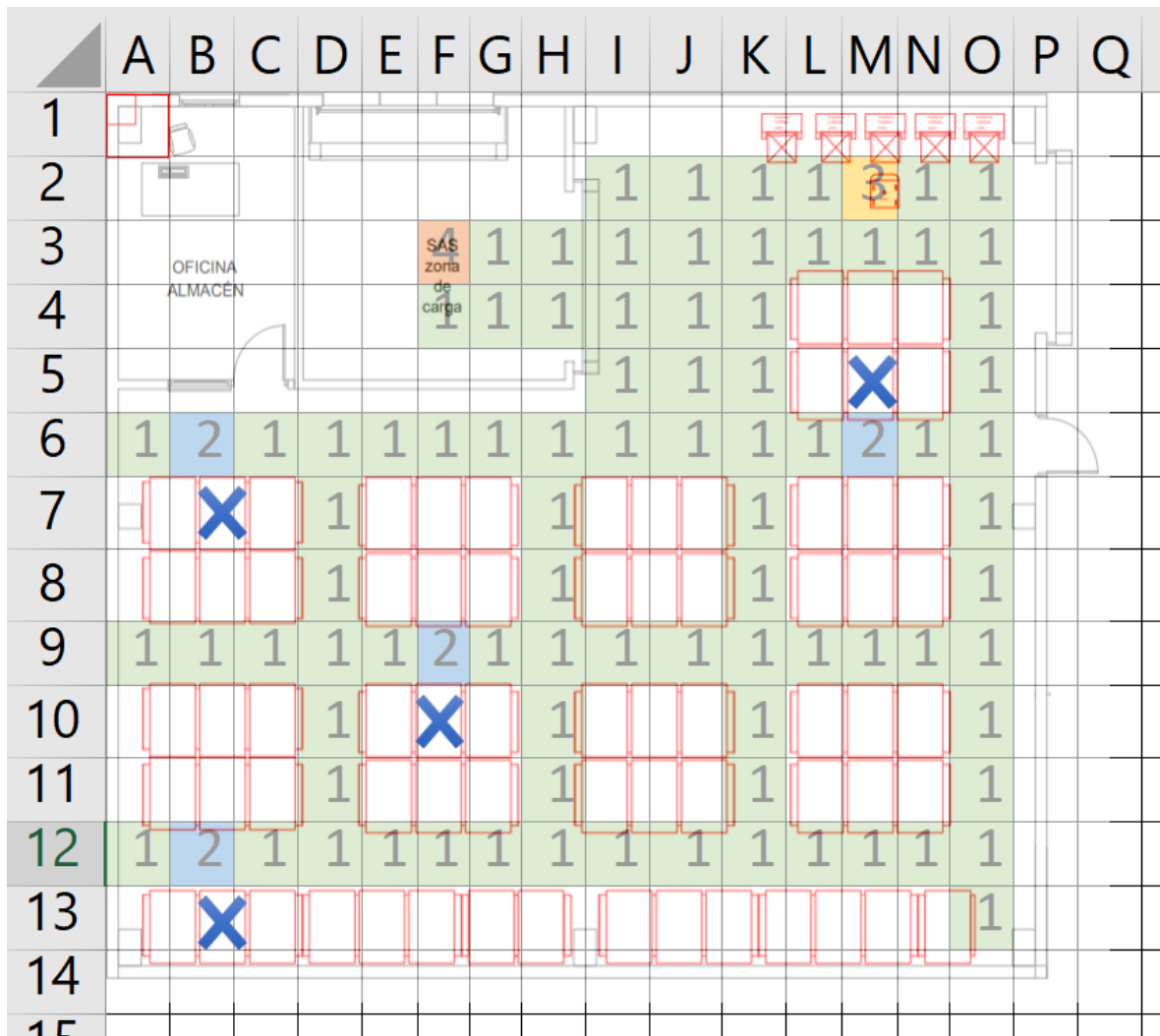


Figura 49 Hoja "Mapa" del excel "Datos_ACO" caso complejo (Fuente: Elaboración propia)

A continuación, se actualizan los datos de la hoja auxiliar adecuándolos a este caso, y finalmente, se utiliza la hoja "Datos". Al igual que en el caso anterior por medio de los botones creados para este propósito en primer lugar se borran todos los puntos que pueda haber de casos anteriores, de forma que se evite cometer errores por quedar datos residuales de otros casos. Tras esto se utilizan el botón "Puntos Posibles" seguido del botón "Puntos Obligatorios" para generar los datos correspondientes a este ejemplo que más tarde se utilizarán en la aplicación de MatLab. La hoja "Datos" una vez se ha hecho esto queda como se ve en la figura 50.

DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	
1		aux	coordenada X	coordenada Y	Indice		X	Y	puntos obligatori			longitud													
2		213	2	13	1		6	2	29			90													
3		29	2	9	2		6	13	40																
4		210	2	10	3		9	6	56																
5		211	2	11	4		12	2	75																
6		212	2	12	5				0																
7		214	2	14	6				0																
8		215	2	15	7				0																
9		37	3	7	8				0																
10		38	3	8	9				0																
11		39	3	9	10				0																
12		310	3	10	11				0																
13		311	3	11	12				0																
14		312	3	12	13				0																
15		313	3	13	14				0																
16		314	3	14	15				0																
17		315	3	15	16				0																
18		46	4	6	17				0																
19		47	4	7	18				0																
20		48	4	8	19				0																
21		49	4	9	20				0																
22		410	4	10	21				0																
23		411	4	11	22				0																
24		415	4	15	23				0																

Figura 50 Hoja "Datos" del Excel "Datos_ACO" caso complejo

Debido al largo tiempo de procesamiento que se necesita para ejecutar este modelo y que los medios disponibles son limitados para este tipo de procesos, se toma la decisión de trabajar con un menor número de hormigas y de iteraciones. Esta decisión influye en la precisión del algoritmo; sin embargo, escoger de 50 hormigas recomendadas 10, y ejecutarlo con un menor número de iteraciones, pasando de las 500 recomendadas a 150, permite obtener unos resultados razonables y válidos para poder extraer conclusiones relevantes para los objetivos marcados en este TFG.

A continuación, se muestran los mensajes obtenidos en la ventana de comandos en Matlab en la figura 51 y el gráfico obtenido por medio de la función "drawGraph" en la figura 52.

DISEÑO Y DESARROLLO DEL ALGORITMO APLICADO A LOGÍSTICA DE ALMACENES

```
Iteration #34 Shortest length = 90002.3137
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Iteration #35 Shortest length = 89999.8284
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
Iteration #36 Shortest length = 89999.8284
Hormiga finalizada
Hormiga finalizada
Hormiga finalizada
```

Figura 51 Mensajes obtenidos en la ventana de comandos de MatLab (Fuente: Elaboración propia)

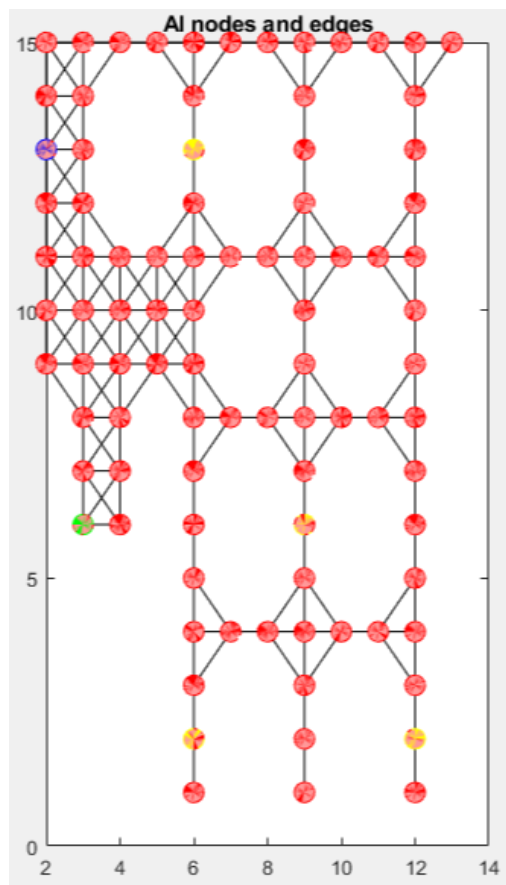


Figura 52 Gráfico de todos los puntos y uniones del caso complejo (Fuente: Elaboración propia)

Se puede observar en la Figura que no ocurren fallos, sino que el único problema es el tiempo de procesamiento ya que no aparece ningún tipo de error y el programa se ejecuta correctamente. Este caso no ha sido posible ejecutarlo completamente ya que el tiempo que tarda es demasiado elevado, por lo cual no se ha podido obtener la solución óptima. Sin embargo, se considera que con los medios adecuados se puede llegar a ejecutar correctamente como ocurrió con el caso simple.

