Cátedra Nissan

-PROTHIUS-

Aplicación de los algoritmos de hormigas para la resolución del problema de equilibrado de líneas de montaje SALBP-E y aplicación a un caso real – Memoria

Javier Bretón Blas y Jose Antonio Fernández Ros

SW-118/2010
(CN 2010-BF)

Departamento de Organización de Empresas

Universidad Politécnica de Cataluña

Publica:

Universitat Politècnica de Catalunya
www.upc.edu

Edita:

Cátedra Nissan
www.nissanchair.com
director@nissanchair.com

TÍTOL	**Aplicación de los algoritmos de hormigas para la resolución del problema de equilibrado de líneas de montaje SALBP-E y aplicación a un caso real.**
ALUMNE	Javier Bretón Blas; Jose Antonio Fernández Ros
ASSIGNATURA	Proyecto final del carrera
PROFESSOR	D. Joaquín Bautista Valhondo
DATA	4 de octubre de 2000
DOCUMENT	Memoria

RESUM

La memoria incluye una introducción a la problemática del equilibrado de líneas de montaje, el modelo matemático que se utilizará para modelizar el SALBP-E y una descripción de diferentes procedimientos para su resolución. Así mismo, contiene una introducción a los algoritmos de hormigas y la propuesta y desarrollo de dos métodos heurísticos basados en éstos últimos. Además, se resumen los resultados de la experiencia computacional y se analizan para obtener las conclusiones del trabajo y las futuras líneas de investigación. Por último, se describe el presupuesto total del proyecto, desglosado en sus partidas más importantes.

MEMORIA

RESUMEN **1**

CAPÍTULO 1: **INTRODUCCIÓN AL EQUILIBRADO DE LÍNEAS DE MONTAJE** **3**

1.1 CONTEXTO 4
1.2 LÍNEAS DE MONTAJE O PRODUCCIÓN 6
1.2.1 Sistemas de producción orientados al producto *6*
1.2.2 Líneas de montaje. Descripción *7*
1.3 TIPOS DE LÍNEAS DE MONTAJE 8
1.3.1 Número de productos *8*
1.3.2 Líneas con buffer entre estaciones *8*
1.3.3 Variabilidad de los tiempos de las tareas *8*
1.3.4 Tipos de restricciones *8*
1.3.5 Distribución en planta *9*
1.4 EL PROBLEMA DE EQUILIBRADO DE UNA LÍNEA 11
1.4.1 Conceptos generales *11*
1.4.2 Objetivos del equilibrado *11*
1.5 CLASIFICACIÓN DE LOS PROBLEMAS DE EQUILIBRADO 13

CAPÍTULO 2: **EL PROBLEMA SALB-E** **15**

2.1 TERMINOLOGIA BÁSICA 16
2.1.1 Notación *16*
2.1.2 Grafo de precedencias *16*
2.1.3 Asignación de tareas *17*
2.1.4 Capacidad de la línea *17*
2.1.5 Eficiencia de la línea *17*
2.2 DESCRIPCIÓN DEL PROBLEMA SALB-E 18
2.3 SIMETRÍA DEL PROBLEMA 19
2.4 COTAS DE LA FUNCIÓN OBJETIVO 20
2.4.1 Cotas inferiores para el SALBP-E *20*
2.4.2 Cotas superiores para el SALBP-E *20*
2.5 MODELO MATEMÁTICO 21

CAPÍTULO 3: **MÉTODOS DE RESOLUCIÓN** **23**

3.1 COMPLEJIDAD 24
3.2 MÉTODOS EXACTOS 25
3.3 INCONVENIENTES DE LOS MÉTODOS EXACTOS 26
3.4 MÉTODOS HEURÍSTICOS 27
3.4.1 Introducción *27*
3.4.2 Método Fix and Relax *27*
3.4.3 Método de exploración de entornos *28*
3.4.4 Heurísticas inspiradas en fenómenos naturales *29*

CAPÍTULO 4: **ALGORITMOS ACO** **32**

4.1 INTRODUCCIÓN 33
4.2 ALGORITMOS DE HORMIGAS 34
4.2.1 Descripción de los algoritmos de hormigas *34*

4.2.2 Algoritmos ACO *36*
4.2.3 Descripción del algoritmo *38*
4.3 APLICACIÓN AL PROBLEMA TSP 39
4.3.1 El problema TSP *39*
4.3.2 Ant System (AS) *39*
4.3.3 Ant Colony System (ACS) *41*
4.3.4 Max-Min Ant System (MMAS) *42*
4.3.5 Versión "Rank-Based" del AS *42*
4.3.6 Técnicas adicionales *43*
4.4 PROPIEDADES DE LOS ALGORITMOS ACO 45
4.4.1 Parámetros de control *45*
4.4.2 Papel de la heurística *45*
4.4.3 Evaluación implícita de las soluciones *45*
4.4.4 Autocatálisis *46*
4.4.5 Paralelización del algoritmo *46*
4.5 OTRAS APLICACIONES 48
4.5.1 Problemas combinatorios estáticos *48*
4.5.2 Problemas combinatorios dinámicos *48*

CAPÍTULO 5: **APLICACIÓN DE LOS ALGORITMOS ACO A LA RESOLUCIÓN DEL SALBP-E** **50**

5.1 META-HEURÍSTICA ACO PARA EL SALBP-E 51
5.1.1 Introducción *51*
5.1.2 Función objetivo del SALBP-E *52*
5.1.3 Funcionamiento general: una colonia de hormigas *52*
5.1.4 Funcionamiento interno: la subcolonia *57*
5.1.5 Criterio de finalización *63*
5.1.6 Configuración de los parámetros básicos *63*
5.2 META-HEURISTICA ACO PARA EL SALBP-E COMBINADA CON UNA MEJORA LOCAL 65
5.2.1 Objetivos de la mejora local dentro del algoritmo de hormigas *65*
5.2.2 Funcionamiento de la mejora local *66*
5.2.3 Generación de nuevas secuencias de tareas durante la mejora local *68*
5.2.4 Condición de final *69*
5.2.5 Evaluación de la solución *69*

CAPÍTULO 6: **EXPERIENCIA COMPUTACIONAL** **75**

6.1 DESCRIPCION DE LAS PRUEBAS 76
6.1.1 Introducción *76*
6.1.2 Pruebas con el número de soluciones fijado *76*
6.1.3 Pruebas con el tiempo de resolución fijado *77*
6.2 ENTORNO COMPUTACIONAL 79
6.2.1 El lenguaje Java *79*
6.2.2 Maquinas utilizadas *79*
6.2.3 Juego de datos *79*
6.3 RIEJU 81
6.3.1 Características del juego de datos *81*
6.3.2 Resultados *81*

CAPÍTULO 7: **CONCLUSIONES** **82**

7.1 LOS ALGORITMOS ACO 83
7.2 CONCLUSIONES DE LA EXPERIENCIA COMPUTACIONAL 84
7.2.1 Heurística de referencia *84*
7.2.2 Evaluación de la solución *84*
7.2.3 Optimización de los parámetros y backtracking *85*
7.2.4 Comparación de métodos heurísticos *87*
7.3 APLICACIÓN AL CASO RIEJU 92

CAPÍTULO 8: **LÍNEAS FUTURAS DE INVESTIGACIÓN** **93**

CAPÍTULO 9: **PRESUPUESTO** **95**

9.1 INVESTIGACIÓN Y DESARROLLO (I+D) 96
9.1.1 Costes de personal *96*
9.1.2 Costes de material *96*
9.1.3 Otros costes *97*
9.1.4 Resumen costes *97*
9.2 APLICACIÓN AL CASO RIEJU 98
9.2.1 Costes de puesta en marcha *98*
9.2.2 Costes de investigación *98*
9.2.3 Costes totales de la aplicación al caso Rieju *98*

BIBLIOGRAFIA Y REFERENCIAS **99**

ANEXO I:EXPERIENCIA COMPUTACIONAL.RESUMEN PRUEBAS

Carpeta 1

RESUMEN **2**

GRUPO I: **7 A 30 TAREAS** **3**

I.1 MERTENS (N=7) 4
I.2 BOWMAN (N=8) 6
I.3 JAESCHKE (N=9) 8
I.4 JACKSON (N=11) 12
I.5 MANSOOR (N=11) 15
I.6 MITCHELL (N=21) 17
I.7 ROSZIEG (N=25) 21
I.8 HESKIAOFF (N=28) 25
I.9 BUXEY (N=29) 31
I.10 SAWYER (N=30) 38

