APLICACIÓN DE COLONIA DE HORMIGAS BASADAS EN UNA HEURÍSTICA DE INSERCIÓN PARA UNA FAMILIA DE PROBLEMAS DE RUTEO DE HELICÓPTEROS

M. VLADIMIR ROSERO BERNAL

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERIA
DEPARTAMENTO DE INGENIERIA INDUSTRIAL
MAESTRIA EN INGENIERIA INDUSTRIAL
BOGOTA, D.C.
2005

APLICACIÓN DE COLONIA DE HORMIGAS BASADAS EN UNA HEURÍSTICA DE INSERCIÓN PARA UNA FAMILIA DE PROBLEMAS DE RUTEO DE HELICÓPTEROS

M. VLADIMIR ROSERO BERNAL

Trabajo de grado para optar el título de
Magíster en Ingeniería Industrial

Asesor
DR. FIDEL TORRES

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERIA
DEPARTAMENTO DE INGENIERIA INDUSTRIAL
MAESTRIA EN INGENIERIA INDUSTRIAL
BOGOTA, D.C.
2005

A

Dios, a mi madre, a mi padre y en especial, a Vane

AGRADECIMIENTOS

A Dios por todas las oportunidades que me ha presentado.

A mi madre y a mi padre por su constante y desinteresado apoyo.

Al Doctor Fidel Torres por su asesoría y ayuda en el desarrollo de este proyecto.

A todos mis compañeros por su apoyo incondicional.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

INTRODUCCIÓN

Los problemas de programación y de ruteo de vehículos, tienen una gran importancia desde el punto de vista práctico y de investigación; por un lado, la solución a estos problemas, pueden traer grandes ahorros a empresas que día a día buscan optimizar el uso de sus recursos para ser más competitivos, y por el otro, alienta a los investigadores a desarrollar nuevas metodologías que permitan resolver problemas de alta complejidad. Como un ejemplo de este tipo de problemas, se puede citar al Problema de Ruteo de Helicópteros con capacidad finita y con restricciones de precedencia.

En el problema de Ruteo de un Helicóptero, un número de solicitudes de transporte, tienen que ser atendidas, las cuales consisten de un nodo origen, un nodo destino y un número determinado de pasajeros a transportar. El objetivo de dicho problema, es encontrar una ruta que satisfaga las necesidades de transporte de todos los pasajeros, minimizando el tiempo o la distancia total recorrida, sin violación de las restricciones de precedencia y de capacidad.

En el presente trabajo, se propone dar solución a una familia de problemas reales de Ruteo de Helicópteros, mediante el uso de la Meta-heurística Colonia de Hormigas basada en una heurística de inserción. El uso de esta Meta-heurística, se fundamentó en varios aspectos, de los cuales se puede mencionar: 1) a pesar de ser un algoritmo relativamente nuevo (según Maniezzo, Gambardella & Luigi (2001), fue introducido por primera vez por Colorni, Dorigo y Maniezzo en los inicios de la década de los noventa), ha demostrado ser competitivo frente a otros algoritmos, cuando se ha probado en problemas de ruteo de vehículos (Dorigo & Gambardella, 1997a, 1997b; Gambardella, Taillard & Agazzi, 1999; Doerner, Hartl & Reiman, 2000); y 2) por ser un método heurístico, provee soluciones cercanas a las óptimas a problemas tipo NP-hard, a un bajo costo computacional.

El presente documento, se encuentra dividido en siete capítulos; en el primero, se señalará la importancia del problema a estudiar, los antecedentes, los objetivos y la metodología utilizada.

En el segundo capítulo, se hace una presentación general del problema de ruteo de helicópteros estudiado y de los procedimientos empleados para su solución.

En el tercer y cuarto capítulo, se expondrá respectivamente, la Meta-heurística Colonia de Hormigas y la heurística de inserción sobre la cual se basa la primera.

En el quinto capítulo, se hace una descripción detallada del algoritmo implementado, de sus extensiones y de los problemas utilizados. En el sexto capítulo se discuten los resultados y finalmente, en el séptimo se dan las conclusiones y recomendaciones para trabajos futuros.

1 CONSIDERACIONES GENERALES

Los Sistemas de Transporte utilizados hoy en día, son objeto de estudio por parte de los investigadores, debido a que de estos sistemas depende en gran medida el éxito de las empresas. Desde el transporte de artículos (materias primas o productos terminados) hasta el transporte del recurso humano, el sistema utilizado es crítico en el desempeño eficiente de la empresa.

Particularmente, el transporte de personas ha recibido mucha atención durante los últimos 30 años (Cordeau & Laporte, 2003). Se han estudiado sistemas tan simples como el servicio puerta a puerta, en donde el pasajero es recogido en su origen y llevado directamente a su destino, o sistemas tan complejos en donde un grupo de personas con distintos orígenes y destinos, comparten un mismo vehículo (Bergvinsdottir, 2004). Para estos últimos, hay ventajas desde el punto de vista económico, ya que disminuyen los costos de transporte, pero tienen la desventaja de ser un problema de difícil solución, que exigen técnicas avanzadas de procesamiento y un alto gasto computacional. Como ejemplos de estos sistemas, se puede citar el transporte escolar, el transporte de personas de la tercera edad y de discapacitados, el transporte de personal de trabajo, entre otros.

1.1 ANTECEDENTES

El tema de estudio del presente trabajo, está basado en la solución de 40 problemas reales de ruteo de helicópteros, los cuales son utilizados por una empresa para el transporte de su personal a distintas zonas geográficas. Hace poco tiempo, estos problemas de ruteo, eran solucionados por expertos programadores de rutas, quienes contaban con la experiencia y habilidad suficiente para encontrar buenas soluciones en periodos cortos de tiempo. Sin embargo, la calidad de la solución ofrecida, dependía de

varios factores, como por ejemplo, el tamaño del problema (un problema podría tener hasta 48 solicitudes de transporte, con 12 nodos), del tiempo disponible para la programación de la ruta, de imprevistos como cancelaciones de ruta, cambios de último momento de los puntos de origen o de destino, o de factores variables propios del programador como el estado de ánimo, su disposición, etc.

Para evitar esta variabilidad y para mejorar tanto la calidad de la solución como del tiempo necesario para obtener una respuesta, Torres (2004) propuso el uso del la Meta-heurística de Algoritmos Genéticos basado en una heurística de inserción; en este trabajo, el cromosoma representaba una solución factible del problema, el cual era construido insertando aquellos nodos que minimizaban la extensión de la ruta. Bajo este enfoque, Torres consiguió mejorar los resultados de la programación manual, logrando una mejora máxima del 26%.

1.2 METODOLOGÍA

Para solucionar el problema de Ruteo de un Helicóptero, el trabajo se dividió en etapas; en la primera, se hizo una revisión bibliográfica sobre los antecedentes del problema de ruteo de helicópteros; se revisó la literatura concerniente a problemas de ruteo de vehículos o VRP (de sus siglas en inglés *Vehicle Routing Problems*), de problemas de entrega y recogida (*Pickup and Delivery Problem)* y del problema *Dial-a-Ride* DARP, con el objeto de ubicar al problema de estudio en un contexto más general. También se revisó, los métodos utilizados para su solución, la Meta-heurística Colonia de Hormigas y distintas heurísticas de inserción.

En una segunda etapa, se definió el problema a trabajar y la estructura del algoritmo a implementar; esta estructura, se fundamentaría en varios aportes de trabajos que no estaban dirigidos específicamente a la solución de un problema de ruteo de helicópteros, pero que por su semejanza a él y por la calidad de los resultados reportados, influyeron en la estructura utilizada. En la tercera etapa, se implementó el algoritmo de Colonia de Hormigas basada en una heurística de inserción.

En la cuarta etapa, se hicieron pruebas sobre una librería de 40 problemas reales (Torres, 2004), para determinar el desempeño del algoritmo propuesto. De acuerdo al número de solicitudes que manejaban, dichos problemas fueron clasificados en problemas de tamaño pequeño, mediano y grande. Para cada grupo, se corrió un diseño de experimentos, con el cual se determinó los parámetros y los exponentes más significativos.

En una última etapa, se analizaron los resultados obtenidos y se compararon con los reportados por Torres (2004). Como parámetros de comparación, se tuvo en cuenta tanto la calidad de la solución y el tiempo de ejecución.

1.3 OBJETIVO GENERAL

Utilizar la Meta-heurística Colonia de Hormigas basada en una heurística de Inserción, para dar solución a una familia de problemas de Ruteo de un Helicópteros.

1.4 OBJETIVOS ESPECÍFICOS

Adaptar la meta-heurística de Colonia de Hormigas basada en una heurística de inserción al problema de Ruteo de Helicópteros, con el objetivo de minimizar la distancia total de recorrido.

Determinar por medio de un diseño de experimentos, los parámetros más significativos que influyen en el comportamiento del algoritmo propuesto.

Analizar el desempeño del algoritmo con base en la calidad de solución y en el tiempo de ejecución.

Realizar comparaciones entre los resultados obtenidos y las mejores soluciones reportadas para el problema de estudio.

2 MARCO TEÓRICO

El Problema de Ruteo de un Helicóptero, consiste en la construcción de una ruta factible, la cual debe ser seguida por el vehículo, en orden de satisfacer unas solicitudes de transporte (Salvelsbergh & Sol, 1995; Torres, 2004). Las solicitudes de transporte, están conformadas por un nodo origen, un nodo destino y una carga a transportar, la cual no debe sobrepasar en cualquier trayecto de la ruta, la carga máxima del helicóptero. Para este problema, las siguientes condiciones también se tienen que cumplir: a) restricciones de precedencia (un nodo destino, siempre tiene que ser visitado después de un nodo origen), b) el nodo de inicio y de terminación de la ruta de un vehículo no necesariamente son los mismos; c) los nodos de origen y destino de los pasajeros, pueden ser visitados más de una vez, y d) se supone que los pasajeros tienen una disponibilidad total (Torres, 2004).

Bajo estos lineamientos, el Problema de Ruteo de un Helicóptero, puede ser clasificado como un caso más del Problema General de Entrega y Recogida (GPDP, de sus siglas en inglés, *General Pickup and Delivery Problem*). Según la clasificación de Salvelsbergh y Sol (1995), el GPDP tiene tres extensiones muy bien estudiadas: el Problema de Entrega y Recogida (PDP), el Dial-a-Ride (DARP) y el de Ruteo de Vehículos (VRP).

En el Dial-a-Ride, un conjunto de rutas tienen que diseñarse y programarse para n usuarios, los cuales especifican: a) las ubicaciones de su origen y destino y b) el ancho de las ventanas de tiempo, ya sea para el origen o para el destino (Cordeau & Laporte, 2003b). El DARP, tiene un grado de dificultad más alto que los otros dos problemas, ya que su solución, aparte de tener en cuenta los costos de transporte (propios de un problema de entrega y de recogida o de ruteo de vehículos), también tiene presente los inconvenientes causados al usuario, como por ejemplo, tiempos de espera, retrasos en la hora de entrega o de recogida, largas permanencias en el vehículo, entre otras. (Cordeau & Laporte, 2003a).

La literatura considera que hay dos clases del DARP: uno estático, en donde las solicitudes de transporte son conocidas previamente y otro dinámico, donde la ruta se construye a través de una programación en tiempo real, a medida que las solicitudes entran al sistema (Salvelsbergh & Sol, 1995; Cordeau & Laporte, 2003a, b). En la práctica, hay pocos estudios que han atacado la versión dinámica del DARP (Cordeau & Laporte, 2003a), ya que este problema puede resolverse como una secuencia de problemas estáticos, en donde cada vez que llegue una nueva solicitud, la ruta es actualizada (Salvelsbergh & Sol).

El Ruteo de Helicópteros, como lo define Fiala y Pulleyblank (1992), es un problema Dial-a-Ride estático de un solo vehículo, sin restricciones de ventanas de tiempo, el cual ha sido atacado por varios autores desde distintos puntos de vista. Para este problema en especial, métodos de optimización y de aproximación han sido propuestos. Dentro del primer método, Salvelsbergh y Sol (1995) al igual que Cordeau y Laporte (2003b) reportan el trabajo realizado por Psaraftis, el cual aplicó un algoritmo de programación dinámica, con el que se pudo resolver instancias hasta de 10 clientes con 20 nodos. Otros trabajos como el de Kalantary, Hill y Arora (en Salvelsbergh & Sol), utilizaron el algoritmo de Branch and bound, con que se pudo resolver instancias de hasta 18 clientes.

Del lado de los métodos de aproximación, encontramos el trabajo realizado por Fiala y Pulleyblank (1992), quienes basaron su algoritmo de solución, en una heurística de inserción, seguida por un intercambio 3-opt; por medio de la primera, los autores en cada iteración, extendían una ruta factible, insertando aquellos nodos que maximizaban el costo de la ruta. Con esta metodología, garantizaban que la estructura principal de la ruta quedara definida desde el inicio del algoritmo y que los arcos cortos podían permanecer hasta el final de la ruta. Una vez insertados todos los nodos y con el fin de mejorar la solución, se utilizaba un intercambio 3-opt al final de cada iteración. El algoritmo propuesto fue probado en instancias reales, y los autores reportaron mejoras de hasta un 15 % con respecto las soluciones obtenidas en ese momento.

En el 2004, Torres inspirado en el trabajo de Fiala y Pullerblank (1992), utilizó un algoritmo genético basado en una heurística de inserción, para dar solución a una familia de problemas de ruteo de helicópteros; los resultados reportados por el autor, superaron o

al menos igualaron las soluciones que habían sido construidas manualmente por expertos programadores de ruta.

2.1 PROBLEMA DE RUTEO DE UN HELICÓPTERO

En la actualidad, formulaciones matemáticas no han sido presentadas para el problema en estudio. Existen formulaciones de problemas muy cercanos al de ruteo de un helicóptero, como por ejemplo, para el problema de entrega y de recogida (Salvelsbergh & Sol, 1995; Lau & Liang, 2002) y para el problema Dial-a-Ride estático multivehículo (Salvelsbergh & Sol; Bergvinsdottir, 2004); sin embargo, estas formulaciones no pueden ser utilizadas para el problema bajo estudio, porque no permiten visitar los nodos más de una vez, situación que se repite constantemente en el Ruteo de Helicópteros.

Torres (2004), hace una descripción general del problema, basado en la definición de dos conjuntos: un conjunto **N** de nodos y un conjunto **P** de pasajeros; con estos conjuntos el problema de Ruteo de un helicóptero, puede ser establecido como:

El helicóptero inicia su recorrido en el nodo $e \in N$; este nodo puede coincidir con el nodo de inicio de recorrido $I(q) \in N$ del pasajero $q \in P$. Si el helicóptero tiene capacidad de carga $C \in \{1,2,....\}$, la cual es revisada en cada movimiento, avanza al nodo $n_1 \in N$, recorriendo una distancia $d(e,n_1)$; el nodo n_1, puede ser el nodo final $F(q) \in N$ del pasajero $q \in P$ (se cumple que $I(q) \neq F(q)$) o puede ser el nodo de inicio de recorrido de otro pasajero. El siguiente paso es moverse a un nuevo nodo, si la capacidad lo permite y respetando la restricción de precedencia. El proceso se repite, hasta que todas las solicitudes de transporte hallan sido atendidas; en tal caso, el helicóptero termina su recorrido en el nodo $s \in N$. El objetivo del problema, es encontrar ruta factible $R = (e, n_1, n_2, ..., n_m, s)$ que optimice el recorrido total $F(R) = d(e,n_1) + \sum_{i=1}^{m-1} d(n_i, n_{i+1}) + d(n_m, s)$ de la ruta.

Este tipo de problemas de ruteo, que consideran restricciones de precedencia y de capacidad, son considerados del tipo NP-hard, los cuales son atacados en su mayoría por métodos de aproximación, como se mencionó anteriormente.

3 COLONIA DE HORMIGAS

3.1 ORIGEN

En los últimos años, un nuevo campo entre los límites de la Investigación de Operaciones y la Inteligencia Artificial, ha sido objeto de estudio y de interés por parte de los investigadores: las heurísticas provenientes de la naturaleza (Colorni, et al., 1996). Entre estas técnicas se encuentran la Búsqueda Tabú, los Algoritmos Genéticos, el Recocido Simulado, las Redes Neuronales, la Colonia de Hormigas, etc., que han sido utilizadas según Glover y Greenberg (1989) y Reeves (1993) "para obtener muy buenos resultados en problemas de optimización combinatoria NP-hard" (Colorni, et al., 1996, p.1).

Este tipo de métodos heurísticos, se basan en dos principios básicos extraídos de la naturaleza: la selección, que le permite a ella premiar a los buenos individuos y penalizar a los malos (idea básica para la optimización), y la mutación, que es el elemento de aleatoriedad que permite el nacimiento de nuevos individuos (idea básica para la búsqueda no determinística) (Colorni, et al., 1996).

Según Maniezzo, Gambardella & Luigi (2001), el algoritmo de Colonia de Hormigas, fue introducido por primera vez por Colorni, Dorigo y Maniezzo (1991, 1992) bajo el nombre de Ant System. Este algoritmo se fundamenta en el comportamiento de las hormigas reales, quienes son capaces de encontrar la trayectoria más corta desde la colonia hasta la fuente de alimento (Beckers, Deneubourg & Goss; Goss, Aron, Deneubourg & Pasteels, en Dorigo & Gambardella, 1997a) sin usar pistas visuales (Hölldobler & Wilson, en Dorigo & Gambardella). Este comportamiento puede ser explicado por medio de la Figura 1.

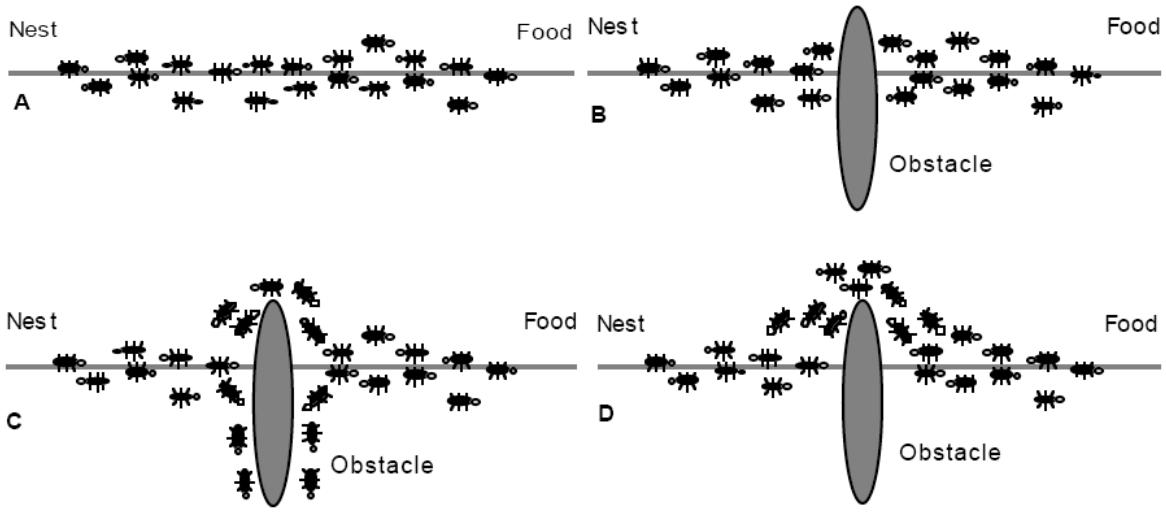

Figura 1 Comportamiento de las hormigas reales.
Fuente: (Dorigo & Gambardella, 1996, p. 3)

La labor de una hormiga, es explorar el territorio en busca de alimento; a medida que ésta avanza en su búsqueda, deja un rastro de feromona que sirve de guía para el retorno a la colonia o como un medio de comunicación para transmitir a las demás hormigas la ubicación exacta de la fuente de alimento (Figura 1A). Si este rastro o camino de feromona es obstruido (Figura 1B), las hormigas buscan distintas rutas para reestablecerlo. En este punto, hay hormigas que giran a la izquierda o a la derecha del obstáculo tratando de reconstruir el rastro de feromona (Figura 1C); pasado un intervalo de tiempo, el camino más corto tiene una mayor cantidad de feromona que el camino largo, ya que por él han pasado más hormigas por la misma unidad de tiempo. Si una hormiga llega al punto de bifurcación y debido a que las hormigas tienen preferencia por caminos más cortos y cargados con una mayor cantidad de feromona, la hormiga tenderá a girar a la derecha, en busca de la trayectoria más corta. (Figura 1D). Este proceso es repetido una y otra vez hasta que se reconstruye completamente y de forma rápida el camino de feromona. Este comportamiento presentado por las hormigas reales, puede ser simulado y adaptado por hormigas artificiales o agentes, para tratar de resolver problemas de optimización combinatoria NP-hard, como por ejemplo el bien conocido problema del agente viajero, el de asignación cuadrática o problemas de ruteo de vehículos (Dorigo & Gambardella, 1997a).

3.2 THE ANT SYSTEM

Según Dorigo y Gambardella (1997b), el algoritmo Ant System fue aplicado por primera vez al Problema del Agente Viajero o TSP. En este problema, el algoritmo comienza ubicando a cada hormiga en una ciudad diferente la cual es escogida de manera aleatoria. La siguiente ciudad a visitar por cada hormiga, es escogida de acuerdo a la Regla Proporcional Aleatoria, la cual se basa en: 1) una medida de deseabilidad o nivel feromona $\tau(r,s)$, que hace referencia a la cantidad de feromona que puede llegar a tener el arco conformado por dos ciudades distintas *(r, s);* y 2) en un valor heurístico o de Visibilidad $\eta(r,s)$. En el TSP, el valor heurístico es igual al inverso de la distancia entre dos ciudades distintas. Dorigo y Gambardella, definen a la Regla Proporcional Aleatoria como:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]^{\alpha} \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)]^{\alpha} [\eta(r,u)]^{\beta}} & \text{Si } s \in J_k(r) \\ 0 & \text{d.l.c} \end{cases} \qquad (1)$$

donde $J_k(r)$ es el conjunto de ciudades por visitar para la hormiga k-ésima; α y β son parámetros que controlan la importancia entre el nivel de feromona y la visibilidad (Dorigo, Maniezzo & Colorni, 1996).

Cada vez que una ciudad es visitada, ésta queda registrada en la memoria temporal de cada hormiga M_k, la cual es actualizada permanentemente, evitando reprogramaciones.

Cuando todas las hormigas han construido una ruta, el nivel de feromona de todos los arcos es actualizado por medio de la regla de Actualización Global, definida por la siguiente ecuación:

$$\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k(r,s) \qquad (2)$$

Donde $\Delta\tau_k(r,s) = \begin{cases} 1/L_k & \text{si (r,s)} \in \text{al tour de la hormiga k} \\ 0 & \text{d.l.c} \end{cases}$

L_k es la longitud de la ruta para la hormiga k y ρ es un parámetro de evaporación. La Ecuación 2, indica que todos los arcos utilizados por las hormigas reciben una cantidad adicional de feromona, mientras que en los otros arcos, el nivel de feromona es reducido o evaporado una cantidad (1 – ρ) (Dorigo & Gambardella, 1997b).

Stützle y Dorigo (1999), definen a la feromona "como un tipo de información numérica distribuida" que las hormigas, una vez se produce la actualización global, transfieren a los arcos sirviendo como un medio de comunicación indirecto entre hormigas de distintas generaciones. Así, la feromona se convierte en otro tipo de memoria de largo plazo, característica importante de esta meta-heurística.

3.3 SISTEMA DE COLONIA DE HORMIGAS

Este sistema también propuesto por Dorigo y Gambardella (1997b), se diferencia del anterior en los siguientes puntos:

3.3.1 Regla de transición de estado

Esta regla permite hacer un balance entre el aprovechamiento de la información almacenada en los arcos (nivel de feromona) y la exploración de nuevas rutas (Dorigo y Gambardella, 1997b). Esta regla establece que la decisión de avanzar desde la ciudad *r* a la ciudad *s,* está influenciado por la siguiente fórmula probabilística:

$$s = \begin{cases} \arg\max_{u \notin M_k} \left\{ [\tau(r,u)]^{\alpha} \cdot [\eta(r,u)]^{\beta} \right\} & \text{Si } q \leq q_0 \\ S & \text{d.l.c.} \end{cases} \qquad (3)$$

Donde q es un número aleatorio uniformemente distribuido entre [0,1], q_0 es un parámetro ($0 \leq q_0 \leq 1$), α es igual a 1 y S es una variable aleatoria escogida de acuerdo a la Ecuación 1 (Dorigo & Gambardella, 1997a, 1997b).

3.3.2 Actualización Global

Al igual que en el Ant System, la actualización global también se lleva a cabo cuando todas las hormigas hallan construido su ruta y se diferencia en que solo aquellos arcos que pertenecen a la ruta más corta encontrada por la hormiga k, reciben una cantidad adicional de feromona. La regla de actualización global, está dada por:

$$\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \rho \cdot \Delta\tau(r,s) \qquad (4)$$

Donde

$$\Delta\tau(r,s) = \begin{cases} (L_{gb})^{-1} & \text{Si (r,s) } \in \text{al mejor tour global} \\ 0 & \text{d.l.c} \end{cases}$$

L_{gb} es la longitud del mejor tour global (Dorigo & Gambardella, 1997a, 1997b).

3.3.3 Actualización Local

Es el cambio de feromona realizado cada vez que una hormiga escoge moverse de la ciudad *r* a la ciudad *s*. Está dada por la siguiente fórmula:

$$\tau(r,s) = (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \qquad (5)$$

Donde τ_0 es un parámetro (Dorigo & Gambardella, 1997a, 1997b).

3.3.4 Aplicaciones de Colonia de Hormigas

Esta nueva Meta-heurística, ha sido aplicada para resolver distintos problemas del campo de la Investigación de Operaciones, de los cuales se puede mencionar, el problema del Agente Viajero (Colorni, et al., 1996; Dorigo & Gambardella, 1997a, 1997b; Dorigo, Maniezzo & Colorni, 1996); problema de Asignación Cuadrática (Colorni, et al., 1996); Ruteo de Vehículos (Gambardella, Taillard & Agazzi,1999; Doerner, Hartl & Reiman 2000); Programación de la Producción (Buitrago & González, 2004); entre otros.

4 ALGORITMO DE INSERCIÓN

El algoritmo de inserción, es un procedimiento goloso e iterativo, que parte de una secuencia preseleccionada de nodos para la construcción de una ruta (Torres, 2004).

Este algoritmo, ha sido combinado con otras heurísticas para resolver problemas del tipo NP-hard. Por ejemplo, Jaw, Odoni, Psaraftis, et al. (1986), emplearon un algoritmo de esta clase, para resolver el problema Dial-a-Ride estático multi-vehículo con ventanas de tiempo. En este trabajo, la heurística insertaba un cliente a la vez en la ruta actual de algún vehículo; este proceso se repetía hasta que todas las solicitudes de transporte habían sido procesadas; en este punto, una fase de optimización era aplicada, para encontrar la mejor inserción factible.

Fiala y Pullerblank (1992) y Torres (2004), también hicieron uso de este algoritmo para resolver el Problema de Ruteo de un Helicóptero. Fiala y Pullerblank, utilizaron esta heurística combinada con intercambios 3-opt, mientras que Torres, utilizó los algoritmos genéticos con operadores clásicos de cruce y mutación.

4.1 ALGORITMO DE INSERCIÓN APLICADO AL PROBLEMA DE RUTEO DE HELICÓPTEROS

En pocas palabras, el algoritmo funciona de la siguiente manera. Suponga que tiene un conjunto **P** de pasajeros, cada uno con una solicitud de transporte diferente. En el arranque del algoritmo, un pasajero es seleccionado aleatoriamente, para construir la ruta inicial **R: (e, n1, n2, s)** con sus nodos de origen y de destino. Con base en esta ruta, se examina todas las inserciones factibles de los nodos de cada uno de los pasajeros restantes del conjunto **P**; simultáneamente y por medio del procedimiento explicado más adelante, se calcula el costo de inserción para cada nodo. A continuación se selecciona, aquel con el menor costo y se actualiza la ruta **R**. El proceso continúa hasta que todas las solicitudes de transporte hallan sido atendidas (Torres, 2004).

La Figura 2, presenta el flujograma de este algoritmo.

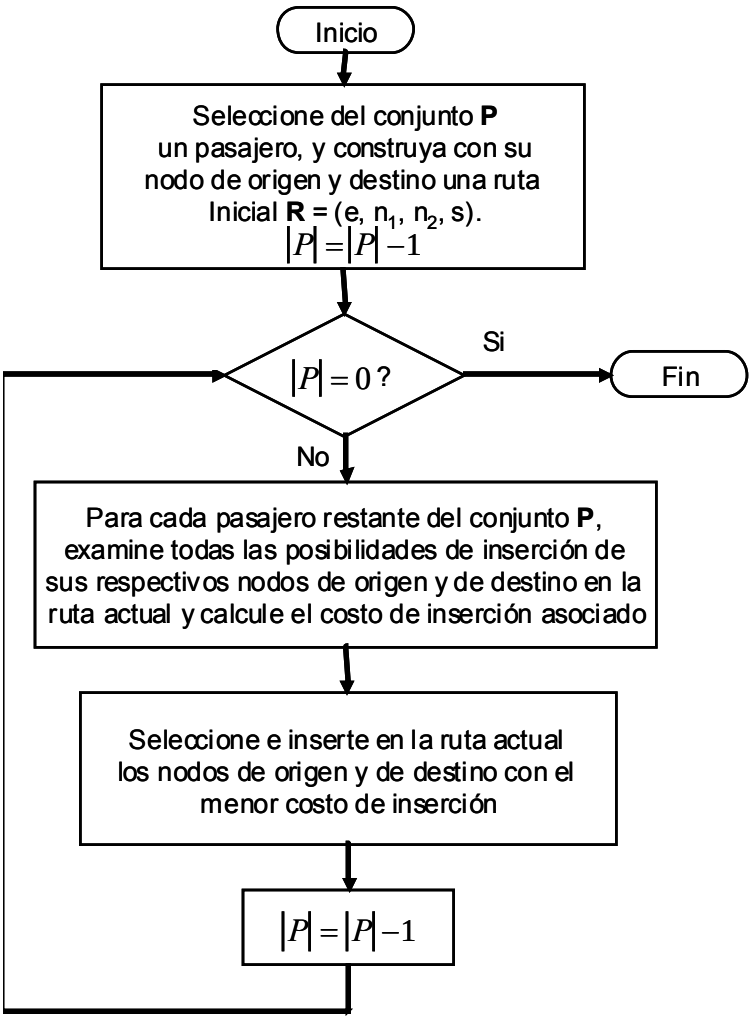

Figura 2. Flujograma de la Heurística de Inserción

El costo de inserción puede ser calculado por medio de uno de los siguientes 3 casos, en donde el nodo *i* y el nodo *j* hacen referencia al nodo origen y destino respectivamente (Torres, 2004).