En el siguiente capítulo se tratan y analizan en profundidad los resultados obtenidos en los dos casos realizados.

5 ANÁLISIS DE LOS RESULTADOS

En este capítulo se van a analizar los resultados obtenidos en los dos casos desarrollados, además de los resultados obtenidos modificando las variables número de hormigas e iteraciones del algoritmo para comprobar la variación de las soluciones en función de estos parámetros.

5.1 ANÁLISIS DEL ALGORITMO LOGÍSTICA DE TRANSPORTE

En este apartado se va a presentar el análisis de los resultados obtenidos utilizando el algoritmo aplicado a la logística del transporte expuesto en el capítulo 3. Este algoritmo se ha utilizado en dos casos similares. El primero de los casos es sencillo, el cual se puede resolver sin necesidad de métodos metaheurísticos, y se ha realizado para comprobar la efectividad del algoritmo diseñado. El segundo de los casos, indicado como caso complejo, se asemeja a un caso real, en el que es necesario utilizar este tipo de algoritmos por la gran cantidad de posibles soluciones existentes. A continuación, se analizan los resultados obtenidos en ambos casos, para la logística de transporte.

5.1.1 Análisis del caso simple

Este caso se ha realizado para comprobar la efectividad del programa creado a partir de un algoritmo ACO. Éste consiste en obtener la mejor ruta cerrada entre cuatro ciudades.

Se ha obtenido la solución óptima de marea exacta calculando el tiempo que se tarda en realizar cada una de las combinaciones posibles, obteniendo el resultado óptimo.

A continuación, se ha utilizado el programa en Matlab basado en el algoritmo ACO con los mismos datos, del cual se ha obtenido el mismo resultado que se ha calculado anteriormente. El programa se ha ejecutado en varias ocasiones, obteniendo en todos los casos el mismo resultado.

Con esto se ha podido comprobar que el programa creado proporciona resultados válidos.

Al ser un caso sencillo, en todas las ocasiones se ha llegado al óptimo, puesto que existen pocas combinaciones posibles. Además, se ha necesitado valores pequeños tanto para la variable número de hormigas como para el número de iteraciones.

Para casos más complejos, y debido a la aleatoriedad del algoritmo, cabe esperar que no se obtenga en todas las ocasiones la misma solución, puesto que puede ocurrir que no se llegue a la solución óptima en todas las ejecuciones del programa, sin embargo, sí que será una buena solución en todo caso. Además, la calidad de la solución obtenida dependerá de los valores asignados a las variables. Esto se explicará con mayor detalle en los siguientes apartados.

5.1.2 Análisis del caso complejo

En esta ocasión se ha utilizado un caso que puede corresponder a un caso real, en el que hay que trazar la mejor ruta cerrada entre 10 localidades.

Para ello, se ha ejecutado el programa setenta veces, con diferentes valores para las variables número de hormigas y número de iteraciones. A continuación, se muestran los resultados obtenidos para la variable *colony.queen.fitness*, tiempo que se tarda en realizar la mejor ruta obtenida en la tabla 7. Con estos resultados se va a realizar una representación gráfica de forma que se pueda comprobar cómo varía la calidad de las soluciones al modificar las variables principales.

Nº hormigas	Nº iteraciones	colony.queen.fitness en cada ejecución (min)										Mejor resultado (min)	Peor resultado (min)
		1	2	3	4	5	6	7	8	9	10		
5	20	131	130	128	127	131	129	127	128	125	127	125	131
5	100	120	125	119	115	123	124	122	119	125	121	115	125
5	300	122	120	112	117	120	121	121	121	117	114	112	122
20	20	115	117	123	128	117	127	122	124	123	130	115	130
20	100	120	115	114	115	117	117	121	112	115	122	112	122
20	300	114	115	117	115	112	115	115	117	115	114	112	117
40	400	112	114	112	115	114	115	112	114	115	112	112	115

Tabla 7 Resultados colony.queen.fitness (Fuente: Elaboración propia)

En la figura 53 se muestra un gráfico de cajas donde se pueden ver los resultados obtenidos para las 7 combinaciones de valores de las variables utilizadas.

Se emplea un diagrama de cajas y bigotes dado que tenemos una serie de números distribuidos a lo largo de una recta, correspondientes a los resultados obtenidos de cada una de las combinaciones de las variables, estando esto representado por cada una de las cajas, diferenciadas por colores.

En cada una de las cajas se agrupan los datos en cuatro secciones limitadas en los extremos de los bigotes por el valor máximo y el valor mínimo y representando con la caja el primer y tercer cuartil, ubicando una línea en el interior de la caja que representa la mediana. Cada sección del diagrama contiene un cuarto de los datos.

En el caso de que se hubiese ejecutado el código un número muy elevado de veces, además de estas secciones aparecerían asteriscos por fuera de los límites de los bigotes, representando valores atípicos. En este caso esto no ocurre, ya que se ha ejecutado el programa únicamente diez veces para cada situación, pero se considera que para un análisis simple es suficiente ya que el comportamiento de los resultados es razonable.