GRUPO II: **30 A 60 TAREAS** **46**

II.1 LUTZ1 (N=32) 47
II.2 GUNTHER (N=35) 53
II.3 KILBRIDGE (N=45) 61
II.4 HAHN (N=53) 67
II.5 WARNECKE (N=58) 71

GRUPO III: **60 A 90 TAREAS** **86**

III.1 TONGE (N=70) 87
III.2 WEE-MAG (N=75) 99
III.3 ARCUS1 (N=83) 121
III.4 LUTZ2 (N=89) 135
III.5 LUTZ3 (N=89) 157

Carpeta 2

GRUPO IV: **90 A 150 TAREAS** **171**

IV.1 ARCUS2 (N=111) 172
IV.2 BARTHOLD (N=148) 192
IV.3 BARTHOLD2 (N=148) 201

GRUPO V: **MÁS DE 150 TAREAS** **231**

V.1 SCHOLL (N=297) 232

ANEXO II: EXPERIENCIA COMPUTACIONAL.SOLUCIÓN DE LOS JUEGOS DE DATOS

Carpeta **1**

RESUMEN **2**

GRUPO I: 7 A 30 TAREAS **3**

I.1 MERTENS (N=7) 4
I.2 BOWMAN (N=8) 6
I.3 JAESCHKE (N=9) 8
I.4 JACKSON (N=11) 12
I.5 MANSOOR (N=11) 15
I.6 MITCHELL (N=21) 17
I.7 ROSZIEG (N=25) 21
I.8 HESKIAOFF (N=28) 25
I.9 BUXEY (N=29) 31
I.10 SAWYER (N=30) 38

GRUPO II: 30 A 60 TAREAS **46**

II.1 LUTZ1 (N=32) 47
II.2 GUNTHER (N=35) 53
II.3 KILBRIDGE (N=45) 61
II.4 HAHN (N=53) 67
II.5 WARNECKE (N=58) 72

GRUPO III: 60 A 90 TAREAS **87**

III.1 TONGE (N=70) 88
III.2 WEE-MAG (N=75) 100
III.3 ARCUS1 (N=83) 127
III.4 LUTZ2 (N=89) 141
III.5 LUTZ3 (N=89) 168

Carpeta **2**

GRUPO IV: 90 A 150 TAREAS **182**

IV.1 ARCUS2 (N=111) 183
IV.2 BARTHOLD (N=148) 203
IV.3 BARTHOLD2 (N=148) 212

GRUPO V: MÁS DE 150 TAREAS **258**

V.1 SCHOLL (N=297) 259

ANEXO III: APLICACIÓN AL CASO RIEJU

RESUMEN **1**

CAPÍTULO 1: **ENUNCIADO DEL PROBLEMA** **2**

1.1 LA EMPRESA 3
1.2 ENUNCIADO ORIGINAL: PRECEDENCIAS 4
1.3 ENUNCIADO SIMPLIFICADO: PRECEDENCIAS 7

CAPÍTULO 2: **SOLUCIONES DEL SALBP-E** **12**

2.1 RESUMEN DE LA EXPERIENCIA COMPUTACIONAL 13
2.1.1 Características del juego de datos *13*
2.1.2 Tabla de resultados *13*
2.1.3 Distribución de tareas del enunciado original *14*
2.1.4 Distribución de tareas del enunciado simplificado *14*

CAPÍTULO 3: **CONCLUSIONES** **16**

ANEXO IV: LISTADOS DEL PROGRAMA

RESUMEN **1**

CAPÍTULO 1: **ALGORITMO DE HORMIGAS** **2**

1.1 DESCRIPCIÓN DEL PROGRAMA 3
1.2 LISTADO DE LOS OBJETOS 5

CAPÍTULO 2: **ALGORITMO DE LAS HORMIGAS CON MEJORA LOCAL** **58**

2.1 DESCRIPCIÓN DEL PROGRAMA 59
2.2 LISTADO DE LOS OBJETOS 60

CAPÍTULO 3: **MULTI-START** **68**

3.1 DESCRIPCIÓN DEL PROGRAMA 69
3.2 LISTADO DE OBJETOS 70

RESUMEN

En este trabajo se propone un nuevo método heurístico basado en los algoritmos de hormigas para la resolución del problema de equilibrado de líneas de montaje con función objetivo producto del número de estaciones y el tiempo de ciclo (SALBP-E; *Simple Assembly Line Balancing Problem*).

Los algoritmos de hormigas son procedimientos heurísticos utilizados en la resolución de problemas de optimización discreta que está basado en el comportamiento de las hormigas. Sus principales características son: *(1)* la utilización de "*feed-back*" positivo (acelera el descubrimiento de soluciones de gran calidad), *(2) computación distribuida* (la estructura de estos algoritmos permite su paralelización de forma muy simple y natural), y *(3)* el *uso de heurísticas Greedy constructivas* (ayuda a encontrar soluciones aceptables en las primeras etapas del proceso de exploración, además de mantener un nivel mínimo de exploración durante todo el proceso).

Para comprobar su funcionamiento, se realizará una extensa experiencia computacional en la que se utilizará un juego de datos extraídos de *Scholl (1999)*, el cual ha sido utilizado por gran número de autores, y del que se conocen sus soluciones óptimas, con lo que se podrá comprobar la calidad de la solución obtenida en cada juego de datos. Una vez comprobado el funcionamiento con el juego de datos anterior, el procedimiento heurístico se probará con un juego de datos reales extraídos de la empresa Rieju.

Por lo tanto, los objetivos principales del trabajo son:

- Proponer un nuevo procedimiento heurístico para resolver el problema de equilibrado de líneas de montaje SALBP-E (*Simple Assembly Line Balancing Problem*).
- Comprobar la viabilidad de la meta-heurística ACO como sistema de resolución del problema de equilibrado de líneas de montaje SALB-E.
- Comprobar el funcionamiento de la heurística propuesta en un caso real.

El trabajo consta de 2 volúmenes:

Volumen I:

- Memoria.
- Anexo I: Experiencia computacional. Resumen de las pruebas (carpetas 1 y 2).
- Anexo III: Aplicación al caso Rieju
- Anexo V: CD-ROM.

Volumen II:

- Anexo II: Experiencia computacional. Soluciones de los juegos de datos(carpetas 1 y 2).
- Anexo IV:. Listados del programa.

La memoria se ha dividido en 9 capítulos: 1) Introducción al equilibrado de líneas de montaje; 2) El problema SALB-E; 3) Métodos de resolución; 4) Algoritmos ACO; 5) Aplicación de los algoritmos ACO a la resolución del SALBP-E; 6)Experiencia computacional; 7) Conclusiones; 8) Líneas futuras de investigación y 9) Presupuesto.

El apartado 1 pretende introducir la problemática del equilibrado de líneas de montaje, así como los conceptos necesarios para su comprensión. En el segundo apartado se describe detalladamente el modelo matemático que se utilizará para modelizar el SALPB-E así como las posibles interpretaciones de su función objetivo. En el tercer apartado se exponen procedimientos de resolución para el equilibrado de líneas de montaje. El capítulo 4 está dedicado en su totalidad a presentar los algoritmos ACO. En el capítulo 5 se proponen dos métodos heurísticos basados en los algoritmos ACO. El capítulo 6 resume los resultados de la experiencia computacional. En los capítulos 7 y 8 se presentan las conclusiones del trabajo y las futuras líneas de investigación. Y finalmente, en el capítulo 9 se describe el presupuesto total del proyecto, desglosado en sus partidas más importantes.

CAPÍTULO 1: ***INTRODUCCIÓN AL EQUILIBRADO DE LÍNEAS DE MONTAJE***

Este capítulo contiene una introducción a los sistemas de producción que utilizan las denominadas líneas de producción o montaje. Además, se presenta uno de los principales problemas inherentes a este tipo de sistemas productivos: el equilibrado de líneas de montaje, que junto con el problema de secuenciación de unidades, tienen como objetivo optimizar la eficiencia de estos sistemas productivos.

1.1 CONTEXTO

Actualmente, la división del trabajo es un factor muy importante dentro de la industria, especialmente en aquellas empresas que producen grandes cantidades de productos con un elevado grado de homogeneidad (e.g. electrodomésticos, automóviles, electrónica de consumo, etc.).