CASO 1: El nodo *i* del nuevo pasajero está en la ruta actual.

En este caso, se tiene que examinar todas las inserciones factibles del nodo destino *j* después de cada aparición del nodo *i*. La Figura 3 ilustra este proceso.

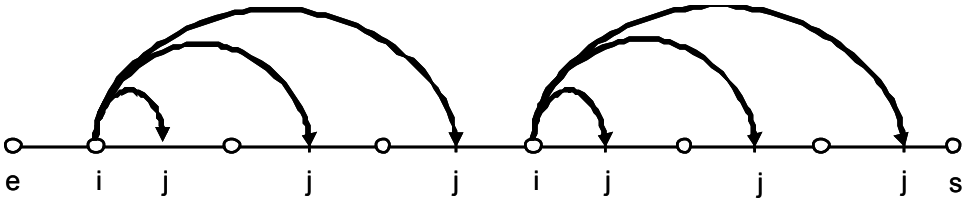

Figura 3. Inserción para el caso 1. Fuente: (Torres, 2004)

El costo de inserción *A(i,j)*, es calculado con la siguiente ecuación:

$$A(i,j) = \begin{cases} 0 & \text{Si j está en la ruta actual} \\ \delta(j, n_k) & \text{Si j se localiza después del nodo } n_k \end{cases} \quad (6)$$

CASO 2: El nodo *j* del nuevo pasajero está en la ruta actual

Se examinan todas las inserciones factibles del nodo *i* antes de cada aparición del nodo *j*, como se ilustra en la Figura 4.

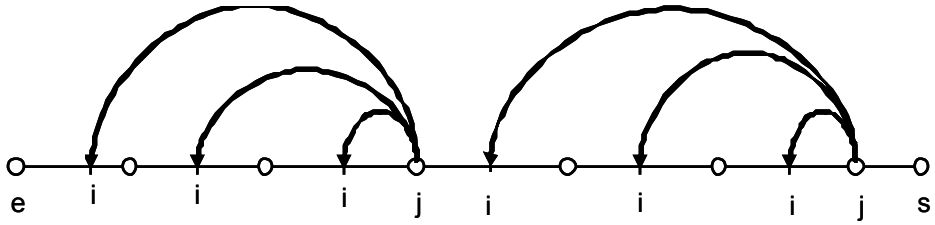

Figura 4. Inserción para el caso 2. Fuente: (Torres, 2004)

El costo de inserción *A(i,j)*, es calculado con la siguiente ecuación:

$$A(i,j) = \begin{cases} 0 & \text{Si i está en la ruta actual} \\ \delta(i, n_k) & \text{Si i se localiza después del nodo } n_k \end{cases} \quad (7)$$

CASO 3: Los nodos *i* y *j* del nuevo pasajero, no se encuentran en la ruta actual.

Se tiene que examinar todos los casos posibles de inserción de los nodos *i, j*, como se ilustra en la Figura 5.

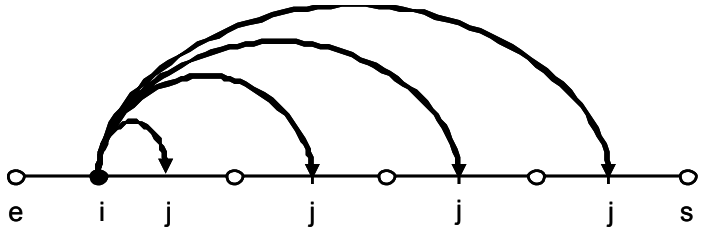

Figura 5. Inserción para el caso 3. Fuente: (Torres, 2004)

Suponiendo que *i* se localiza después del nodo n_k y *j* después del nodo n_l:

$$A(i,j)=\begin{cases}\delta(i,n_k)+\delta(j,n_l) & \text{Si } n_k \neq n_l \\ \delta(i,n_k)+d(i,n_{k+1})-d(i,j)-d(j,n_{k+1}) & \text{Si } n_k = n_l\end{cases} \quad (8)$$

Si *i* se localiza después del nodo n_k, $\delta(i, n_k)$ se define como:

$$\delta(i,n_k)=d(n_k,n_{k+1})-d(n_k,i)-d(i,n_{k+1}) \quad (9)$$

5 EL ALGORITMO DE COLONIA DE HORMIGAS PROPUESTO

El problema que se pretende solucionar con el algoritmo propuesto, “es el de determinar la ruta del helicóptero que permita transportar a todos los pasajeros en un tiempo o una distancia total mínimos, respetando la capacidad del aparato en cada trayecto” (Torres, 2004).

Para resolver este problema, se implementó la Meta-heurística Colonia de Hormigas, la cual se adaptó al problema de estudio y se le propusieron mejoras con el objetivo de obtener mejores resultados. A continuación se describe en forma detallada el algoritmo propuesto.

5.1 COLONIA DE HORMIGAS APLICADA AL PROBLEMA DE RUTEO DE UN HELICÓPTERO

Para la solución del problema planteado, se utilizó dos extensiones de la Meta-heurística de Colonia de Hormigas, las cuales han sido designadas por ACO1 y ACO2; el tratamiento del problema en cuestión, con estas dos extensiones, se hizo con el fin de comparar la calidad de solución dada, por un lado con un algoritmo *Ant System* modificado (ACO1, basado en la propuesta de Doerner, Hartl y Reimann (2000)) y por el otro, con una Colonia de Hormigas inspirado en su forma natural (algoritmo ACO2, basado en la propuesta dada por Dorigo y Gambardella (1997a)).

5.1.1 Adaptación al problema

Realmente, el problema de Ruteo de un Helicóptero está conformado por dos sub-problemas: el primero, consiste en secuenciar el orden de atención de las solicitudes de

transporte (muy parecido al problema del agente viajero) y el segundo, el de hallar posiciones factibles de inserción para los nodos de origen y de destino de una solicitud, respetando restricciones de precedencia y de capacidad. Estos dos problemas, exigen ser resueltos de manera secuencial, ya que persiguen un solo objetivo: la minimización de la distancia o tiempo total de recorrido.

Con esto en mente, el enfoque que se utilizó para resolver el problema en cuestión, es el siguiente: la Colonia de Hormigas es empleada para secuenciar las solicitudes de transporte, la cual se basa en los costos de inserción hallados por el algoritmo o heurística de inserción. A continuación, se explica de manera detallada las adaptaciones propuestas.

5.1.2 Heurística de Inserción

Una vez que las rutas iniciales son establecidas, la heurística de inserción examina para cada hormiga y para cada solicitud de transporte no perteneciente a memoria temporal M_k, todas las posiciones factibles de inserción tanto del nodo de origen como del nodo de entrega; simultáneamente, también calcula el costo de inserción, $A(\delta, n_k)$. Como resultado, se obtiene para cada solicitud una lista con todos los costos asociados a las inserciones posibles de sus respectivos nodos. La Figura 6, ilustra este procedimiento con un ejemplo.

Ruta actual, R: (e n_1 s)

Solicitud	Nodo origen	Posibles Inserciones	Nodo destino	Posibles Inserciones	Costo
n	n_2	Antes de e	n_3	Antes de e, después de n_2	C_1
	n_2	Antes de e	n_3	Entre e y n_1	C_2
	n_2	Antes de e	n_3	Entre n_1 y s	C_3
	n_2	Antes de e	n_3	Después de s	C_4
...	...	...	...	...	...

Figura 6. Ejemplo del procedimiento utilizado en la heurística de inserción

Posteriormente, el módulo selecciona aquella inserción con el menor costo, y lo transfiere a la tabla de Mínimo Costo de Inserción. Este proceso se repite con todas las solicitudes no programadas. Ver Figura 7.

Solicitud	Mejor posición de inserción del nodo origen	Mejor posición de inserción del nodo destino	Costo
1	Antes de e	Entre n_1 y s	C_1
2	Entre e y n_1	Entre n_1 y s	C_2
….	…..	…..	…..
n	Entre n_k y n_{k+1}	Entre n_k y n_{k+1}	C_n

Figura 7. Ejemplo de la tabla de Mínimo Costo de Inserción

La Figura 8, muestra el diagrama de flujo desarrollado para el módulo generador de rutas factibles.

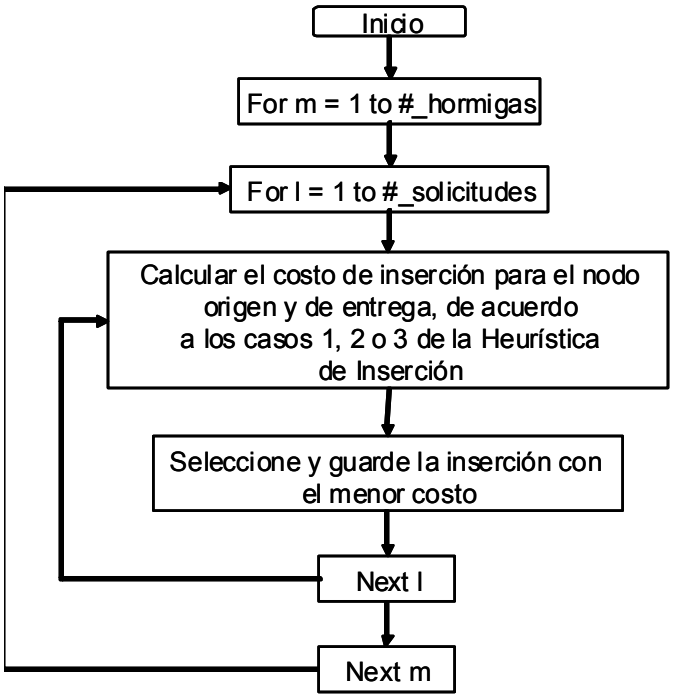

Figura 8. Flujograma del Módulo Generador de Rutas Factibles

Con base en la tabla de Mínima Costo de Inserción, la elección de qué solicitud atender, estará determinada por la heurística implementada: ACO1 o ACO2.

5.1.3 Heurística ACO1

Está construida con las bases de un algoritmo *Ant System* y modificada según la propuesta realizada por Bullnheimer, Hartl y Strauss (1999).

En la fase inicial de cada iteración, el algoritmo asigna de manera aleatoria, una solicitud de transporte a cada hormiga, la cual genera una ruta inicial con los respectivos nodos de origen y destino. Esta primera solicitud, queda grabada en la memoria temporal de cada hormiga. En el paso siguiente, se hace un llamado a la heurística de inserción para que calcule los costos respectivos que serán utilizados por las hormigas para secuenciar las solicitudes restantes; estos pasos se repiten, hasta que todas las solicitudes sean atendidas.

Regla Proporcional Aleatoria

Las hormigas utilizan la regla proporcional aleatoria (Ecuación 1), para decidir qué solicitud programar; para efectos de explicación, esta ecuación es repetida.

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]^{\alpha} \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)]^{\alpha} [\eta(r,u)]^{\beta}} & \text{Si } s \in J_k(r) \\ 0 & \text{d.l.c} \end{cases} \qquad \text{(1 repetida)}$$

donde $J_k(r)$ es el conjunto de solicitudes por programar para la k-ésima hormiga, $\tau(r,s)$ es el nivel de feromona del "arco" entre la solicitud *r* y *s*, $\eta(r,s)$ es la función heurística y α y β son parámetros.

Función Heurística

La función heurística o de visibilidad $\eta(r,s)$, está definida por la siguiente ecuación:

$$\eta(r,s) = Exp\ (-\ C \cdot Mínimo\ Costo\ de\ Inserción) \qquad (10)$$

Donde *C* es un parámetro de afinación.

La Ecuación 10, permite discriminar entre aquellas solicitudes que originan bajos y altos costos de inserción; solicitudes con un costo nulo o casi cero, tendrán una visibilidad cercana a uno, lo que significa tener una mayor probabilidad de ser escogido, cuando la hormiga aplica la Ecuación 1. Por el contrario, solicitudes con altos costos de inserción, reducen casi a cero su valor heurístico y por consiguiente, su probabilidad de ser seleccionada.

En la Figura 9 se grafica el comportamiento de la función heurística utilizada versus distintos valores de costos de inserción y del parámetro *C*.

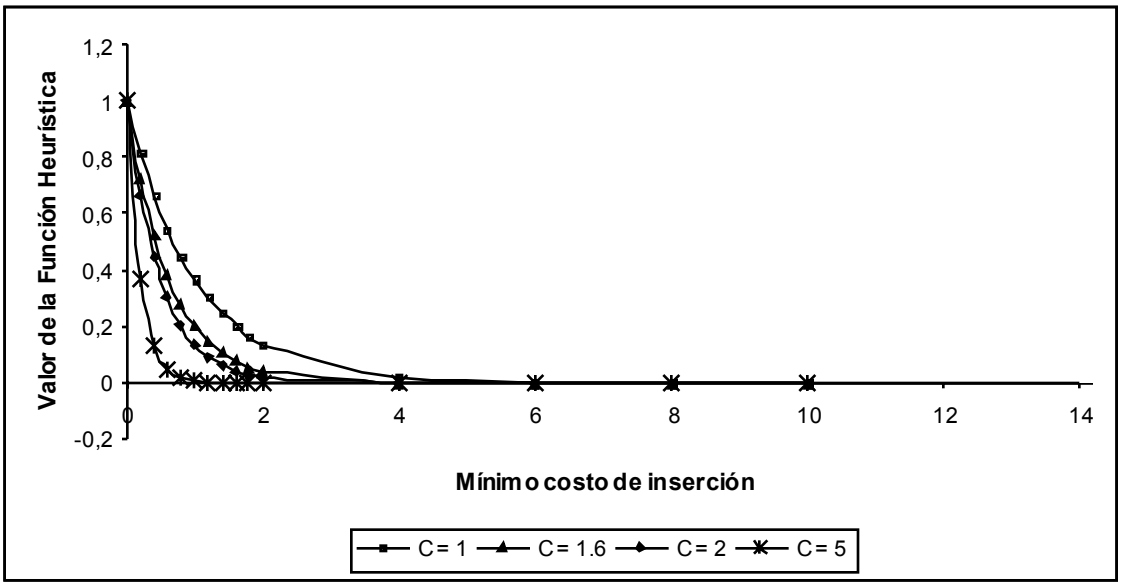

Figura 9. Comportamiento de la Función Heurística, para distinto valores de costos de inserción y del parámetro C.

De la Figura 9, también se puede observar que entre más alto sea el valor del parámetro *C*, más rápido cae a cero el valor heurístico, incluso para valores pequeños del costo de inserción. Para el modelo propuesto, este parámetro se dejo en un valor intermedio de 1.6.

Actualización Global

La modificación más relevante que tuvo ACO1, fue la actualización global, la cual se basó en el trabajo realizado por Bullnheimer, Hartl y Strauss (1999), quienes plantearon que esta actualización debería ser ejecutada sobre un grupo de mejores hormigas, escogidas de acuerdo a la calidad de su solución. Bullnheimer, Hartl y Strauss, proponen la siguiente ecuación para realizar la actualización global:

$$\tau(r,s) = \rho \cdot \tau(r,s) + \sum_{\lambda=1}^{\Lambda} \Delta\tau_{(r,s)}^{\lambda} \qquad (11)$$

Donde ρ es la persistencia de la feromona, Λ son las mejores hormigas y $\Delta\tau_{(r,s)}^{\lambda}$ está dada por la ecuación:

$$\Delta\tau_{(r,s)}^{\lambda} = \begin{cases} 1 - \dfrac{\lambda - 1}{\Lambda} & \text{Si } 1 \le \lambda \le \Lambda \\ 0 & \text{d.l.c.} \end{cases} \qquad (12)$$

5.1.4 Heurística ACO2

Esta basada en el trabajo desarrollado por Dorigo y Gambardella (1997a), quienes aplicaron la Colonia de Hormigas al problema del Agente Viajero. A continuación, se explica el procedimiento utilizado en ACO2, aplicado al problema de Ruteo de un Helicóptero.

Regla de Transición de Estado

Esta regla, propia de un Sistema de Colonia de Hormigas, es utilizada por las hormigas para decidir qué solicitud atender; está definida por la Ecuación 3.

$$s = \begin{cases} \arg\max_{u \notin M_k} \left\{ [\tau(r,u)]^{\alpha} \cdot [\eta(r,u)]^{\beta} \right\} & \text{si } q \leq q_0 \\ S & \text{d.l.c} \end{cases} \qquad \text{(3 repetida)}$$

donde S esta dada por la Ecuación 1:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau(r,s)]^{\alpha} \cdot [\eta(r,s)]^{\beta}}{\sum_{u \notin M_k} [\tau(r,u)] \cdot [\eta(r,u)]^{\beta}} & \text{si s € a } M_k \\ 0 & \text{d.l.c} \end{cases} \qquad \text{(1 repetida)}$$

Donde $\tau(r,s)$ vuelve a ser el nivel de feromona sobre el "arco" formado por las solicitudes *r* y *s*, $\eta(r,s)$ la función heurística, β un parámetro, *α es igual a 1*, *q* un valor escogido aleatoriamente con probabilidad uniforme [0,1] y q_0 un valor entre ($0 \leq q_0 \leq 1$).

Función Heurística o de Visibilidad, η(r,s)

Para ACO2, se probó otra función heurística que no dependiera de ningún parámetro, pero que al igual que en ACO1, de mayor relevancia a las solicitudes con un costo de inserción nulo o casi cero. La función escogida está en la Ecuación 13 y su comportamiento se describe en la Figura 10.

$$\eta(r,s) = \frac{1}{Exp(mínimo.\cos to.inserción)} \qquad (13)$$

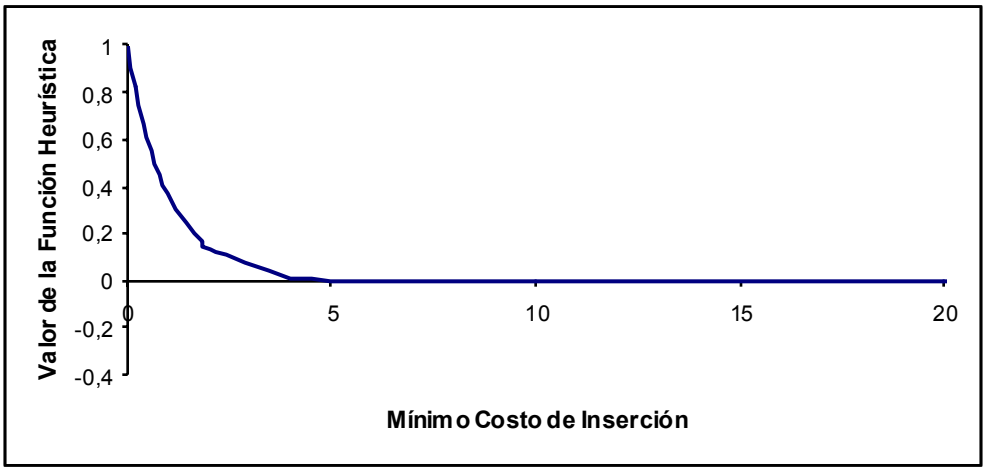

Figura 10. Comportamiento de la Función Heurística Vs. Mínimo Costo de Inserción

Actualización Local

Cada vez que una hormiga secuencie una solicitud, el nivel de feromona entre solicitudes, es actualizado por medio de la Ecuación 5. Como lo menciona Dorigo y Gambardella (1997a), esta actualización "evapora" la feromona existente entre los "arcos" de las solicitudes que no han sido utilizados, haciéndolos menos deseables para las hormigas de las próximas iteraciones.

Actualización Global

Al igual que en ACO1, se utilizó la propuesta de Bullnheimer, Hartl y Strauss (1999) para la actualización global, la cual es una estrategia del tipo elitista. Con esta estrategia, se asegura que los arcos más utilizados y por ende más cortos, predominen sobre toda la ejecución del algoritmo.

5.2 DEFINICIÓN DE PARÁMETROS Y DISEÑO DE EXPERIMENTOS

Tal vez, una de la desventaja más grande que se tiene con el uso de meta-heurísticas, es su dependencia con los parámetros; muchas veces, la literatura especializada recomienda

el uso de ciertos valores (Dorigo & Gambardella, 1997a, b), los cuales carecen de una explicación más detallada de cómo fueron encontrados.

En el problema bajo estudio, se recurrió a un diseño de experimentos, que según la definición de Buitrago y González (2004), es la metodología que basándose en ensayos y cálculos realizados sobre una muestra, se puede obtener una inferencia acerca de la población.

En un diseño factorial, a través de un ensayo completo del experimento, se puede investigar el efecto de todas las posibles combinaciones de los niveles de los factores (Montgomery, 2001). Cuando se tienen k factores, con dos niveles cada uno, se habla de un diseño factorial 2^k. Sin embargo, en algunas ocasiones es muy difícil ejecutar las 2^k corridas de un diseño completo (para nuestro caso, con 5 factores y con una sola réplica, se tienen 32 corridas por cada heurística y para cada grupo de problemas). Por esta razón y asumiendo que las interacciones de alto nivel son despreciables, se recurren a diseños que corriendo una fracción de todo el experimento, se puede obtener una cantidad suficiente y confiable de información. Este tipo de experimentos, se conocen como Diseños Factoriales Fraccionales.

Para el problema de estudio, se definieron 5 factores cada uno con dos niveles, haciendo necesario correr un diseño factorial fraccional 2^{5-1}resolución V. La resolución indica, si en un diseño el efecto del Factor *p,* está aliado o confundido con cualquier otro efecto. En un diseñó con resolución V, los efectos principales o las interacciones dobles no se encuentran confundidos con otros efectos principales ni con otras interacciones dobles, pero las interacciones dobles si se confunden con interacciones triples (Montgomery, 2001, p. 307).

En las siguientes tablas, se encuentran los niveles de los factores estudiados para ACO1 y ACO2.

Tabla 1. Niveles utilizados en el diseño fraccional para ACO1

Factor	Representación utilizada	Nivel Bajo	Nivel Alto
Número de hormigas	A	10	60
α	B	1	9
β	C	1	9
ρ	D	0.1	0.9
% de mejores hormigas	E	0.3	0.7

Tabla 2. Niveles utilizados en el diseño fraccional para ACO2

Factor	Representación utilizada	Nivel Bajo	Nivel Alto
Número de hormigas	A	10	60
q_0	B	0.1	0.9
β	C	1	9
ρ	D	0.1	0.9
% de mejores hormigas	E	0.3	0.7

5.3 PROBLEMAS DE PRUEBA

5.3.1 Nomenclatura

Para probar el desempeño de los algoritmos propuestos, ACO1 y ACO2 se probaron sobre una librería de 40 problemas reales, los cuales fueron divididos en 3 grupos de acuerdo al número de pasajeros: pequeño, mediano y grande. En la Figura 11 se explica la nomenclatura utilizada y en la Tabla 3, se indica la información acerca del número de pasajeros y del número de nodos, para cada problema.

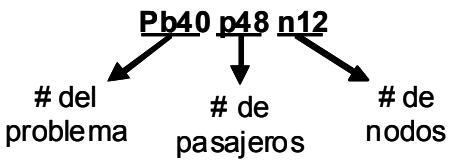

Figura 11. Nomenclatura utilizada para los problemas de prueba

Información más detallada acerca de esta librería, estará disponible en la página web http://pylo.uniandes.edu.co

Tabla 3. Número de solicitudes y de nodos, para los problemas de prueba (Torres, 2004)

Tamaño del Problema	Problema	# de Pasajeros	# de Nodos
Pequeño	Pb1p9n7	9	7
	Pb2p12n6	12	6
	Pb3p13n7	13	7
	Pb4p14n7	14	7
	Pb5p15n4	15	4
	Pb6p15n7	15	7
	Pb7p16n8	16	8
	Pb8p16n9	16	9
	Pb9p17n7	17	7
	Pb10p17n8	17	8
	Pb11p17n9	17	9
	Pb12p18n5	18	5
Mediano	Pb13p20n8	20	8
	Pb14p20n9	20	9
	Pb15p21n7	21	7
	Pb16p21n8	21	8
	Pb17p23n9	23	9
	Pb18p25n10	25	10
	Pb19p25n10	25	10
	Pb20p26n3	26	3
	Pb21p26n9	26	9
	Pb22p28n7	28	7
	Pb23p28n9	28	9
	Pb24p29n8	29	8
	Pb25p29n9	29	9
Grande	Pb26p30n9	30	9
	Pb27p30n12	30	12
	Pb28p31n10	31	10
	Pb29p31n12	31	12
	Pb30p32n13	32	13
	Pb31p33n10	33	10
	Pb32p33n12	33	12
	Pb33p33n12	33	12
	Pb34p35n10	35	10
	Pb35p37n11	37	11
	Pb36p38n11	38	11
	Pb37p44n11	44	11
	Pb38p44n11	44	11
	Pb39p44n12	44	12
	Pb40p48n12	48	12

6 ANÁLISIS DE RESULTADOS

Los algoritmos propuestos se desarrollaron en el lenguaje de programación Visual Basic con Excel (anexo 2 y 3), y se corrieron en un computador con procesador Pentium IV de 2.8 GHz y de 256 MB en RAM. Los resultados fueron los siguientes.

6.1 DISEÑO DE EXPERIMENTOS

6.1.1 Algoritmo ACO1

Inicialmente, se tomó un problema de cada grupo y se corrió el diseño de experimentos. Los resultados se pueden observar en las siguientes tablas.

Tabla 4. Diseño de experimentos para el problema Pb8p16n9

Tratamiento	A	B	C	D	E	Respuesta (m)	Tiempo (hh: mm: ss)
e	10	1	1	0,1	0,7	73996,7302	00:00:06
a	60	1	1	0,1	0,3	73996,7302	00:00:32
b	10	2	1	0,1	0,3	73996,7302	00:00:05
abe	60	2	1	0,1	0,7	73996,7302	00:00:33
c	10	1	2	0,1	0,3	73996,7302	00:00:06
ace	60	1	2	0,1	0,7	73996,7302	00:00:33
bce	10	2	2	0,1	0,7	73996,7302	00:00:06
abc	60	2	2	0,1	0,3	73996,7302	00:00:31
d	10	1	1	0,9	0,3	73996,7302	00:00:06
ade	60	1	1	0,9	0,7	73996,7302	00:00:33
bde	10	2	1	0,9	0,7	73996,7302	00:00:06
abd	60	2	1	0,9	0,3	73996,7302	00:00:32
cde	10	1	2	0,9	0,7	73996,7302	00:00:05
acd	60	1	2	0,9	0,3	73996,7302	00:00:31
bcd	10	2	2	0,9	0,3	73996,7302	00:00:06
abcde	60	2	2	0,9	0,7	73996,7302	00:00:32
				Tiempo total de ejecución			**00:05:03**

Tabla 5. Diseño de experimentos para el problema Pb21p26n9

Tratamiento	A	B	C	D	E	Respuesta (m)	Tiempo (hh: min: ss)
e	10	1	1	0,2	0,7	120545,4214	00:00:11
a	60	1	1	0,2	0,3	120545,4214	00:01:02
b	10	4	1	0,2	0,3	120545,4214	00:00:11
abe	60	4	1	0,2	0,7	120545,4214	00:01:02
c	10	1	4	0,2	0,3	120545,4214	00:00:11
ace	60	1	4	0,2	0,7	120545,4214	00:01:04
bce	10	4	4	0,2	0,7	120545,4214	00:00:11
abc	60	4	4	0,2	0,3	120545,4214	00:01:05
d	10	1	1	0,8	0,3	120545,4214	00:00:10
ade	60	1	1	0,8	0,7	120545,4214	00:01:04
bde	10	4	1	0,8	0,7	120545,4214	00:00:11
abd	60	4	1	0,8	0,3	120545,4214	00:01:06
cde	10	1	4	0,8	0,7	120545,4214	00:00:10
acd	60	1	4	0,8	0,3	120545,4214	00:01:03
bcd	10	4	4	0,8	0,3	120545,4214	00:00:10
abcde	60	4	4	0,8	0,7	120545,4214	00:01:05
				Tiempo total de ejecución			00:09:56

Tabla 6. Diseño de experimentos para el problema Pb40p48n12

Tratamiento	A	B	C	D	E	Respuesta (m)	Tiempo (hh: min: ss)
e	10	1	1	0,1	0,7	126247,2597	00:01:07
a	60	1	1	0,1	0,3	115975,7465	00:06:40
b	10	9	1	0,1	0,3	115975,7465	00:01:07
abe	60	9	1	0,1	0,7	115975,7465	00:06:43
c	10	1	9	0,1	0,3	116997,9736	00:01:03
ace	60	1	9	0,1	0,7	115975,7465	00:06:19
bce	10	9	9	0,1	0,7	115975,7465	00:01:04
abc	60	9	9	0,1	0,3	115975,7465	00:06:15
d	10	1	1	0,9	0,3	117114,5315	00:01:07
ade	60	1	1	0,9	0,7	115975,7465	00:06:43
bde	10	9	1	0,9	0,7	115975,7465	00:01:07
abd	60	9	1	0,9	0,3	115975,7465	00:06:38
cde	10	1	9	0,9	0,7	115975,7465	00:01:03
acd	60	1	9	0,9	0,3	115975,7465	00:06:14
bcd	10	9	9	0,9	0,3	124048,8679	00:01:02
abcde	60	9	9	0,9	0,7	115975,7465	00:06:19
				Tiempo total de ejecución			01:00:31

De las tablas 4, 5 y 6, se puede concluir que los parámetros bajo estudio, no son significantes y que no afectan el valor de la respuesta obtenida. En las figuras 12, 13 y 14, se muestra la convergencia para los tres problemas anteriores. Resultados similares tanto para el diseño de experimentos como para la convergencia, se encontraron para el resto de problemas.