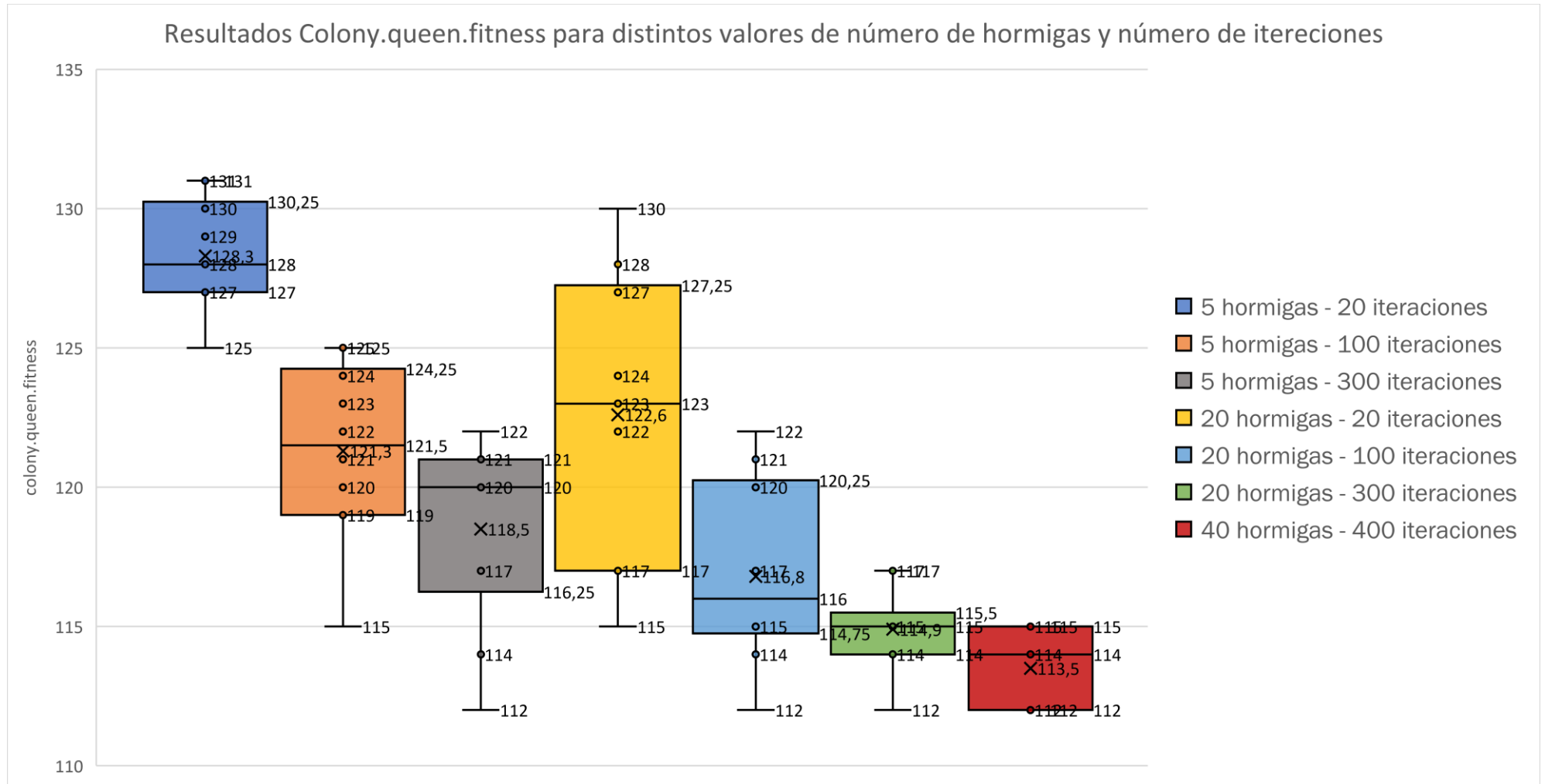


Figura 53 Gráfico cajas y bigotes de los resultados obtenidos para colony.queen.fitness (Fuente: Elaboración propia)

En el gráfico se puede ver la dispersión que tienen los datos para cada caso, de forma que en los que el número de hormigas y número de variables es elevado la dispersión disminuye y el algoritmo ofrece buenos resultados. En el caso en el que las dos variables tienen valores menores pese a tener poca dispersión los resultados obtenidos son de peor calidad. Para casos intermedios la dispersión es mayor pero los resultados mejoran cuanto mayor sean las variables.

También se puede comprobar la calidad de los resultados, puesto que cuanto menor sea el resultado obtenido mejor será el resultado.

Los resultados de peor calidad se obtienen para el caso 20 hormigas y 20 iteraciones ya que, pese a tener un número razonable de hormigas 20 iteraciones son muy poco para este tipo de algoritmos.

Además, se pueden analizar por bloques en los que el número de hormigas coincide, y únicamente varía el número de iteraciones. De esta forma, los tres primeros forman un bloque en el que el número de hormigas es 5 y los tres siguientes otro bloque para un número de hormigas 20. En ambos bloques conjunto se aumenta el número de iteraciones igual, teniendo 20, 100 y 300. Para ambos casos se puede observar que a medida que aumentan el número de iteraciones la calidad de las soluciones mejora.

De igual manera, se puede observar que, si agrupamos dos a dos los casos que tienen un mismo número de iteraciones, de forma que la variable que varía sea el número de hormigas, éstos son el primero con el cuarto, el segundo con el quinto y el tercero con el sexto, se puede ver que cuanto mayor número de hormigas se obtienen mejores resultados.

Finalmente, el último de los casos realizados tiene un mayor número de hormigas y de iteraciones que todos los anteriores (40 y 400 respectivamente). Se puede observar que en este caso los resultados son de mejor calidad, principalmente en cuanto a la dispersión de los datos, pero no existe demasiada diferencia con el caso inmediatamente anterior, correspondiente a 20 hormigas y 300 iteraciones.

Con esto último se deduce que a la hora de aplicar el algoritmo se debe decidir si se prefieren resultados de mejor calidad aumentando el coste computacional, o si con unos resultados razonablemente buenos es suficiente, de forma que se reduzca el coste computacional.

Cabe mencionar que el aumento en una unidad de la variable número de hormigas es más significativo en los resultados que el número de iteraciones. De esta forma, para obtener mejoras en los resultados, aumentando el número de iteraciones se deben realizar aumentos mínimos de múltiplos de 10.

Esta variación de los resultados obtenidos utilizando los mismos valores para las variables principales se deben a la aleatoriedad del Esto quiere decir que puede ocurrir que no se obtenga la misma solución en todas las ocasiones.

La calidad de la solución depende del valor de las variables, puesto que en función de éstas se comprobarán más o menos opciones, ya que cuantas más hormigas se utilicen más posibles soluciones se comprobarán en cada iteración y cuantas más iteraciones se hagan más posibles soluciones se comprobarán en general, y del factor aleatorio del algoritmo. Debido a esto, para obtener la solución óptima es necesario seleccionar adecuadamente el valor de estas variables. Esto se ha comprobado en las diferentes ejecuciones de este caso, ya que a medida que aumentaban los valores los resultados eran mejores.

Por último, cabe mencionar que, en algunas ocasiones, se pueden obtener el resultado óptimo con valores bajos en las variables o en muy pocas iteraciones, pero se debe tener en cuenta que esto únicamente ocurre por la aleatoriedad del algoritmo.

5.1.3 Modificación de las variables

Como se ha indicado anteriormente, se ha realizado el caso complejo para distintos valores en las variables número de hormigas y número de iteraciones. Se ha ejecutado el algoritmo utilizando los valores de número de hormigas 5, 20 y 40 y para el número de iteraciones se han utilizado 20, 100, 300 y 400.

Como regla general se ha comprobado que a medida que se aumenta el valor de estas variables el resultado obtenido es mejor, de forma que para un valor suficientemente grande en todas las ocasiones se alcanzaría el valor óptimo.

El incremento de estas variables incide directamente en un aumento del coste computacional. Por lo que en una aplicación real se debe valorar si es necesario obtener en todas las ocasiones el valor óptimo, o si, por el contrario, resulta más interesante obtener una buena solución que en algunas ocasiones puede no ser la óptima, pero obtener los resultados en menor tiempo.

Además, se ha comprobado que únicamente aumentando una de las variables los resultados ya mejoran, y que si se aumenta una variable y la otra se disminuye los resultados no empeoran; de forma que, por ejemplo, si se disminuye el número de hormigas, esta disminución se puede compensar aumentando el número de iteraciones para que las soluciones obtenidas no empeoren.

5.1.4 Conclusiones obtenidas

En los dos casos para los que se ha ejecutado el algoritmo los resultados obtenidos han sido satisfactorios. Además, se ha comprobado que la calidad de los resultados mejora a medida que se aumentan los valores de las variables número de hormigas e iteraciones.

El valor de estas variables depende del tamaño del problema a resolver. Para este caso en cuanto mayor sea el número de localidades para las cuales se quiera obtener la ruta, mayor será el valor que deban tomar estas variables.

Como valores generalizados, según la bibliografía consultada y los resultados obtenidos, los valores del número de hormigas se recomiendan que se encuentren entre 20 y 50, mientras que el número de iteraciones debe estar entre 300 y 600. Para casos en los que el tiempo de procesado no sea muy elevado se recomienda realizar varias pruebas con distintos valores para las variables, de forma que se puedan seleccionar los valores que sean adecuados en cada caso en concreto.