Un caso muy habitual está representado por aquellas empresas que fabrican diferentes versiones de un mismo producto estandarizado, al que se le pueden añadir diferentes componentes y/o equipamiento opcional, en una misma línea de montaje.

Estas empresas se caracterizan por utilizar una distribución en planta orientada al producto. En este tipo de distribución es fundamental la ordenación de los puestos de trabajo, colocándose uno a continuación del otro en el orden en el que se suceden las operaciones a realizar, moviéndose el producto de un punto a otro.

El flujo de productos semielaborados va pasando por una cadena o línea de estaciones (maquinas u operarios), ya sea de forma continua o intermitente. En cada una de las estaciones, los productos reciben una cierta de cantidad de trabajo, y la suma de todos los elementos de trabajo recibidos en cada estación corresponde al total de operaciones que requiere el producto.

En este tipo de sistemas de producción se deben tomar básicamente dos tipos de decisiones:

- El *problema de equilibrado de la línea* consiste en subdividir el trabajo de montaje (de cada modelo) entre las diferentes estaciones de trabajo para que el personal y los equipos sean utilizados de la forma más ajustada y eficiente posible a lo largo del proceso, y de forma que se cumpla la tasa de producción deseada en la línea. Este tipo de problema debe ser revisado a medio plazo.

- En caso de líneas de montaje mixtas (apartado 1.3.1) también se debe decidir *la secuencia* de productos o modelos que deben introducirse en la cadena para cumplir con la demanda y optimizar los recursos utilizados. Obviamente, este problema está directamente relacionado con el equilibrado de la línea, no obstante, a diferencia del primero, este tipo de decisión es a corto plazo, y por lo tanto, se debe tomar con mayor frecuencia.

Los problemas anteriores pertenecen a un conjunto de problemas combinatorios discretos catalogados como *NP-hard*. La resolución *óptima* de este tipo de problemas requiere un consumo de recursos computacionales (memoria y tiempo) excesivo. Por ello, en muchos casos se opta por procedimientos heurísticos, mucho más rápidos y flexibles, que no pueden garantizar soluciones óptimas, pero que proporcionan soluciones de gran calidad con un consumo de recursos aceptable.

El equilibrado de líneas de montaje pretende, pues, optimizar los recursos de los sistemas productivos orientados al producto, los cuales cada vez están más sujetos a frecuentes modificaciones. La utilización de herramientas eficaces y potentes para su resolución resultará

muy importante a la hora de implantar filosofías productivas como el JIT, ya que dotarán al sistema de gran flexibilidad.

1.2 LÍNEAS DE MONTAJE O PRODUCCIÓN

1.2.1 Sistemas de producción orientados al producto

Una característica importante de la industria manufacturera es el diseño de los procesos de producción. Como ya se ha comentado, la distribución en planta orientada al producto se adopta cuando la producción de los productos es en masa o en series de gran tamaño, y está organizada de forma continua o repetitiva. El caso más característico es el de las cadenas de montaje.

Las principales ventajas de este tipo de distribución son:

- La eficiencia del sistema es muy alta y los tiempos de producción son pequeños.
- El stock de productos en curso se mantiene muy reducido.
- El flujo de material es muy regular y se puede controlar de forma simple.
- Apenas es necesario material de logística interna para mover los productos semielaborados dentro de la planta, ya que dichos productos o piezas se transfieren de forma mecánica gracias a sistemas como por ejemplo cintas transportadoras.
- El espacio de la planta requerido se disminuye debido al menor espacio necesario para el almacenamiento y movimiento de material.
- Debido a la estricta división de las tareas, se requieren operarios menos cualificados y tiempos de aprendizaje menores.

Sin embargo, también existen desventajas que restringen la aplicación de este tipo de sistemas en algunas industrias:

- La instalación de estos sistemas requiere grandes inversiones de capital, especialmente si es necesario equipamiento automatizado.
- La satisfacción de los empleados con su trabajo normalmente es baja, sobre todo si el grado de especialización es elevado debido a tareas muy simples y monótonas. Todo esto conduce a un elevado absentismo laboral y la necesidad de rotación entre los operarios.
- Debido al elevado grado de especialización, los sistemas de producción basados en líneas de montaje pueden carecer en ciertos momentos de la flexibilidad necesaria para aquellas industrias que manufacturan productos con un ciclo de vida relativamente reducido.
- El mantenimiento y las reparaciones son las cuestiones más críticas en este tipo de sistemas, ya que cualquier anomalía, ya sean averías puntuales de alguna máquina o la falta de material, puede provocar la parada de todo el sistema.

- Los controles de calidad se deben integrar dentro de la misma línea en estaciones de inspección, ya que si ocurre algún fallo toda la línea queda afectada.

En resumen, y después de ver todas las ventajas y desventajas anteriores se puede concluir que este tipo de sistema de producción se pueden instalar cuando se cumplen las siguientes condiciones: productos estandarizados, volumen de producción elevado, demanda de productos estable y suministro de material continuo.

1.2.2 Líneas de montaje. Descripción

Una línea de producción o montaje consiste en una secuencia ordenada de tareas o elementos de trabajo. Cada tarea es un conjunto de operaciones elementales de un proceso productivo que deben hacerse a la vez (necesitan una misma herramienta, utilizan una misma guía o elemento de instalación, etc.). Por lo tanto, las tareas no pueden subdividirse y deben ser asignadas a una misma estación, atendida por un operario, un equipo de operarios, o incluso por un robot. Cada estación, por tanto, tendrá asignado un cierto subconjunto de operaciones, y la unión de todos estos subprocesos constituirá el proceso completo.

Las unidades del producto, a medida que se elaboran, se transfieren de una estación a otra de forma ordenada. Las unidades empiezan a elaborarse en la primera estación y salen terminadas, o en el grado de elaboración que corresponda, por la última estación. La trayectoria de los materiales puede tener formas diversas, pero lo esencial es que no hay retrocesos, es decir, un producto semielaborado situado en una estación, no puede pasar a otra situada aguas abajo.

El flujo de productos semielaborados va pasando por la cadena o línea de estaciones (maquinas u operarios), ya sea de forma continua o intermitente. En cada una de las estaciones, los productos reciben una cierta de cantidad de trabajo, y la suma de todos los elementos de trabajo recibidos en cada estación corresponde al total de operaciones que requiere el producto.

1.3 TIPOS DE LÍNEAS DE MONTAJE

1.3.1 Número de productos

Una posible clasificación de los tipos de línea hace referencia a los productos que se fabrican en ella. Según C*ompanys y Corominas (1994)* se pueden diferenciar básicamente tres tipos de líneas: *(1) monomodelo,* donde sólo se fabrica un tipo de producto, *(2) multimodelo,* donde se producen varios tipos de productos, pero su producción se realiza por lotes. Y finalmente, las líneas *(3) mixtas*, donde se producen diferentes tipos de productos indistintamente. Este último caso es muy común cuando los productos que se deben fabricar en la línea tiende a tener tareas y diagramas de precedencia similares.

1.3.2 Líneas con buffer entre estaciones

En las *líneas "sin buffer"* todas las estaciones disponen de la misma cantidad de tiempo (tiempo de ciclo) para realizar todas las tareas asignas a la misma. En cambio, en las *líneas "con buffer"* las estaciones se descomponen en dos partes: un buffer limitado donde se almacenan las piezas que vienen de la estación anterior, y la parte de la estación donde se realizan las tareas. Por lo tanto, en este tipo de línea el tiempo concedido a cada estación puede ser variable, lo cual dota al sistema de cierta flexibilidad. Una estación sólo quedará bloqueada si el buffer de la siguiente estación está lleno.

1.3.3 Variabilidad de los tiempos de las tareas

En función de la naturaleza de las tareas y los operarios, los tiempos de éstas pueden variar más o menos. En el caso de tareas muy simples la variación de los tiempos de operación no debe ser muy grande, en cambio, si las tareas son complejas y sensibles a posibles fallos esta variación aumenta. Esta variación es especialmente evidente en el caso del trabajo humano, ya que los tiempos de las tareas están sujetos a factores físicos, psíquicos y sociales.