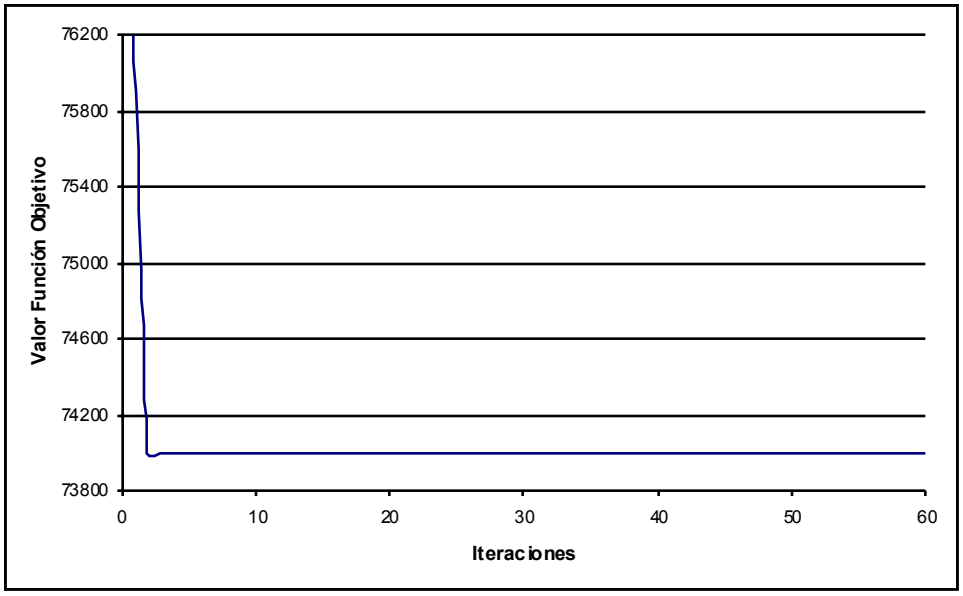

Figura 12. Evolución de la Convergencia para el problema 8

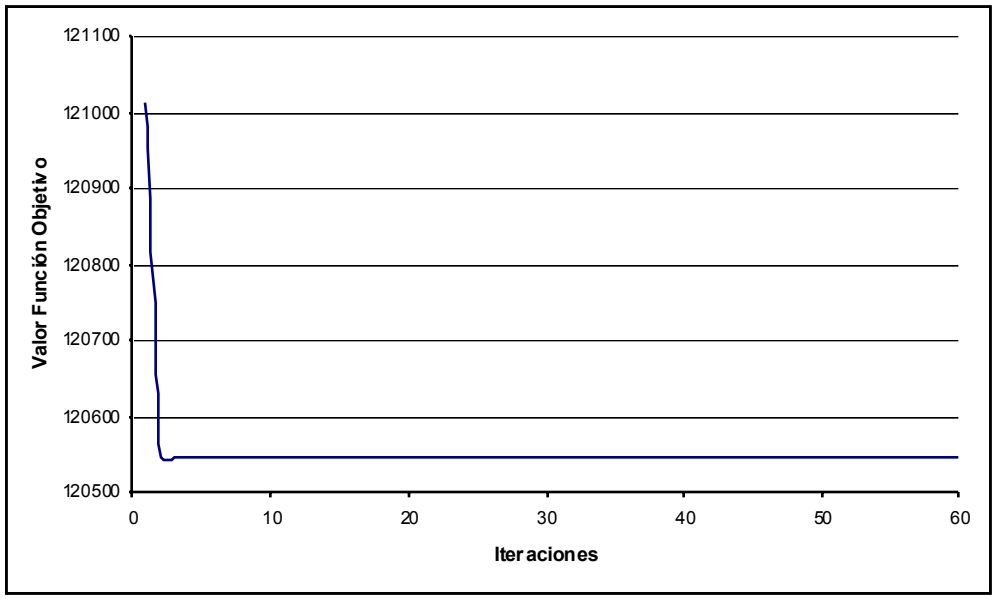

Figura 13. Evolución de la Convergencia para el problema 21

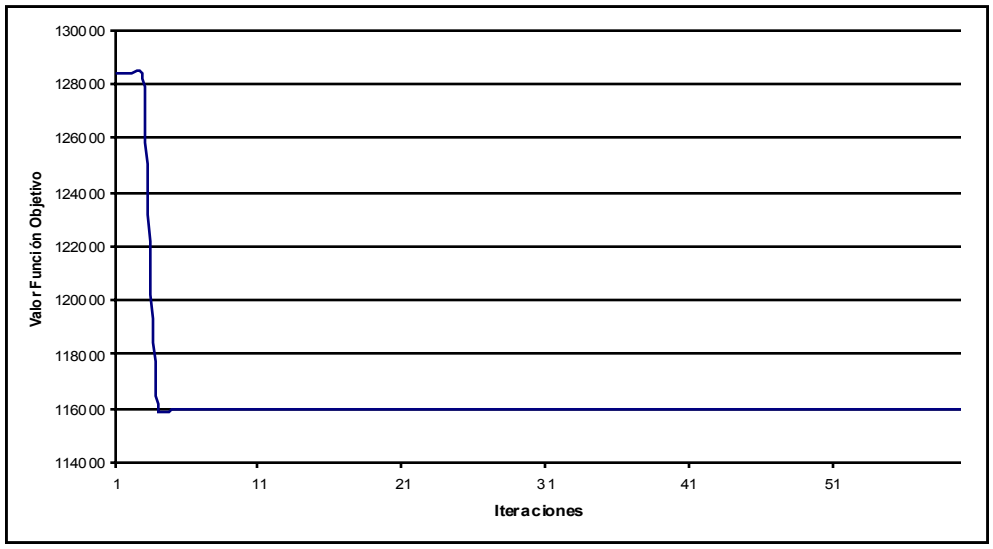

Figura 14. Evolución de la Convergencia para el problema 40

6.1.2 Algoritmo ACO2

Al igual que en las tablas 6, 7 y 8, y en las figuras 12, 13 y 14, se obtuvieron resultados para los grupos de problemas pequeño y mediano, cuando se aplicó el algoritmo ACO2. Los resultados se muestran a continuación.

Tabla 7. Diseño de Experimentos para el problema Pb8p16n9 con ACO2

Tratamiento	A	B	C	D	E	Respuesta (m)	Tiempo (hh:mm:ss)
e	10	0.3	1	0.1	0.7	73996.72833	0:00:45
a	60	0.3	1	0.1	0.3	73996.72779	0:04:25
b	10	0.9	1	0.1	0.3	73996.72835	0:00:44
abe	60	0.9	1	0.1	0.7	73996.72779	0:04:27
c	10	0.3	9	0.1	0.3	73996.72807	0:00:44
ace	60	0.3	9	0.1	0.7	73996.72833	0:04:28
bce	10	0.9	9	0.1	0.7	73996.72779	0:00:45
abc	60	0.9	9	0.1	0.3	73996.72779	0:04:22
d	10	0.3	1	0.9	0.3	73996.72833	0:00:45
ade	60	0.3	1	0.9	0.7	73996.72807	0:04:26
bde	10	0.9	1	0.9	0.7	73996.72833	0:00:45
abd	60	0.9	1	0.9	0.3	73996.72779	0:04:24
cde	10	0.3	9	0.9	0.7	73996.72833	0:00:44
acd	60	0.3	9	0.9	0.3	73996.72833	0:04:23
bcd	10	0.9	9	0.9	0.3	73996.72807	0:00:44
abcde	60	0.9	9	0.9	0.7	73996.72779	0:04:24
				Tiempo total de ejecución			0:41:15

Tabla 8. Diseño de Experimentos para el problema Pb21p26n9 con ACO2

Tratamiento	A	B	C	D	E	Respuesta (m)	Tiempo (hh:mm:ss)
e	10	0.3	1	0.1	0.7	120608.9348	0:01:40
a	60	0.3	1	0.1	0.3	120545.4212	0:09:55
b	10	0.9	1	0.1	0.3	120545.4214	0:01:40
abe	60	0.9	1	0.1	0.7	120545.4194	0:10:12
c	10	0.3	9	0.1	0.3	120608.9345	0:01:42
ace	60	0.3	9	0.1	0.7	120608.9345	0:10:52
bce	10	0.9	9	0.1	0.7	120545.4215	0:01:44
abc	60	0.9	9	0.1	0.3	120608.9341	0:10:52
d	10	0.3	1	0.9	0.3	120608.9346	0:01:37
ade	60	0.3	1	0.9	0.7	120545.4219	0:09:51
bde	10	0.9	1	0.9	0.7	121293.6554	0:01:40
abd	60	0.9	1	0.9	0.3	120608.934	0:09:52
cde	10	0.3	9	0.9	0.7	121491.5215	0:01:47
acd	60	0.3	9	0.9	0.3	120608.9344	0:10:32
bcd	10	0.9	9	0.9	0.3	120604.2444	0:01:42
abcde	60	0.9	9	0.9	0.7	120608.9341	0:10:47
				Tiempo Total de ejecución			1:36:25

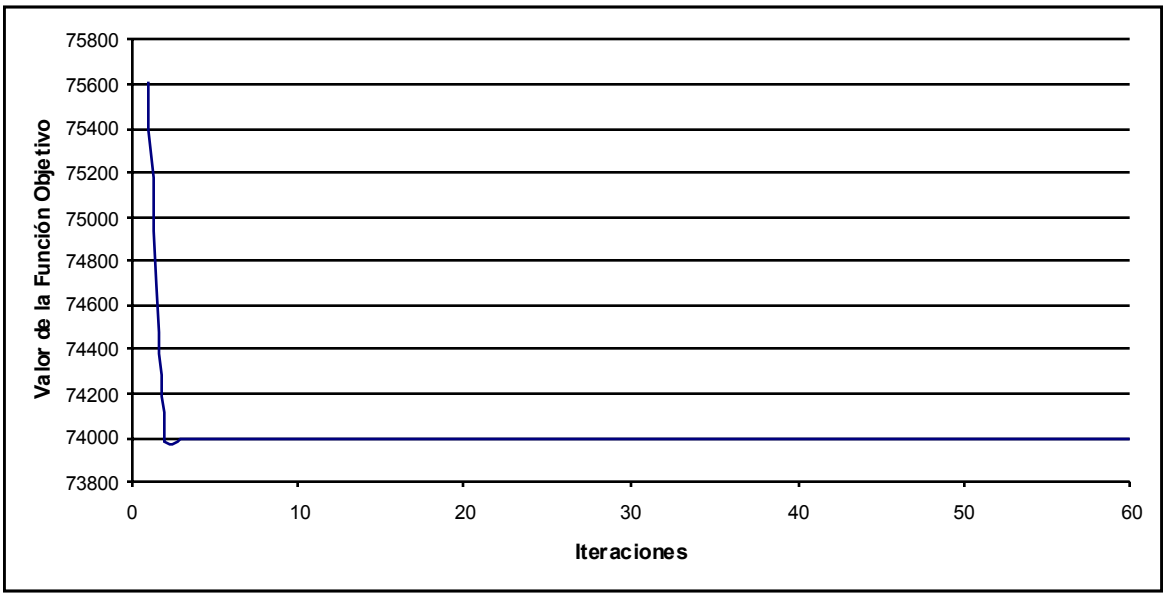

Figura 15. Evolución de la convergencia para el problema Pb8p16n9 con el Algoritmo ACO2

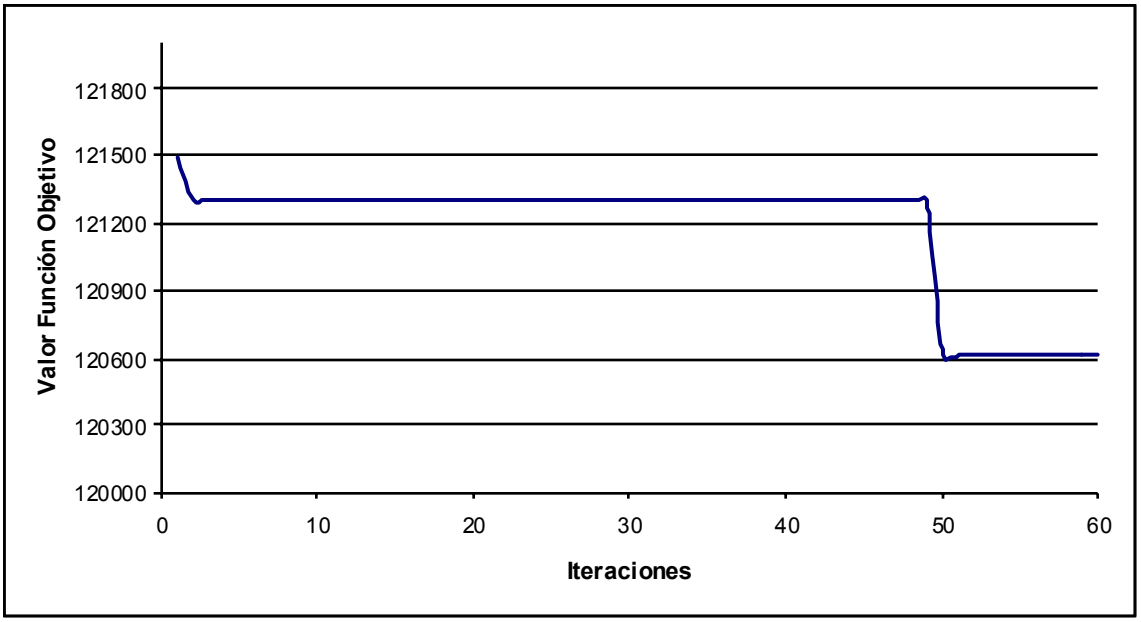

Figura 16. Evolución de la Convergencia para el problema Pb21p26n9 con el Algoritmo ACO2

Un comportamiento totalmente diferente, se observo cuando se hizo las pruebas sobre el problema Pb40p48n12, donde dos factores si fueron significantes. Con los resultados obtenidos del diseño de experimentos, se realizó un análisis de varianza, obteniendo los *P-value* registrados en la siguiente tabla.

Tabla 9. *P-value* para cada factor estudiado en el problema Pb40p48n12 con ACO2

Representación Utilizada	Factor	*P-value*
A	Número de hormigas	0.005
B	q_0	0.013
C	β	0.154
D	ρ	0.223
E	% de mejores hormigas	0.143

La Tabla 9, indica que los parámetros más significativos son el número de hormigas y el factor q_0 (en el anexo 1 están los resultados obtenidos con Minitab). Los parámetros

menos significativos son los coeficientes ρ y β, junto con el porcentaje de mejores hormigas.

Con estos resultados, se hizo un análisis de regresión obteniendo el siguiente modelo; en la Figura 17 se muestra la gráfica de probabilidad normal de los residuos.

Regression Analysis: Respuesta versus A, B

```
The regression equation is
Respuesta = 1.42 - 0.0772 A - 0.0657 B

Predictor        Coef     SE Coef          T        P
Constant      1.41596     0.02173      65.15    0.000
A            -0.07717     0.02173      -3.55    0.005
B            -0.06571     0.02173      -3.02    0.013
C             0.03355     0.02173       1.54    0.154
D            -0.02826     0.02173      -1.30    0.223
E            -0.03456     0.02173      -1.59    0.143

S = 0.08693     R-Sq = 73.9%     R-Sq(adj) = 60.9%
```

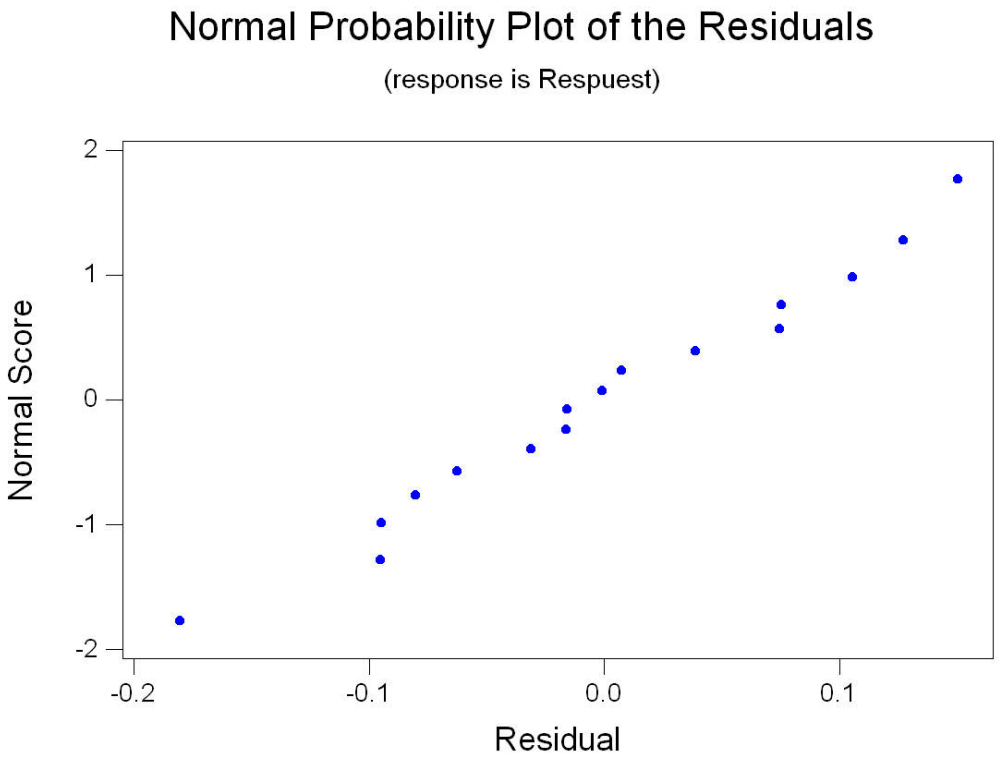

Figura 17. Probabilidad Normal de los residuos para el problema Pb40p48n12 con ACO2

Del análisis de regresión se puede concluir que el Número de Hormigas (Factor A) y q_0 (Factor B), tienen una relación inversa con la respuesta, es decir, que para minimizar la distancia total recorrida es necesario mantener en sus niveles altos dichos factores. De la Figura 17, en la cual se grafica la probabilidad normal de los residuos, se puede concluir que la suposición de normalidad se mantuvo y que el modelo es adecuado.

En la siguiente tabla, se muestra el valor de los factores que se utilizaron en la corrida de los problemas restantes (los valores de los factores no significativos, se dejaron en un nivel cero o intermedio).

Tabla 10. Valores de los factores utilizados en ACO2

Factor	Valor utilizado
Número de hormigas	60
q_0	0.9
β	5
ρ	0.5
% mejores hormigas	0.5

Con estos valores, se corrió el problema Pb40p48n12, del cual se obtuvo la Figura 18 que indica la convergencia de la respuesta.

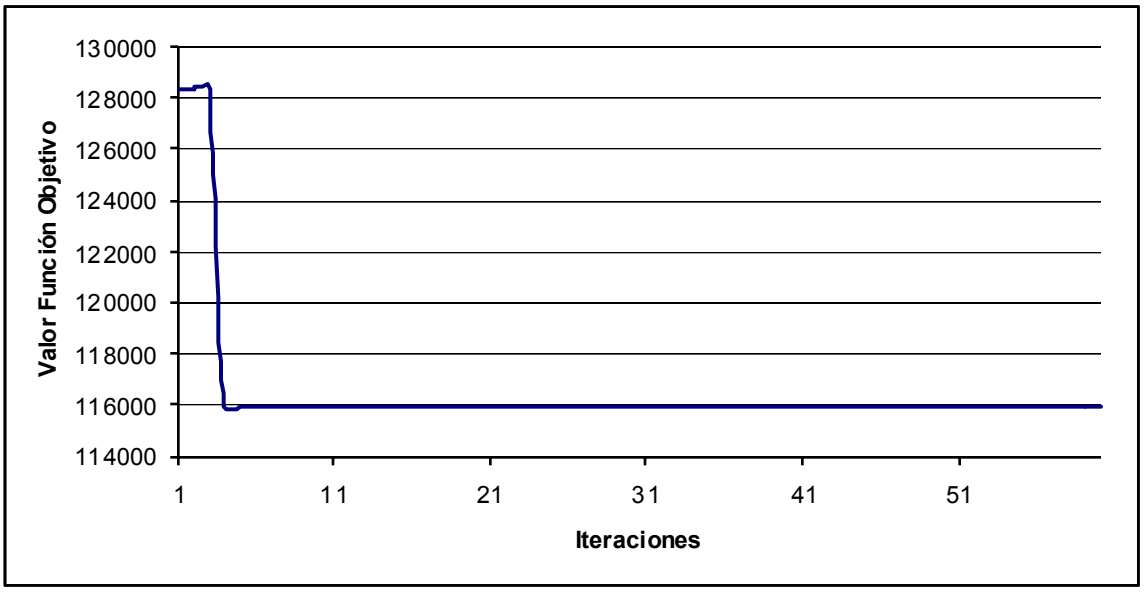

Figura 18. Convergencia para el problema Pb40p48n12 corrido con el algoritmo ACO2

En resumen del diseño de experimentos, se obtuvieron las siguientes gráficas, en las cuales se indica de manera general, el comportamiento de la respuesta frente a distintas combinaciones de los parámetros.

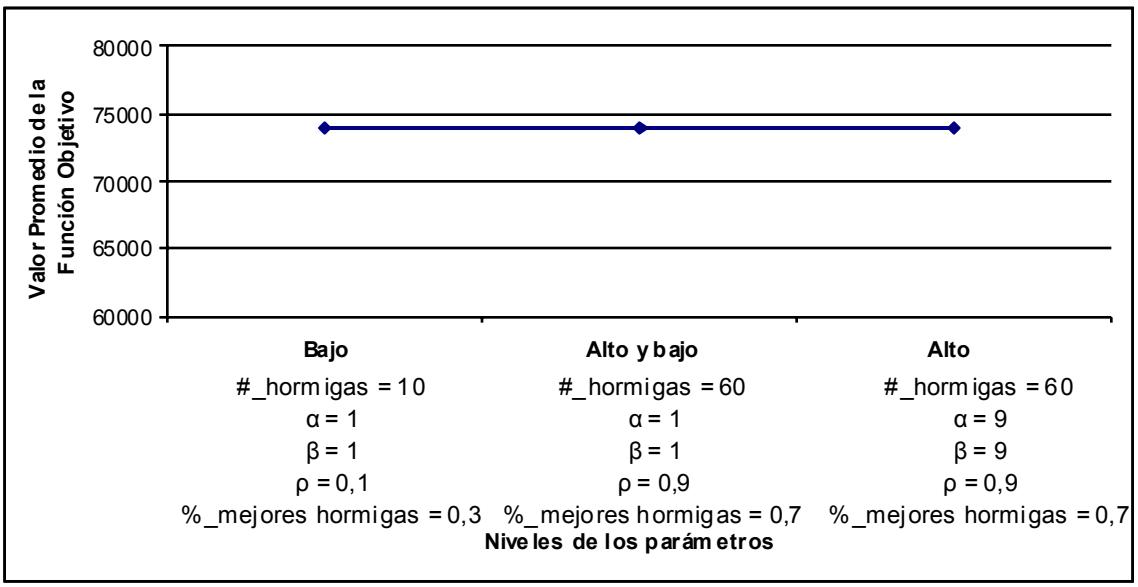

Figura 19. Resumen del comportamiento de la respuesta frente a distintas combinaciones de parámetros. Algoritmo ACO1. Problema Pb8p16n8. Número de corridas = 15

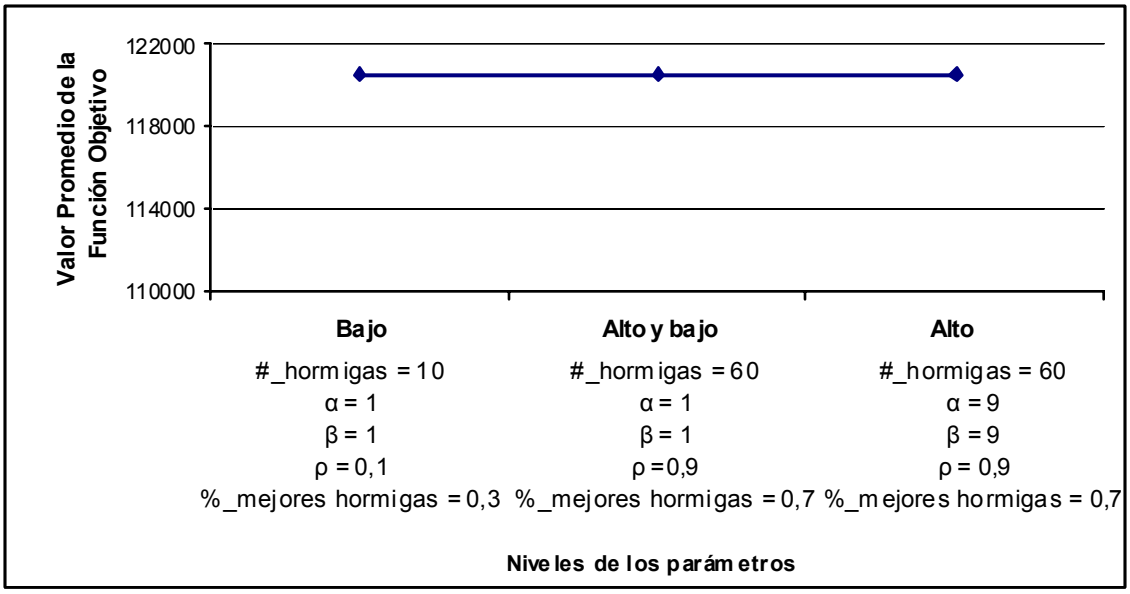

Figura 20. Resumen del comportamiento de la respuesta frente a distintas combinaciones de parámetros. Algoritmo ACO1. Problema Pb21p26n9. Número de corridas = 15

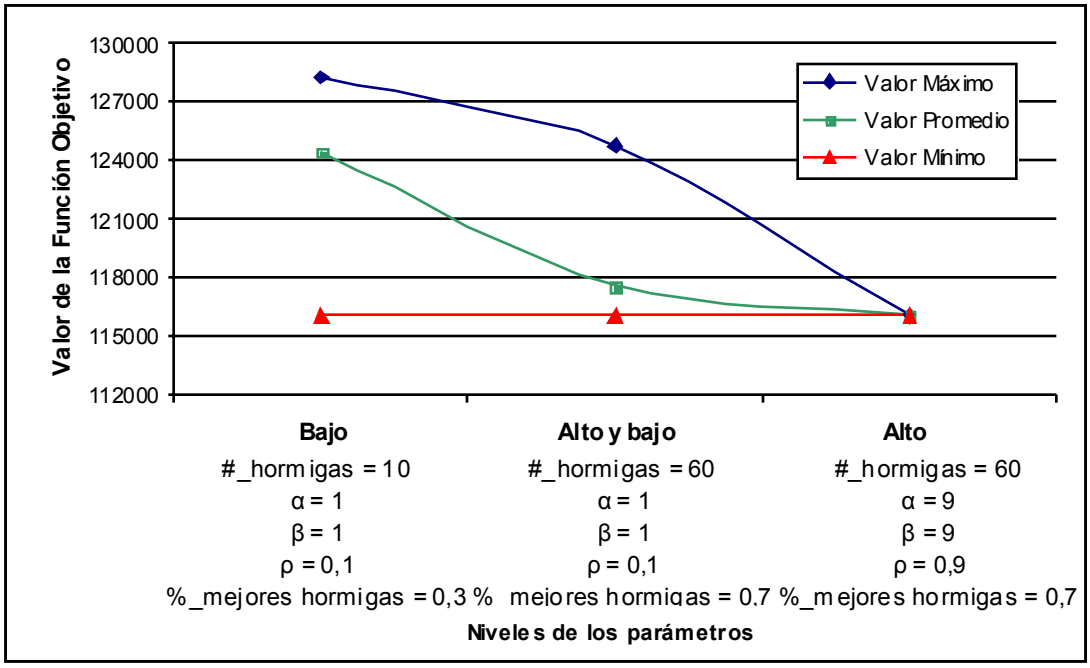

Figura 21. Resumen del comportamiento de la respuesta frente a distintas combinaciones de parámetros. Algoritmo ACO1. Problema Pb40p48n12. Número de corridas = 15

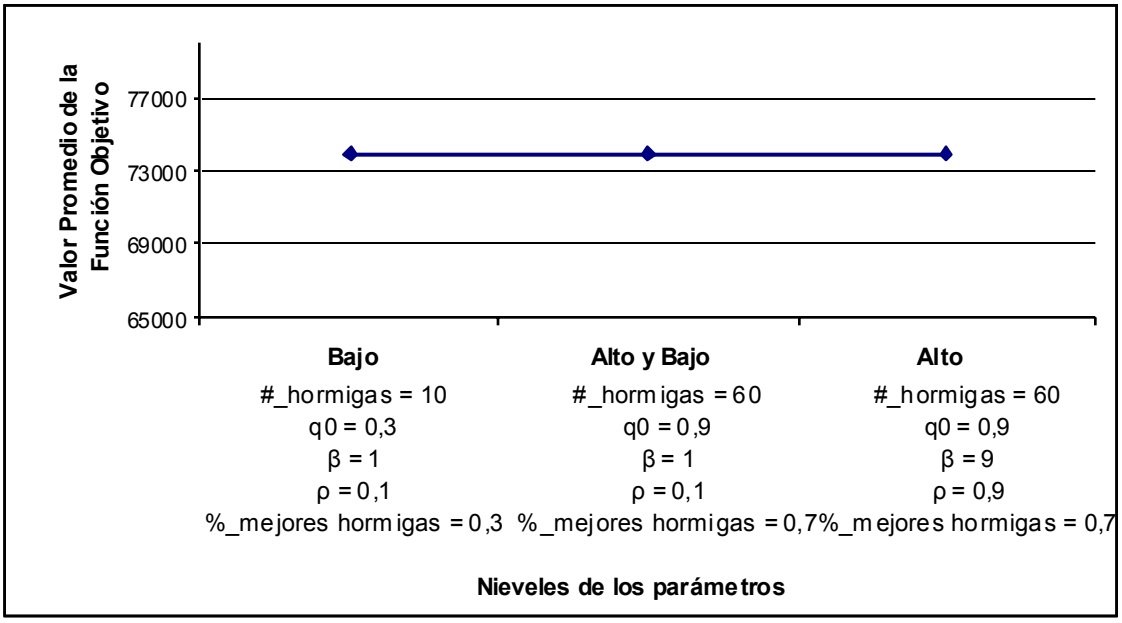

Figura 22. Resumen del comportamiento de la respuesta frente a distintas combinaciones de parámetros. Algoritmo ACO2. Problema Pb8p16n8. Número de corridas = 15