5.2. ANÁLISIS DEL ALGORITMO PARA LOGÍSTICA DE ALMACENES

En este apartado se presenta el análisis de los resultados obtenidos en el capítulo 4 en el caso básico, ya que el caso complejo no fue posible ejecutarlo completamente. Para este caso se realizó una simulación completa del algoritmo, ejecutando las 500 iteraciones recomendadas y varias simulaciones no completas modificando el número de hormigas.

Se ha realizado una única simulación completa debido al tiempo de ejecución del algoritmo en el equipo utilizado, el cual ha tardado unas 100 horas en completarse (más de 4 días). Debido a esto no se han podido realizar más simulaciones completas, puesto que el equipo de trabajo no se tenía con disponibilidad completa, y más en las circunstancias de la pandemia acontecidas durante la realización de este trabajo.

5.2.1 Modificación del número de hormigas

Para este caso se ha ejecutado varias veces con diferente número de hormigas, aunque no en todas se ha podido ejecutar el algoritmo completo, sino únicamente unas cuantas iteraciones.

Finalmente se ha ejecutado el modelo completo con 40 hormigas, ya que se considera un número adecuado y suficiente para obtener resultados fiables.

Se ha probado con más hormigas, 50, 60 y 80 pero los resultados no variaban demasiado respecto a 40, aumentando de manera importante el tiempo de procesamiento, por lo que se considera que para este caso con 40 hormigas es suficiente.

También se ha ejecutado el algoritmo con menos hormigas, 20, 15, 10 e incluso 4. En estos casos se considera que el número de hormigas es insuficiente, puesto que a pesar de bajar el tiempo de procesamiento, no se realizan suficientes caminos distintos, por lo que el mejor camino obtenido no es válido. Además, dedica mucho más tiempo entre una iteración y otra para mejorar el camino, por lo que en este caso disminuir el número de hormigas implicaría un aumento del número de iteraciones para obtener resultados razonables, de forma que el ahorro de coste computacional producido por la disminución de hormigas no es relevante, ya que esta disminución se compensaría con el aumento de número de iteraciones.

Esto se ha comprobado con el caso básico, que es un caso de dimensiones reducidas. Al tomar casos de mayor tamaño es posible que sea necesario aumentar el número de hormigas; aunque a partir de la bibliografía consultada y tras las distintas pruebas realizadas no se recomienda utilizar más de 50 o 60 hormigas, ya que estos números influyen enormemente en el tiempo de procesamiento. Así mismo no se considera oportuno ejecutar el algoritmo con un número muy reducido de hormigas.

5.2.2 Modificación del número de iteraciones

Para comprobar las variaciones de modificar el número de iteraciones es necesario ejecutar el algoritmo completo, y debido al tiempo de ejecución de éste no ha sido posible realizar gran cantidad de pruebas para este propósito.

Sin embargo, en la ejecución del caso simple que se realizó completa para un total de 500 iteraciones se considera que no hubieran sido necesarias tantas, sino que con unas 400 hubiera sido suficiente. En dicho caso se dejó terminar el algoritmo con las 500 iteraciones para, al menos, ejecutar el caso completo.

Al igual que en el número de hormigas, el número de iteraciones depende del tamaño del caso a resolver, de tal forma que para casos sencillos se necesitan un menor número de iteraciones y para casos más complejos va aumentando el número de iteraciones necesarias.

Finalmente se considera que, para un problema de tamaño medio, a los que está destinado este algoritmo, se recomiendan mínimo unas 500 iteraciones para obtener un buen resultado.

5.2.3 Análisis del paso por puntos obligatorios

Este es uno de los aspectos más importantes de este análisis, puesto que si el algoritmo no pasa por todos los puntos considerados como obligatorios los resultados no serán válidos. Además de esto, se debe comenzar en el punto seleccionado como inicio y acabar en el punto final. Si se cumple todo ello, los resultados obtenidos son válidos, pero en caso contrario no.

Respecto al paso por los puntos obligatorios, lo cual es una de las funciones principales del algoritmo, se comprueba que en todos los casos el recorrido obtenido pasa por todos los puntos obligatorios en todas las simulaciones realizadas, tanto en la que se ha ejecutado completamente como en las que se han detenido sin terminar todas las iteraciones, independientemente del número de hormigas.

Una vez observado todo esto, se puede concluir que el algoritmo no produce fallos por no pasar por alguno de los puntos obligatorios, además de comenzar en todos los casos en el inicio y acabar en el punto final.

5.2.4 Conclusiones del análisis

A pesar de haber podido realizar únicamente una simulación completa, el resultado ha sido satisfactorio, ya que se han obtenido resultados coherentes, sin ocurrir fallos durante la ejecución del algoritmo.

Una vez se han realizado las distintas simulaciones se puede concluir que el algoritmo es válido y fiable; y que con los medios adecuados sería posible utilizar este algoritmo para casos de mayores dimensiones o complejidad.

5.2.5 Análisis del caso complejo

Para el caso complejo no ha podido realizarse una simulación completa con todas las iteraciones necesarias, ya que el tiempo de computación es muy elevado. Se ha ejecutado el algoritmo para unas 30 iteraciones para comprobar que funciona correctamente y se obtienen resultados coherentes, aunque no se ha obtenido un resultado óptimo, ya que se han realizado muy pocas iteraciones. Se considera que si se hubiera podido ejecutar el algoritmo completo con las 500 iteraciones recomendadas, se hubiera llegado a una buena solución.

Además, se ha ejecutado únicamente con 10 hormigas, a pesar de que según la bibliografía consultada se recomiendan unas 50, para poder obtener unas cuantas iteraciones en un tiempo razonable. Se ha ejecutado el algoritmo durante aproximadamente 36 horas.

Se considera que si se tienen los medios adecuados se podría utilizar este método, ya que no aparecen errores al ejecutar, pero no es posible comprobar una ejecución del algoritmo completo con el número de hormigas y de iteraciones recomendados.

5.3 ANÁLISIS CONJUNTO

En este apartado se muestran por una parte los resultados de un análisis conjunto del caso de la logística de transporte, visto en el capítulo 3, y el caso de la logística de almacenes, correspondiente al capítulo 4. Y también se exponen las conclusiones conjuntas que se pueden obtener analizando ambos casos, ya que parten del mismo planteamiento de optimización por colonia de hormigas.

5.3.1 Principales diferencias entre los algoritmos

Los dos programas en Matlab que se han utilizado en este trabajo parten de la misma filosofía, un algoritmo de optimización por colonia de hormigas, por lo que tienen una parte común, en ambos casos. Sin embargo, hay varias diferencias destacables entre ambos.

Los dos casos tienen en común la base del algoritmo ACO, además que se han realizado de forma similar, utilizando el programa en Matlab y un fichero Excel para

pasar los datos necesarios de manera simple. Además, los dos casos proporcionan rutas o recorridos a seguir.

Pese a partir de la misma base, existen diferencias destacables entre ambos casos, la cuales se comentan a continuación. Una de las diferencias principales son los tipos de datos que se proporcionan al algoritmo. En el caso de los camiones se proporcionan los datos del tiempo que se tarda en ir de un punto a otro, sin importarnos el recorrido que siga o la velocidad, únicamente teniendo en cuenta la característica del tiempo, de forma que el mejor resultado es el que haga la ruta en menor tiempo.

En el caso de los AGVs se proporcionan la ubicación de los puntos en coordenadas cartesianas, teniendo en cuenta que para este caso la velocidad será constante, por lo que el tiempo irá directamente relacionado con la longitud del recorrido. En este caso el resultado a obtener es el recorrido que debe seguir el AGV para pasar por todos los puntos necesarios haciendo la ruta más corta.