Cuando se pueda justificar que esta variación es lo suficientemente pequeña se podrá asumir la hipótesis de que los *tiempos de las operaciones son deterministas*. Si por el contrario no se puede hacer esta hipótesis, se deberá hacer un estudio de la variabilidad de dichos tiempos y asumir que los *tiempos de operaciones son estocásticos*.

1.3.4 Tipos de restricciones

Cuando se diseña una línea de montaje se deben tener en cuenta todas las posibles restricciones, a parte de las de precedencia. Estas restricciones pueden ser de diferentes tipos:

- Relaciones de precedencia.
- Incompatibilidades entre tareas.
- Incompatibilidades entre tareas y estaciones.

- Complementariedad (dos tareas han de ser asignadas en la misma estación).

Más información sobre los diferente tipos de restricciones y su modelización mediante programación lineal entera puede encontrarse resumida en *Valero (1991)* y *Ferrer (2000)*.

1.3.5 Distribución en planta

Una línea de montaje puede presentar básicamente dos disposiciones diferentes, una disposición lineal, o una disposición en forma de U. Dependiendo de las características o limitaciones de la planta, también se pueden implementar otras formas que resultan de la combinación de las anteriores. En *Domínguez Machuca et al (1995)* se describen las más habituales, y que aparecen en la figura 1.3.

En la *disposición lineal*, las estaciones están colocadas en una línea recta. En ella, los operarios se pueden desplazar a lo largo de la misma. Sus principales inconvenientes son la pérdida de tiempo productivo cuando los operarios se desplazan desde la última tarea asignada a la primera, y por otro lado, la falta de comunicación entre operarios, sobre todo en líneas de grandes dimensiones.

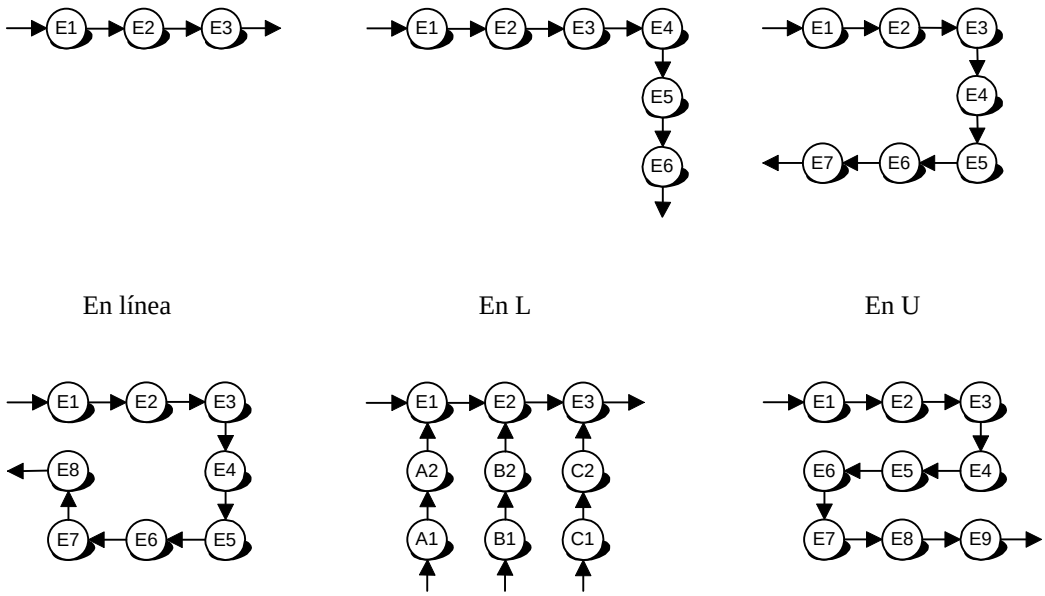


Figura 1.3

Las líneas con una *disposición en forma de U* mejoran estos dos aspectos. En primer lugar, se reduce las distancias entre operarios, ya que la entrada y la salida de la cadena estarán más cerca (en el caso ideal, la entrada y la salida de la cadena será controlada por un mismo operario). De esta forma, resulta más fácil controlar el ritmo de producción, y se acelera la reacción ante la posible aparición de problemas. Otra ventaja de este sistema es la mayor flexibilidad a la hora de calcular el tiempo de ciclo, ya que los operarios podrán acceder a un

mayor número de combinaciones de tareas. Todo ello quedará reflejado en un aumento de la eficiencia de la línea.

1.4 EL PROBLEMA DE EQUILIBRADO DE UNA LÍNEA

1.4.1 Conceptos generales

El problema general del equilibrado de una línea de montaje se suele plantear después del diseño y montaje de la propia línea. Este diseño fijará varios parámetros del problema como pueden ser los tiempos de las tareas y el diagrama de precedencias. Sin embargo, en algunas ocasiones es conveniente revisar el diseño de la línea (e.g. rediseñando tareas, precedencias, asignaciones de herramientas, etc.) para aumentar su eficiencia.

En la primera etapa de montaje también se fijan los parámetros de capacidad, número máximo de estaciones, tipo de las estaciones, coste de las estaciones, categorías de personal, etc. Estas decisiones, pues, se deben de tomar al inicio de la fabricación de un producto y deberían ser actualizadas a medio/largo plazo.

Las decisiones sobre el equilibrado propiamente dicho se refieren a la posibilidad de dividir el flujo de trabajo lo suficiente y asignar las diferentes tareas elementales (no fraccionables) en las que se ha dividido el proceso, a las distintas estaciones de trabajo. Estas tareas elementales se han de ejecutar sin interrupción, y por consiguiente, en una misma estación para que el personal y los equipos sean utilizados de la forma más eficiente posible a lo largo de toda la línea.

En el caso más frecuente de que una de las operaciones del proceso requiera más tiempo para ser ejecutada que las demás, se convertirá en lo que se conoce como un cuello de botella, que restringirá la capacidad de todo el proceso. Uno de los objetivos del equilibrado de la cadena es ajustar y equilibrar las cargas de todas las estaciones de la línea de tal forma que se eviten estos picos de carga en las estaciones.

En el caso de líneas de montaje sin buffer el tiempo del que dispone cada estación para realizar las tareas que se le han asignado es limitado e igual para todas las estaciones. Este tiempo límite se denomina *tiempo de ciclo* TC. Cada uno de los productos que circulan por la línea llega a una estación y otro la abandonan cada TC unidades de tiempo, respectivamente. Debido a la restricción del tiempo de ciclo las líneas sin buffer se caracterizan por una *tasa de producción constante* (unidades producidas por unidad de tiempo). Este parámetro es necesario para poder plantear el problema, ya sea un dato, o una incógnita del problema. El caso más corriente es fijar la tasa de producción impuesta por la demanda, y a partir de ella calcular el tiempo de ciclo:

$$TC = \frac{tiempo\ de\ producción\ \ disponible}{unidades\ \ producidas}$$

1.4.2 Objetivos del equilibrado

La instalación de una línea de montaje es una decisión a largo plazo y requiere una gran inversión de capital. Por ello, es muy importante que el sistema funcione de la manera más eficiente posible. Como ya se ha comentado en los apartados anteriores, la línea se debe

equilibrar periódicamente debido a posibles cambios en el proceso o en el programa de producción. Debido a los efectos a medio plazo de las decisiones de equilibrado, los objetivos que se quieren conseguir con las mismas se deben de escoger cuidadosamente y en función de los objetivos de la empresa.

El principal objetivo del equilibrado es ajustar y equilibrar las cargas de las estaciones. Sin embargo, dependiendo del problema, este objetivo puede cambiar, o simplemente complementarse con otros índices de eficiencia.

Los dos objetivos principales cuando se equilibra una línea son: minimizar el número de estaciones y/o minimizar el tiempo de ciclo. Si se combinan estos dos objetivos se puede obtener el mínimo tiempo de ciclo una vez fijado el número de estaciones mínimo.

Si se fija un tiempo de ciclo, y se minimiza el número de estaciones, también se está consiguiendo automáticamente que se minimice el tiempo ocioso o tiempo muerto, que se define como el tiempo improductivo total en la fabricación de una unidad de producto para el conjunto de todas las estaciones de trabajo. La relación entre el tiempo "teóricamente" requerido para la fabricación del producto, y el tiempo realmente necesario o empleado definirá la eficiencia de la cadena (apartado 2.2).