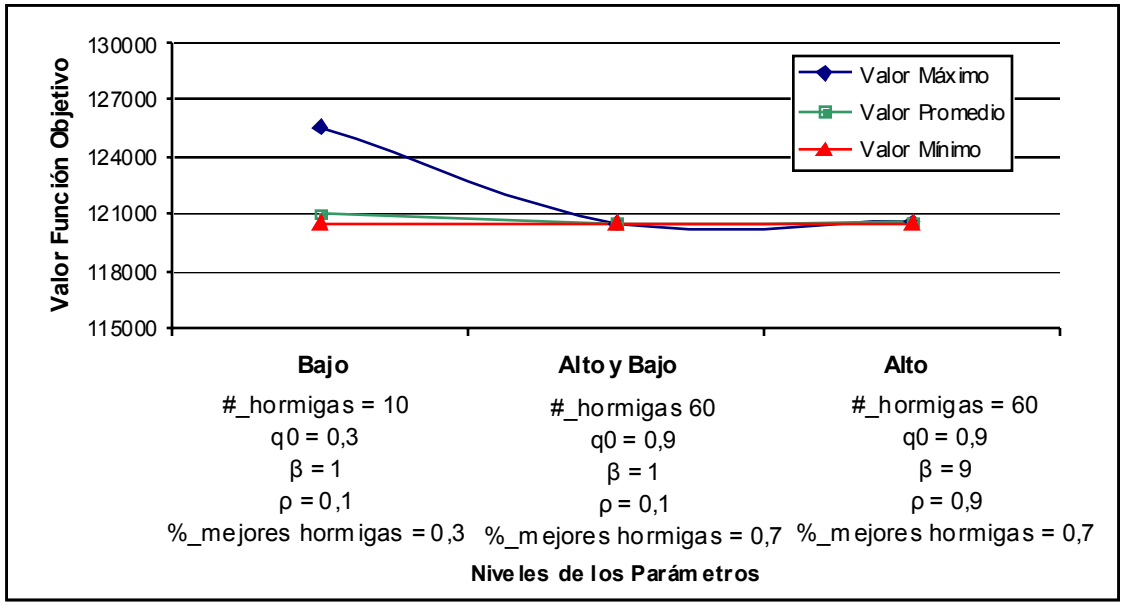

Figura 23. Resumen del comportamiento de la respuesta frente a distintas combinaciones de parámetros. Algoritmo ACO2. Problema Pb21p26n9. Número de corridas = 15

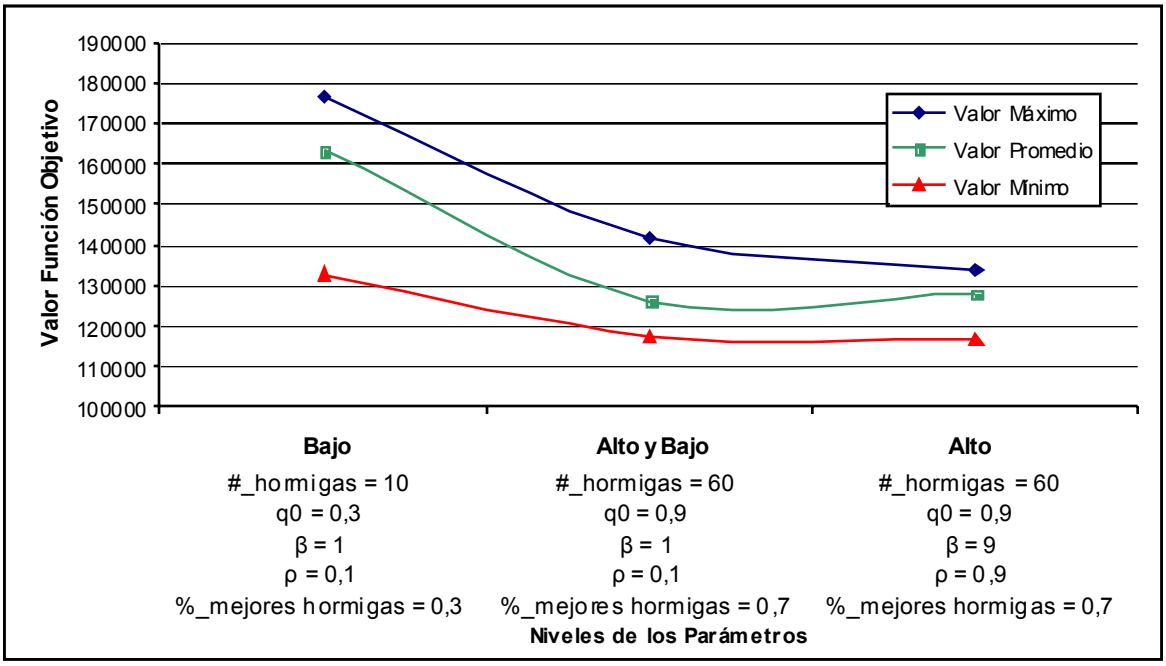

Figura 24. Resumen del comportamiento de la respuesta frente a distintas combinaciones de parámetros. Algoritmo ACO2. Problema Pb40p48n12. Número de corridas = 15

6.2 COMPARACIÓN DE LOS RESULTADOS OBTENIDOS CON LOS MEJORES REPORTADOS

Con el objetivo de medir el desempeño de los algoritmos propuestos, se hace una comparación con las respuestas realizadas manualmente por expertos programadores de ruta y con las obtenidas por medio de un algoritmo genético, reportados en Torres (2004).

6.2.1 Comparación de los resultados obtenidos entre la programación manual y las heurísticas propuestas.

Los resultados se encuentran contenidos en las tablas 13, 14, 15 y 16.

Tabla 11. Comparación programación Manual Vs ACO1 y ACO2. Grupo de problemas: Pequeño

Problema	Manual (m)	ACO1 (m)	% Mejora Manual Vs ACO1	ACO2 (m)	% Mejora Manual Vs ACO2
Pb1p9n7	88378	88378.71884	0.00%	88378.71789	0.00%
Pb2p12n6	56810	56810.43154	0.00%	56810.4312	0.00%
Pb3p13n7	43427	43427.84638	0.00%	43427.8457	0.00%
Pb4p14n7	104648	103261.7952	1.34%	103261.7944	1.34%
Pb5p15n4	62276	62276.28513	0.00%	62276.28498	0.00%
Pb6p15n7	82496	82496.27598	0.00%	82496.27476	0.00%
Pb7p16n8	123679	123612.7094	0.05%	123612.7088	0.05%
Pb8p16n9	80590	73996.7302	8.91%	73996.72779	8.91%
Pb9p17n7	94259	94259.3679	0.00%	94259.3662	0.00%
Pb10p17n8	74132	74132.77408	0.00%	74132.77332	0.00%
Pb11p17n9	93026	93026.31262	0.00%	93026.31138	0.00%
Pb12p18n5	97971	89225.31777	9.80%	89225.31637	9.80%

Tabla 12. Comparación programación Manual Vs ACO1 y ACO2. Grupo de problemas Mediano

Problema	Manual (m)	ACO1 (m)	% Mejora Manual Vs ACO1	ACO2 (m)	% Mejora Manual Vs ACO2
Pb13p20n8	85588	83874.52067	2.04%	83874.51927	2.04%
Pb14p20n9	125241	123071.9652	1.76%	123071.9632	1.76%
Pb15p21n7	89712	88070.69135	1.86%	88070.68884	1.86%
Pb16p21n8	92629	90457.49092	2.40%	90457.48979	2.40%
Pb17p23n9	123387	123322.6852	0.05%	123322.6838	0.05%
Pb18p25n10	124292	124292.81	0.00%	124292.8088	0.00%
Pb19p25n10	95048	87306.0941	8.87%	87306.09149	8.87%
Pb20p26n3	98563	98563.07186	0.00%	98563.07183	0.00%
Pb21p26n9	124747	120545.4214	3.49%	120545.419	3.49%
Pb22p28n7	130122	130122.8606	0.00%	130122.8578	0.00%
Pb23p28n9	124639	124081.0215	0.45%	124081.019	0.45%
Pb24p29n8	90210	88536.75265	1.89%	88536.75222	1.89%
Pb25p29n9	91246	91138.03694	0.12%	91138.0367	0.12%

Tabla 13. Comparación programación Manual Vs ACO1 y ACO2. Grupo de problemas Grande

Problema	Manual (m)	ACO1 (m)	% Mejora Manual Vs ACO1	ACO2 (m)	% Mejora Manual Vs ACO2
Pb26p30n9	89127	86559.69341	2.97%	89753.4661	-0.70%
Pb27p30n12	143799	140262.1992	2.52%	146508.9621	-1.88%
Pb28p31n10	94254	92489.19726	1.91%	92489.19655	1.91%
Pb29p31n12	133119	132010.0124	0.84%	132010.0098	0.84%
Pb30p32n13	152085	140971.8307	7.88%	140971.8286	7.88%
Pb31p33n10	96854	94839.70921	2.12%	94839.70864	2.12%
Pb32p33n12	167980	141750.561	18.50%	141750.5606	18.50%
Pb33p33n12	120272	95022.64372	26.57%	95022.64064	26.57%
Pb34p35n10	90129	89160.0215	1.09%	89160.02178	1.09%
Pb35p37n11	120927	120927.655	0.00%	120927.6534	0.00%
Pb36p38n11	164810	159804.0551	3.13%	159804.0534	3.13%
Pb37p44n11	113848	103232.5387	10.28%	119268.7307	-4.76%
Pb38p44n11	136153	130995.8458	3.94%	131955.7049	3.18%
Pb39p44n12	132671	129823.237	2.19%	130964.4592	1.30%
Pb40p48n12	136488	115975.7465	17.69%	117793.2134	15.87%

Tabla 14. Comparación de resultados totales de la programación Manual Vs ACO1 y ACO2

Distancia Total por programación Manual (m)	Distancia Total por ACO1 (m)	% Total de mejora Manual Vs ACO1	Distancia Total por ACO2 (m)	% Total de mejora Manual Vs ACO2
4289632	4132112,934	3,81%	4161508,164	3,08%

De las tablas anteriores, se puede decir que ACO1 consiguió igualar o mejorar los resultados obtenidos por la programación manual para el 100% de los problemas; esta situación, no se pudo conseguir para ACO2 en donde 3 problemas (Pb26p30n9, Pb27p30n12 y Pb37p44n11), incrementaron el valor de la distancia total recorrida.

A pesar de esto, se puede decir que el comportamiento general de los algoritmos en cuanto a la calidad de la respuesta, es aceptable, consiguiendo una mejora total del 3.81% para ACO1 y del 3.08% para ACO2. Para ambas heurísticas, se consiguió un máximo ahorro del 26.57%, conseguido para un problema considerado de gran tamaño.

6.2.2 Comparación entre los resultados obtenidos por Algoritmos Genéticos y ACO1 y ACO2

Las siguientes tablas tienen consignados estos resultados.

Tabla 15. Comparación AG Vs ACO1 y ACO2. Grupo de problemas Pequeño

Problema	Manual (m)	Algoritmos Genéticos (AG) (m)	ACO1 (m)	Comparación entre AG y ACO1 (%)	Tiempo (seg)	ACO2 (m)	Comparación entre AG y ACO2 (%)	Tiempo (seg)
Pb1p9n7	88378	88378	88378.71884	0.00%	11	88378.71789	0.00%	81
Pb2p12n6	56810	56810	56810.43154	0.00%	16	56810.4312	0.00%	123
Pb3p13n7	43427	43427	43427.84638	0.00%	18	43427.8457	0.00%	147
Pb4p14n7	104648	103261	103261.7952	0.00%	24	103261.7944	0.00%	175
Pb5p15n4	62276	62276	62276.28513	0.00%	16	62276.28498	0.00%	214
Pb6p15n7	82496	82496	82496.27598	0.00%	26	82496.27476	0.00%	224
Pb7p16n8	123679	123676	123612.7094	0.05%	29	123612.7088	0.05%	265
Pb8p16n9	80590	73996	73996.7302	0.00%	33	73996.72779	0.00%	266
Pb9p17n7	94259	94259	94259.3679	0.00%	29	94259.3662	0.00%	300
Pb10p17n8	74132	74132	74132.77408	0.00%	32	74132.77332	0.00%	433
Pb11p17n9	93026	93026	93026.31262	0.00%	33	93026.31138	0.00%	435
Pb12p18n5	97971	89225	89225.31777	0.00%	34	89225.31637	0.00%	415

Tabla 16. Comparación AG Vs ACO1 y ACO2. Grupo de problemas Mediano

Problema	Manual (m)	Algoritmos Genéticos (AG) (m)	ACO1 (m)	Comparación entre AG y ACO1 (%)	Tiempo (seg)	ACO2 (m)	Comparación entre AG y ACO2 (%)	Tiempo (seg)
Pb13p20n8	85588	83874	83874.52067	0.00%	48	83874.51927	0.00%	339
Pb14p20n9	125241	123071	123071.9652	0.00%	54	123071.9632	0.00%	347
Pb15p21n7	89712	88841	88070.69135	0.87%	46	88070.68884	0.87%	360
Pb16p21n8	92629	90457	90457.49092	0.00%	44	90457.48979	0.00%	374
Pb17p23n9	123387	123322	123322.6852	0.00%	66	123322.6838	0.00%	454
Pb18p25n10	124292	124292	124292.81	0.00%	82	124292.8088	0.00%	540
Pb19p25n10	95048	87306	87306.0941	0.00%	85	87306.09149	0.00%	510
Pb20p26n3	98563	98563	98563.07186	0.00%	46	98563.07183	0.00%	540
Pb21p26n9	124747	120545	120545.4214	0.00%	65	120545.419	0.00%	600
Pb22p28n7	130122	130122	130122.8606	0.00%	75	130122.8578	0.00%	720
Pb23p28n9	124639	124081	124081.0215	0.00%	91	124081.019	0.00%	840
Pb24p29n8	90210	88536	88536.75265	0.00%	90	88536.75222	0.00%	840
Pb25p29n9	91246	91138	91138.03694	0.00%	122	91138.0367	0.00%	820

Tabla 17. Comparación AG Vs ACO1 y ACO2. Grupo de problemas Grande

Problema	Manual (m)	Algoritmos Genéticos (AG) (m)	ACO1 (m)	Comparación entre AG y ACO1 (%)	Tiempo (seg)	ACO2 (m)	Comparación entre AG y ACO2 (%)	Tiempo (seg)
Pb26p30n9	89127	86296	86559.69341	-0.31%	174	89753.4661	-4.01%	1440
Pb27p30n12	143799	142402	140262.1992	1.53%	142	146508.9621	-2.88%	1403
Pb28p31n10	94254	92848	92489.19726	0.39%	148	92489.19655	0.39%	1300
Pb29p31n12	133119	132010	132010.0124	0.00%	151	132010.0098	0.00%	1433
Pb30p32n13	152085	140971	140971.8307	0.00%	166	140971.8286	0.00%	1450
Pb31p33n10	96854	94839	94839.70921	0.00%	172	94839.70864	0.00%	1450
Pb32p33n12	167980	141750	141750.561	0.00%	215	141750.5606	0.00%	1380
Pb33p33n12	120272	95022	95022.64372	0.00%	193	95022.64064	0.00%	1200
Pb34p35n10	90129	89160	89160.0215	0.00%	198	89160.02178	0.00%	1500
Pb35p37n11	120927	120927	120927.655	0.00%	231	120927.6534	0.00%	1680
Pb36p38n11	164810	160359	159804.0551	0.35%	224	159804.0534	0.35%	1697
Pb37p44n11	113848	105826	103232.5387	2.51%	285	119268.7307	-12.70%	2940
Pb38p44n11	136153	130995	130995.8458	0.00%	265	131955.7049	-0.73%	2462
Pb39p44n12	132671	129893	129823.237	0.05%	319	130964.4592	-0.82%	2640
Pb40p48n12	136488	115984	115975.7465	0.01%	403	117793.2134	-1.56%	3360

Tabla 18. Comparación de Resultados Totales AG Vs ACO1 y ACO2

Manual (m)	Algoritmos Genéticos (AG) (m)	ACO1 (m)	Comparación entre AG y ACO1 (%)	ACO2 (m)	Comparación entre AG y ACO2 (%)
4289632	4138392	4132112,934	0,15%	4161508,164	-0,56%

En cuanto a la calidad de la solución encontrada, se pude concluir que para problemas pequeños y medianos (número de solicitudes menor a 30), las heurísticas propuestas tienen un comportamiento similar al algoritmo genético propuesto por Torres (2004),

incluso, en algunos de ellos se puede conseguir pequeñas mejoras. Para problemas de gran tamaño (mayores a 30 solicitudes), solo ACO1 mantiene este desempeño con bajos tiempos de ejecución (menor a 5 minutos). Por su parte, ACO2 disminuye la calidad de la solución encontrada y aumenta de forma drástica el tiempo computacional.

Globalmente, ACO1 reporta una mejora del 0.15% frente al algoritmo genético, mientras que ACO2 pierde un 0.56% contra esta misma meta-heurística.

7 CONCLUSIONES Y RECOMENDACIONES PARA TRABAJOS FUTUROS

En este trabajo, se utilizó una metodología híbrida, que combinó una meta-heurística relativamente nueva, la colonia de hormigas con una herramienta muy versátil como la heurística de inserción. Este híbrido permitió encontrar y mejorar las soluciones que hasta el momento habían sido reportadas para una familia de problemas de Ruteo de un helicóptero.

En instancias pequeñas de esta familia de problemas, se puede concluir que no hay diferencias sustanciales entre los resultados obtenidos por medio de la programación manual y las heurísticas propuestas en este trabajo y en Torres (2004). La diferencia, se aprecia para problemas grandes (con un número de solicitudes mayor a 30), en donde las implementaciones computacionales tuvieron un gran desempeño en cuanto a calidad de la solución y tiempo de ejecución.

Por medio del diseño de experimentos, se puede concluir los algoritmos propuestos no dependen de sus parámetros para problemas relativamente pequeños (menor a 30 solicitudes), consiguiendo para cualquier combinación de ellos, la mejor respuesta. Caso contrario ocurre, para grandes instancias, donde los parámetros si influyen directamente con la calidad de la solución.

ACO1, una heurística basada en el procedimiento del *Ant System*, fue el algoritmo de mejor desempeño, superando en algunas instancias, al algoritmo genético propuesto por Torres (2004), que éste a su vez, había superado a la programación manual. Además, con relación al tiempo de ejecución, ACO1 produjo buenas soluciones en un tiempo que no excedió los 5 minutos.

Con relación a ACO2, se puede concluir que también alcanzó en la mayoría de los casos, las soluciones óptimas, pero con elevados tiempos de ejecución.

Como trabajos futuros, se propone utilizar esta técnica de solución híbrida, para atacar problemas más realistas, que tengan en cuenta el uso de una flota de vehículos, de restricciones como ventanas de tiempo, inconvenientes causados a los pasajeros (por ejemplo, el exceso de permanencia en el vehículo) y capacidad variable del helicóptero.

Además, también se propone hacer uso de otro tipo de meta-heurísticas, como la búsqueda tabú y recocido simulado, con el objetivo de comparar su desempeño que las técnicas ya estudiadas.

LISTA DE REFERENCIAS

Bergvinsdottir, K. B. (2004). The Genetic Algorithm for solving the Dial-a-Ride Problem. Theses of master non publicated. Technical University of Denmark. Recuperado el 20 de octubre de 2004 de http://www.imm.dtu.dk/pubdb/views/publication_details.php?id=3229

Bullnheimer, B., Hartl, R. & Strauss, Ch. (1999). A new rank based version of the Ant System. Central European Journal of Operations Research 7(1), p. 25-38. Recuperado el 1 de octubre de 2005 de la base de datos EBSCO.

Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., & Trubian, M. (1996). Heuristic from Nature for Hard Combinatorial Optimization Problems. Int. Trans. Opl. Res. Vol. 3. No. 1, pp, 1-21.

Cordeau, J. F. & Laporte, G. (2003a). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. Transportation Research Part B 37 pp. 579 – 594. Recuperado el 7 de noviembre de 2004 de http://www.iro.umontreal.ca/~marcotte/PLU6000/PLU6000_H04/Cordeau2.pdf

Cordeau, J. F. & Laporte, G. (2003b). The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms. Quartely Journal of the Belgian, French and Italian Operations Research Societies, 4OR 1, p. 89-101.

Doerner, K., Hartl, R. F. & Reimann, M. (2000). Ant Colony Optimization applied to the Pickup and Delivery Problem. SFB Adaptive Information System and Modelling in Economics and Mangment Science. Vienna University of Economics and Business Administration Working Paper No. 76. Recuperado el 1 de noviembre de 2004 de http://epub.wu-wien.ac.at/dyn/virlib/wp/mediate/epub-wu-01_e4.pdf?ID=epub-wu-01_e4

Dorigo, M., & Gambardella, L. M. (1997a). Ant Colonies for the Traveling Salesman Problem. TR/IRIDIA/1996-3. Université Libre de Bruxelles. Belgium. pp. 1-10. Recuperado el 1 de octubre de 2004 de www.idsia.ch/~luca/acs-bio97.pdf

Dorigo, M., & Gambardella, L. M. (1997b). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation. Vol. 1, No. 1. Recuperado el 1 de octubre de 2004 de http://iridia.ulb.ac.be/~mdorigo/ACO/publications.html

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, Vol. 26, No. 1, pp. 1-13.

Fiala, M. & Pulleyblank W. (1992). Precedence Constrained Routing and Helicopter Scheduling: Heuristic Design. Interfaces 22: 3 May – June, p. 100 – 111. Recuperado el 10 de octubre de 2005 de la base de datos EBSCO.

Gambardella, L. M., Taillard, E. & Agazzi, G. (1999). MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. Technical Report IDSIA. IDSIA-06-99. Lugano, Switzerland. Recuperado el 1 de octubre de 2004 de http://citeseer.ist.psu.edu/450222.html

Jaw, J.J., Odoni, A. R., Psaraftis, H.N & Wilson, N.H.. (1986). A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. Transportation Research, part B, 20B(3), pp. 243-257.

Lau & Liang (2002). "Pick Up and Delivery with Time Windows: Algorithms and test Case Generation"; International Journal on Artificial Intelligence Tools, Vol. 11, No. 3, page 455 – 472.

Maniezzo, V., Gambardella, Luca M. & Luigi, F. Ant Colony Optimization. New Optimization Techniques in Engineering, by Onwubolu, G. C., and B. V. Babu, Springer-Verlag Berlin Heidelberg, 101-117, 2004. Recuperado el 10 de octubre de 2004 de http://www.idsia.ch/~luca/aco2004.pdf

Montgomery, Douglas. (Fifth Edition). (2001). Design and Analysis of Experiments. United States of America: John Wiley & Sons, Inc.

Savelsberch, M. & Sol, M. (1995). The General Pickup and Delivery Problem. Transportation Science, Vol. 29, No. 1, 17-29. Recuperado el 1 de noviembre de 2004 de la base de datos EBSCO.

Torres, F. (2004). Un Algoritmo Genético basado en heurística de inserción para el problema de ruteo de helicópteros. XII Congreso Latino Iberoamericano de Investigación de Operaciones. Ciudad de Habana. Cuba. Octubre 4-8.

ANEXO 1 RESULTADOS OBTENIDOS CON MINITAB PARA EL PROBLEMA 40 CON ACO2

Factor	Type	Levels	Values
A	fixed	2	-1 1
B	fixed	2	-1 1
C	fixed	2	-1 1
D	fixed	2	-1 1
E	fixed	2	-1 1

Analysis of Variance for Respuest, using Adjusted SS for Tests

Source	DF	Seq SS	Adj SS	Adj MS	F	P
A	1	0.095286	0.095286	0.095286	12.61	0.005
B	1	0.069093	0.069093	0.069093	9.14	0.013
C	1	0.018007	0.018007	0.018007	2.38	0.154
D	1	0.012774	0.012774	0.012774	1.69	0.223
E	1	0.019108	0.019108	0.019108	2.53	0.143
Error	10	0.075571	0.075571	0.007557		
Total	15	0.289837				

Regression Analysis: Respuesta versus A, B

The regression equation is
Respuesta = 1.42 - 0.0772 A - 0.0657 B

Predictor	Coef	SE Coef	T	P
Constant	1.41596	0.02173	65.15	0.000
A	-0.07717	0.02173	-3.55	0.005
B	-0.06571	0.02173	-3.02	0.013
C	0.03355	0.02173	1.54	0.154
D	-0.02826	0.02173	-1.30	0.223
E	-0.03456	0.02173	-1.59	0.143

S = 0.08693 R-Sq = 73.9% R-Sq(adj) = 60.9%

Analysis of Variance

Source	DF	SS	MS	F	P
Regression	5	0.214267	0.042853	5.67	0.010
Residual Error	10	0.075571	0.007557		
Total	15	0.289837			