Esta diferencia es importante ya que la entrada de datos pasa de un vector en el primer caso a dos vectores en el segundo. Además de que el resultado obtenido es completamente diferente.

En el caso del transporte el programa proporciona únicamente un vector con el orden a seguir en la ruta, lo que se puede traducir directamente en el orden de las localidades, puesto que cada una tiene un número asignado,

En el caso de los AGVs se obtiene la ruta por medio de un vector indicando el orden de los puntos por los que pasa, pero, además, se obtiene gráficamente el recorrido a seguir, resaltando los puntos importantes con un código de colores, lo que proporciona soluciones muy visuales, y por tanto eficaces para las empresas de logística.

El representar la solución gráficamente no procede en el primer caso, puesto que únicamente se necesitan los datos de tiempo independientemente de la posición que ocupe cada una de las localidades, por lo que la ubicación no se tiene en cuenta en este caso. Esto se hace porque la velocidad no es constante, sino que puede ocurrir que sea menor el tiempo entre dos localidades más alejadas que entre dos que se encuentran más próximas pero que, debido a la comunicación entre ellas por el tipo de vía, el tiempo que se tarda en ir de una a otra sea mayor.

Por último, otra de las grandes diferencias entre ambos casos es que en el caso de los camiones el algoritmo debe pasar por todos los puntos (localidades), mientras que en el caso de los AGVs existen puntos obligatorios por los que debe pasar el algoritmo y puntos por los que es posible pasar para comunicar los obligatorios, pero que, en caso de no ser necesarios, no se utilizarán. En relación con esto también se debe tener en cuenta que en el caso de los camiones únicamente se pasa una vez por cada punto, pero en el segundo caso, de AGVs, puede que la ruta utilice varias

veces los mismos puntos, lo cual permite ir a ubicaciones en las que la entrada y la salida transcurre por el mismo pasillo.

5.3.2 Modificación de las variables

La modificación de las variables de número de iteraciones y número de hormigas se ha analizado en cada uno de los casos. En este apartado se muestran las conclusiones obtenidas tras realizar dos casos diferentes de un algoritmo ACO.

Se ha comprobado que el valor que se debe dar a estas variables aumenta a medida que aumentan las posibles soluciones, ya que cuanto mayores sean estas variables más soluciones se comprobarán. Sin embargo, no es necesario comprobar todas las soluciones, porque debido a las “feromonas” del algoritmo, éste busca nuevas soluciones partiendo principalmente de buenos resultados. Aun así, debido a la parte aleatoria del algoritmo, también busca soluciones independientes a éstas, por lo que puede encontrar un mejor recorrido que no esté relacionado con recorridos anteriores.

Además, se ha comprobado, principalmente a partir de primer caso en el que se hicieron varias pruebas, que en ocasiones no es necesario aumentar las dos variables, sino que aumentando únicamente una de ellas es suficiente para mejorar los resultados obtenidos. Esto se debe a que al aumentar cualquiera de ellas se están aumentando el número de posibles soluciones que se comprueban.

Sin embargo, sí que existen unos valores recomendados para este tipo de algoritmos. En el caso del número de hormigas no es recomendable utilizar menos de 20 ni más de 60 en casos complejos. Esto se debe a que un valor inferior perdería parte de la efectividad del algoritmo de comprobar un número elevado de posibles soluciones por iteración, y un valor muy elevado en el número de hormigas aumenta considerablemente el tiempo de computación, llegando a cierto punto en el que no resulta interesante aumentar más esta variable sino optar por aumentar el número de iteraciones.

En el caso de la variable número de iteraciones funciona de manera similar al número de hormigas, ya que cuantas más iteraciones se realicen mayor cantidad de posibles soluciones se comprobarán. Los valores recomendados para esta variable, siempre dependiendo del tamaño del problema a resolver, pueden ir como mínimo de unas 250 o 300 iteraciones para obtener buenos resultados en casos sin demasiada complejidad, a 600 o 700 iteraciones para casos muy complejos. Siempre hay que tener en cuenta que cuantas más iteraciones se realicen, mejor calidad tendrá la solución, pero mayor será el coste computacional. Por tanto si se considera que no es necesario obtener la solución óptima, sino que con una buena solución es suficiente, se puede utilizar un número de iteraciones menor para obtener los resultados en menor tiempo. Siendo este criterio uno de los fundamentos de economía de recursos aplicado en el sector ingenieril.

5.3.3 Conclusiones del análisis global

Como conclusiones finales tras la realización de los dos tipos de casos y una vez expresado cada aspecto destacable en los apartados anteriores, cabe mencionar que los resultados obtenidos han sido satisfactorios en ambos casos.

Sin embargo, cabe mencionar que han sido más interesantes los resultados obtenidos en el caso de los camiones, puesto que al tener menor coste computacional y ser un algoritmo más ligero, se ha podido ejecutar completo varias veces utilizando distintos valores, de forma que se han podido obtener mejores resultados, los cuales han sido analizados detenidamente en los apartados anteriores.

En el caso de los AGVs, pese a no dar problemas el equipo utilizado no tenía suficiente potencia, por lo que se han obtenido menos resultados. Sin embargo, todos los resultados obtenidos son coherentes con lo esperado de la ejecución del algoritmo y no ha dado errores, únicamente no ha sido posible ejecutarlo completamente debido al tiempo necesario para hacerlo con los medios de los que se ha dispuesto en este trabajo.

5.4. RECOMENDACIONES PARA LOS ALGORITMOS ACO

En este apartado se presentan las principales ventajas e inconvenientes que tiene emplear la aplicación de ACO desarrollada, así como posibles aplicaciones en otros casos.

5.4.1 Ventajas

- Obtención de rutas para camiones de forma muy eficiente una vez se tenga una base de datos con los tiempos que se tarda en ir de cada una de las localidades a todas las demás.
- Con la base de datos verificada permite definir diariamente la ruta personalizada, dependiendo de las localidades a recorrer cada día.
- Genera soluciones precisas siempre y cuando los medios de simulación sean adecuados y el tiempo de ejecución apto.
- Permite obtener las rutas previstas en una etapa de diseño, ya que indica un camino pseudo-óptimo para el recorrido de un AGV a partir del plano de implantación y los programas utilizados,
- Flexibilidad y comodidad para definir rutas de AGVs ante cambios en implantaciones.
- Para el caso de los AGVs permite esquivar obstáculos, siempre y cuando exista un centro computacional potente, ya que se podría ejecutar el algoritmo eliminando el punto que tiene el obstáculo de los puntos posibles, de forma que se obtenga un nuevo recorrido en tiempo real.
- Solventa problemas de callejones sin salida en vehículos autónomos, dotándoles de un mayor grado de inteligencia.

5.4.2 Inconvenientes

- Pese a la precisión del algoritmo, sobre todo cuando el número de iteraciones y hormigas es elevado, el algoritmo se sustenta en la pseudo-aleatoriedad y por tanto, no se obtiene en todos los casos la solución óptima, sino que en algunos casos se obtienen soluciones muy buenas sin llegar al óptimo.
- El tiempo de Ejecución del algoritmo es muy elevado, por lo que sería necesario contar con unos equipos que cuenten con una capacidad de computación elevada.
- Para la utilización de los algoritmos ACO es necesario tener conocimiento de los programas informáticos empleados basados en este método metaheurístico.

5.4.3 Aplicaciones

Aplicando el algoritmo ACO a empresas de reparto de mercancías, y teniendo como dato la distancia o el tiempo de tránsito entre distintos nodos, podemos visualizar algunos ejemplos ilustrativos.

Por ejemplo, como se ve en la figura 54.