A parte de estos dos objetivos básicos, se pueden definir otros igualmente válidos:

- Minimizar el producto del número de estaciones por el tiempo de ciclo.
- Minimizar el porcentaje de tiempo muerto medio de todas las estaciones.
- Minimizar la suma ponderada del número de estaciones y el tiempo de ciclo.
- Minimizar la suma de las discrepancias entre las cargas de trabajo de cada estación y el tiempo de ciclo global (*Plans y Corominas, 1998*).
- Minimizar el número de operarios inexpertos (*Riera 1997*).

En el caso de líneas mixtas, donde varios productos se fabrican en la misma línea, la eficiencia de la línea también depende de la secuencia de productos que se introduce en la línea, ya que el tiempo de ensamblaje necesario para los diferentes productos puede variar sensiblemente. Por lo tanto, en este tipo de líneas también se debe considerar el problema de secuenciación.

1.5 CLASIFICACIÓN DE LOS PROBLEMAS DE EQUILIBRADO

Existen diferentes criterios para clasificar los problemas de equilibrado. A continuación de describe la clasificación propuesta por Baybars (extraída de *Ferrer (2000)*), ya que fue una de las primeras clasificaciones realizadas.

Según Baybars se pueden distinguir dos grandes problemas de equilibrado:

SALBP (*Simple Assembly Line Balancing Problem*):

Todos los problemas que se incluyen dentro de este grupo deben cumplir las siguientes condiciones:

- Todos los parámetros se conocen con certidumbre.
- Una tarea sólo puede asignarse a una estación.
- Existen secuencias de precedencias a respetar.
- Todas las tareas se deben asignar.
- Los costes asociados a cada estación son iguales.
- El tiempo que se requiere para realizar una tarea en una estación es determinista, e independiente de la estación donde se realiza.
- Cualquier tarea se puede realizar en cualquier estación.
- Se aplica en líneas de montaje con un único producto.

Dentro de este grupo se puede distinguir tres problemas particulares:

- *SALBP-1*:el objetivo es minimizar el número de estaciones dado el TC.
- *SALBP-2*: el objetivo es minimizar el TC fijado el número de estaciones.
- *SALBP-E*: el objetivo es minimizar el tiempo muerto sujeto a la variación tanto del número de estaciones, como del tiempo de ciclo.

GALBP (*General Assembly Line Balancing Problem*):

En este grupo se encuentran todos aquellos problemas con alguna relajación de alguna de las condiciones anteriores (e.g. existencia de estaciones en paralelo, elaboración de diferentes productos en las líneas, incompatibilidades entre tareas, etc.).

La clasificación de Baybars, pese a no ser demasiado minuciosa, es suficiente para hacerse una idea de los diferentes problemas que pueden encontrarse a la hora de equilibrar una línea de montaje. Sin embargo, no se hace ninguna distinción entre los diferentes tipos de línea descritos en el apartado 1.3. En *Scholl (1999)* se proponen dos nuevos tipos de problema:

UALBP (*U-line Assembly Line Balancing Problem*):

Este problema considera el caso de las líneas en forma de U monomodelo, donde los operarios son colocados formando una U lo suficientemente estrecha para que puedan trabajar en ambos lados de dicha U. De esta forma, se tendrán que considerar modificaciones en las restricciones de precedencias. De forma análoga al SALBP, se pueden considerar problemas particulares.

MALBP (*Mixed Assembly Line Balancing Problem*):

El problema de equilibrado de líneas de montaje multimodelo tiene en cuenta los tiempos de las tareas para cada modelo que se fabrica dentro de la línea, por lo tanto, estará relacionado directamente con el problema de secuenciación de unidades en las líneas de montaje. El objetivo del MALBP será doble, por un lado decidir el equilibrado de la línea, pero al mismo tiempo decidir la secuencia o *mix* de productos, de tal forma que dicha combinación (equilibrado y secuencia) minimice las sobrecargas y el riesgo de interrupciones en la línea.

CAPÍTULO 2: ***EL PROBLEMA SALB-E***

En este capítulo se presenta el problema de equilibrado de líneas de montaje SALBP-E (Simple Assembly Line Balancing Problem). En primer lugar se explican todos los conceptos y notaciones necesarias para la comprensión del problema, a continuación se describe el problema y sus principales características, y finalmente, se presenta todo lo explicado en el capítulo mediante un modelo matemático.

2.1 TERMINOLOGIA BÁSICA

2.1.1 Notación

En la tabla 2.1 se resumen los términos y notaciones necesarias para la descripción matemática de los problemas de equilibrado de líneas de montaje, y en particular del problema SALB-E.

n	*Número de tareas*
V	*Conjunto de todas las tareas (= {1,..., n})*
i	*Índice de las tareas (j = 1,...,n)*
TC	*Tiempo de ciclo*
p	*Tasa de producción (= 1/TC)*
m	*Número de estaciones*
j	*Índice de las estaciones (j = 1,...,m)*
t_i	*Tiempo necesario para realizar la tarea i*
t_{max}	*Tiempo de la tarea de mayor duración (= max { t_j / j = 1,...,n })*
t_{min}	*Tiempo de la tarea de menor duración (= min { t_j / j = 1,...,n })*
t_{sum}	*Suma de los tiempos de todas las tareas (= $\Sigma_j t_j$)*
A	*Conjunto de relaciones de precedencias inmediatas (= {(i,j) / i ∈ V y j∈ IS_i})*
IP_i	*Conjunto de tareas predecesoras inmediatas de la tarea i*
P_i	*Conjunto de tareas predecesoras de la tarea i*
IS_i	*Conjunto de tareas sucesoras inmediatas de la tarea i*
S_i	*Conjunto de tareas sucesoras de la tarea i*
S_k	*Carga asociada a la estación k, conjunto de tareas asignadas a la estación k*
$t(S_k)$	*Tiempo que necesita la estación k para realizar la carga S_k*
$j_{min(i)}$	*Cota inferior de la estación a la cual puede ser asignada la tarea i*
$j_{max(i)}$	*Cota superior de la estación a la cual puede ser asignada la tarea i*

Tabla 2.1: Notaciones

2.1.2 Grafo de precedencias

Un grafo de precedencias *G = (V , A , t)* es un grafo sin circuitos que permite representar una relación estricta de orden entre los nudos, con un conjunto *V* de nodos y un conjunto *A* de arcos. Los nodos simbolizan las tareas, y los arcos representan la relaciones directas de precedencias entre las tareas. Si una tarea *i* debe ser completada antes que una tarea *j* , se dirá que la tarea *i* es predecesora de la tarea *j*, o lo que es lo mismo, que la tarea *j* es sucesora de la tarea *i*.

Una relación de precedencia entre dos tareas *i* y *j* es *directa* (inmediata) si ninguna de las tareas subsiguientes de *i* es predecesora de *j*. El conjunto de tareas sucesoras inmediatas de la tarea *i* se notará como IS_i. El peso del nodo *j* es t_j, y representa el tiempo de operación de la tarea *j*. Un nodo correspondiente a una tarea sin ningún predecesor (sucesor) se denominará *nodo inicial* (*nodo final*).

El grafo inverso de G se notará G^r y se obtendrá invirtiendo la orientación de todos los arcos de G. Los nodos finales se convierten en nodos iniciales, los predecesores se convierten en sucesores, y viceversa.

2.1.3 Asignación de tareas

Una división del conjunto V de todas las tareas en m subconjuntos independientes S_k (cargas asignadas a cada una de las m estaciones) con $k= 1,...,m$ se denomina asignación de tareas a las estaciones, si una tarea $i \in S_h$ y $j \in S_k$ se debe cumplir la relación $h \leq k$ para cada arco $(i,j) \in A$. Esto se traduce en dos ideas básicas del problema de equilibrado SALB, por una parte una tarea se asigna sólo a una estación, y por otra, los sucesores de la tarea i no pueden ser asignados a una estación anterior a la estación de i.

Otra condición para que la asignación sea factible es que el tiempo de ninguna carga ($t(S_k)$) asignada a alguna estación supere el límite impuesto por el tiempo de ciclo TC. Una asignación factible de tareas es el denominado *equilibrado de la línea*. Una condición necesaria para que exista un equilibrado es que $t_{max} \leq TC$ debido a la indivisibilidad de las tareas.