ANEXO 2 IMPLEMENTACION DEL CODIGO PARA ACO1

```
Public capacidad As Integer
Public pasajeros As Integer
Public nodos As Integer
Public coordenada_x(1000)
Public coordenada_y(1000)
Public origen(1000) As Integer
Public destino(1000) As Integer
Public nodo_origen As Integer
Public nodo_destino As Integer
Public d(100, 100)
Public r(1000) As Integer
Public N As Integer
Public vx
Public carga(1000) As Integer
Public indice_origen(1000) As Integer
Public indice_fin(1000) As Integer
Public probabilidad(100, 100)
Public eta(100, 100)
Public tao(100, 100)
Public max_iteraciones As Integer
Public hormigas As Integer
Public tao_0
Public secuencia_p(100) As Integer
Public r_pasajeros(100) As Integer
Public alfa
Public beta
Public ro
Public mejores_hormigas As Integer
Public v_malos(300) As Integer
Public v_fobjs(300)
Public mejores_pasajeros(300, 200) As Integer
Public mejores_rutas(300, 200) As Integer
Public mejores_N(300) As Integer
Public nodo_visitado(100) As Boolean
Public Lista_salen(100, 100) As Integer
Public Lista_entran(100, 100) As Integer
Public numero_salen(100) As Integer
Public numero_entran(100) As Integer
Public suma
'

Function genere_entero_entre(a, b)
u = Rnd()
genere_entero_entre = Int(a * (1 - u) + (b + 1) * u)
End Function

Sub inicialice(vx, l)
vx = 1
Randomize
Cells(vx, 1).Value = "Ruteo de Helicópteros"
vx = vx + 1
Cells(vx, 1).Value = "Capacidad = "
capacidad = Cells(vx, 2).Value
vx = vx + 1
Cells(vx, 1).Value = "Nodos= "
nodos = Cells(vx, 2).Value
vx = vx + 1
Cells(vx, 1).Value = "Pasajeros= "
```

```
pasajeros = Cells(vx, 2).Value
vx = vx + 1
Cells(vx, 1).Value = "Nodo de origen= "
'nodo_origen = genere_entero_entre(1, nodos)
'Cells(vx, 2).Value = nodo_origen
nodo_origen = Cells(vx, 2).Value
vx = vx + 1
Cells(vx, 1).Value = "Nodo de destino= "
'nodo_destino = genere_entero_entre(1, nodos)
'Cells(vx, 2).Value = nodo_destino
nodo_destino = Cells(vx, 2).Value
'vx = vx + 1
'
'Cells(vx, 1).Value = "Coordenadas de Nodos "
'vx = vx + 1
'Cells(vx, 1).Value = "Nodo"
'Cells(vx, 2).Value = "Coordenada x"
'Cells(vx, 3).Value = "Coordenada y"
'vx = vx + 1
'For i = 1 To nodos
'    Cells(vx, 1).Value = i
'    coordenada_x(i) = genere_entero_entre(0, 100)
'    coordenada_y(i) = genere_entero_entre(0, 100)
'    Cells(vx, 2).Value = coordenada_x(i)
'    Cells(vx, 3).Value = coordenada_y(i)
'    coordenada_x(i) = Cells(vx, 2).Value
'    coordenada_y(i) = Cells(vx, 3).Value
'    vx = vx + 1
'Next i
vx = vx + 1
Cells(vx, 1).Value = "Origenes y Destinos de pasajeros "
vx = vx + 1
Cells(vx, 1).Value = "Pasajero"
Cells(vx, 2).Value = "Origen"
Cells(vx, 3).Value = "Destino"
vx = vx + 1
For i = 1 To pasajeros
   Cells(vx, 1).Value = i
'    origen(i) = genere_entero_entre(1, nodos)
'    destino(i) = genere_entero_entre(1, nodos)
'    While origen(i) = destino(i)
'        origen(i) = genere_entero_entre(1, nodos)
'        destino(i) = genere_entero_entre(1, nodos)
'    Wend
'    Cells(vx, 2).Value = origen(i)
'    Cells(vx, 3).Value = destino(i)
   origen(i) = Cells(vx, 2).Value
   destino(i) = Cells(vx, 3).Value
   vx = vx + 1
Next i
vx = vx + 1
Cells(vx, 1).Value = "Matriz de distancias "
vx = vx + 1
'For i = 1 To nodos
'    d(i, i) = 0
'    For j = i + 1 To nodos
'        d(i, j) = Sqr((coordenada_x(i) - coordenada_x(j)) ^ 2 + (coordenada_y(i) - coordenada_y(j)) ^ 2)
'        d(j, i) = d(i, j)
'    Next j
'
'Next i
For i = 1 To nodos
   For j = 1 To nodos
'        Cells(vx, j).Value = d(i, j)
       d(i, j) = Cells(vx, j).Value
   Next j
   vx = vx + 1
```

```
Next i
For i = 0 To 1000
   carga(i) = 0
Next i

max_iteraciones = 60
tao_0 = 0.1
hormigas = Worksheets("grande").Cells(17 + l, 25)
alfa = Worksheets("grande").Cells(17 + l, 26)
beta = Worksheets("grande").Cells(17 + l, 27)
ro = Worksheets("grande").Cells(17 + l, 28)
mejores_hormigas = Math.Round(hormigas * Worksheets("grande").Cells(17 + l, 29), 0)

For i = 1 To 100
For j = 1 To 100
   tao(i, j) = tao_0
Next j
Next i
End Sub

Sub escribir_ruta(vx)
'vx = vx + 1
'Cells(vx, 1).Value = "Ruta actual "
'vx = vx + 1

'For i = 0 To N + 1
'    Cells(vx, i + 1).Value = r(i)
'
'Next i
'vx = vx + 1
'Cells(vx, 1).Value = "Carga actual "
'vx = vx + 1
'For i = 0 To N
'    Cells(vx, i + 1).Value = carga(i)
'
'Next i
'suma = 0
'For i = 1 To N - 1
'    suma = suma + d(r(i), r(i + 1))
'Next i
'vx = vx + 1
'Cells(vx, 1).Value = "Longitud  actual ="
'Cells(vx, 2).Value = suma
'vx = vx + 1
End Sub

Sub escribir_ruta_v2(vx)
vx = vx + 1
Cells(vx, 1).Value = "Ruta actual "
vx = vx + 1

For i = 0 To N + 1
   Cells(vx, i + 1).Value = r(i)

Next i
vx = vx + 1
Cells(vx, 1).Value = "Carga actual "
vx = vx + 1
For i = 0 To N
   Cells(vx, i + 1).Value = carga(i)

Next i
'suma = 0
'For i = 1 To N - 1
'    suma = suma + d(r(i), r(i + 1))
'Next i
'vx = vx + 1
```

```
'Cells(vx, 1).Value = "Longitud  actual ="
'Cells(vx, 2).Value = suma
'vx = vx + 1
End Sub


Sub calcule_parametros_insercion(p, i, j, mejor_costo, indice_o_i, indice_o_j)
i = origen(p)
j = destino(p)
k = 0
'vx = vx + 1
'Cells(vx, 1).Value = p
'Cells(vx, 3).Value = i
'Cells(vx, 4).Value = j
'vx = vx + 1
primera_vez = True

While k <= -N
'    Cells(vx, 1).Value = "Examina segmento "
'    Cells(vx, 2).Value = k
'    vx = vx + 1

   If carga(k) < capacidad Then

      If i <> r(k) Then
         costo1 = d(r(k), i) + d(i, r(k + 1)) - d(r(k), r(k + 1))

      Else
         costo1 = 0
      End If

      indice_o_1_i = k

      'el nodo i se inserta entre k y k+1


      l = k + 1

      puede_seguir = True

      If j <> r(l) Then
         costo2 = d(r(k), i) + d(i, j) + d(j, r(k + 1)) - d(r(k), r(k + 1))

         indice_o_2_i = k

         indice_o_2_j = k


         If carga(l) < capacidad Then
            If l <= N Then
               costo3 = costo1 + d(r(l), j) + d(j, r(l + 1)) - d(r(l), r(l + 1))
               indice_o_3_i = k

               indice_o_3_j = l

               If primera_vez Then
                  primera_vez = False
                  If costo2 < costo3 Then
                     mejor_costo = costo2
                     indice_o_i = indice_o_2_i

                     indice_o_j = indice_o_2_j

                  Else
                     mejor_costo = costo3
                     indice_o_i = indice_o_3_i
```

```
                indice_o_j = indice_o_3_j

            End If
        Else
            If costo2 < mejor_costo Then
                mejor_costo = costo2
                indice_o_i = indice_o_2_i

                indice_o_j = indice_o_2_j

            End If
            If costo3 < mejor_costo Then
                mejor_costo = costo3
                indice_o_i = indice_o_3_i

                indice_o_j = indice_o_3_j

            End If
        End If
    Else
        If primera_vez Then
            primera_vez = False
            mejor_costo = costo2
            indice_o_i = indice_o_2_i

            indice_o_j = indice_o_2_j

        Else
            If costo2 < mejor_costo Then
                mejor_costo = costo2
                indice_o_i = indice_o_2_i

                indice_o_j = indice_o_2_j

            End If
        End If
    End If
Else
'NO PUEDE SEGUIR INCREMENTANDO L
    If primera_vez Then
        primera_vez = False
        mejor_costo = costo2
        indice_o_i = indice_o_2_i

        indice_o_j = indice_o_2_j

    Else
        If costo2 < mejor_costo Then
            mejor_costo = costo2
            indice_o_i = indice_o_2_i

            indice_o_j = indice_o_2_j

        End If
    End If
    puede_seguir = False
End If
Else
    indice_o_1_j = l

    costo = costo1
    If primera_vez Then
        primera_vez = False
        mejor_costo = costo
        indice_o_i = indice_o_1_i

        indice_o_j = indice_o_1_j
```

```
    Else
        If costo < mejor_costo Then
            mejor_costo = costo
            indice_o_i = indice_o_1_i

            indice_o_j = indice_o_1_j

        End If
    End If
End If



If puede_seguir Then
    l = k + 2
Else
    l = N + 1
End If


'el nodo i se inserta entre k y k+1

While l <= N
    If j <> r(l) Then
        If carga(l - 1) < capacidad Then
            costo2 = costo1 + d(r(l - 1), j) + d(j, r(l)) - d(r(l - 1), r(l))

            indice_o_2_i = k

            indice_o_2_j = l - 1



            If primera_vez Then
                primera_vez = False

                mejor_costo = costo2
                indice_o_i = indice_o_2_i

                indice_o_j = indice_o_2_j


            Else
                If costo2 < mejor_costo Then
                    mejor_costo = costo2
                    indice_o_i = indice_o_2_i

                    indice_o_j = indice_o_2_j

                End If
            End If
        Else
        'NO PUEDE SEGUIR InCREMENTADO L
            l = N + 1
        End If

        If l <= N Then
            If carga(l) < capacidad Then
                costo3 = costo1 + d(r(l), j) + d(j, r(l + 1)) - d(r(l), r(l + 1))

                indice_o_3_i = k

                indice_o_3_j = l

                If primera_vez Then
```

```
                primera_vez = False
                mejor_costo = costo3
                indice_o_i = indice_o_3_i

                indice_o_j = indice_o_3_j

              Else
                If costo3 < mejor_costo Then
                  mejor_costo = costo3
                  indice_o_i = indice_o_3_i

                  indice_o_j = indice_o_3_j

                End If
              End If
            Else
            'NO PUEDE SEGUIR InCREMENTADO L
              l = N + 1
            End If
          End If

        Else
          costo = costo1
          indice_o_1_j = l

          If primera_vez Then
            primera_vez = False
            mejor_costo = costo
            indice_o_i = indice_o_1_i

            indice_o_j = indice_o_1_j

          Else
            If costo < mejor_costo Then
              mejor_costo = costo
              indice_o_i = indice_o_1_i

              indice_o_j = indice_o_1_j

            End If
          End If
        End If



        l = l + 1
      Wend


    Else

    End If
    k = k + 1
  Wend

  'vx = vx + 1
  'Cells(vx, 1).Value = "Mejor Costo = "
  'Cells(vx, 2).Value = mejor_costo
  'vx = vx + 1
  'Cells(vx, 1).Value = "El nodo "
  'Cells(vx, 2).Value = i
  'Cells(vx, 3).Value = "Entra entre las posiciones"
  'Cells(vx, 4).Value = indice_o_i
  'Cells(vx, 5).Value = indice_o_i + 1
  'vx = vx + 1
  'Cells(vx, 1).Value = "El nodo "
  'Cells(vx, 2).Value = j
```

```
'Cells(vx, 3).Value = "Entra entre las posiciones"
'Cells(vx, 4).Value = indice_o_j
'Cells(vx, 5).Value = indice_o_j + 1
'vx = vx + 1

k = 0
While k <= N
   l = k
   costo1 = d(r(k), i) + d(i, r(k + 1)) - d(r(k), r(k + 1))
   While l <= N
      If k = 0 And l = 0 Then
         mejor_costo = d(r(0), i) + d(i, j) + d(j, r(1))
         indice_o_i = k
         indice_o_j = k
         l = l + 1
      Else
         If k = l Then
            If carga(l) < capacidad Then
               costo2 = d(r(k), i) + d(i, j) + d(j, r(k + 1)) - d(r(k), r(k + 1))
               If costo2 < mejor_costo Then
                  mejor_costo = costo2
                  indice_o_i = k
                  indice_o_j = l
               End If
               l = l + 1
            Else

               l = N + 1
            End If
         Else
            If carga(l) < capacidad Then
               costo2 = costo1 + d(r(l), j) + d(j, r(l + 1)) - d(r(l), r(l + 1))
               If costo2 < mejor_costo Then
                  mejor_costo = costo2
                  indice_o_i = k
                  indice_o_j = l
               End If
               l = l + 1
            Else

               l = N + 1
            End If
         End If
      End If
   Wend
   k = k + 1
Wend

'vx = vx + 1
'Cells(vx, 1).Value = "Mejor Costo = "
'Cells(vx, 2).Value = mejor_costo
'vx = vx + 1
'Cells(vx, 1).Value = "El nodo "
'Cells(vx, 2).Value = i
'Cells(vx, 3).Value = "Entra entre las posiciones"
'Cells(vx, 4).Value = indice_o_i
'Cells(vx, 5).Value = indice_o_i + 1
'vx = vx + 1
'Cells(vx, 1).Value = "El nodo "
'Cells(vx, 2).Value = j
'Cells(vx, 3).Value = "Entra entre las posiciones"
'Cells(vx, 4).Value = indice_o_j
'Cells(vx, 5).Value = indice_o_j + 1
'vx = vx + 1


'If i = r(indice_o_i) Or i = r(indice_o_i + 1) Then
```

```
'    If j = r(indice_o_j) Or j = r(indice_o_j + 1) Then
       If indice_o_i < indice_o_j Then
'          vx = vx + 1
'          Cells(vx, 1).Value = "MMMEJJJORRA"
'          vx = vx + 1
           buscar_ind = True
           kk = indice_o_i
           While buscar_ind
'              If r(kk) = r(indice_o_i) And r(kk + 1) = r(indice_o_i + 1) Then
              If r(kk) = i Or r(kk + 1) = i Or (r(kk) = r(indice_o_i) And r(kk + 1) = r(indice_o_i + 1)) Then
                 mejor_o_i = kk
              End If
              kk = kk + 1
              If kk = indice_o_j Then buscar_ind = False
           Wend

           mejor_o_j = indice_o_j
           kk = indice_o_j
           buscar_ind = True
           While buscar_ind
'              If r(kk) = r(indice_o_j) And r(kk + 1) = r(indice_o_j + 1) Then
              If r(kk) = j Or r(kk + 1) = j Or (r(kk) = r(indice_o_j) And r(kk + 1) = r(indice_o_j + 1)) Then
                 mejor_o_j = kk
              End If
              kk = kk - 1
              If kk = mejor_o_i Then buscar_ind = False
           Wend
'          Cells(vx, 1).Value = "El nodo "
'          Cells(vx, 2).Value = i
'          Cells(vx, 3).Value = "Entra entre las posiciones"
'          Cells(vx, 4).Value = mejor_o_i
'          Cells(vx, 5).Value = mejor_o_i + 1
'          vx = vx + 1
'          Cells(vx, 1).Value = "El nodo "
'          Cells(vx, 2).Value = j
'          Cells(vx, 3).Value = "Entra entre las posiciones"
'          Cells(vx, 4).Value = mejor_o_j
'          Cells(vx, 5).Value = mejor_o_j + 1
'          vx = vx + 1
           indice_o_i = mejor_o_i
           indice_o_j = mejor_o_j
       End If

'    End If
'End If
End Sub

Sub mejore_origen_fin_pasajeros(n_p, vx)
'vx = vx + 1
'Cells(vx, 1).Value = "Pasajeros = "
'Cells(vx, 2).Value = n_p
'vx = vx + 1
'Cells(vx, 1).Value = "kkk"
'Cells(vx, 2).Value = "L"
'Cells(vx, 3).Value = "F"
'Cells(vx, 4).Value = "i"
'Cells(vx, 5).Value = "j"
'vx = vx + 1
For kk = 1 To n_p
   kkk = r_pasajeros(kk)
   i = origen(kkk)
   j = destino(kkk)
   'nodos de origen y destino del pasajero kkk
   l = indice_origen(kkk)
   'L es el indice de origen del pasajero kkk en la ruta r
   F = indice_fin(kkk)
   'F es el indice de fin del pasajero kkk en la ruta r
```

```
'    Cells(vx, 1).Value = kkk
'    Cells(vx, 2).Value = L
'    Cells(vx, 3).Value = F
'    Cells(vx, 4).Value = i
'    Cells(vx, 5).Value = j
'    vx = vx + 1
    k = l + 1
    buscar = True
    L_nuevo = l
    While buscar
        If r(k) = i Then L_nuevo = k
        If k = F Then buscar = False
        k = k + 1
    Wend
    k = F - 1
    buscar = True
    F_nuevo = F
    While buscar
        If r(k) = j Then F_nuevo = k
        If k = L_nuevo Then buscar = False
        k = k - 1
    Wend
    indice_origen(kkk) = L_nuevo
    indice_fin(kkk) = F_nuevo

'    If L <> L_nuevo Then
'        vx = vx + 1
'        Cells(vx, 1).Value = "Pasajeor "
'        Cells(vx, 2).Value = kkk
'        Cells(vx, 3).Value = "Nuev indice origen del psajero = "
'        Cells(vx, 4).Value = L_nuevo
'        vx = vx + 1
'    End If
'
'    If F <> F_nuevo Then
'        vx = vx + 1
'        Cells(vx, 1).Value = "Pasajeor "
'        Cells(vx, 2).Value = kkk
'        Cells(vx, 3).Value = "Nuev indice fin del psajero = "
'        Cells(vx, 4).Value = F_nuevo
'        vx = vx + 1
'    End If

Next kk
End Sub

Sub agrege_pasajero_en_ruta(n_p, p, i, j, mejor_costo, indice_o_i, indice_o_j, Long_ruta)
'n_p es el numero del pasajero en la ruta actual (con p agragado)
'p es el pasajero
If indice_o_i = indice_o_j Then
    If i <> r(indice_o_i) Then
    'Inserta i entre las posiciones indice_o_i y indice_f_i
        k = N + 1
        While k >= indice_o_i + 1
            r(k + 1) = r(k)
            k = k - 1
        Wend
        r(indice_o_i + 1) = i
        indice_origen(p) = indice_o_i + 1
        N = N + 1

        For kk = 1 To n_p - 1
            kkk = r_pasajeros(kk)
            If indice_origen(kkk) >= indice_origen(p) Then
                indice_origen(kkk) = indice_origen(kkk) + 1
            End If
            If indice_fin(kkk) >= indice_origen(p) Then
```

```
            indice_fin(kkk) = indice_fin(kkk) + 1
          End If
        Next kk

        indice_o_j_t = indice_o_j + 1

        If indice_o_i + 1 = N Then
          r(N + 1) = nodo_destino
          r(N + 2) = nodo_destino
'         indice_o_j_t = indice_o_j_t + 1
          N = N + 1
        End If

        If indice_o_i = 0 Then
          k = N + 1
          While k >= 0
            r(k + 1) = r(k)
            k = k - 1
          Wend
          r(0) = nodo_origen
          N = N + 1
          indice_o_j_t = indice_o_j_t + 1
          indice_o_i = indice_o_i + 1
          indice_origen(p) = indice_o_i + 1
          For kk = 1 To n_p - 1
            kkk = r_pasajeros(kk)
            If indice_origen(kkk) >= indice_origen(p) Then
              indice_origen(kkk) = indice_origen(kkk) + 1
            End If
            If indice_fin(kkk) >= indice_origen(p) Then
              indice_fin(kkk) = indice_fin(kkk) + 1
            End If
          Next kk
        End If

        Call escribir_ruta(vx)
      Else
        indice_origen(p) = indice_o_i
        indice_o_j_t = indice_o_j
      End If

'     If j <> r(indice_o_j + 1) Then
      If j <> r(indice_o_j_t + 1) Then
      'Inserta j entre las posiciones indice_o_j_t y indice_f_j_t
        k = N + 1
        While k >= indice_o_j_t + 1
          r(k + 1) = r(k)
          k = k - 1
        Wend
        r(indice_o_j_t + 1) = j
        indice_fin(p) = indice_o_j_t + 1

        N = N + 1

        For kk = 1 To n_p - 1
          kkk = r_pasajeros(kk)
          If indice_fin(kkk) >= indice_fin(p) Then
            indice_fin(kkk) = indice_fin(kkk) + 1
          End If
          If indice_origen(kkk) >= indice_fin(p) Then
            indice_origen(kkk) = indice_origen(kkk) + 1
          End If
        Next kk

        If indice_o_j_t + 1 = N Then
          r(N + 1) = nodo_destino
          r(N + 2) = nodo_destino
```

```
            N = N + 1
        End If


        If indice_o_j_t = 0 Then
            k = N + 1
            While k >= 0
                r(k + 1) = r(k)
                k = k - 1
            Wend
            r(0) = nodo_origen
            indice_o_j_t = indice_o_j_t + 1
            indice_fin(p) = indice_o_j_t + 1
            N = N + 1
            For kk = 1 To n_p - 1
                kkk = r_pasajeros(kk)
                If indice_fin(kkk) >= indice_fin(p) Then
                    indice_fin(kkk) = indice_fin(kkk) + 1
                End If
                If indice_origen(kkk) >= indice_fin(p) Then
                    indice_origen(kkk) = indice_origen(kkk) + 1
                End If
            Next kk
        End If

'       vx = vx + 1
'       Cells(vx, 1).Value = "indice_o_i="
'       Cells(vx, 2).Value = indice_o_i
'       vx = vx + 1
'       Cells(vx, 1).Value = "indice_o_j_t="
'       Cells(vx, 2).Value = indice_o_j_t
'       vx = vx + 1
'
'       Cells(vx, 1).Value = "indice origen pasajero 1="
'       Cells(vx, 2).Value = indice_origen(p)
'       vx = vx + 1
'       Cells(vx, 1).Value = "indice fin pasajero 1="
'       Cells(vx, 2).Value = indice_fin(p)
'       vx = vx + 1

        Call escribir_ruta(vx)
    Else
    'Cuando j = r(indice_o_j + 1)
        indice_fin(p) = indice_o_j_t + 1
    End If

Else

    If i = r(indice_o_i) Or i = r(indice_o_i + 1) Then
        If j = r(indice_o_j) Or j = r(indice_o_j + 1) Then
'           vx = vx + 1
'           Cells(vx, 1).Value = "indice_o_i="
'           Cells(vx, 2).Value = indice_o_i
'           vx = vx + 1
'           Cells(vx, 1).Value = "indice_o_j="
'           Cells(vx, 2).Value = indice_o_j
'           vx = vx + 1

            If i = r(indice_o_i + 1) Then
                indice_L_i = indice_o_i + 1
            Else
                indice_L_i = indice_o_i
            End If
            If j = r(indice_o_j) Then
                indice_L_j = indice_o_j
            Else
                indice_L_j = indice_o_j + 1
```

```
            End If

            indice_origen(p) = indice_L_i
            indice_fin(p) = indice_L_j

'           Cells(vx, 1).Value = "indice origen pasajero 2="
'           Cells(vx, 2).Value = indice_origen(p)
'           vx = vx + 1
'           Cells(vx, 1).Value = "indice fin pasajero 2="
'           Cells(vx, 2).Value = indice_fin(p)
'           vx = vx + 1


            Call escribir_ruta(vx)
          Else
          'Inserta j entre las posiciones indice_o_j y indice_f_j
            k = N + 1
            While k >= indice_o_j + 1
               r(k + 1) = r(k)
               k = k - 1
            Wend
            r(indice_o_j + 1) = j
            indice_fin(p) = indice_o_j + 1
            N = N + 1

            For kk = 1 To n_p - 1
               kkk = r_pasajeros(kk)
               If indice_fin(kkk) >= indice_fin(p) Then
                  indice_fin(kkk) = indice_fin(kkk) + 1
               End If
               If indice_origen(kkk) >= indice_fin(p) Then
                  indice_origen(kkk) = indice_origen(kkk) + 1
               End If
            Next kk

            If indice_o_j + 1 = N Then
               r(N + 1) = nodo_destino
               r(N + 2) = nodo_destino
               N = N + 1
            End If

            If indice_o_j = 0 Then
               k = N + 1
               While k >= 0
                  r(k + 1) = r(k)
                  k = k - 1
               Wend
               r(0) = nodo_origen
               indice_o_j = indice_o_j + 1
               indice_fin(p) = indice_o_j + 1
               N = N + 1
               For kk = 1 To n_p - 1
                  kkk = r_pasajeros(kk)
                  If indice_fin(kkk) >= indice_fin(p) Then
                     indice_fin(kkk) = indice_fin(kkk) + 1
                  End If
                  If indice_origen(kkk) >= indice_fin(p) Then
                     indice_origen(kkk) = indice_origen(kkk) + 1
                  End If
               Next kk
            End If

            If i = r(indice_o_i + 1) Then
               indice_L_i = indice_o_i + 1
            Else
               indice_L_i = indice_o_i
            End If
```

```
            indice_origen(p) = indice_L_i

'           vx = vx + 1
'           Cells(vx, 1).Value = "indice_o_i="
'           Cells(vx, 2).Value = indice_o_i
'           vx = vx + 1
'           Cells(vx, 1).Value = "indice_o_j="
'           Cells(vx, 2).Value = indice_o_j
'           vx = vx + 1
'
'           Cells(vx, 1).Value = "indice origen pasajero 3="
'           Cells(vx, 2).Value = indice_origen(p)
'           vx = vx + 1
'           Cells(vx, 1).Value = "indice fin pasajero 3="
'           Cells(vx, 2).Value = indice_fin(p)
'           vx = vx + 1

            Call escribir_ruta(vx)
        End If
    Else
    'Inserta i entre las posiciones indice_o_i y indice_f_i
        k = N + 1
        While k >= indice_o_i + 1
            r(k + 1) = r(k)
            k = k - 1
        Wend
        r(indice_o_i + 1) = i
        indice_origen(p) = indice_o_i + 1
        N = N + 1

        For kk = 1 To n_p - 1
            kkk = r_pasajeros(kk)
            If indice_origen(kkk) >= indice_origen(p) Then
                indice_origen(kkk) = indice_origen(kkk) + 1
            End If
            If indice_fin(kkk) >= indice_origen(p) Then
                indice_fin(kkk) = indice_fin(kkk) + 1
            End If
        Next kk

        indice_o_j_t = indice_o_j + 1

        If indice_o_i + 1 = N Then
            r(N + 1) = nodo_destino
            r(N + 2) = nodo_destino
            N = N + 1
'           indice_o_j_t = indice_o_j_t + 1
        End If

        If indice_o_i = 0 Then
            k = N + 1
            While k >= 0
                r(k + 1) = r(k)
                k = k - 1
            Wend
            r(0) = nodo_origen
            N = N + 1
            indice_o_j_t = indice_o_j_t + 1
            indice_o_i = indice_o_i + 1
            indice_origen(p) = indice_o_i + 1

            For kk = 1 To n_p - 1
                kkk = r_pasajeros(kk)
                If indice_origen(kkk) >= indice_origen(p) Then
                    indice_origen(kkk) = indice_origen(kkk) + 1
                End If
```

```
            If indice_fin(kkk) >= indice_origen(p) Then
               indice_fin(kkk) = indice_fin(kkk) + 1
            End If
         Next kk
      End If



      Call escribir_ruta(vx)
      If j = r(indice_o_j_t) Or j = r(indice_o_j_t + 1) Then

'        vx = vx + 1
'        Cells(vx, 1).Value = "indice_o_i="
'        Cells(vx, 2).Value = indice_o_i
'        vx = vx + 1
'        Cells(vx, 1).Value = "indice_o_j_t="
'        Cells(vx, 2).Value = indice_o_j_t
'        vx = vx + 1

         If j = r(indice_o_j_t) Then