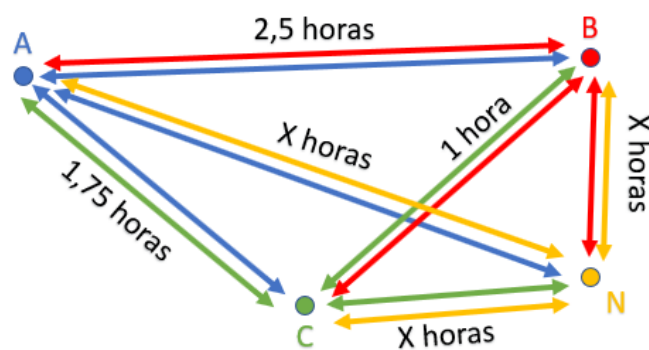


Figura 54 Esquema correspondiente al ejemplo de reparto (Fuente: Elaboración propia)

El algoritmo nos indicaría el camino óptimo para hacer el reparto.

Existen muchas empresas que actualmente diseñan las rutas de forma manual de sus camiones de reparto el día anterior al día de reparto. Aplicando el Algoritmo ACO indicando el tiempo de transporte entre cada uno de sus clientes, se resolvería el problema de una forma eficiente. Suponiendo, además, unos ahorros importantes en tiempo de planificación y solventando posibles errores humanos en el trazado.

- Diseño de un sistema de AGVs para cualquier industria a nivel de estudio. Es decir, permite definir cuáles son las rutas óptimas de dichos vehículos

cuando estamos diseñando una planta. Si tenemos los caminos de los vehículos predefinidos, el algoritmo nos indicará cual es el orden más adecuado que debe seguir el AGV. Si no se dispone de los caminos definidos, pero sí se es consciente de los lugares por los que no pueden pasar los AGVs el algoritmo nos indica cuál es el mejor camino. En este caso como se dispone de tiempo de planificación (días) se puede ejecutar una simulación con muchas hormigas y muchas iteraciones de tal manera que tendremos un resultado preciso.

- Para sistemas más avanzados, que requieren un coste elevado de computación, se puede implementar un sistema de AGVs que evite obstáculos creando rutas alternativas.
Si se dispone de una central que controla los AGV, cuando un AGV se ha detenido debido a un obstáculo en su ruta, mientras espera a que un “pastor de AGVs”, persona encargada de ir a revisar que ocurre cuando un AGV se ha parado, eliminar los obstáculos en el camino y activar el AGV para que vuelva a ponerse en marcha, resuelva el obstáculo, éste enviaría una señal a la central, quien ejecutaría el Algoritmo ACO indicándole el bloqueo. Si el algoritmo es capaz de encontrar una ruta alternativa en el tiempo en el que está bloqueado, el AGV podría continuar.
- En rutas de AGVs ya establecidas, como por ejemplos los robots de limpieza. Se pueden mejorar las rutas haciendo recorridos óptimos de manera que se minimiza el tiempo y la distancia de recorrido, permitiendo de esta manera un ahorro energético o por consiguiente, un mayor alcance de distancia con la misma batería.
El algoritmo ACO puede ser lanzado en los tiempos de carga y/o descanso del dispositivo y de esta manera no se requiere un costo computacional a mayores.

6 ESTUDIO ECONÓMICO

Este capítulo está propuesto para presentar el coste económico que supondría realizar las diferentes etapas del proyecto desarrollado en este Trabajo Fin de Grado, en caso de haber sido desarrollado de forma profesional.

Este Trabajo Fin de Grado tiene como objetivo la investigación, diseño y desarrollo de una serie de algoritmos en Matlab, con la ayuda de hojas de cálculo Excel para la introducción de los datos, para poder definir rutas óptimas de AGVs a nivel de diseño de un proyecto, así como establecer el mejor orden en el que se debería llevar a cabo el transporte de mercancías en el ámbito de la logística industrial.

Por ser parte esencial de este trabajo el desarrollo de algoritmos para varios casos de logística, los costes que se valorarán serán únicamente los que derivan del tiempo y del coste de las licencias que se han empleado durante el diseño y desarrollo del mismo, así como el equipo (ordenador personal, puesto que dadas las circunstancias excepcionales de 2020, no se ha podido contar con otros equipos más potentes de computación).

6.1 DESCRIPCIÓN DE LAS ETAPAS DE DESARROLLO

En este apartado se van a indicar las distintas etapas que tiene el proyecto.

- **Etapas 1: Análisis e investigación previos**

En esta etapa se realiza un análisis del problema. Se busca información acerca de cómo integrar métodos metaheurísticos en la industria (concretamente el de Colonia de hormigas), se llega a la conclusión de la posible utilidad y viabilidad del método en la generación del rutado de AGVs, así como la posibilidad de emplearlo para la gestión de recursos de transporte en las que se tiene un vehículo que ha de realizar un recorrido entre diferentes puntos.

Durante esta etapa se recopila información bibliográfica científico-técnica y se marca la línea de trabajo, que irán siendo modificados a medida que evoluciona el mismo. En esta etapa se delimita el alcance y las líneas generales de actuación en el trabajo.

- **Etapas 2 Análisis en profundidad y acopio de Recursos**

Una vez se ha abordado el problema de forma general, se procede a un análisis más exhaustivo.

Se busca, estudia y analiza bibliografía rigurosa acerca de los distintos métodos de optimización existentes, así como del método seleccionado para este proyecto, método de optimización por colonia de hormigas. En principio el trabajo se basa en aplicaciones simples de Colonias de Hormigas

programadas, a partir de las cuales poder comprender su funcionamiento y desarrollar una aplicación más compleja.

Es imprescindible analizar la viabilidad del proyecto en este punto, ya que supondría un aumento considerable en los costes tener que retroceder hasta el principio una vez se inicie el desarrollo de la aplicación.

• **Etapa 3 Diseño de los programas**

En esta etapa se realizan los esquemas del núcleo del funcionamiento de los programas, se define la necesidad de emplear ficheros Excel que sirvan como entrada de datos, así como las necesidades generales que se deben exigir al código.

• **Etapa 4 Desarrollo de los programas**

Una vez definidos los parámetros del diseño, se desarrollan los programas definitivos a partir de las aplicaciones básicas obtenidas durante la etapa 2.

En esta etapa se prueba también el funcionamiento de los programas para comprobar que es correcto. Este testeo consiste en someter a los programas a pruebas con el fin de detectar errores, los cuales se corregirán en caso de existir, y es la forma de garantizar el correcto funcionamiento de los algoritmos programados.

• **Etapa 5 Obtención de resultados y Análisis**

En esta etapa se realizan todos los distintos casos, cuatro para este trabajo, y se obtienen los resultados proporcionados por el algoritmo en cada uno de ellos. También se analizan los resultados obtenidos, caso por caso, y conjuntamente.

• **Etapa 6 Edición de la Documentación**

En esta última etapa se realiza la redacción de la documentación, en este caso la redacción de este Trabajo Fin de Grado. En esta documentación aparece detallado lo realizado en todas las etapas anteriores, por lo que se hace en paralelo a las demás etapas.

A continuación, en la tabla 8, se muestra un diagrama de Gantt en el que se puede ver con más detalle la duración de cada una de las etapas.

Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Etapa 1 Análisis e investigación previos																								
Etapa 2 Análisis en profundidad y acopio de Recursos																								
Etapa 3 Diseño de los programas																								
Etapa 4 Desarrollo de los programas																								
Etapa 5 Obtención de resultados y análisis																								
Etapa 6 Edición de la Documentación																								

Tabla 8 Diagrama de Gantt, etapas del proyecto (Fuente: Elaboración propia)

6.2 CÁLCULO DE LOS COSTES

A continuación, se realiza un cálculo de las horas efectivas anuales para después determinar las tasas por hora del salario. Los días efectivos de trabajo anual se calculan en función de datos recopilados de años anteriores.