2.1.4 Capacidad de la línea

Se denomina capacidad de la línea T ($T = m \cdot c$) al tiempo total disponible para el ensamblaje de cada pieza en la línea de montaje, esta T representa la máxima capacidad de la línea. Obviamente, la capacidad de la línea debe ser mayor o igual que la suma de tiempos de todas las tareas (t_{sum}).

2.1.5 Eficiencia de la línea

La capacidad de utilización de una línea de montaje se mide por su índice de eficiencia: $E=t_{sum} / T$. La parte de capacidad de la línea no utilizada, también llamado tiempo muerto es igual a $(T - t_{sum})$. El tiempo muerto de una estación k con carga S_k se define como $I_k = TC-t(S_k)$.

2.2 DESCRIPCIÓN DEL PROBLEMA SALB-E

El problema del SALBP-E es una extensión del problema SALB, y consiste en encontrar una combinación de tiempo de ciclo (*TC*) y número de estaciones (*m*), además del correspondiente equilibrado de la línea, donde se asignan todas las tareas a las *m* estaciones cumpliendo la restricción de *TC* ($t(S_k) \leq \mathrm{TC} \ \forall k$) para cada estación, tal que se maximize la eficiencia de la línea $E = t_{sum} / T$.

El problema sólo tiene sentido si existe una cota del número de estaciones mínimo superior a 1, si no existe esta restricción, la máxima eficiencia de la línea se obtiene escogiendo $m=1$ y $TC = t_{sum}$. El significado de esa cota inferior esta relacionado con el grado de especialización mínimo requerido en la línea. Por otra parte, una cota superior del tiempo de ciclo c_{max} puede ser impuesta por la tasa de producción deseada. Debido a la condición $m \cdot TC \geq t_{sum}$, se puede extraer una cota del número mínimo de estaciones m_{min} ($m_{min} = \lceil t_{sum} / TC_{\max} \rceil$).

A parte de las restricciones anteriores, el SALBP-E también puede estar restringido por un número de estaciones máximo resultante de limitaciones de espacio. Así como, por un tiempo de ciclo mínimo ($TC \geq t_{max}$). En resumen, las restricciones de un problema particular de SALBP-E están definido por dos intervalos [m_{min} , m_{max}] y [$TC_{\min}$, $TC_{\max}$].

Debido a que la eficiencia *E* depende de la constante t_{sum} y del término no lineal $m \cdot TC$, la eficiencia de la línea se maximiza minimizando el producto $m \cdot TC$. Por lo tanto, la maximización de la eficiencia de la línea también puede ser entendida como:

- Minimizar el tiempo muerto total de la línea:

$$Tiempo\ muerto = TC \cdot m - \sum_j t_j \qquad (donde \sum_j t_j = cte)$$

- Minimizar el coste por unidad de producto fabricada:

$$Coste\left[\frac{\$}{unidad}\right] = TC \cdot m = TC\left[\frac{tiempo}{unidad}\right] \cdot m[operarios] \cdot k\left[\frac{\$}{tiempo \cdot operario}\right]$$

(donde k=cte es el coste anual por operario)

2.3 SIMETRÍA DEL PROBLEMA

Dado un ejemplar de un problema SALBP-E definido por un grafo de precedencias, se denomina ejemplar inverso al problema que se obtiene del original o directo invirtiendo en grafo de precedencias. Dada una solución del problema obtenida con el ejemplar directo (distribución de las tareas en las diferentes estaciones), en el ejemplar inverso dicha solución también es factible si se invierte el orden de las estaciones. Ambos ejemplares, directo e inverso, tienen la misma solución óptima, y una permutación óptima de uno de ellos puede deducirse de directamente por inversión de una permutación óptima para el otro.

2.4 COTAS DE LA FUNCIÓN OBJETIVO

2.4.1 Cotas inferiores para el SALBP-E

Una cota inferior para el SALBP-E se puede obtener básicamente de dos maneras:

- Sea $CI_{SALBP\text{-}1}(TC)$ una cota inferior del problema SALBP-1 para el tiempo de ciclo *TC*. Si se conoce todo el intervalo de tiempos de ciclo factibles [TC_{min} , TC_{max}], una cota del SALBP-E es igual a:

$$CI_{SALBP-E} = \min\left\{ CI_{SALBP-1}(TC) \cdot TC \;\middle|\; TC \in \left[TC_{\min}, TC_{\max} \right] \right\}$$

- Análogamente, si lo que se conoce a priori es el número máximo y mínimo de estaciones (m_{min} , m_{max}), se puede obtener una cota del SALBP-E aprovechando las cotas inferiores del SALBP-2 ($CI_{SALBP\text{-}2}(m)$) de la siguiente forma:

$$CI_{SALBP-E} = \min\left\{ CI_{SALBP-2}(m) \cdot m \;\middle|\; m \in \left[m_{\min}, m_{\max} \right] \right\}$$

El objetivo de este trabajo no es presentar todas estas cotas, no obstante, en *Scholl 1999* se encuentran explicadas detalladamente diferentes cotas del problema SALB-1 y 2.

2.4.2 Cotas superiores para el SALBP-E

Al igual que en el resto de problemas SALB, cualquier solución factible del problema, en este caso el SALBP-E, es a su vez una cota superior de la solución óptima.

2.5 MODELO MATEMÁTICO

Como ya se ha comentado, la función objetivo del SALBP-E no es lineal, por ello, el modelo MILP que se utiliza para resolver el SALBP-E incluye la linealización de la función objetivo propuesta por *Plans, Corominas (2000)*:

Datos iniciales:

n : número de tareas.
d_i *: duración de cada tarea (i=1,..,m).*
m_{min} *: número mínimo de estaciones.*
m_{max} *: número máximo de estaciones.*
IS_i *Conjunto de sucesores inmediato, tal que (i,j)∈P significa que la tarea i se debe realizar antes que la tarea j.*

A partir de estos datos anteriores se pueden calcular los siguientes:

T_{min} *: Cota inferior del tiempo de ciclo.*
T_{max} *: Cota superior del tiempo de ciclo.*
m' : Cota superior del número máximo de tareas que pueden ser asignadas a una estación.

Variables:

n : número de estaciones.
TC : tiempo de ciclo.
$z = n \cdot t$ *:* función objetivo

$x_{i,j} \in \{0,1\}$: Donde la variable $x_{i,j}$ =1 si la tarea *i* es asignada a la estación *j* ($i=1,..,m$; $j=j_{min}(i),..,j_{max}(i)$)

$y_j \in \{0,1\}$: Donde la variable y_j =1 indica la existencia de la estación *j* ($j=m_{min},..,m_{max}$)

Si tenemos en cuenta que $(y_j = 1) \Rightarrow (z \geq j \cdot t)$, se puede linealizar la función objetivo introduciendo las restricciones $j \cdot t - z \leq M_j \cdot (1 - y_j)$, donde M_j es una cota superior de $j \cdot t - z$ (donde $M_j = j \cdot TC - \sum_{i=1}^{m} t_i$), ya que $\sum_{i=1}^{m} t_i$ es una cota inferior de *z*).

Con todo lo anterior ya se puede formular el PLE para el SALBP-E:

[MIN]z

s.a.

$$\sum_{j=j_{\min}(i)}^{j=j_{\max}(i)} x_{ij} = 1 \quad (i = 1,..,m) \quad (1)$$

$$\sum_{i \in I_j} t_i \cdot x_{ij} \leq TC \quad (j = 1,..,m_{\max}) \quad (2)$$

$$\sum_{i \in I_j} x_{ij} \leq m' \cdot y_j \quad (j = m_{\min} + 1,..,m_{\max}) \quad (3)$$

$$m_{\min} \cdot TC - z \leq 0 \qquad (4')$$

$$j \cdot t - z \leq M_j \cdot (1 - y_j) \quad (j = m_{\min} + 1,..,m_{\max}) \quad (4'')$$

$$\sum_{j=j_{\min}(i)}^{j=j_{\max}(i)} j \cdot x_{ij} = \sum_{j=j_{\min}(k)}^{j=j_{\max}(k)} j \cdot x_{kj} \quad \forall (i,k) \in IP \quad (5)$$

$$y_{j+1} \leq y_j \quad (j = m_{\min} + 1,..,m_{\max} - 1) \quad (6)$$

$$x_{ij} \in \{0,1\} \quad \forall i, j \quad ; \quad y_j \in \{0,1\} \quad \forall j \quad (7)$$

La restricción (1) garantiza que todas las tareas se asignen a una estación; (2) impone que ninguna estación supere el tiempo de ciclo máximo; (3) fuerza que una estación se abra cuando se le asigna alguna tarea; (4') y (4'') sirven para linealizar la función (5) corresponde a las restricciones o ligaduras de precedencias; finalmente, (6) impone que las estaciones se abran en orden.