            indice_L_j = indice_o_j_t
         Else
            indice_L_j = indice_o_j_t + 1
         End If

         indice_origen(p) = indice_o_i + 1
         indice_fin(p) = indice_L_j

'        Cells(vx, 1).Value = "indice origen pasajero 4="
'        Cells(vx, 2).Value = indice_origen(p)
'        vx = vx + 1
'        Cells(vx, 1).Value = "indice fin pasajero 4="
'        Cells(vx, 2).Value = indice_fin(p)
'        vx = vx + 1

         Call escribir_ruta(vx)
      Else
      'Inserta j entre las posiciones indice_o_j_t y indice_f_j_t
         k = N + 1
         While k >= indice_o_j_t + 1
            r(k + 1) = r(k)
            k = k - 1
         Wend
         r(indice_o_j_t + 1) = j
         indice_fin(p) = indice_o_j_t + 1
         N = N + 1

         For kk = 1 To n_p - 1
            kkk = r_pasajeros(kk)
            If indice_fin(kkk) >= indice_fin(p) Then
               indice_fin(kkk) = indice_fin(kkk) + 1
            End If
            If indice_origen(kkk) >= indice_fin(p) Then
               indice_origen(kkk) = indice_origen(kkk) + 1
            End If
         Next kk

         If indice_o_j_t + 1 = N Then

            r(N + 1) = nodo_destino
            r(N + 2) = nodo_destino
            N = N + 1

         End If

         If indice_o_j_t = 0 Then
            k = N + 1
```

```
            While k >= 0
               r(k + 1) = r(k)
               k = k - 1
            Wend
            r(0) = nodo_origen
            indice_o_j_t = indice_o_j_t + 1
            indice_fin(p) = indice_o_j_t + 1
            N = N + 1
            For kk = 1 To n_p - 1
               kkk = r_pasajeros(kk)
               If indice_fin(kkk) >= indice_fin(p) Then
                  indice_fin(kkk) = indice_fin(kkk) + 1
               End If
               If indice_origen(kkk) >= indice_fin(p) Then
                  indice_origen(kkk) = indice_origen(kkk) + 1
               End If
            Next kk
         End If

'         vx = vx + 1
'         Cells(vx, 1).Value = "indice_o_i="
'         Cells(vx, 2).Value = indice_o_i
'         vx = vx + 1
'         Cells(vx, 1).Value = "indice_o_j_t="
'         Cells(vx, 2).Value = indice_o_j_t
'         vx = vx + 1

         indice_origen(p) = indice_o_i + 1

'         Cells(vx, 1).Value = "indice origen pasajero 5="
'         Cells(vx, 2).Value = indice_origen(p)
'         vx = vx + 1
'         Cells(vx, 1).Value = "indice fin pasajero 5="
'         Cells(vx, 2).Value = indice_fin(p)
'         vx = vx + 1

         Call escribir_ruta(vx)
      End If
   End If
End If

Call escriba_pasajeros(n_p, vx)
Call escribir_ruta(vx)

If nodo_origen = nodo_destino Then
   If r(N) = r(N - 1) Then
      N = N - 1
'      vx = vx + 1
'      Cells(vx, 1).Value = "Ruta Corregida"
'      vx = vx + 1
      Call escribir_ruta(vx)
   End If
End If

Call depure_ruta(n_p, vx, Long_ruta, depure)
If depure Then
'   vx = vx + 1
'   Cells(vx, 1).Value = "DEPURE"
'   vx = vx + 1
   Call escriba_pasajeros(n_p, vx)
   Call escribir_ruta(vx)
End If

End Sub

Sub peluquee_ruta(n_p, vx, Long_ruta, depure)
For l = 1 To N
```

```
   numero_salen(l) = 0
   numero_entran(l) = 0
Next l

For kk = 1 To n_p
   kkk = r_pasajeros(kk)
   i = origen(kkk)
   j = destino(kkk)
   'nodos de origen y destino del pasajero kkk
   l = indice_origen(kkk)
   'L es el indice de origen del pasajero kkk en la ruta r
   F = indice_fin(kkk)
   'F es el indice de fin del pasajero kkk en la ruta r
   numero_salen(l) = numero_salen(l) + 1
   'numero_salen(L) es el número de pasajeros que salen del nodo de índice L , 1<=L<=N
   Lista_salen(l, numero_salen(l)) = kkk
   'Lista_salen(L, numero_salen(L)) es la lista de pasajeros que salen del nodo de índice L , 1<=L<=N
   numero_entran(F) = numero_entran(F) + 1
   'numero_entran(F) es el número de pasajeros que entran al nodo de índice F , 1<=F<=N
   Lista_entran(F, numero_entran(F)) = kkk
   'Lista_entran(F, numero_entran(F)) es la lista de pasajeros que entran al nodo de índice F , 1<=F<=N
Next kk

'vx = vx + 1
'Cells(vx, 1).Value = "Lista de pasajeros que entran a cada nodo de la ruta actual"
'vx = vx + 1

'For L = 1 To N
'    For np = 1 To numero_entran(L)
'        Cells(vx, np).Value = Lista_entran(L, np)
'    Next np
'    vx = vx + 1
'Next L

'vx = vx + 1
'Cells(vx, 1).Value = "Lista de pasajeros que salen de cada nodo de la ruta actual"
'vx = vx + 1

'For L = 1 To N
'    For np = 1 To numero_salen(L)
'        Cells(vx, np).Value = Lista_salen(L, np)
'    Next np
'    vx = vx + 1
'Next L

For l = 1 To N
   If numero_entran(l) = 0 Then
      If numero_salen(l) = 1 Then

         p = Lista_salen(l, 1)
         F = indice_fin(p)
         i = origen(p)
         j = destino(p)
         buscar_inicio = True
         k = 1
         While k < l And buscar_inicio
            If carga(k) < capacidad Then
               If r(k) = i Then

                  m = k + 1
                  While m < l And buscar_inicio
                     If carga(m) < capacidad Then
                        If r(m) = j Then
                           indice_origen(p) = k
                           indice_fin(p) = m
                           buscar_inicio = False
'                           vx = vx + 1
```

```
'                        Cells(vx, 1).Value = "CAMBIA PASAJERO ANTES"
'                        Cells(vx, 2).Value = p
'                        vx = vx + 1
'                        Cells(vx, 1).Value = "Nuevo Indice origen del pasajero ="
'                        Cells(vx, 2).Value = k
'                        vx = vx + 1
'                        Cells(vx, 1).Value = "Nuevo Indice fin del pasajero ="
'                        Cells(vx, 2).Value = m
'                        vx = vx + 1
                         Call depure_ruta(n_p, vx, Long_ruta, depure)
                      End If
                      m = m + 1
                   Else
                      m = l
                   End If
                Wend
             End If
             k = k + 1
          Else
             k = l
          End If
       Wend
       If buscar_inicio Then
          k = F + 1
          While k <= N And buscar_inicio
             If carga(k) < capacidad Then
                If r(k) = i Then

                   m = k + 1
                   While m <= N And buscar_inicio
                      If carga(m) < capacidad Then
                         If r(m) = j Then
                            indice_origen(p) = k
                            indice_fin(p) = m
                            buscar_inicio = False
'                           vx = vx + 1
'                           Cells(vx, 1).Value = "CAMBIA PASAJERO DESPUES"
'                           Cells(vx, 2).Value = p
'                           vx = vx + 1
'                           Cells(vx, 1).Value = "Nuevo Indice origen del pasajero ="
'                           Cells(vx, 2).Value = k
'                           vx = vx + 1
'                           Cells(vx, 1).Value = "Nuevo Indice fin del pasajero ="
'                           Cells(vx, 2).Value = m
'                           vx = vx + 1
                            Call depure_ruta(n_p, vx, Long_ruta, depure)
                         End If
                         m = m + 1
                      Else
                         m = N + 1
                      End If
                   Wend
                End If
                k = k + 1
             Else
                k = N + 1
             End If
          Wend
       End If

    End If
  End If
Next l

For F = 1 To N
  If numero_entran(F) = 1 Then
    If numero_salen(F) = 0 Then
```

```
            p = Lista_entran(F, 1)
            l = indice_origen(p)
            i = origen(p)
            j = destino(p)
            buscar_inicio = True
            k = 1
            While k < l And buscar_inicio
               If carga(k) < capacidad Then
                  If r(k) = i Then

                     m = k + 1
                     While m < l And buscar_inicio
                        If carga(m) < capacidad Then
                           If r(m) = j Then
                              indice_origen(p) = k
                              indice_fin(p) = m
                              buscar_inicio = False
'                              vx = vx + 1
'                              Cells(vx, 1).Value = "CAMBIA PASAJERO ANTES fff"
'                              Cells(vx, 2).Value = p
'                              vx = vx + 1
'                              Cells(vx, 1).Value = "Nuevo Indice origen del pasajero ="
'                              Cells(vx, 2).Value = k
'                              vx = vx + 1
'                              Cells(vx, 1).Value = "Nuevo Indice fin del pasajero ="
'                              Cells(vx, 2).Value = m
'                              vx = vx + 1
                              Call depure_ruta(n_p, vx, Long_ruta, depure)
                           End If
                           m = m + 1
                        Else
                           m = l
                        End If
                     Wend
                  End If
                  k = k + 1
               Else
                  k = l
               End If
            Wend
            If buscar_inicio Then
               k = F + 1
               While k <= N And buscar_inicio
                  If carga(k) < capacidad Then
                     If r(k) = i Then

                        m = k + 1
                        While m <= N And buscar_inicio
                           If carga(m) < capacidad Then
                              If r(m) = j Then
                                 indice_origen(p) = k
                                 indice_fin(p) = m
                                 buscar_inicio = False
'                                 vx = vx + 1
'                                 Cells(vx, 1).Value = "CAMBIA PASAJERO DESPUES fff"
'                                 Cells(vx, 2).Value = p
'                                 vx = vx + 1
'                                 Cells(vx, 1).Value = "Nuevo Indice origen del pasajero ="
'                                 Cells(vx, 2).Value = k
'                                 vx = vx + 1
'                                 Cells(vx, 1).Value = "Nuevo Indice fin del pasajero ="
'                                 Cells(vx, 2).Value = m
'                                 vx = vx + 1
                                 Call depure_ruta(n_p, vx, Long_ruta, depure)
                              End If
                              m = m + 1
```

```
                    Else
                        m = N + 1
                    End If
                Wend
            End If
            k = k + 1
        Else
            k = N + 1
        End If
    Wend
End If

End If
End If
Next F


End Sub

Sub depure_ruta(n_p, vx, Long_ruta, depure)
For nn = 1 To N
    nodo_visitado(nn) = False
Next nn

For nn = 1 To n_p
    p = r_pasajeros(nn)
    nodo_visitado(indice_origen(p)) = True
    nodo_visitado(indice_fin(p)) = True
Next nn

'vx = vx + 1
'Cells(vx, 1).Value = "Vector nodo visitado"
'vx = vx + 1
'For nn = 1 To N
'    Cells(vx, 1).Value = nn
'    Cells(vx, 2).Value = nodo_visitado(nn)
'    vx = vx + 1
'Next nn

nn = 2
depure = False
While nn <= N - 1
    If Not (nodo_visitado(nn)) Then
'       vx = vx + 1
'       Cells(vx, 1).Value = "Nodo No VISITADO ="
'       Cells(vx, 2).Value = nn
'       vx = vx + 1
        depure = True
        'DEbe eliminar de la ruta actual el nodo de indice i
        delta = d(r(nn - 1), r(nn)) + d(r(nn), r(nn + 1)) - d(r(nn - 1), r(nn + 1))
        Long_ruta = Long_ruta - delta
        For kk = nn To N
            r(kk) = r(kk + 1)
            nodo_visitado(kk) = nodo_visitado(kk + 1)

        Next kk
        For iii = 1 To n_p
            kkk = r_pasajeros(iii)
            If indice_origen(kkk) >= nn Then
                indice_origen(kkk) = indice_origen(kkk) - 1
            End If
            If indice_fin(kkk) >= nn Then
                indice_fin(kkk) = indice_fin(kkk) - 1
            End If
        Next iii
        N = N - 1
        Call escriba_detalle_pasajeros(n_p, vx)
```

```
   Else
      nn = nn + 1
   End If
Wend

End Sub

Sub inserte_pasajero_en_ruta(n_p, p)
Call calcule_parametros_insercion(p, i, j, mejor_costo, indice_o_i, indice_o_j)
Call agrege_pasajero_en_ruta(n_p, p, i, j, mejor_costo, indice_o_i, indice_o_j, Long_ruta)
End Sub

Sub escriba_pasajeros(n_p, vx)
Call mejore_origen_fin_pasajeros(n_p, vx)
'vx = vx + 1
'Cells(vx, 1).Value = "Pasajeros, indices de origen y destinos actuales"
'vx = vx + 1
'Cells(vx, 1).Value = "i"
'Cells(vx, 2).Value = "indice_origen(i)"
'Cells(vx, 3).Value = "indice_fin(i)"
'vx = vx + 1
For kk = 0 To N
   carga(kk) = 0
Next kk
For i = 1 To n_p
   p = r_pasajeros(i)
   For kk = indice_origen(p) To indice_fin(p) - 1
      carga(kk) = carga(kk) + 1
   Next kk
'    Cells(vx, 1).Value = p
'    Cells(vx, 2).Value = indice_origen(p)
'    Cells(vx, 3).Value = indice_fin(p)
'    Cells(vx, 5).Value = r(indice_origen(p))
'    Cells(vx, 6).Value = r(indice_fin(p))
'    Cells(vx, 7).Value = origen(p)
'    Cells(vx, 8).Value = destino(p)
'    vx = vx + 1
Next i
End Sub

Sub escriba_detalle_pasajeros(n_p, vx)

'vx = vx + 1
'Cells(vx, 1).Value = "Pasajeros, indices de origen y destinos actuales"
'vx = vx + 1
'Cells(vx, 1).Value = "i"
'Cells(vx, 2).Value = "indice_origen(i)"
'Cells(vx, 3).Value = "indice_fin(i)"
'vx = vx + 1
For kk = 0 To N
   carga(kk) = 0
Next kk
For i = 1 To n_p
   p = r_pasajeros(i)
   For kk = indice_origen(p) To indice_fin(p) - 1
      carga(kk) = carga(kk) + 1
   Next kk
'    Cells(vx, 1).Value = p
'    Cells(vx, 2).Value = indice_origen(p)
'    Cells(vx, 3).Value = indice_fin(p)
'    Cells(vx, 5).Value = r(indice_origen(p))
'    Cells(vx, 6).Value = r(indice_fin(p))
'    Cells(vx, 7).Value = origen(p)
'    Cells(vx, 8).Value = destino(p)
'    vx = vx + 1
Next i
End Sub
```

```
Sub escriba_mejores_hormigas(total_h, mejores_h, ite)
For ii = 1 To total_h
   For jj = ii + 1 To total_h
      If v_fobjs(ii) > v_fobjs(jj) Then
         Nt = mejores_N(ii)
         v_objt = v_fobjs(ii)
         For l = 1 To Nt
            r_pasajeros(l) = mejores_rutas(ii, l)
         Next l
         For l = 1 To mejores_N(jj)
            mejores_rutas(ii, l) = mejores_rutas(jj, l)
         Next l
         For l = 1 To Nt
            mejores_rutas(jj, l) = r_pasajeros(l)
         Next l
         mejores_N(ii) = mejores_N(jj)
         v_fobjs(ii) = v_fobjs(jj)
         mejores_N(jj) = Nt
         v_fobjs(jj) = v_objt
      End If
   Next jj
Next ii

vx = vx + 1
Cells(vx, 1).Value = "Rutas de las mejores hormigas INICIALES"
vx = vx + 1
For h = 1 To mejores_h
   N = mejores_N(h)
   Cells(vx, 1).Value = "Hormiga"
   Cells(vx, 2).Value = h
   Cells(vx, 4).Value = "Longitud N ="
   Cells(vx, 5).Value = N
   Cells(vx, 7).Value = "Longitud Ruta ="
   Cells(vx, 8).Value = v_fobjs(h)
   If h = 1 Then
      Worksheets("grande").Cells(17 + ite, 33) = v_fobjs(h)
   End If
   vx = vx + 1
   For i = 1 To N
      Cells(vx, i).Value = mejores_rutas(h, i)
   Next i
   vx = vx + 1
Next h
End Sub

Sub modifique_feromonas_totales(vx, mejores_h)
For i = 1 To pasajeros
   For j = 1 To pasajeros
      tao(i, j) = ro * tao(i, j)
   Next j
Next i

For h = 1 To mejores_h
   delta = 1 - (h - 1) / mejores_h
   For i = 1 To pasajeros - 1
      nodo_ini = mejores_pasajeros(h, i)
      nodo_fin = mejores_pasajeros(h, i + 1)
      tao(nodo_ini, nodo_fin) = tao(nodo_ini, nodo_fin) + delta
   Next i
Next h

'vx = vx + 1
'Cells(vx, 1).Value = "MATRIZ TAO"
'vx = vx + 1
'
'For i = 1 To pasajeros
```

```
'    For j = 1 To pasajeros
'        Cells(vx, j).Value = tao(i, j)
'    Next j
'    vx = vx + 1
'Next i

End Sub

Sub genere_soluciones_aleatorias(vx)
For h = 1 To hormigas
    r(0) = nodo_origen
    r(1) = nodo_origen
    r(2) = nodo_destino
    r(3) = nodo_destino
    N = 2
    For i = 0 To N + 1
        carga(i) = 0
    Next i
    Call escribir_ruta(vx)
    For p = 1 To pasajeros
        secuencia_p(p) = p
    Next p
    p_selecto = genere_entero_entre(1, pasajeros)
    numero_pasajeros = 1
    r_pasajeros(1) = p_selecto
    For p = p_selecto + 1 To pasajeros
        secuencia_p(p - 1) = secuencia_p(p)
    Next p
    pasajeros_t = pasajeros - 1
    Call calcule_parametros_insercion(p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j)
    Call agrege_pasajero_en_ruta(numero_pasajeros, p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j, Longitud_ruta)
    Longitud_ruta = mejor_costo + d(nodo_origen, nodo_destino)
    While pasajeros_t >= 1
        ip_selecto = genere_entero_entre(1, pasajeros_t)
        p_selecto = secuencia_p(ip_selecto)
'           vx = vx + 1
'           Cells(vx, 1).Value = "Selecciona el pasajero #="
'           Cells(vx, 2).Value = p_selecto
'           vx = vx + 1
            numero_pasajeros = numero_pasajeros + 1
            r_pasajeros(numero_pasajeros) = p_selecto
            For p = ip_selecto + 1 To pasajeros_t
                secuencia_p(p - 1) = secuencia_p(p)
            Next p
            pasajeros_t = pasajeros_t - 1
            Call calcule_parametros_insercion(p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j)
            Call agrege_pasajero_en_ruta(numero_pasajeros, p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j, Longitud_ruta)
            Longitud_ruta = Longitud_ruta + mejor_costo
            Call peluquee_ruta(numero_pasajeros, vx, Longitud_ruta, depure)
    Wend
    Call peluquee_ruta(numero_pasajeros, vx, Longitud_ruta, depure)
    mejores_N(h) = N
    For i = 1 To N
        mejores_rutas(h, i) = r(i)
    Next i

    vx = vx + 1
    Cells(vx, 1).Value = "Longitud de ruta = "
    Cells(vx, 2).Value = Longitud_ruta
    v_fobjs(h) = Longitud_ruta
    vx = vx + 1

Next h
End Sub

Sub ACO(vx)
vx = vx + 1
```

```
Cells(vx, 1).Value = "ALGORITMO ACO"
vx = vx + 1
For ite = 1 To max_iteraciones
   For h = 1 To hormigas
      r(0) = nodo_origen
      r(1) = nodo_origen
      r(2) = nodo_destino
      r(3) = nodo_destino
      N = 2
      For i = 0 To N + 1
         carga(i) = 0
      Next i
      Call escribir_ruta(vx)
      For p = 1 To pasajeros
         secuencia_p(p) = p
      Next p
      p_selecto = genere_entero_entre(1, pasajeros)
      numero_pasajeros = 1
      r_pasajeros(1) = p_selecto
      For p = p_selecto + 1 To pasajeros
         secuencia_p(p - 1) = secuencia_p(p)
      Next p
      pasajeros_t = pasajeros - 1

      Call calcule_parametros_insercion(p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j)
      Call agrege_pasajero_en_ruta(numero_pasajeros, p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j, Longitud_ruta)
'      vx = vx + 1
'      Cells(vx, 1).Value = "probabilidades"
'      vx = vx + 1
'      Cells(vx, 1).Value = "pasajero #"
'      Cells(vx, 2).Value = "ETA"
'      Cells(vx, 3).Value = "probabilidad selección"
'      vx = vx + 1
'      vxt = vx
      Longitud_ruta = mejor_costo + d(nodo_origen, nodo_destino)
      While pasajeros_t >= 1


         suma = 0
         contador = 0
         For ip = 1 To pasajeros_t
            p = secuencia_p(ip)
            Call calcule_parametros_insercion(p, i, j, mejor_costo, indice_o_i, indice_o_j)
            If mejor_costo > 10000000 Then
               eta(p_selecto, p) = 0

            Else
               If mejor_costo < 0 Then
                  eta(p_selecto, p) = Exp(1.6 * mejor_costo)
               Else
                  eta(p_selecto, p) = Exp(-1.6 * mejor_costo)
               End If
            End If
            'Cells(vx, 1).Value = p
            'Cells(vx, 2).Value = eta(p_selecto, p)

            'vx = vx + 1
            probabilidad(p_selecto, p) = (tao(p_selecto, p) ^ alfa) * (eta(p_selecto, p) ^ beta)
            suma = suma + probabilidad(p_selecto, p)
         Next ip
         If suma = 0 Then
'            vx = vx + 1
'            Cells(vx, 1).Value = "AJUSTE DE ETAAS EQUIVALENTES"
'            vx = vx + 1
            For ip = 1 To pasajeros_t
               p = secuencia_p(ip)
               eta(p_selecto, p) = 0.00001
```

```
            probabilidad(p_selecto, p) = 0.001 '(tao(p_selecto, p) ^ alfa) * (eta(p_selecto, p) ^ beta)
            suma = suma + probabilidad(p_selecto, p)
          Next ip
        End If

'       vx = vx + 1
'       Cells(vx, 1).Value = "probabilidades"
'       vx = vx + 1
'       Cells(vx, 1).Value = "pasajero #"
'       Cells(vx, 2).Value = "probabilidad sin dividir"
'       Cells(vx, 3).Value = "probabilidad selección"
'       Cells(vx, 5).Value = "SUMA= "
'       Cells(vx, 6).Value = suma
'       vx = vx + 1
        suma_1 = 0
        u = Rnd()
        ip = 0
'       Cells(vx, 1).Value = "u="
'       Cells(vx, 2).Value = u
'       vx = vx + 1
        While u > suma_1
          ip = ip + 1
          p = secuencia_p(ip)
'         Cells(vx, 2).Value = probabilidad(p_selecto, p)
          'If suma = 0 Then
          '    suma = 0.00000000001
          'End If
          probabilidad(p_selecto, p) = probabilidad(p_selecto, p) / suma
          suma_1 = suma_1 + probabilidad(p_selecto, p)
'         Cells(vx, 1).Value = p
'
'         Cells(vx, 3).Value = probabilidad(p_selecto, p)
'         vx = vx + 1

        Wend


        ip_selecto = ip
        p_selecto = secuencia_p(ip_selecto)
'       vx = vx + 1
'       Cells(vx, 1).Value = "Selecciona el pasajero #="
'       Cells(vx, 2).Value = p_selecto
'       vx = vx + 1
        numero_pasajeros = numero_pasajeros + 1
        r_pasajeros(numero_pasajeros) = p_selecto
        For p = ip_selecto + 1 To pasajeros_t
          secuencia_p(p - 1) = secuencia_p(p)
        Next p
        pasajeros_t = pasajeros_t - 1
        Call calcule_parametros_insercion(p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j)
        Call agrege_pasajero_en_ruta(numero_pasajeros, p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j, Longitud_ruta)

        Longitud_ruta = Longitud_ruta + mejor_costo
        Call peluquee_ruta(numero_pasajeros, vx, Longitud_ruta, depure)
      Wend
      Call peluquee_ruta(numero_pasajeros, vx, Longitud_ruta, depure)
      Call escriba_detalle_pasajeros(numero_pasajeros, vx)
      Call escribir_ruta(vx)
      mejores_N(h) = N
      For i = 1 To N
        mejores_rutas(h, i) = r(i)
      Next i
      For i = 1 To pasajeros
        mejores_pasajeros(h, i) = r_pasajeros(i)
      Next i
      vx = vx + 1
      Cells(vx, 1).Value = "Longitud de ruta = "
```

```
        Cells(vx, 2).Value = Longitud_ruta
        v_fobjs(h) = Longitud_ruta
        vx = vx + 1
'       Call escribir_ruta_v2(vx)
    Next h
    Call escriba_mejores_hormigas(hormigas, mejores_hormigas, ite)
    Call modifique_feromonas_totales(vx, mejores_hormigas)
Next ite
End Sub

Sub leer_secuencia_pasajeros(vx)
vx = vx + 1
Cells(vx, 1).Value = "Secuenca Pasajeros"
vx = vx + 1
For np = 1 To pasajeros
    secuencia_p(np) = Cells(vx, 1).Value
    vx = vx + 1
Next np
numero_pasajeros = 0
r(0) = nodo_origen
r(1) = nodo_origen
r(2) = nodo_destino
r(3) = nodo_destino
N = 2
For i = 0 To N + 1
    carga(i) = 0
Next i

Longitud_ruta = d(nodo_origen, nodo_destino)
For np = 1 To pasajeros
    p_selecto = secuencia_p(np)
        vx = vx + 1
        Cells(vx, 1).Value = "Selecciona el pasajero #="
        Cells(vx, 2).Value = p_selecto
        vx = vx + 1
    numero_pasajeros = numero_pasajeros + 1
    r_pasajeros(numero_pasajeros) = p_selecto
    Call calcule_parametros_insercion(p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j)
    Call agrege_pasajero_en_ruta(numero_pasajeros, p_selecto, i, j, mejor_costo, indice_o_i, indice_o_j, Longitud_ruta)
    Longitud_ruta = Longitud_ruta + mejor_costo
Next np
vx = vx + 1
Cells(vx, 1).Value = "Longitud de ruta AAAA = "
Cells(vx, 2).Value = Longitud_ruta
vx = vx + 1
End Sub

Sub principal()

ini_experimento = Time
n_experimentos = 16
For l = 1 To n_experimentos
    inicio = Time
    Range("a72:z20000").ClearContents
    vx = 0
    Call inicialice(vx, l)
    'Call leer_secuencia_pasajeros(vx)
    'Call escriba_detalle_pasajeros(pasajeros, vx)
    'Call escribir_ruta(vx)

    'Call genere_soluciones_aleatorias(vx)

    'Call escriba_mejores_hormigas(hormigas, mejores_hormigas)

    'r(0) = nodo_origen
    'r(1) = nodo_origen
    'r(2) = nodo_destino
```

```
    'r(3) = nodo_destino
    'N = 2
    'For i = 0 To N + 1
    '    carga(i) = 0
    'Next i


    'Call escribir_ruta(vx)
    'For p = 1 To pasajeros
    '    Call inserte_pasajero_en_ruta(p)'
    '
    'Next p

    Call ACO(vx)

    Set RangeACO1 = Worksheets("grande").Range("AG18:AG15000")
    respuesta = Application.WorksheetFunction.Min(RangeACO1)
    Worksheets("grande").Cells(17 + l, 30) = respuesta
    final = Time
    tiempo_total = final - inicio
    Worksheets("grande").Cells(17 + l, 31) = tiempo_total
    'Worksheets("grande").Range("AG18:AG15000").ClearContents

Next l
fin_experimento = Time
tiempo_experimento = fin_experimento - ini_experimento
Worksheets("grande").Cells(34, 31) = tiempo_experimento

End Sub
```