De esta manera, se puede establecer el número de horas necesarias que se deberían contratar para la realización de este trabajo y con ello poder calcular posteriormente el coste en personal. Se utilizará para el cálculo un solo trabajador, (en el caso de querer acelerar el desarrollo del trabajo se podrían contratar más de uno).

Días laborales empleados	240	Días
Fines de Semana	67	Días
Festivos	8	Días
Total estimado días efectivos	165	Días
Total estimado horas (4 horas/día)	659	Horas

Tabla 9 Horas efectivas trabajadas (Fuente: Elaboración propia)

En la Tabla 9 se recoge la estimación de las horas efectivas trabajadas en los ocho meses que se ha realizado la ejecución de este trabajo (estimado desde enero de 2020 hasta septiembre de 2020, sin incluir periodos vacacionales).

6.2.1 Costes de personal

Los costes de personal van a ser los más importantes y grandes del proyecto, ya que se trata de un trabajo de investigación donde solamente es necesario horas humanas, un equipo informático y el software específico para el desarrollo, así como software de ofimática.

Además, al ser realizado este proyecto en una universidad que dispone de los medios de software, la realización de este trabajo se vuelve mucho menos costosa, ya que los costes serían únicamente humanos, pese a ello se van a evaluar también los gastos en equipo y licencias de software.

Se supone que el personal contratado para realizar este trabajo es un ingeniero junior para realizar tanto la programación, las pruebas y el análisis de resultados como la redacción de la memoria.

En la Tabla 10 se hace una estimación de las horas invertidas en cada etapa del desarrollo del trabajo.

ETAPA	INGENIERO JUNIOR
Etapa 1 Análisis e investigación previos	35
Etapa 2 Análisis en profundidad y acopio de Recursos	94
Etapa 3 Diseño de los programas	33
Etapa 4 Desarrollo de los programas	95
Etapa 5 Obtención de resultados y análisis	108
Etapa 6 Edición de la Documentación	295
TOTAL HORAS	660

Tabla 10 Horas dedicadas a cada etapa (Fuente: Elaboración propia)

Una vez establecido el número de horas empleadas en cada etapa, se puede proceder al cálculo del salario. En la Tabla 11 se ven reflejados el salario total y la retribución horaria.

REMUNERACIÓN	INGENIERO JUNIOR
Sueldo bruto anual	21.000,00
Sueldo bruto en periodo considerado	14.000,00
Seguridad social -35%	4.900,00
Coste total de la Empresa	18.900,00
COSTE/ HORA	28,64

Tabla 11 Salarios en euros (Fuente: Elaboración propia)

A continuación, se unifica la información de las dos tablas anteriores (tabla 10 y tabla 11) con el objetivo de visualizar los costes finales por etapa del TFG. Esto se puede ver en la tabla 12.

ETAPA	COSTE PERSONAL POR ETAPA	% SOBRE TOTAL PROYECTO
Etapa 1 Análisis e investigación previos	1.002,27	5,30%
Etapa 2 Análisis en profundidad y acopio de Recursos	2.691,82	14,24%
Etapa 3 Diseño de los programas	945,00	5,00%
Etapa 4 Desarrollo de los programas	2.720,45	14,39%
Etapa 5 Obtención de resultados y análisis	3.092,73	16,36%
Etapa 6 Edición de la Documentación	8.447,73	44,70%
TOTAL	18.900,00	100%

Tabla 12 Costes por etapas en euros (Fuente: Elaboración propia)

6.2.2 Costes de amortización

Además de los costes de personal, existen otro tipo de costes, los cuales pese a ser significativamente menores, han de tenerse en cuenta.

Entre ellos se encuentran los que se muestran en este apartado y los dos siguientes.

En primer lugar, se encuentran los costes de amortización de equipos y programas software utilizados para el desarrollo del proyecto. En la tabla 13 se encuentran estos costes estimados y la amortización (considerada lineal a cuatro años) del hardware y software empleados.

EQUIPO	COSTE
Ordenador portátil HP ENVY m6 Monitor Philips Ratón HP	694,00
Microsoft Windows 10 Pro	259,00
Microsoft Office 365	149,00
MatLab 2017	2.000,00
AutoCAD 2020	2.227,00
Total	5.329,00
Amortización anual	1.332,25
Amortización en el periodo considerado	888,17

Tabla 13 Coste y amortización de los equipos en euros (Fuente: Elaboración propia)

Como se ha indicado anteriormente, al realizarse este trabajo en una universidad que dispone de los medios software necesarios, se han aplicado los costes indicados anteriormente que corresponden a los costes que tendría una empresa (ya que las licencias para universidades suponen costes menores, especiales para dichas instituciones).

6.2.3 Costes generales

Finalmente, se indican los costes indirectos producidos por los recursos consumidos durante la realización de este trabajo. Estos costes se reflejan en la tabla 14.

RECURSO	COSTE
Alquiler de oficina	659,20
Electricidad	144,00
Internet	120,00
Material consumible de oficina	25,00
TOTAL	948,20

Tabla 14 Costes generales en euros (Fuente: Elaboración propia)

Estos costes corresponden a una estimación del precio del alquiler y los servicios necesarios en una oficina de una empresa privada, en un edificio destinado para oficinas, durante los días estimados de duración del proyecto.

6.3 COSTES TOTALES

En este apartado se indica el coste total estimado de la realización de este TFG. Este coste se calcula a partir de la suma de los diferentes costes indicados en los apartados anteriores. Además, se indica el porcentaje del total que supone cada uno de ellos. A continuación, se muestra en la tabla 15 la información correspondiente a los costes, y en la figura 55 un gráfico de sectores en el que se puede apreciar el peso que tiene cada uno de los costes sobre el total.

CONCEPTO	COSTE	% SOBRE EL TOTAL DEL PROYECTO
Costes de personal	18.900,00	91,14%
Costes de amortización	888,17	4,28%
Costes generales	948,20	4,57%
TOTAL	20.736,37	100%

Tabla 15 Coste total del proyecto en euros (Fuente: Elaboración propia)

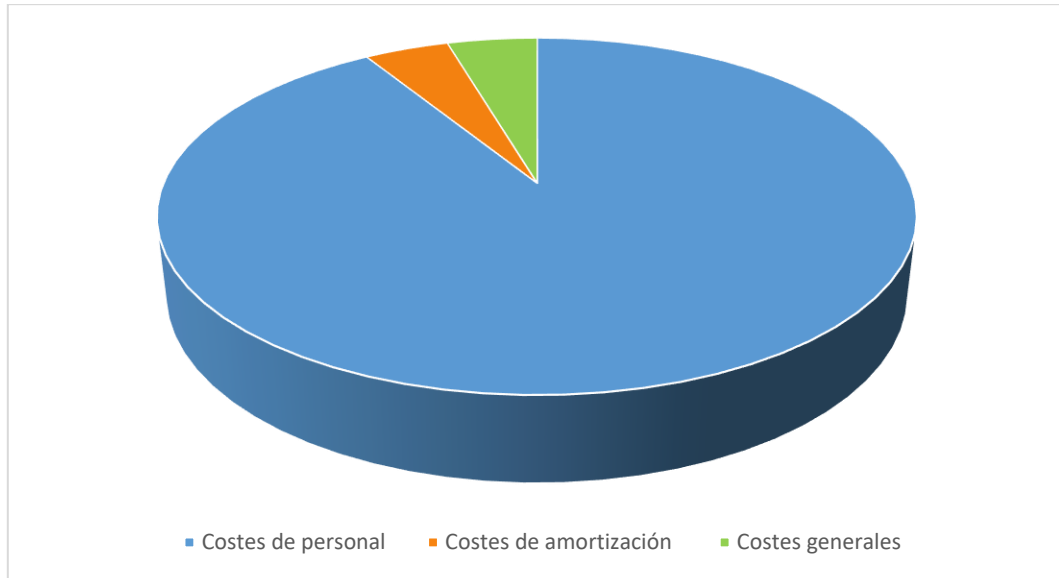


Figura 55 Gráfico de costes totales del proyecto (Fuente: Elaboración propia)

Como se indica al principio de este análisis económico, para este proyecto los costes más importantes son los correspondientes a los costes de personal.