La forma de calcular las cotas T_{min} , T_{max}, el número de estaciones mínimo ($j_{min(i)}$) y máximo ($j_{max(i)}$), y el número máximo de tareas en una estación se describe en el apartado 3.1, ya que su misión es la de agilizar la resolución del problema.

CAPÍTULO 3: ***MÉTODOS DE RESOLUCIÓN***

En este capítulo se describen los diferentes métodos que se utilizan para resolver el problema de equilibrado en general, y en particular de su versión SALBP-E. LA primera parte se dedica a presentar los métodos exactos y sus inconvenientes, y en la segunda parte, se explican diferentes métodos heurísticos propuestos para la resolución de los problemas de equilibrado.

3.1 COMPLEJIDAD

La complejidad computacional de un problema depende del tiempo de cálculo necesario para su resolución óptima, así como de los recursos de memoria que utiliza. Para clasificar los diferentes problemas matemáticos que existen se definen básicamente dos tipos: P y NP. Los problemas pertenecientes a la clase **P** necesitan un tiempo de resolución polinomial, es decir, el tiempo de resolución se puede acotar mediante un polinomio que será función de los parámetros del problema. Estos problemas se pueden resolver de forma eficiente mediante la utilización de algoritmos. Sin embargo, la mayoría de problemas de optimización pertenecen la clase **NP**. Para este tipo de problemas no se conoce ningún algoritmo polinomial de resolución.

Dentro de la clase **NP** se define la clase **NP-hard**, un tipo de problema especialmente difícil de resolver, y al cual pertenece el SALBP, por lo que una vez se demuestra que un problema pertenece a la clase NP-hard suele plantearse la posibilidad de resolver el problema de forma heurística y no óptima.

3.2 MÉTODOS EXACTOS

En lo referente a los métodos exactos, existen básicamente dos procedimientos para la resolución del SALBP: los procedimientos basados en la programación dinámica y los procedimientos de exploración dirigida, también denominados *B&B* (*Branch and Bound*).

La primera formalización de un modelo de programación dinámica para resolver el problema del equilibrado fue hecha por Held, Karp y Sharesian. Las conclusiones de su trabajo fueron positivas para problemas reducidos, pero inviables cuando aumentaba la dimensión del problema.

El método *B&B*, por su parte, se basa en una estrategia de fijación de variables (del programa lineal utilizado para modelizar el problema), ramificando y acotando las diferentes soluciones en curso. El proceso consiste en forzar sistemática y progresivamente que las variables binarias/enteras pertenezcan o no a la solución, de tal forma que se obtengan colecciones de soluciones posibles cuya calidad pueda apreciarse a través de una cota. El resultado es un procedimiento de exploración arborescente, basado en la partición y acotación de cada una de las ramas del grafo o árbol.

La cota de una rama se puede definir como el mejor valor posible de la función objetivo que se puede conseguir por esa rama eliminando alguna de las restricciones del problema. Es la herramienta básica de este tipo de procedimientos, ya que gracias a ella, no es necesario explorar todas las posibles soluciones del problema combinatorio.

En los paquetes informáticos como pueden ser CPLEX o LINDO, para calcular esta cota se resuelve el programa lineal entero suponiendo que las variables no son enteras, sino reales, con lo cual, para cada cota, se tiene que resolver un PL (Programa Lineal) mediante el método *Simplex* (método recurrente utilizado para resolver PL's).

De todo lo explicado, se puede deducir la importancia de partir con una solución heurística inicial, y calcular la cota de la forma más ajustada posible. Ya que, una vez se conoce una solución factible del problema, no es necesario seguir explorando aquellas ramas con una cota peor.

La estrategia de exploración de las ramas es otro parámetro básico a la hora de diseñar un procedimiento de este tipo. Se debe utilizar toda la información que se pueda extraer del problema que se pretende resolver, tanto para calcular una cota lo más ajustada posible, como para decidir el orden en que se van a explorar las ramas del árbol.

Para que el proceso concluya con la solución óptima, será necesario haber encontrado al menos una solución factible del problema, y que el resto de ramas pendientes de exploración tengan una cota peor o igual que la mejor solución hallada hasta el momento.

3.3 INCONVENIENTES DE LOS MÉTODOS EXACTOS

Los métodos exactos que se utilizan para resolver problemas combinatorios suelen requerir tiempos de resolución muy largos. Un problema de *n* variables binarias (nº tareas) implica que el "árbol" que se generará durante el Branch & Bound tendrá 2^n ramas, es decir, la relación entre el tiempo de resolución y el número de variables es exponencial, si el número de variables es muy elevado el tiempo de resolución puede ser excesivo.

Otro problema asociado a estos procedimientos de ramificación y acotación es, una vez encontrada la solución óptima en alguno de los nodos del árbol, comprobar que dicha solución es la óptima, es decir, cerrar todos lo vértices con una cota inferior a la mejor solución encontrada hasta el momento. En muchos casos, el tiempo que necesita el B&B para confirmar el óptimo es mayor que el tiempo en llegar a dicho valor óptimo. Es por ello, que a veces se opta por detener el proceso cuando la diferencia entre la mejor solución hallada hasta el momento y la mejor cota en ese instante, es menor que una tolerancia ε, definida antes de inicializar el proceso. Aunque con este método no se podrá garantizar el óptimo.

Todos estos problemas quedan reflejados en *Bautista, Bretón y Fernández (2000)*, donde se lleva a cabo una experiencia computacional en la que se intenta resolver el problema SALB-E de forma exacta con el optimizador matemático CPLEX. Las conclusiones no son muy esperanzadoras, ya que pese ha utilizar una solución inicial de gran calidad y el uso de prioridades, la mayoría de problemas de más de 25 tareas no han podido ser resueltos en el tiempo límite impuesto de 3600 segundos. En muchos casos ni tan siquiera se ha mejorado la solución inicial.

3.4 MÉTODOS HEURÍSTICOS

3.4.1 Introducción

Dado un índice de eficiencia, los procedimientos heurísticos proporcionan una solución factible, que puede o no ser la óptima, a diferencia de los procedimientos exactos, que tratan de hallar la solución que optimice dicho índice.

En algunos tarbajos como B*autista, Mateo, Ferrer, Pereira y Companys (2000)* se tienen en cuenta tres grupos: Greedy determinista, GRASP (Greedy Randomized Adaptative Search Procedure) y búsqueda de exploración de entornos (Bautista Suárez, Mateo y Companys (2000)).

Las heurísticas Greedy están basadas en el empleo de reglas de prioridad estáticas que condicionan el orden de asignación de las tareas a las estaciones. Existen una gran variedad de reglas de prioridad que combinan diferentes aspectos tales como el número de estaciones que falta por completar, el tiempo de las estaciones precedentes a dicha tarea, la duración de la tarea, etc.

Los algoritmos GRASP y GRWASP añaden a las heurísticas Greedy una componente aleatoria, es decir, las tareas serán asignadas en función de su prioridad y el azar.

Por último, se encuentran las heurísticas de mejora local o procedimientos de exploración de entornos. La característica básica de estos métodos es la definición del vecindario a explorar y el número de soluciones que se quiere explorar. Aunque tienen el inconveniente que si el vecindario se define de forma muy general, se puede perder el conocimiento sobre el problema específico, y ser simplemente una búsqueda de diferentes combinaciones basada en el azar.

Otra posible clasificación de las heurísticas, que complementa la anterior, se puede hacer según la forma en que se construye la solución. Las heurísticas directas construyen la solución paso a paso, es decir, de forma progresiva. Las heurísticas con retroceso (*backtracking*) construyen una solución con la posibilidad, en cualquier momento de la construcción, de desechar los últimos pasos, y retomar la marcha en otra dirección. Las heurísticas de mejora, a partir de una solución, buscan otra por transformaciones sucesivas. Y, finalmente, las heurísticas mixtas combinan elementos de las anteriores.