ANEXO 3 IMPLEMENTACION DEL CODIGO PARA ACO2

```
Public n_experimentos As Integer
Public n_iteraciones As Integer
Public n_solicitudes As Integer
Public n_hormigas As Integer
Public n_nodos As Integer
Public beta, alfa, q0, d(), dist_ruta(), tao(), tao0, q
Public comb As Integer
Public capacidad As Integer
Public infactible As Integer
Public con_tabla2 As Integer
Public nodo_i As Integer
Public nodo_f As Integer
Public origen_temp As Integer
Public destino_temp As Integer
Public con_s() As Integer
Public vec_s() As Integer
Public matriz_s() As Integer
Public matriz_c() As Integer
Public largo_vec_ruta() As Integer
Public tabla3() As Integer
Public vec_ruta() As Integer
Public capa_acum() As Integer
Public capa_acum_t() As Integer
Public dist_mejor()
Public mejor_vec_s() As Integer
Public mejor_vec_ruta() As Integer
Public mejor_capa_acum() As Integer
Public mejor As Integer
Public mejor_tabla3() As Integer
Public vx As Integer
Public tabla() As Single
Public tabla2()
Public tabla4() As Single
Public max_ride_time As Integer
Public pasa_comb As Boolean
Public tabla2_sim_1 As Integer
Public tabla2_sim_2 As Integer
Public tabla2_sim_3 As Integer
Public tabla_origen() As Integer
Public tabla_destino() As Integer
Public con_ultimo As Integer
Public tabla_ultimo() As Integer
Public mejores_hormigas As Integer
Public delta

Sub main()

Inicio_total = Time
n_experimentos = 16
n_solicitudes = Cells(5, 3)
n_nodos = Cells(6, 3)
nodo_i = Cells(7, 3)
nodo_f = Cells(8, 3)
capacidad = Cells(9, 3)
ReDim matriz_s(1 To n_solicitudes, 1 To 2)
ReDim matriz_c(1 To n_solicitudes, 1 To n_nodos)
```

```
ReDim d(1 To n_nodos, 1 To n_nodos)
Call leer_datos

For l = 1 To n_experimentos
   Range("aj12:ce59").ClearContents
   Range("b62:gr10000").ClearContents
   inicio = Time

   n_iteraciones = 60
   n_hormigas = Worksheets("grande").Cells(17 + l, 25)
   q0 = Worksheets("grande").Cells(17 + l, 26)
   beta = Worksheets("grande").Cells(17 + l, 27)
   alfa = Worksheets("grande").Cells(17 + l, 28)
   mejores_hormigas = Math.Round(n_hormigas * Worksheets("grande").Cells(17 + l, 29), 0) '
Worksheets("grande").Cells(17 + l, 29)

   tao0 = 0.1
   Randomize

   For landa = 1 To mejores_hormigas
      delta = delta + (1 - ((landa - 1) / mejores_hormigas))
   Next landa

   Call tao_inicial
   vx = 0
   For m = 1 To n_iteraciones
      ReDim con_s(1 To n_hormigas)
      ReDim vec_s(1 To n_hormigas, 1 To n_solicitudes)
      ReDim vec_ruta(1 To n_hormigas, 1 To 100)
      ReDim capa_acum(1 To n_hormigas, 1 To 100)
      ReDim largo_vec_ruta(1 To n_hormigas)
      ReDim dist_ruta(1 To n_hormigas)
      ReDim tabla3(1 To n_hormigas, 1 To n_solicitudes, 1 To 3)   'Mide el maximo Ride Time de un pasajero
      For n = 1 To n_solicitudes
         'Con n = 1, se hace la insercion inicial de solicitudes y nodos
         'para todas las hormigas
         If n = 1 Then
            For k = 1 To n_hormigas
               con_s(k) = con_s(k) + 1
               solicitud_0 = Int(n_solicitudes * Rnd() + 1)
               'Cells(25 + k, 11) = solicitud_0
               vec_s(k, con_s(k)) = solicitud_0
               Call insercion_inicial(k, solicitud_0)
               For o = 1 To largo_vec_ruta(k) - 1
                  dist_ruta(k) = dist_ruta(k) + d(vec_ruta(k, o), vec_ruta(k, o + 1))
               Next o
               '= Impresion tabla3 muestra las posiciones dentro de la ruta ========
               'For aa = 1 To con_s(k)
               '    For bb = 1 To 3
               '        Cells(75 + aa, 18 + bb + (4 * k)) = tabla3(k, aa, bb)
               '    Next bb
               'Next aa
               '=====================================================================
            Next k
         Else
            For k = 1 To n_hormigas
               q = Rnd()
               If q <= q0 Then
                  Call exploit(k)
               Else
                  Call exploration(k)
               End If
               '= Impresion tabla3 ===================================
               'For aa = 1 To con_s(k)
               '    For bb = 1 To 3
               '        Cells(75 + aa, 18 + bb + (4 * k)) = tabla3(k, aa, bb)
               '    Next bb
```

```
                'Next aa
                '=========================================================
                Range("l62:u105").ClearContents
            Next k
        End If
    Next n
    Call actualizacion_global
    'Call guardar_mejor(m, mejor)
  Next m

  Set myRange = Range("CH62:CH15000")
  respuesta = Application.WorksheetFunction.Min(myRange)

  Worksheets("grande").Cells(17 + l, 30) = respuesta
  final = Time
  tiempo_total = final - inicio
  Worksheets("grande").Cells(17 + l, 31) = tiempo_total

Next l

Final_total = Time
tiempo_total_exp = Final_total - Inicio_total
Worksheets("grande").Cells(34, 31) = tiempo_total_exp


End Sub

Sub leer_datos()

'Matriz de solicitudes
For i = 1 To n_solicitudes
  For j = 1 To 2
    matriz_s(i, j) = Cells(11 + i, 2 + j)
  Next j
Next i

'Matriz de carga
For i = 1 To n_solicitudes
  For j = 1 To n_nodos
    If Cells(11 + i, 20 + j) <> 1 Then
      matriz_c(i, j) = 0
    Else
      matriz_c(i, j) = 1
    End If
  Next j
Next i

'Matriz distancia
For i = 1 To n_nodos
  For j = 1 To n_nodos
    d(i, j) = Cells(11 + i, 6 + j) / 100000
  Next j
Next i

End Sub
Sub tao_inicial()

ReDim tao(1 To n_solicitudes, 1 To n_solicitudes)

'Tao Inicial
For i = 1 To n_solicitudes
  For j = 1 To n_solicitudes
    If i = j Then
      tao(i, j) = 0
    Else
      tao(i, j) = tao0 '0.1
    End If
```

```
    Next j
Next i

End Sub

Sub insercion_inicial(k, solicitud_0)

If nodo_i <> matriz_s(solicitud_0, 1) Then
    If nodo_f <> matriz_s(solicitud_0, 2) Then
        largo_vec_ruta(k) = 4
        vec_ruta(k, 1) = nodo_i
        vec_ruta(k, 2) = matriz_s(solicitud_0, 1)
        vec_ruta(k, 3) = matriz_s(solicitud_0, 2)
        vec_ruta(k, 4) = nodo_f
        capa_acum(k, 1) = 0 'matriz_c(solicitud_0, nodo_i)
        capa_acum(k, 2) = matriz_c(solicitud_0, matriz_s(solicitud_0, 1))
        capa_acum(k, 3) = matriz_c(solicitud_0, matriz_s(solicitud_0, 1)) - matriz_c(solicitud_0, matriz_s(solicitud_0, 2))
        capa_acum(k, 4) = 0 'matriz_c(solicitud_0, nodo_f)
        pos_origen = 2
        pos_destino = 3
    ElseIf nodo_f = matriz_s(solicitud_0, 2) Then
        largo_vec_ruta(k) = 3
        vec_ruta(k, 1) = nodo_i
        vec_ruta(k, 2) = matriz_s(solicitud_0, 1)
        vec_ruta(k, 3) = nodo_f
        capa_acum(k, 1) = 0 'matriz_c(solicitud_0, nodo_i)
        capa_acum(k, 2) = matriz_c(solicitud_0, matriz_s(solicitud_0, 1))
        capa_acum(k, 3) = 0 'matriz_c(solicitud_0, nodo_f)
        pos_origen = 2
        pos_destino = 3
    End If
ElseIf nodo_i = matriz_s(solicitud_0, 1) Then
    If nodo_f <> matriz_s(solicitud_0, 2) Then
        largo_vec_ruta(k) = 3
        vec_ruta(k, 1) = nodo_i
        vec_ruta(k, 2) = matriz_s(solicitud_0, 2)
        vec_ruta(k, 3) = nodo_f
        capa_acum(k, 1) = matriz_c(solicitud_0, nodo_i)
        capa_acum(k, 2) = matriz_c(solicitud_0, nodo_i) - matriz_c(solicitud_0, matriz_s(solicitud_0, 2))
        capa_acum(k, 3) = 0 'matriz_c(solicitud_0, nodo_f)
        pos_origen = 1
        pos_destino = 2
    ElseIf nodo_f = matriz_s(solicitud_0, 2) Then
        largo_vec_ruta(k) = 2
        vec_ruta(k, 1) = nodo_i
        vec_ruta(k, 2) = nodo_f
        capa_acum(k, 1) = matriz_c(solicitud_0, nodo_i)
        capa_acum(k, 2) = 0 'matriz_c(solicitud_0, nodo_f)
        pos_origen = 1
        pos_destino = 2
    End If
End If

tabla3(k, con_s(k), 1) = solicitud_0    'Solicitud
tabla3(k, con_s(k), 2) = pos_origen     'posicion en vector ruta de origen
tabla3(k, con_s(k), 3) = pos_destino    'posicion en vector ruta de destino

'For i = 1 To largo_vec_ruta(k)
'    Cells(25 + k, 11 + i) = capa_acum(k, i)
'Next i

End Sub

Sub exploit(k)

Call heuristica_insercion(k)
Call hallar_mayor_solicitud(k, con_tabla2)
```

```
Call actualizacion_local(k)

End Sub

Sub exploration(k)

Call heuristica_insercion(k)
For i = 1 To con_tabla2
   If i = 1 Then
      tabla2(i, 7) = tabla2(i, 5) / tabla2(1, 6)
   Else
      tabla2(i, 7) = tabla2(i - 1, 7) + tabla2(i, 5) / tabla2(1, 6)
   End If
Next i
aleatorio = Rnd()

'= Imprime tabla2(i, 7) o pk ========================================
'For i = 1 To con_tabla2
'    Cells(61 + i, 20) = tabla2(i, 7)
'Next i
'Cells(62, 21) = aleatorio
'=====================================================================
j = 0
bandera6 = True
While bandera6
   j = j + 1
   If j > con_tabla2 Then
      bandera6 = False
   ElseIf 0 < aleatorio And aleatorio <= tabla2(j, 7) Then
      Call nueva_solicitud(k, j)
      bandera6 = False
   ElseIf tabla2(j, 7) < aleatorio And aleatorio <= tabla2(j + 1, 7) Then
      Call nueva_solicitud(k, j)
      bandera6 = False
   End If
Wend
Call actualizacion_local(k)

End Sub

Sub actualizacion_local(k)

tao(vec_s(k, con_s(k) - 1), vec_s(k, con_s(k))) = (1 - alfa) * tao(vec_s(k, con_s(k) - 1), vec_s(k, con_s(k))) + (alfa * tao0)
'Cells(11 + vec_s(k, con_s(k) - 1), 35 + vec_s(k, con_s(k))) = tao(vec_s(k, con_s(k) - 1), vec_s(k, con_s(k)))

End Sub
Sub actualizacion_global()

ReDim dist_mejor(1 To mejores_hormigas)
ReDim mejor_largo_vec_ruta(1 To mejores_hormigas)
ReDim mejor_vec_s(1 To mejores_hormigas, 1 To n_solicitudes)
ReDim mejor_vec_ruta(1 To mejores_hormigas, 1 To 100)
ReDim mejor_capa_acum(1 To mejores_hormigas, 1 To 100)
ReDim mejor_tabla3(1 To mejores_hormigas, 1 To n_solicitudes, 1 To 3)

For i = 1 To n_solicitudes
   For j = 1 To n_solicitudes
      If i = j Then
         tao(i, j) = 0
      Else
         tao(i, j) = alfa * tao0 '0.1
      End If
   Next j
Next i

For b_ant = 1 To mejores_hormigas
   'j tiene el menor
```

```
    j = 1
    l = 2
    bandera7 = True
    While bandera7
        If dist_ruta(j) > dist_ruta(l) Then
            j = l
        End If
        l = l + 1
        If l > n_hormigas Then
            For i = 1 To n_solicitudes - 1
                tao(vec_s(j, i), vec_s(j, i + 1)) = tao(vec_s(j, i), vec_s(j, i + 1)) + delta '(1 / dist_ruta(j))
                'Cells(11 + vec_s(j, i), 35 + vec_s(j, i + 1)) = tao(vec_s(j, i), vec_s(j, i + 1))
            Next i
            dist_mejor(b_ant) = dist_ruta(j)
            mejor_largo_vec_ruta(b_ant) = largo_vec_ruta(j)
            For o = 1 To n_solicitudes
                mejor_vec_s(b_ant, o) = vec_s(j, o)
            Next o
            For o = 1 To largo_vec_ruta(j)
                mejor_vec_ruta(b_ant, o) = vec_ruta(j, o)
                mejor_capa_acum(b_ant, o) = capa_acum(j, o)
            Next o
            For y = 1 To n_solicitudes
                For z = 1 To 3
                    mejor_tabla3(b_ant, y, z) = tabla3(j, y, z)
                Next z
            Next y

            dist_ruta(j) = 1000
            bandera7 = False
        End If
    Wend
Next b_ant
' = Impresion por Iteración ==========================================================================
Cells(62 + vx, 85) = "OTRA"
For b_ant = 1 To mejores_hormigas
    Cells(62 + vx, 86) = dist_mejor(b_ant)
    'For i = 1 To n_solicitudes
    '    Cells(62 + vx, 87 + i) = mejor_vec_s(m, i)
    'Next i
    'vx = vx + 1
    For i = 1 To mejor_largo_vec_ruta(b_ant)
        Cells(62 + vx, 87 + i) = mejor_vec_ruta(b_ant, i)
        Cells(62 + vx + 1, 87 + i) = mejor_capa_acum(b_ant, i)
    Next i
    vx = vx + 3
    ' ============================================================================================================
    '= Impresion tabla3 muestra las posiciones dentro de la ruta ========
    'For aa = 1 To n_solicitudes
    '    For bb = 1 To 3
    '        Worksheets("Hoja3").Cells((1 + m) + aa + (n_solicitudes * m), bb) = mejor_tabla3(m, aa, bb)
    '    Next bb
    'Next aa
    '=====================================================================
Next b_ant

End Sub
Sub guardar_mejor(m, mejor)

dist_mejor(m) = dist_ruta(mejor)
For i = 1 To n_solicitudes
    mejor_vec_s(m, i) = vec_s(mejor, i)
Next i

For i = 1 To largo_vec_ruta(mejor)
    mejor_vec_ruta(m, i) = vec_ruta(mejor, i)
    mejor_capa_acum(m, i) = capa_acum(mejor, i)
```

```
Next i

For y = 1 To n_solicitudes
   For z = 1 To 3
      mejor_tabla3(m, y, z) = tabla3(mejor, y, z)
   Next z
Next y

' = Impresion por Iteración ==========================================================================
Cells(62 + vx, 86) = dist_mejor(m)
For i = 1 To n_solicitudes
   Cells(62 + vx, 87 + i) = mejor_vec_s(m, i)
Next i
vx = vx + 1
For i = 1 To largo_vec_ruta(mejor)
   Cells(62 + vx, 87 + i) = mejor_vec_ruta(m, i)
   Cells(62 + vx + 1, 87 + i) = mejor_capa_acum(m, i)
Next i
vx = vx + 3
' ============================================================================================================
'= Impresion tabla3 muestra las posiciones dentro de la ruta ========
For aa = 1 To n_solicitudes
   For bb = 1 To 3
      Worksheets("Hoja3").Cells((1 + m) + aa + (n_solicitudes * m), bb) = mejor_tabla3(m, aa, bb)
   Next bb
Next aa
'=====================================================================
End Sub

Sub heuristica_insercion(k)

'Identificar que la solicitud no este dentro de la memoria
'spa = Solicitud Por Atender
'sa = Solicitudes Atendidas
'tabla() da los resultados de las distintas combinaciones
'tabla2() guarda la mejor combinacion para cada solicitud por entrar y en la_
'columna 5 el producto tao*(eta)^beta
'bandera1, bandera2,bandera3, bandera4,'bandera5, bandera6,

'Definicion de eta

ReDim tabla2(1 To n_solicitudes, 1 To 7)
con_tabla2 = 0
For spa = 1 To n_solicitudes
   bandera1 = True
   bandera2 = True
   sa = 1
   While bandera1
      If spa = vec_s(k, sa) Then
         bandera1 = False
         bandera2 = False
      ElseIf sa > con_s(k) Then
         bandera1 = False
      End If
      sa = sa + 1
   Wend
   con_origen = 0
   con_destino = 0
   comb = 0
   con_ultimo = 0
   ReDim tabla(1 To 500, 1 To 3)
   ReDim tabla_origen(1 To 100)
   ReDim tabla_destino(1 To 100)
   ReDim tabla_ultimo(1 To 300, 1 To 2)
   While bandera2
      origen_temp = matriz_s(spa, 1)
      destino_temp = matriz_s(spa, 2)
```

```
For i = 1 To largo_vec_ruta(k) ' - 1
   If origen_temp = vec_ruta(k, i) Then
      con_origen = con_origen + 1
      tabla_origen(con_origen) = i
      For j = i + 1 To largo_vec_ruta(k)
         If origen_temp = vec_ruta(k, j) Then
            con_origen = con_origen + 1
            tabla_origen(con_origen) = j
            j = largo_vec_ruta(k) + 1
         Else                              'If origen_temp <> vec_ruta(k, j) Then
            If destino_temp = vec_ruta(k, j) Then
               a = j - 1
               Call verificacion3(k, i, a, spa)    'Verificacion capacidad origen y destino _temp existen
               If infactible = 0 Then              'en la ruta actual
                  comb = comb + 1
                  tabla(comb, 1) = i - 1
                  tabla(comb, 2) = j - 1
                  tabla(comb, 3) = 0
               Else
                  con_ultimo = con_ultimo + 1
               '    tabla_ultimo(con_ultimo, 1) = i
               '    tabla_ultimo(con_ultimo, 2) = j
                  j = largo_vec_ruta(k) + 1
               End If
               j = largo_vec_ruta(k) + 1
            End If
         End If
      Next j

   ElseIf destino_temp = vec_ruta(k, i) Then
      If i <> 1 Then
         con_destino = con_destino + 1
         tabla_destino(con_destino) = i
      End If
   End If
Next i

If comb <> 0 Then      'origen y destino _temp estuvieron en la ruta actual
   bandera2 = False    'no hay necesidad de buscar más, la capacidad no se superó
   bandera3 = False    'origen_emp está repetido en la ruta
   bandera4 = False    'destino_temp está repetido en la ruta
   bandera5 = False    'ni origen ni destino _temp están en ruta
   bandera6 = False    'caso especial, se activa con_ultimo
Else
   If con_origen <> 0 Then
      bandera3 = True
      bandera5 = False
   Else
      bandera3 = False
      bandera5 = True
   End If
   If con_destino <> 0 Then
      bandera4 = True
      bandera5 = False
   Else
      bandera4 = False
      If bandera3 = True Then
         bandera5 = False
      Else
         bandera5 = True
      End If
   End If
   bandera6 = False
End If                 'Posiblemente la capacidad se ha superado

While bandera3
   For i = 1 To con_origen
```

```
            If i > 1 Then
               For j = tabla_origen(i) To (tabla_origen(i + 1) - 1)
                  If destino_temp <> vec_ruta(k, j + 1) Then
                     a = j
                     Call verificacion3prima(k, i, a, spa)
                     If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = tabla_origen(i) - 1
                        tabla(comb, 2) = j
                           tabla(comb, 3) = d(vec_ruta(k, j), destino_temp) + d(destino_temp, vec_ruta(k, j + 1)) - d(vec_ruta(k, j),
vec_ruta(k, j + 1))
                     Else
                        j = tabla_origen(i + 1)
                     End If
                  Else
                     j = tabla_origen(i + 1)
                  End If
               Next j
            Else
               For j = tabla_origen(i) To largo_vec_ruta(k) - 1
                  If destino_temp <> vec_ruta(k, j + 1) Then
                     a = j
                     Call verificacion3prima(k, i, a, spa)
                     If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = tabla_origen(i) - 1
                        tabla(comb, 2) = j
                           tabla(comb, 3) = d(vec_ruta(k, j), destino_temp) + d(destino_temp, vec_ruta(k, j + 1)) - d(vec_ruta(k, j),
vec_ruta(k, j + 1))
                     Else
                     '    con_ultimo = con_ultimo + 1
                     '    tabla_ultimo(con_ultimo, 1) = tabla_origen(i)
                     '    tabla_ultimo(con_ultimo, 2) = j + 1
                        j = largo_vec_ruta(k)
                     End If
                  Else
                     con_ultimo = con_ultimo + 1
                  '    tabla_ultimo(con_ultimo, 1) = tabla_origen(i)
                  '    tabla_ultimo(con_ultimo, 2) = j + 1
                     j = largo_vec_ruta(k)
                  End If
               Next j
            End If
         Next i
         bandera3 = False
      Wend

      While bandera4
         i = con_destino
         While i >= 1
            If i > 1 Then
               j = tabla_destino(i)
               While j > tabla_destino(i - 1)
                  If origen_temp <> vec_ruta(k, j - 1) Then
                     Call verificacion5(k, i, j, spa)
                     If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = j - 1
                        tabla(comb, 2) = tabla_destino(i) - 1
                           tabla(comb, 3) = d(vec_ruta(k, j - 1), origen_temp) + d(origen_temp, vec_ruta(k, j)) - d(vec_ruta(k, j - 1),
vec_ruta(k, j))
                        j = j - 1
                     Else
                        j = tabla_destino(i - 1)
                     End If
                  Else
                     j = tabla_destino(i - 1)
```

```
                End If
            Wend

        Else
            j = tabla_destino(i)
            While j >= 2
                If origen_temp <> vec_ruta(k, j - 1) Then
                    Call verificacion5(k, i, j, spa)
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = j - 1
                        tabla(comb, 2) = tabla_destino(i) - 1
                            tabla(comb, 3) = d(vec_ruta(k, j - 1), origen_temp) + d(origen_temp, vec_ruta(k, j)) - d(vec_ruta(k, j - 1),
vec_ruta(k, j))
                        j = j - 1
                    Else
                        con_ultimo = con_ultimo + 1
                    '    tabla_ultimo(con_ultimo, 1) = j - 1
                    '    tabla_ultimo(con_ultimo, 2) = tabla_destino(i)
                        j = 1
                    End If
                Else
                    con_ultimo = con_ultimo + 1
                '    tabla_ultimo(con_ultimo, 1) = j - 1
                '    tabla_ultimo(con_ultimo, 2) = tabla_destino(i)
                    j = 1
                End If
            Wend
        End If
        i = i - 1
    Wend
    bandera4 = False
Wend

If bandera5 = False Then
    If con_ultimo = 0 Then
        If comb = 0 Then
            con_ultimo = con_ultimo + 1
            tabla_ultimo(con_ultimo, 1) = largo_vec_ruta(k)
            tabla_ultimo(con_ultimo, 2) = 1
        End If
    End If
End If

While bandera5
    For i = 0 To largo_vec_ruta(k)
        For j = i To largo_vec_ruta(k)
            If i = 0 Then
                If j = 0 Then
                    If destino_temp <> vec_ruta(k, j + 1) Then 'YAAAAAA
                        comb = comb + 1
                        tabla(comb, 1) = i
                        tabla(comb, 2) = i
                            tabla(comb, 3) = d(nodo_i, origen_temp) + d(origen_temp, destino_temp) + d(destino_temp, vec_ruta(k,
1))
                    End If
                Else
                    j = largo_vec_ruta(k) + 1
                End If

            ElseIf i > 0 And i < largo_vec_ruta(k) Then

                If origen_temp <> vec_ruta(k, i) And origen_temp <> vec_ruta(k, i + 1) Then
                    If j = i Then
                        If destino_temp <> vec_ruta(k, j + 1) Then 'YAAAAAA
                            temp_carga = capa_acum(k, i)
                            temp_carga = temp_carga + 1
```

```
                        If temp_carga <= capacidad Then
                            comb = comb + 1
                            tabla(comb, 1) = i
                            tabla(comb, 2) = i
                                tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, destino_temp) + d(destino_temp,
vec_ruta(k, i + 1)) - d(vec_ruta(k, i), vec_ruta(k, i + 1))
                        Else
                            con_ultimo = con_ultimo + 1
                            tabla_ultimo(con_ultimo, 1) = largo_vec_ruta(k)
                            tabla_ultimo(con_ultimo, 2) = 1
                        End If
                    End If
                ElseIf j > i And j < largo_vec_ruta(k) Then 'ESTEEEE
                    If destino_temp <> vec_ruta(k, j) And destino_temp <> vec_ruta(k, j + 1) Then  'YAAAAAA
                        a = j
                        Call verificacion4(k, i, a, spa)
                        If infactible = 0 Then
                            comb = comb + 1
                            tabla(comb, 1) = i
                            tabla(comb, 2) = j
                                tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, vec_ruta(k, i + 1)) - d(vec_ruta(k,
i), vec_ruta(k, i + 1)) + d( vec_ruta(k, j), destino_temp) + d(destino_temp, vec_ruta(k, j + 1)) - d( vec_ruta(k, j), vec_ruta(k, j +
1))
                        Else
                            con_ultimo = con_ultimo + 1
                            tabla_ultimo(con_ultimo, 1) = largo_vec_ruta(k)
                            tabla_ultimo(con_ultimo, 2) = 1
                        End If
                    End If
                End If
            End If

        ElseIf i = largo_vec_ruta(k) Then
            If origen_temp <> vec_ruta(k, i) Then  'YAAAAAAA
                comb = comb + 1  'YAAAAAAA
                tabla(comb, 1) = i
                tabla(comb, 2) = i
                tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, destino_temp) + d(destino_temp, nodo_f)
            End If
        End If
      Next j
    Next i
    bandera5 = False
  Wend

  If comb = 0 Then
    If con_ultimo <> 0 Then
      '= Impresion con_ultimo ================================================
      'For yy = 1 To con_ultimo
      '    Cells(61 + yy, 73) = yy
      '    For xx = 1 To 2
      '        Cells(61 + yy, 73 + xx) = tabla_ultimo(yy, xx)
      '    Next xx
      'Next yy
      '=========================================================================
      bandera6 = True
      While bandera6
        Call heuristica1(k, spa)
        bandera6 = False
      Wend
    End If
  End If
  bandera2 = False
Wend

'= Imprimir tabla ================================
'Cells(62, 2) = spa
```

```
    'For i = 1 To largo_vec_ruta(k)
    '    Cells(62, 2 + i) = vec_ruta(k, i)
    'Next i
    'For i = 1 To comb
    '    Cells(61 + i, 7) = i
    '    For j = 1 To 3
    '        Cells(61 + i, 7 + j) = tabla(i, j)
    '    Next j
    'Next i
    '=====================================================
    If comb <> 0 Then
        Call hallar_menor_ext(k, spa, comb)
        tao_temp = tao(vec_s(k, con_s(k)), spa)
        eta_temp = 1 / Exp(tabla2(con_tabla2, 4))
        tabla2(con_tabla2, 5) = tao_temp * (eta_temp) ^ beta
        If q > q0 Then
            tabla2(1, 6) = tabla2(1, 6) + tabla2(con_tabla2, 5)
        End If
        '= Impresion tabla2, 5 columna ============================
        'Cells(61 + con_tabla2, 16) = tabla2(con_tabla2, 5)
        'Cells(61 + con_tabla2, 17) = tao_temp
        'Cells(61 + con_tabla2, 18) = eta_temp
        'Cells(62, 19) = tabla2(1, 6)
        '==========================================================
    End If
    Range("b62:j200").ClearContents
Next spa

End Sub

Sub heuristica1(k, spa)

For x = 1 To con_ultimo
    For y = 1 To 2
        If y = 1 Then
            inicio = 0
            final = tabla_ultimo(x, 1)
        Else
            inicio = tabla_ultimo(x, 2)
            final = largo_vec_ruta(k)
        End If
        For i = inicio To final
            For j = i To final
                If i = 0 Then
                    If origen_temp <> vec_ruta(k, i + 1) Then
                        'i o d i111f
                        If j = 0 Then
                            If destino_temp <> vec_ruta(k, j + 1) Then  'YAAAAAA
                                comb = comb + 1
                                tabla(comb, 1) = i
                                tabla(comb, 2) = i
                                    tabla(comb, 3) = d(nodo_i, origen_temp) + d(origen_temp, destino_temp) + d(destino_temp, vec_ruta(k,
1))
                            End If
                        ElseIf j > 0 And j < final Then
                            'i o id111f o i o i11d11f
                            If destino_temp = vec_ruta(k, j) Then 'YAAAAAA
                                Call verificacion1(k, i, j, spa)
                                If infactible = 0 Then
                                    comb = comb + 1
                                    tabla(comb, 1) = i
                                    tabla(comb, 2) = j - 1
                                    tabla(comb, 3) = d(nodo_i, origen_temp) + d(origen_temp, vec_ruta(k, 1))
                                End If
                            'i o i d 111f
                            ElseIf destino_temp <> vec_ruta(k, j) And destino_temp <> vec_ruta(k, j + 1) Then  'YAAAAAA
                                Call verificacion2(k, i, j, spa)
```

```
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = i
                        tabla(comb, 2) = j
                            tabla(comb, 3) = d(nodo_i, origen_temp) + d(origen_temp, vec_ruta(k, 1)) + d(vec_ruta(k, j),
destino_temp) + d(destino_temp, vec_ruta(k, j + 1)) - d(vec_ruta(k, j), vec_ruta(k, j + 1))
                    End If
                End If

            ElseIf j = final Then
                'i o i111fd
                If destino_temp = vec_ruta(k, j) Then 'YAAAAAAA
                    Call verificacion1(k, i, j, spa)
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = i
                        tabla(comb, 2) = j - 1
                        tabla(comb, 3) = d(nodo_i, origen_temp) + d(origen_temp, vec_ruta(k, 1))
                    End If
                'i o i1111f d f
                ElseIf destino_temp <> vec_ruta(k, j) Then 'YAAAAAA
                    Call verificacion2(k, i, j, spa)
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = i
                        tabla(comb, 2) = j
                            tabla(comb, 3) = d(nodo_i, origen_temp) + d(origen_temp, vec_ruta(k, 1)) + d(vec_ruta(k, j),
destino_temp) + d(destino_temp, nodo_f)
                    End If
                End If
            End If
        ElseIf origen_temp = vec_ruta(k, i + 1) Then
            j = final
        End If

    ElseIf i > 0 And i < final Then

        If origen_temp = vec_ruta(k, i) Then
            If j >= i And j < final Then
                If destino_temp <> vec_ruta(k, j) And destino_temp <> vec_ruta(k, j + 1) Then 'ESTEEEEE
                    a = j
                    Call verificacion3(k, i, a, spa)
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = i - 1
                        tabla(comb, 2) = j
                            tabla(comb, 3) = d(vec_ruta(k, j), destino_temp) + d(destino_temp, vec_ruta(k, j + 1)) - d(vec_ruta(k, j),
vec_ruta(k, j + 1))
                    End If
                ElseIf destino_temp = vec_ruta(k, j) Then 'YAAAAAAA
                    a = j - 1
                    Call verificacion3(k, i, a, spa)
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = i - 1
                        tabla(comb, 2) = j - 1
                        tabla(comb, 3) = 0
                    End If
                End If
            ElseIf j = final Then
                If destino_temp <> vec_ruta(k, j) Then 'YAAAAAA
                    a = j
                    Call verificacion3(k, i, a, spa)
                    If infactible = 0 Then
                        comb = comb + 1
                        tabla(comb, 1) = i - 1
                        tabla(comb, 2) = j
```

```
                    tabla(comb, 3) = d(vec_ruta(k, j), destino_temp) + d(destino_temp, nodo_f)
                End If
            ElseIf destino_temp = vec_ruta(k, j) Then 'YAAAAAAAA
                a = j - 1
                Call verificacion3(k, i, a, spa)
                If infactible = 0 Then
                    comb = comb + 1
                    tabla(comb, 1) = i - 1
                    tabla(comb, 2) = j - 1
                    tabla(comb, 3) = 0
                End If
            End If
        End If
    ElseIf origen_temp <> vec_ruta(k, i) And origen_temp <> vec_ruta(k, i + 1) Then
        If j = i Then
            If destino_temp <> vec_ruta(k, j + 1) Then 'YAAAAAA
                temp_carga = capa_acum(k, i)
                temp_carga = temp_carga + 1
                If temp_carga <= capacidad Then
                    comb = comb + 1
                    tabla(comb, 1) = i
                    tabla(comb, 2) = i
                        tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, destino_temp) + d(destino_temp,
vec_ruta(k, i + 1)) - d(vec_ruta(k, i), vec_ruta(k, i + 1))
                End If
            End If
        ElseIf j > i And j < final Then 'ESTEEEE
            If destino_temp <> vec_ruta(k, j) And destino_temp <> vec_ruta(k, j + 1) Then 'YAAAAAA
                a = j
                Call verificacion4(k, i, a, spa)
                If infactible = 0 Then
                    comb = comb + 1
                    tabla(comb, 1) = i
                    tabla(comb, 2) = j
                        tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, vec_ruta(k, i + 1)) - d(vec_ruta(k, i),
vec_ruta(k, i + 1)) + d(vec_ruta(k, j), destino_temp) + d(destino_temp, vec_ruta(k, j + 1)) - d(vec_ruta(k, j), vec_ruta(k, j + 1))
                End If
            ElseIf destino_temp = vec_ruta(k, j) Then 'YAAAAAAAA
                a = j - 1
                Call verificacion4(k, i, a, spa)
                If infactible = 0 Then
                    comb = comb + 1
                    tabla(comb, 1) = i
                    tabla(comb, 2) = j - 1
                        tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, vec_ruta(k, i + 1)) - d(vec_ruta(k, i),
vec_ruta(k, i + 1))
                End If
            End If
        ElseIf j = final Then
            If destino_temp <> vec_ruta(k, j) Then 'YAAAAAA
                a = j
                Call verificacion4(k, i, a, spa)
                If infactible = 0 Then
                    comb = comb + 1
                    tabla(comb, 1) = i
                    tabla(comb, 2) = j
                        tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, vec_ruta(k, i + 1)) - d(vec_ruta(k, i),
vec_ruta(k, i + 1)) + d(vec_ruta(k, j), destino_temp) + d(destino_temp, nodo_f)
                End If
            ElseIf destino_temp = vec_ruta(k, j) Then 'YAAAAAAA
                a = j - 1
                Call verificacion4(k, i, a, spa)
                If infactible = 0 Then
                    comb = comb + 1
                    tabla(comb, 1) = i
                    tabla(comb, 2) = j - 1
```

```
                        tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, vec_ruta(k, i + 1)) - d(vec_ruta(k, i),
vec_ruta(k, i + 1))
                    End If
                End If
            End If
            'i 111 o d 1'111f pero o es = a 1' solo hay un caso
            ElseIf origen_temp <> vec_ruta(k, i) And origen_temp = vec_ruta(k, i + 1) Then
                If j = i Then
                    If destino_temp <> vec_ruta(k, j + 1) Then  'YAAAAAA
                        temp_carga = capa_acum(k, i)
                        temp_carga = temp_carga + 1
                        If temp_carga <= capacidad Then
                            comb = comb + 1
                            tabla(comb, 1) = i
                            tabla(comb, 2) = i
                              tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, destino_temp) + d(destino_temp,
vec_ruta(k, i + 1)) - d(vec_ruta(k, i), vec_ruta(k, i + 1))
                        End If
                    End If
                End If
            End If

        ElseIf i = final Then
            If final = largo_vec_ruta(k) Then
                If origen_temp = vec_ruta(k, i) Then  'YAAAAAAA
                    comb = comb + 1
                    tabla(comb, 1) = i - 1
                    tabla(comb, 2) = i '- 1 CORREGIDO 22 SEP 10:21PM
                    tabla(comb, 3) = d(vec_ruta(k, i), destino_temp) + d(destino_temp, nodo_f)
                Else
                    comb = comb + 1  'YAAAAAAA
                    tabla(comb, 1) = i
                    tabla(comb, 2) = i
                    tabla(comb, 3) = d(vec_ruta(k, i), origen_temp) + d(origen_temp, destino_temp) + d(destino_temp, nodo_f)
                End If
            End If
        End If
      Next j
    Next i
  Next y
Next x

End Sub

Sub verificacion1(k, i, j, spa)

ReDim capa_acum_t(1 To largo_vec_ruta(k))
For o = 1 To largo_vec_ruta(k)
  capa_acum_t(o) = capa_acum(k, o)
Next o

infactible = 0
For o = i To j
  If o > i And o < j Then
    capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
    If capa_acum_t(o) > capacidad Then
      infactible = infactible + 1