6.4. CÁLCULO DEL PRECIO DE VENTA DEL PROYECTO

En este último apartado se indica el precio de venta que debería tener este trabajo, como proyecto, planteando un beneficio del 12%. Esto se ha calculado utilizando los costes estimados anteriormente e indicando también el IVA del 21% que se aplica a la venta del proyecto para obtener el precio de venta final.

Estos datos se indican en la tabla 16.

Coste total estimado	20.736,37
Beneficio del 12%	2.488,36
Total	23.224,73
IVA (21%)	4.877,19
Precio de venta final	28.101,92

Tabla 16 Precio de venta en euros (Fuente: Elaboración propia)

7 CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se van a indicar las conclusiones obtenidas tras la realización de este Trabajo Fin de Grado, y las futuras líneas de investigación y trabajo, que podrían iniciarse tras la finalización de este trabajo.

7.1 CONCLUSIONES

Como conclusiones tras la realización de este trabajo se presentan las siguientes.

Se puede concluir que los métodos metaheurísticos por colonia de hormigas resultan interesantes para la obtención de buenas soluciones a problemas de optimización de rutas, pese a no ser de los más utilizados. En este tipo de problemas resultan adecuados, ya que la base principal del algoritmo es la obtención de rutas óptimas.

En este trabajo los algoritmos ACO se han aplicado a dos casos relacionados con la logística y en concreto, el transporte, en la industria, para los cuales los resultados han sido satisfactorios.

No obstante, cabe mencionar que el caso referido a rutas para camiones ofrece resultados que han sido mejores, puesto que el programa utilizado tenía menor coste computacional y por lo tanto se han podido realizar más experimentos. Tras la realización de un análisis de las distintas ejecuciones se han obtenido conclusiones acerca de la relación de los resultados y la calidad de las soluciones obtenidas en función de los valores de las principales variables del algoritmo, comprobando que dicha calidad mejora al aumentar los valores de las variables principales del algoritmo.

El segundo caso analizado también ha proporcionado resultados favorables, pero se necesitarían equipos con mayor potencia para realizar más experimentos, puesto que con los medios utilizados no se han podido obtener todos los resultados deseados. Sin embargo, sí que se ha podido realizar un análisis a partir de los resultados obtenidos ya que se trata de resultados coherentes y sin errores más allá de que no se hayan podido ejecutar el algoritmo completamente tantas veces como se desearía.

Finalmente, he de indicar que para ambos casos se puede concluir que la utilización de una metodología ACO para este tipo de situaciones resulta adecuado, ya que proporciona buenos resultados.

7.2 LÍNEAS FUTURAS

En este Trabajo Fin de Grado se ha estudiado la utilidad de métodos de optimización por colonia de hormigas para dos casos concretos relacionados con el transporte en la industria. Como futuras líneas de investigación relacionadas con el presente trabajo se mencionan las siguientes:

- Aplicación de algoritmos ACO a otros casos relacionados con la industria aparte de los vistos en este trabajo, como optimización de procesos.

CONCLUSIONES Y LÍNEAS FUTURAS

- Utilización de un método metaheurístico diferente al utilizado en este trabajo, como los algoritmos genéticos, ya que son un método de los más conocidos y estudiados, para la resolución de casos similares y comparación de los resultados.
- Empleo de sistemas de computación potentes, para continuar el trabajo realizado en este Trabajo Fin de Grado, en la aplicación de los algoritmos ACO a casos reales.

8 BIBLIOGRAFÍA

Acosta Buitrago, María Isabel y Zuluaga Muñoz, Camilo Alfonso (2000). Tutorial sobre redes neuronales aplicadas en ingeniería eléctrica y su implementación en un sitio web. Proyecto de grado de Ingeniería eléctrica. Universidad Tecnológica de Pereira.

Alonso, Sergio & Cordon, Oscar & Viana, Iñaki & Herrera, Francisco. (2003). La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques. Trabajo realizado en el marco del proyecto “Mejora de Metaheurísticas mediante Hibridación y sus Aplicaciones” de la Universidad de Granada.

ATRIA INNOVATION, sitio web de un fabricante de AGVs Url: <https://www.atriainnovation.com/agvs-vehiculos-de-guiado-automatico-todavia-no-sabes-lo-que-son/>, última visita realizada: 6 abril 2020.

Bautista-Valhondo, Joaquín & Companys, Ramon & Corominas, Albert & Mateo, Manuel. (1999). GRWASP: una generalización de los algoritmos GRASP. Aplicación al ORVP. A: III Jornadas de Ingeniería de Organización: Barcelona, 16 y 17 de septiembre de 1999. Centre de Publicacions d'Abast (CPDA - ETSEIB).

Cizon, sitio web de un fabricante de AGVs Url: <http://www.cizon.com.cn/en/chanpin/lm4/29.html>, última vista realizada: 10 abril 2020.

Díaz, A., Glover, F., Ghaziri, H.M., et al, (1996) Optimización Heurística y Redes Neuronales. Madrid, Editorial Paraninfo.

Diego-Mas, José Antonio (2006). Optimización de la distribución en planta de instalaciones industriales mediante algoritmos genéticos. Aportación al control de la geometría de las actividades. Tesis doctoral. Universidad politécnica de Valencia.

Glover, F. (1986) Future Paths for Integer Programming and Links to Artificial Intelligence, Computers and Operations Research. 5.

Guerra, Diego (2015). Estado del arte y análisis de métodos de optimización de recursos en plantas de producción. Trabajo fin de grado en ingeniería electrónica industrial y automática. Universidad de Valladolid.

Kivnon, sitio web de un fabricante de AGVs Url:
<https://www.kivnon.com/agvcomplementos.html>, Última visita realizada: 10 abril 2020.

Machuca et al (1995). Dirección de Operaciones: Aspectos estratégicos. Editorial McGraw-Hill.

Martí, R (2003), Procedimientos Metaheurísticos en Optimización Combinatoria, Matemàtiques 1(1).

Melián, Belén; Glover, Fred (2007) Introducción a la Búsqueda Tabú. Procedimientos Metaheurísticos en Economía y Empresa. Tirant lo Blanch.

Morillo, Daniel, & Moreno, Luis, & Díaz, Javier (2014). Metodologías Analíticas y Heurísticas para la Solución del Problema de Programación de Tareas con Recursos Restringidos (RCPSP): una revisión. Parte 1. Ingeniería y Ciencia, 10(19).

Osman, Ibrahim & Kelly, James. (1996). Meta-Heuristics: An Overview. In: Osman I.H., Kelly J.P. (eds) Meta-Heuristics. Springer, Boston, MA.1.

Robles Algarín, C. A. (2010), "Optimización por colonia de hormigas: aplicaciones y tendencias" en Revista Ingeniería Solidaria, vol. 6, núm. 10.

Seyedali Mirjalili. Ant Colony Optimiztion (ACO)
(<https://www.mathworks.com/matlabcentral/fileexchange/69028-ant-colony-optimiztion-aco>), MATLAB Central File Exchange. Última visita realizada: 17 mayo 2020.

Soifer, A., & Loiseau, I. (2015). Algoritmos de colonia de hormigas para el problema del viajante de comercio por familias y para el problema de ruteo de vehículos por

familias. XIII Simposio Argentino de Investigación Operativa (SIO) - JAIIO 44 Rosario, 2015.

Vázquez Espí, M. (1995). Un nuevo algoritmo para la optimación de estructuras: el recocido simulado. *Informes de la Construcción*, 46(436).