    End If
  End If
Next o

End Sub

Sub verificacion2(k, i, j, spa)

ReDim capa_acum_t(1 To largo_vec_ruta(k))
For o = 1 To largo_vec_ruta(k)
```

```
    capa_acum_t(o) = capa_acum(k, o)
Next o

infactible = 0
For o = i To j
    If o <> i Then
        capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
        If capa_acum_t(o) > capacidad Then
            infactible = infactible + 1
        End If
    End If
Next o

End Sub

Sub verificacion3(k, i, a, spa)

ReDim capa_acum_t(1 To largo_vec_ruta(k))
For o = 1 To largo_vec_ruta(k)
    capa_acum_t(o) = capa_acum(k, o)
Next o

infactible = 0
For o = i To a
    capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
    If capa_acum_t(o) > capacidad Then
        infactible = infactible + 1
    End If
Next o

End Sub
Sub verificacion3prima(k, i, a, spa)

ReDim capa_acum_t(1 To largo_vec_ruta(k))
For o = 1 To largo_vec_ruta(k)
    capa_acum_t(o) = capa_acum(k, o)
Next o

infactible = 0
For o = tabla_origen(i) To a
    capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
    If capa_acum_t(o) > capacidad Then
        infactible = infactible + 1
    End If
Next o

End Sub

Sub verificacion4(k, i, a, spa)

ReDim capa_acum_t(1 To largo_vec_ruta(k))
For o = 1 To largo_vec_ruta(k)
    capa_acum_t(o) = capa_acum(k, o)
Next o

infactible = 0
For o = i To a
    If o = i Then
        temp_carga = capa_acum(k, o)
        temp_carga = temp_carga + matriz_c(spa, origen_temp)
        If temp_carga > capacidad Then
            infactible = infactible + 1
        End If
    Else
        capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
        If capa_acum_t(o) > capacidad Then
            infactible = infactible + 1
```

```
      End If
   End If
Next o

End Sub
Sub verificacion5(k, i, j, spa)

ReDim capa_acum_t(1 To largo_vec_ruta(k))
For o = 1 To largo_vec_ruta(k)
   capa_acum_t(o) = capa_acum(k, o)
Next o

infactible = 0
For o = j To tabla_destino(i)
   If o = j Then
      If j = 1 Then
         temp_carga = 0
      Else
         temp_carga = capa_acum(k, o - 1)
      End If
      temp_carga = temp_carga + matriz_c(spa, origen_temp)
      If temp_carga > capacidad Then
         infactible = infactible + 1
      End If
      If o <> tabla_destino(i) Then
         capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
         If capa_acum_t(o) > capacidad Then
            infactible = infactible + 1
         End If
      End If
   Else
      If o <> tabla_destino(i) Then
         capa_acum_t(o) = capa_acum_t(o) + matriz_c(spa, origen_temp)
         If capa_acum_t(o) > capacidad Then
            infactible = infactible + 1
         End If
      End If
   End If
Next o

End Sub

Sub hallar_menor_ext(k, spa, comb)

'Ordena todas las combinaciones de menor a mayor y le halla los nodos entre destino y origen

ReDim tabla4(1 To comb, 1 To 4)

If comb <> 1 Then
   a = 1
   con_tabla4 = 0
   pasadas = True
   While pasadas
      'j tiene el menor y el menor_menor
      j = 1
      l = 2
      saltar = 0
      bandera3 = True
      While bandera3
         If tabla(j, 3) > tabla(l, 3) Then
            j = l
         ElseIf tabla(j, 3) = tabla(l, 3) Then
            nodos_entre_destino_origen_j = tabla(j, 2) - tabla(j, 1)
            nodos_entre_destino_origen_l = tabla(l, 2) - tabla(l, 1)
            If nodos_entre_destino_origen_j >= nodos_entre_destino_origen_l Then
               j = l
            End If
```

```
            End If
            l = l + 1
            If l > comb - ejecutadas Then
               con_tabla4 = con_tabla4 + 1
               tabla4(con_tabla4, 1) = tabla(j, 1)
               tabla4(con_tabla4, 2) = tabla(j, 2)
               tabla4(con_tabla4, 3) = tabla(j, 3)
               tabla4(con_tabla4, 4) = tabla(j, 2) - tabla(j, 1)
               '= Impresion tabla4 ================================================================
               'For imprimir_x = 1 To 4
               '    Cells(61 + con_tabla4, 2 + imprimir_x) = tabla4(con_tabla4, imprimir_x)
               'Next imprimir_x
               '====================================================================================
               bandera3 = False
               For c = 1 To comb - ejecutadas
                  If j = c Then
                     saltar = 1
                  Else
                     vy = c - saltar
                     For dd = 1 To 3
                        tabla(vy, dd) = tabla(c, dd)
                        'Cells(61 + vy, 7 + dd) = tabla(vy, dd)
                     Next dd
                  End If
               Next c
            End If
         Wend
         ejecutadas = ejecutadas + 1
         a = a + 1
         If a = comb Then
            con_tabla4 = con_tabla4 + 1
            tabla4(con_tabla4, 1) = tabla(1, 1)
            tabla4(con_tabla4, 2) = tabla(1, 2)
            tabla4(con_tabla4, 3) = tabla(1, 3)
            tabla4(con_tabla4, 4) = tabla(1, 2) - tabla(1, 1)
            '= Impresion tabla4 ================================================================
            'For imprimir_x = 1 To 4
            '    Cells(61 + con_tabla4, 2 + imprimir_x) = tabla4(con_tabla4, imprimir_x)
            'Next imprimir_x
            '====================================================================================
            pasadas = False
         End If
      Wend
   Else
      con_tabla4 = con_tabla4 + 1
      tabla4(con_tabla4, 1) = tabla(1, 1)
      tabla4(con_tabla4, 2) = tabla(1, 2)
      tabla4(con_tabla4, 3) = tabla(1, 3)
      tabla4(con_tabla4, 4) = tabla(1, 2) - tabla(1, 1)
      '= Impresion tabla4 ================================================================
      'For imprimir_x = 1 To 4
      '    Cells(61 + con_tabla4, 2 + imprimir_x) = tabla4(con_tabla4, imprimir_x)
      'Next imprimir_x
      '====================================================================================
   End If

   'seguir = True
   'pasa_comb = True
   con_seguir = 1
   'While seguir
   '    If tabla4(con_seguir, 4) <= max_ride_time Then
   '        Call reemplazar_valores_para_simulacion(spa, con_seguir)
   '        Call simulacion_nueva_solicitud(k, spa)
   '        While pasa_comb
               con_tabla2 = con_tabla2 + 1
               tabla2(con_tabla2, 1) = spa
               tabla2(con_tabla2, 2) = tabla4(con_seguir, 1)
```

```
          tabla2(con_tabla2, 3) = tabla4(con_seguir, 2)
          tabla2(con_tabla2, 4) = tabla4(con_seguir, 3)
'         seguir = False
'         pasa_comb = False
'      Wend
'   End If
'   con_seguir = con_seguir + 1
'   If con_seguir > comb Then
'      'menor_num_nodos tiene el menor
'      menor_num_nodos = 1
'      siguiente_menor = 2
'      bandera10 = True
'     While bandera10
'         If tabla4(menor_num_nodos, 4) > tabla4(siguiente_menor, 4) Then
'            menor_num_nodos = siguiente_menor
'         End If
'         siguiente_menor = siguiente_menor + 1
'         If siguiente_menor > comb Then
'            bandera10 = False
'         End If
'      Wend
'      con_tabla2 = con_tabla2 + 1
'      tabla2(con_tabla2, 1) = spa
'      tabla2(con_tabla2, 2) = tabla4(menor_num_nodos, 1)
'      tabla2(con_tabla2, 3) = tabla4(menor_num_nodos, 2)
'      tabla2(con_tabla2, 4) = tabla4(menor_num_nodos, 3)
'      seguir = False
'  End If
'Wend


'= Impresion tabla2 ======================================
'For i = 1 To 4
'    Cells(61 + con_tabla2, 11 + i) = tabla2(con_tabla2, i)
'Next i
'==========================================================

End Sub

Sub hallar_mayor_solicitud(k, con_tabla2)

'j tiene el mayor
j = 1
l = 2
bandera4 = True
While bandera4
   If tabla2(j, 5) < tabla2(l, 5) Then
      j = l
   End If
   l = l + 1
   If l > con_tabla2 Then
      Call nueva_solicitud(k, j)
      bandera4 = False
   End If
Wend

End Sub

'Construye los nuevos vectores de solicitud y de ruta con la mejor solicitud encontrada
Sub nueva_solicitud(k, j)

con_s(k) = con_s(k) + 1
vec_s(k, con_s(k)) = tabla2(j, 1)
tabla3(k, con_s(k), 1) = tabla2(j, 1)  'Solicitud

'Están ambos en ceros
If tabla2(j, 2) = 0 And tabla2(j, 3) = 0 Then
```

```
If matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3) + 1) Then
   'i o d i1111f
   If matriz_s(tabla2(j, 1), 1) <> nodo_i Then
      largo_vec_ruta_temp = largo_vec_ruta(k) + 3
      ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
      ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
      For i = 1 To largo_vec_ruta_temp
         If i = 1 Then
            vec_ruta_temp(i) = nodo_i
            capa_acum_temp(i) = 0
         ElseIf i = 2 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
            capa_acum_temp(i) = matriz_c(tabla2(j, 1), vec_ruta_temp(i))
            pos_origen = i
         ElseIf i = 3 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
            capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), vec_ruta_temp(i))
            pos_destino = i
         Else
            vec_ruta_temp(i) = vec_ruta(k, i - 3)
            capa_acum_temp(i) = capa_acum(k, i - 3)
         End If
      Next i
      For a = 1 To con_s(k) - 1
         tabla3(k, a, 2) = tabla3(k, a, 2) + 3
         tabla3(k, a, 3) = tabla3(k, a, 3) + 3
      Next a
   'io d i1111f    AQUIIIIIIIIIIIIIIIIIII
   ElseIf matriz_s(tabla2(j, 1), 1) = nodo_i Then
      largo_vec_ruta_temp = largo_vec_ruta(k) + 2
      ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
      ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
      For i = 1 To largo_vec_ruta_temp
         If i = 1 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
            capa_acum_temp(i) = matriz_c(tabla2(j, 1), vec_ruta_temp(i))
            pos_origen = i
         ElseIf i = 2 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
            capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), vec_ruta_temp(i))
            pos_destino = i
         Else
            vec_ruta_temp(i) = vec_ruta(k, i - 2)
            capa_acum_temp(i) = capa_acum(k, i - 2)
         End If
      Next i
      For a = 1 To con_s(k) - 1
         tabla3(k, a, 2) = tabla3(k, a, 2) + 2
         tabla3(k, a, 3) = tabla3(k, a, 3) + 2
      Next a
   End If
'i o id1111f
ElseIf matriz_s(tabla2(j, 1), 2) = vec_ruta(k, tabla2(j, 3) + 1) Then
   largo_vec_ruta_temp = largo_vec_ruta(k) + 2
   ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
   ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
   For i = 1 To largo_vec_ruta_temp
      If i = 1 Then
         vec_ruta_temp(i) = nodo_i
         capa_acum_temp(i) = 0
      ElseIf i = 2 Then
         vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
         capa_acum_temp(i) = matriz_c(tabla2(j, 1), vec_ruta_temp(i))
         pos_origen = i
      Else
         vec_ruta_temp(i) = vec_ruta(k, i - 2)
         capa_acum_temp(i) = capa_acum(k, i - 2)
```

```
            If tabla2(j, 3) + 3 = i Then    'Esto es igual a i - 3 = tabla2(j,3)
               pos_destino = i
            End If
         End If
      Next i
      For a = 1 To con_s(k) - 1
         tabla3(k, a, 2) = tabla3(k, a, 2) + 2
         tabla3(k, a, 3) = tabla3(k, a, 3) + 2
      Next a
   End If

'Origen en cero y destino <> de cero
ElseIf tabla2(j, 2) = 0 And tabla2(j, 3) <> 0 Then
   'Destino menor que el ultimo del vector
   If tabla2(j, 3) < largo_vec_ruta(k) Then
      If matriz_s(tabla2(j, 1), 1) <> vec_ruta(k, tabla2(j, 2) + 1) Then
         'i o i1111d1111f
         If matriz_s(tabla2(j, 1), 2) = vec_ruta(k, tabla2(j, 3) + 1) Then
            largo_vec_ruta_temp = largo_vec_ruta(k) + 2
            ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
            ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
            For i = 1 To largo_vec_ruta_temp
               If i = 1 Then
                  vec_ruta_temp(i) = nodo_i
                  capa_acum_temp(i) = 0
               ElseIf i = 2 Then
                  vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
                  capa_acum_temp(i) = matriz_c(tabla2(j, 1), vec_ruta_temp(i))
                  pos_origen = i
               Else
                  vec_ruta_temp(i) = vec_ruta(k, i - 2)
                  If i <= tabla2(j, 3) Then
                     capa_acum_temp(i) = capa_acum(k, i - 2) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                  Else
                     capa_acum_temp(i) = capa_acum(k, i - 2)
                  End If
                  If tabla2(j, 3) + 3 = i Then
                     pos_destino = i
                  End If
               End If
            Next i
            For a = 1 To con_s(k) - 1
               tabla3(k, a, 2) = tabla3(k, a, 3) + 2
               tabla3(k, a, 3) = tabla3(k, a, 3) + 2
            Next a
         'i o i1111 d 1111f  PPPPPPPPPPPPPPPPP
         ElseIf matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3) + 1) Then
            largo_vec_ruta_temp = largo_vec_ruta(k) + 3
            ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
            ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
            For i = 1 To largo_vec_ruta_temp
               If i = 1 Then
                  vec_ruta_temp(i) = nodo_i
                  capa_acum_temp(i) = 0
               ElseIf i = 2 Then
                  vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
                  a = 2
                  capa_acum_temp(i) = matriz_c(tabla2(j, 1), vec_ruta_temp(i))
                  pos_origen = i
               ElseIf i - 2 = tabla2(j, 3) + 1 Then
                  vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                  a = 3
                  capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
                  pos_destino = i
               Else
                  vec_ruta_temp(i) = vec_ruta(k, i - a)
                  If i - 2 <= tabla2(j, 3) + 1 Then
```

```
            capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
          Else
            capa_acum_temp(i) = capa_acum(k, i - a)
          End If
        End If
      Next i
      For a = 1 To con_s(k) - 1
        If tabla3(k, a, 2) + 2 >= pos_destino Then
          tabla3(k, a, 2) = tabla3(k, a, 2) + 3
          tabla3(k, a, 3) = tabla3(k, a, 3) + 3
        ElseIf tabla3(k, a, 2) + 2 < pos_destino Then
          tabla3(k, a, 2) = tabla3(k, a, 2) + 2
        End If
        If tabla3(k, a, 3) + 2 >= pos_destino Then
          tabla3(k, a, 3) = tabla3(k, a, 3) + 3
        ElseIf tabla3(k, a, 3) + 2 < pos_destino Then
          tabla3(k, a, 3) = tabla3(k, a, 3) + 2
        End If
      Next a
    End If
  ElseIf matriz_s(tabla2(j, 1), 1) = vec_ruta(k, tabla2(j, 2) + 1) Then
    'io d 11111f  and io111 d 11f   AQUIIIIIIIIIII
    If matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3) + 1) Then
      largo_vec_ruta_temp = largo_vec_ruta(k) + 1
      ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
      ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
      For i = 1 To largo_vec_ruta_temp
        If i = tabla2(j, 3) + 1 Then
          vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
          a = 1
          capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
          pos_destino = i
        Else
          vec_ruta_temp(i) = vec_ruta(k, i - a)
          If i <= tabla2(j, 3) Then
            capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
          Else
            capa_acum_temp(i) = capa_acum(k, i - a)
          End If
          If tabla2(j, 2) + 1 = i Then
            pos_origen = i
          End If
        End If
      Next i
      For a = 1 To con_s(k) - 1
        If tabla3(k, a, 2) + 1 > pos_destino Then
          tabla3(k, a, 2) = tabla3(k, a, 2) + 1
        End If
        If tabla3(k, a, 3) + 1 > pos_destino Then
          tabla3(k, a, 3) = tabla3(k, a, 3) + 1
        End If
      Next a
    'io 1d11111f and io111 1d 11f
    ElseIf matriz_s(tabla2(j, 1), 2) = vec_ruta(k, tabla2(j, 3) + 1) Then
      largo_vec_ruta_temp = largo_vec_ruta(k)
      ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
      ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
      For i = 1 To largo_vec_ruta_temp
        vec_ruta_temp(i) = vec_ruta(k, i)
        If i <= tabla2(j, 3) Then
          capa_acum_temp(i) = capa_acum(k, i) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
        Else
          capa_acum_temp(i) = capa_acum(k, i)
        End If
        If tabla2(j, 2) + 1 = i Then
          pos_origen = i
        ElseIf tabla2(j, 3) + 1 = i Then
```

```
                pos_destino = i
            End If
        Next i
    End If
End If

'Destino igual que el ultimo del vector
ElseIf tabla2(j, 3) = largo_vec_ruta(k) Then

    If matriz_s(tabla2(j, 1), 1) <> vec_ruta(k, tabla2(j, 2) + 1) Then
        'i o i11111f d f
        If matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3)) Then
            largo_vec_ruta_temp = largo_vec_ruta(k) + 4
            ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
            ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
            For i = 1 To largo_vec_ruta_temp
                If i = 1 Then
                    vec_ruta_temp(i) = nodo_i
                    capa_acum_temp(i) = 0
                ElseIf i = 2 Then
                    vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
                    a = 2
                    capa_acum_temp(i) = matriz_c(tabla2(j, 1), vec_ruta_temp(i))
                    pos_origen = i
                ElseIf i = largo_vec_ruta(k) + 3 Then
                    vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                    capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), vec_ruta_temp(i))
                    pos_destino = i
                ElseIf i = largo_vec_ruta(k) + 4 Then
                    vec_ruta_temp(i) = nodo_f
                    capa_acum_temp(i) = 0
                Else
                    vec_ruta_temp(i) = vec_ruta(k, i - a)
                    capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                End If
            Next i
            For a = 1 To con_s(k) - 1
                tabla3(k, a, 2) = tabla3(k, a, 2) + 2
                tabla3(k, a, 3) = tabla3(k, a, 3) + 2
            Next a
        End If

    ElseIf matriz_s(tabla2(j, 1), 1) = vec_ruta(k, tabla2(j, 2) + 1) Then
        'io11111f d f
        If matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3)) Then
            largo_vec_ruta_temp = largo_vec_ruta(k) + 2
            ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
            ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
            For i = 1 To largo_vec_ruta_temp
                If i = largo_vec_ruta(k) + 1 Then
                    vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                    capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), vec_ruta_temp(i))
                    pos_destino = i
                ElseIf i = largo_vec_ruta(k) + 2 Then
                    vec_ruta_temp(i) = nodo_f
                    capa_acum_temp(i) = 0
                Else
                    vec_ruta_temp(i) = vec_ruta(k, i - a)
                    If i <= tabla2(j, 3) Then
                        capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                    Else
                        capa_acum_temp(i) = capa_acum(k, i - a)
                    End If
                    If tabla2(j, 2) + 1 = i Then
                        pos_origen = i
                    End If
                End If
```

```
            Next i
          End If
        End If
    End If

'Origen y destino diferentes de cero   PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
ElseIf tabla2(j, 2) <> 0 And tabla2(j, 3) <> 0 Then
    'Origen menor del vector
    If tabla2(j, 2) < largo_vec_ruta(k) Then
        'Destino menor del vector
        If tabla2(j, 3) < largo_vec_ruta(k) Then
            If matriz_s(tabla2(j, 1), 1) = vec_ruta(k, tabla2(j, 2) + 1) And matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3) + 1)
Then
            'i11o d 11111f
            If tabla2(j, 2) = tabla2(j, 3) Then
                largo_vec_ruta_temp = largo_vec_ruta(k) + 1
                ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
                ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
                For i = 1 To largo_vec_ruta_temp
                    If i - 1 = tabla2(j, 2) Then
                        vec_ruta_temp(i) = vec_ruta(k, i - a)
                        capa_acum_temp(i) = capa_acum(k, i) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                        pos_origen = i
                        i = i + 1
                        vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                        capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
                        a = 1
                        pos_destino = i
                    Else
                        vec_ruta_temp(i) = vec_ruta(k, i - a)
                        capa_acum_temp(i) = capa_acum(k, i - a)
                    End If
                Next i
                For a = 1 To con_s(k) - 1
                    If tabla3(k, a, 2) + 1 > pos_destino Then
                        tabla3(k, a, 2) = tabla3(k, a, 2) + 1
                    End If
                    If tabla3(k, a, 3) + 1 > pos_destino Then
                        tabla3(k, a, 3) = tabla3(k, a, 3) + 1
                    End If
                Next a
            'i11o111 d 11f
            ElseIf tabla2(j, 2) <> tabla2(j, 3) Then
                largo_vec_ruta_temp = largo_vec_ruta(k) + 1
                ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
                ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
                For i = 1 To largo_vec_ruta_temp
                    If i - 1 = tabla2(j, 2) Then
                        'bandera5 = True
                        vec_ruta_temp(i) = vec_ruta(k, i - a)
                        capa_acum_temp(i) = capa_acum(k, i) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                        pos_origen = i
                    ElseIf i - 1 = tabla2(j, 3) Then
                        'While bandera5
                            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                            a = 1
                            'bandera5 = False
                            capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
                            pos_destino = i
                        'Wend
                    Else
                        vec_ruta_temp(i) = vec_ruta(k, i - a)
                        If i > tabla2(j, 2) And i <= tabla2(j, 3) Then
                            capa_acum_temp(i) = capa_acum(k, i) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                        Else
                            capa_acum_temp(i) = capa_acum(k, i - a)
                        End If
```

```
                End If
            Next i
            For a = 1 To con_s(k) - 1
                If tabla3(k, a, 2) + 1 > pos_destino Then
                    tabla3(k, a, 2) = tabla3(k, a, 2) + 1
                End If
                If tabla3(k, a, 3) + 1 > pos_destino Then
                    tabla3(k, a, 3) = tabla3(k, a, 3) + 1
                End If
            Next a
        End If
    'i11o 1d11111f and i111o111 1d 11f
    ElseIf matriz_s(tabla2(j, 1), 1) = vec_ruta(k, tabla2(j, 2) + 1) And matriz_s(tabla2(j, 1), 2) = vec_ruta(k, tabla2(j, 3) + 1)
Then
        largo_vec_ruta_temp = largo_vec_ruta(k)
        ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
        ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
        For i = 1 To largo_vec_ruta_temp
            vec_ruta_temp(i) = vec_ruta(k, i)
            If i > tabla2(j, 2) And i <= tabla2(j, 3) Then
                capa_acum_temp(i) = capa_acum(k, i) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
            Else
                capa_acum_temp(i) = capa_acum(k, i)
            End If
            If tabla2(j, 2) + 1 = i Then
                pos_origen = i
            ElseIf tabla2(j, 3) + 1 = i Then
                pos_destino = i
            End If
        Next i

    ElseIf matriz_s(tabla2(j, 1), 1) <> vec_ruta(k, tabla2(j, 2) + 1) And matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3) +
1) Then
        'i o d 111111f
        If tabla2(j, 2) = tabla2(j, 3) Then
            largo_vec_ruta_temp = largo_vec_ruta(k) + 2
            ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
            ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
            For i = 1 To largo_vec_ruta_temp
                If i - 1 = tabla2(j, 2) Then
                    vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
                    capa_acum_temp(i) = capa_acum_temp(i - 1) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                    pos_origen = i
                    i = i + 1
                    vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                    capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
                    a = 2
                    pos_destino = i
                Else
                    vec_ruta_temp(i) = vec_ruta(k, i - a)
                    capa_acum_temp(i) = capa_acum(k, i - a)
                End If
            Next i
            For a = 1 To con_s(k) - 1
                If tabla3(k, a, 2) + 2 > pos_destino Then
                    tabla3(k, a, 2) = tabla3(k, a, 2) + 2
                End If
                If tabla3(k, a, 3) + 2 > pos_destino Then
                    tabla3(k, a, 3) = tabla3(k, a, 3) + 2
                End If
            Next a

        'i o 11 d 111f
        ElseIf tabla2(j, 2) <> tabla2(j, 3) Then
            largo_vec_ruta_temp = largo_vec_ruta(k) + 2
            ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
            ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
```

```
            'bandera5 = False
            For i = 1 To largo_vec_ruta_temp
               If i - 1 = tabla2(j, 2) Then
                  vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
                  a = 1
                  'bandera5 = True
                  capa_acum_temp(i) = capa_acum_temp(i - 1) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                  pos_origen = i
               ElseIf i - 2 = tabla2(j, 3) Then
                  'While bandera5
                     vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
                     a = 2
                     'bandera5 = False
                     capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
                     pos_destino = i
                  'Wend
               Else
                  vec_ruta_temp(i) = vec_ruta(k, i - a)
                  If i > tabla2(j, 2) And i <= tabla2(j, 3) + 1 Then
                     capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
                  Else
                     capa_acum_temp(i) = capa_acum(k, i - a)
                  End If
               End If
            Next i
            For a = 1 To con_s(k) - 1
               If tabla3(k, a, 2) + 1 > pos_origen Then
                  tabla3(k, a, 2) = tabla3(k, a, 2) + 1
               End If
               If tabla3(k, a, 3) + 1 > pos_origen Then
                  tabla3(k, a, 3) = tabla3(k, a, 3) + 1
               End If
               If tabla3(k, a, 2) + 1 > pos_destino Then
                  tabla3(k, a, 2) = tabla3(k, a, 2) + 1
               End If
               If tabla3(k, a, 3) + 1 > pos_destino Then
                  tabla3(k, a, 3) = tabla3(k, a, 3) + 1
               End If
            Next a
         End If
      'i o d1111f and i1111 o 111d11f AQUI TAL VEZ HAYA OTRO CASO
       ElseIf matriz_s(tabla2(j, 1), 1) <> vec_ruta(k, tabla2(j, 2) + 1) And matriz_s(tabla2(j, 1), 2) = vec_ruta(k, tabla2(j, 3) +
1) Then
         largo_vec_ruta_temp = largo_vec_ruta(k) + 1
         ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
         ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
         For i = 1 To largo_vec_ruta_temp
            If i - 1 = tabla2(j, 2) Then
               vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
               a = 1
               capa_acum_temp(i) = capa_acum_temp(i - 1) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
               pos_origen = i
            Else
               vec_ruta_temp(i) = vec_ruta(k, i - a)
               If i > tabla2(j, 2) And i <= tabla2(j, 3) + 1 Then  'OJOOOOOOO hay un + 1
                  capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
               Else
                  capa_acum_temp(i) = capa_acum(k, i - a)
               End If
               If i - 2 = tabla2(j, 3) Then
                  pos_destino = i
               End If
            End If
         Next i
         For a = 1 To con_s(k) - 1
            If tabla3(k, a, 2) + 1 > pos_origen Then
               tabla3(k, a, 2) = tabla3(k, a, 2) + 1
```

```
            End If
            If tabla3(k, a, 3) + 1 > pos_origen Then
               tabla3(k, a, 3) = tabla3(k, a, 3) + 1
            End If
         Next a
      End If

   'Destino igual del vector
   ElseIf tabla2(j, 3) = largo_vec_ruta(k) Then
      'i11o11111f d f
      If matriz_s(tabla2(j, 1), 1) = vec_ruta(k, tabla2(j, 2) + 1) And matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3)) Then
         largo_vec_ruta_temp = largo_vec_ruta(k) + 2
         ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
         ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
         For i = 1 To largo_vec_ruta_temp
            If i = largo_vec_ruta(k) + 1 Then
               vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
               capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
               pos_destino = i
            ElseIf i = largo_vec_ruta(k) + 2 Then
               vec_ruta_temp(i) = nodo_f
               capa_acum_temp(i) = 0
            Else
               vec_ruta_temp(i) = vec_ruta(k, i)
               If i > tabla2(j, 2) And i <= tabla2(j, 3) Then
                  capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
               Else
                  capa_acum_temp(i) = capa_acum(k, i - a)
               End If
               If i - 1 = tabla2(j, 2) Then
                  pos_origen = i
               End If
            End If
         Next i
      'i o 111f d f
      ElseIf matriz_s(tabla2(j, 1), 1) <> vec_ruta(k, tabla2(j, 2) + 1) And matriz_s(tabla2(j, 1), 2) <> vec_ruta(k, tabla2(j, 3))
Then
         largo_vec_ruta_temp = largo_vec_ruta(k) + 3
         ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
         ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
         For i = 1 To largo_vec_ruta_temp
            If i - 1 = tabla2(j, 2) Then
               vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
               a = 1
               capa_acum_temp(i) = capa_acum_temp(i - 1) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
               pos_origen = i
            ElseIf i = largo_vec_ruta(k) + 2 Then
               vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
               capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
               pos_destino = i
            ElseIf i = largo_vec_ruta(k) + 3 Then
               vec_ruta_temp(i) = nodo_f
               capa_acum_temp(i) = 0
            Else
               vec_ruta_temp(i) = vec_ruta(k, i - a)
               If i > tabla2(j, 2) And i <= tabla2(j, 3) + 1 Then
                  capa_acum_temp(i) = capa_acum(k, i - a) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
               Else
                  capa_acum_temp(i) = capa_acum(k, i - a)
               End If
            End If
         Next i
         For a = 1 To con_s(k) - 1
            If tabla3(k, a, 2) + 1 > pos_origen Then
               tabla3(k, a, 2) = tabla3(k, a, 2) + 1
            End If
            If tabla3(k, a, 3) + 1 > pos_origen Then
```

```
                tabla3(k, a, 3) = tabla3(k, a, 3) + 1
            End If
          Next a
        End If
      End If

    'Origen igual del vector
    ElseIf tabla2(j, 2) = largo_vec_ruta(k) Then
      'i1111f o d f
      If matriz_s(tabla2(j, 1), 2) <> nodo_f Then
        largo_vec_ruta_temp = largo_vec_ruta(k) + 3
        ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
        ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
        For i = 1 To largo_vec_ruta_temp
          If i = largo_vec_ruta(k) + 1 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
            capa_acum_temp(i) = capa_acum_temp(i - 1) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
            pos_origen = i
          ElseIf i = largo_vec_ruta(k) + 2 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
            capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
            pos_destino = i
          ElseIf i = largo_vec_ruta(k) + 3 Then
            vec_ruta_temp(i) = nodo_f
            capa_acum_temp(i) = 0
          Else
            vec_ruta_temp(i) = vec_ruta(k, i)
            capa_acum_temp(i) = capa_acum(k, i - a)
          End If
        Next i
      'i1111f o df
      ElseIf matriz_s(tabla2(j, 1), 2) = nodo_f Then
        largo_vec_ruta_temp = largo_vec_ruta(k) + 2
        ReDim vec_ruta_temp(1 To largo_vec_ruta_temp)
        ReDim capa_acum_temp(1 To largo_vec_ruta_temp)
        For i = 1 To largo_vec_ruta_temp
          If i = largo_vec_ruta(k) + 1 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 1)
            capa_acum_temp(i) = capa_acum_temp(i - 1) + matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 1))
            pos_origen = i
          ElseIf i = largo_vec_ruta(k) + 2 Then
            vec_ruta_temp(i) = matriz_s(tabla2(j, 1), 2)
            capa_acum_temp(i) = capa_acum_temp(i - 1) - matriz_c(tabla2(j, 1), matriz_s(tabla2(j, 1), 2))
            pos_destino = i
          Else
            vec_ruta_temp(i) = vec_ruta(k, i)
            capa_acum_temp(i) = capa_acum(k, i - a)
          End If
        Next i
      End If
    End If
End If

dist_ruta(k) = dist_ruta(k) + tabla2(j, 4)
largo_vec_ruta(k) = largo_vec_ruta_temp
For i = 1 To largo_vec_ruta(k)
  vec_ruta(k, i) = vec_ruta_temp(i)
Next i
For i = 1 To largo_vec_ruta(k)
  capa_acum(k, i) = capa_acum_temp(i)
Next i

tabla3(k, con_s(k), 2) = pos_origen   'posicion en vector ruta de origen
tabla3(k, con_s(k), 3) = pos_destino  'posicion en vector ruta de destino


'= Impresion vec_s, vec_ruta, capa_acum y dist_ruta ==============================
```

```
'For i = 1 To con_s(k)
'    Cells(61 + k, 22 + i) = vec_s(k, i)
'Next i
'For i = 1 To largo_vec_ruta(k)
'    Cells(66 + k, 22 + i) = vec_ruta(k, i)
'Next i
'For i = 1 To largo_vec_ruta(k)
'    Cells(71 + k, 22 + i) = capa_acum(k, i)
'Next i
'Cells(66 + k, 22) = dist_ruta(k)
'================================================================================

End Sub
```