3.4.2 Método Fix and Relax

Dillenberg (ver *Plans y Corominas (2000))* se propone un método heurístico para la resolución de programas lineales denominado meta-algoritmo *fix-and-relax*. Este algoritmo está basado en un proceso iterativo de fijación de variables. En cada iteración, la condición de algunas variables enteras se relaja, y se resuelve el programa lineal. El valor de alguna de las variables que no han sido relajadas se fija a al valor óptimo del problema relajado para la siguiente iteración. El algoritmo acaba cuando todas las variables han sido fijadas. *Plans y Corominas (2000)* utilizan esta heurística para la resolución del problema SALBP-E.

3.4.3 Método de exploración de entornos

Las heurísticas de mejora local consisten en explorar soluciones que se generan a partir de otras soluciones anteriores de forma recurrente. Partiendo de una solución en curso, se genera (explícita o implícitamente) su entorno formado por soluciones vecinas, y se elige entre ellas, una nueva solución en curso, guardándose a lo largo del proceso la mejor solución posible hallada.

Por lo tanto, estos métodos de exploración de entornos vendrán caracterizados por los parámetros siguientes:

- Definición de soluciones vecinas.
- Elección de la solución inicial.
- Elección de la nueva solución en curso.
- Condición para detener el proceso.

La generación del vecindario se hace mediante una transformación parametrizada de las diferentes soluciones en curso. Para cada parámetro y solución se obtiene una nueva solución vecina, y el conjunto de soluciones vecinas constituirá todo el vecindario. En aquellas soluciones que pueden representarse mediante la concatenación de elementos (el SALBP-E pertenece a este tipo de problemas, ya que el resultado final del equilibrado no es más que el orden en que deben ser asignadas las tareas en la línea de montaje), la transformación puede consistir en intercambios binarios de elementos.

La elección de una solución inicial afectará de forma directa el resultado final, ya que al ser una mejora local, el óptimo local que encuentre estará dentro del espacio de soluciones exploradas, que dependerá, entre otras cosas, de la solución inicial. No obstante, el partir de una solución inicial con una mayor calidad no asegura que el resultado final vaya a ser mejor.

A modo de ejemplo, se comentan dos procedimientos de elección de la nueva solución en curso basados en la heurística *hill climbing*: el algoritmo exhaustivo de descenso (*AED*), y el algoritmo no exhaustivo de descenso (*ANED*).

En el *AED*, a partir de la solución en curso se generan y evalúan todos los vecinos, si el mejor de ellos es mejor que la solución en curso, éste se toma como nueva solución en curso y se reitera el procedimiento; en caso contrario, si el mejor vecino es peor o igual que la solución en curso, el procedimiento se da por terminado.

En el método *ANED*, a partir de una solución en curso se generan y evalúan según un orden, sus vecinos, si uno de ellos es mejor que la solución en curso se toma como nueva solución en curso (sin terminar la generación de los vecinos de la solución primitiva) y se prosigue aplicando el procedimiento a los vecinos de la nueva solución en curso; cuando se han generado todos los vecinos de una solución en curso sin que ninguno de ellos sea mejor, el procedimiento se da por terminado.

Como su propio nombre explica, estos procedimientos sólo pueden encontrar un óptimo local dentro del espacio de soluciones vecinas que se defina. Un aspecto muy importante de este tipo de algoritmos es el tratamiento que tienen los empates. La aceptación de soluciones vecinas de igual valor puede conducir a otro subespacio de soluciones, diferente al de la solución en curso, pudiéndose encontrar en éste un mejor óptimo local. En estos casos, se pueden producir bucles, con lo que será conveniente tomar precauciones en ese sentido.

Las diferentes formas de controlar este problema ha conducido a los denominados procedimientos de búsqueda tabú (*TS, Tabu Search*) y los algoritmos de recocido simulado (*SA, Simulated Annealing*) que se comentan en el siguiente apartado. En el primero se conserva la información de las soluciones anteriores para evitar que las soluciones en curso regresen a las mismas. El segundo utiliza el azar para aceptar o no una solución vecina de igual valor que la solución en curso. Mas información sobre el campo de la optimización heurística de entornos puede encontrarse en *Adenso (1996).*

3.4.4 Heurísticas inspiradas en fenómenos naturales

En los últimos años, uno de los campos de investigación más importantes y prometedores ha sido el de las heurísticas inspiradas en fenómenos naturales, las cuales utilizan alguna analogía con los sistemas sociales o naturales *(Colorni et al (1996))* para crear métodos heurísticos no-deterministas que permiten obtener muy buenos resultados en problemas de optimización combinatoria.

Básicamente, son dos los atributos de la naturaleza que inspiran estos algoritmos: *selección* y *mutación*. El primero recompensa los individuos (soluciones) más fuertes (de mayor calidad), y el segundo introduce el elemento aleatorio que permite el nacimiento de nuevos individuos (construcción de nuevas soluciones). En los algoritmos inspirados en la naturaleza la idea de selección equivale a la optimización, y la idea de mutación se traduce en una exploración no-determinista.

A parte de estas dos ideas básicas, estos algoritmos también poseen otras cualidades. En la lista siguiente se resumen las características más importantes que poseen los algoritmo catalogados dentro de este grupo:

- Intentan modelar (con más o menos rigor) algún fenómeno que existe en la naturaleza.
- No son deterministas.
- A menudo presentan una estructura en paralelo (multi-agente) implícita.
- Son adaptativas, es decir, tienen la capacidad de usar información en forma de *feed-back* para modificar su parámetros e incluso su modelo interno.

Gracias a estas cualidades se puede garantizar que el sistema se comporta de forma “lógica y razonable”, y que puede ser denominada “inteligente” (se entiende por inteligencia la capacidad de resolver problemas difíciles). En nuestro caso este comportamiento inteligente se traduce en la obtención de soluciones de gran calidad para problemas combinatorios.

Las ideas anteriores son los pasos básicos para la construcción de las heurísticas basadas en los fenómenos naturales. Las principales heurísticas que se pueden incluir dentro de este grupo son las siguientes:

- Algoritmos genéticos (*GA, Genetic Algorithms*)

  Los algoritmos genéticos son técnicas de búsqueda basadas en la mecánica de selección natural y la genética. Estos algoritmos se basan en un proceso de generación de "poblaciones" (conjuntos) de soluciones de forma iterativa y según unas reglas genéticas. La principales reglas son : reproducción, sobrecruzamiento y mutación.

  La reproducción consiste en obtener una nueva población a partir de aquellas variables de decisión que en la población anterior han proporcionado funciones objetivo de gran calidad. A mayor calidad, mayor probabilidad de que el valor de una variable de decisión perteneciente a dicha solución vuelva a formar parte de la nueva población de soluciones.

  En el proceso de sobrecruazamiento se generan poblaciones cuyos miembros son una combinación de genes (variables de decisión) de varios individuos (soluciones). Finalmente, el proceso de mutación consiste en alterar alguna variable de decisión de alguna solución con una probabilidad que es función de su calidad.

- Recocido simulado (*SA, Simulated Annealing*)

  El recocido simulado hace uso de conceptos originalmente descritos por la mecánica estadística, concretamente del Algoritmo de Metrópolis. Un algoritmo de recocido simulado es un métodos de búsqueda por entornos en el que el criterio de elección son las reglas de transición del algoritmo de Metrópolis.

  El algoritmo selecciona un candidato de entre los que componen el entorno de la solución actual, si el candidato es mejor que dicha solució es acepado como solución actual, en caso contrario, será aceptado con una probabilidad que decrece según aumente la diferencia entre la calidad de la solución candidata y actual. Si el candidato no es aceptado, al algoritmo selecciona aleatoriamente otro candidato y se repite el proceso.

- Búsqueda tabu (*TS, Tabu Search*)

  La meta-heurística de búsqueda tabú es una heurística de exploración de entornos en la que se imponen restricciones al proceso de exploración con el objetivo de guiar el proceso de forma inteligente, evitando la exploración de zonas del espacio de soluciones que ya han sido exploradas anteriormente.

  La restricciones pueden ser de diferentes tipos: excluyendo directamente alternativas de búsqueda o modificando la evaluación y las probabilidades de selección de dichas alternativas. El elemento principal de estos algoritmos es el uso de una memoria flexible que tenga en cuenta la recencia , la frecuencia y la calidad de los movimientos aplicados a la solución en curso.

- Redes Neuronales (*NN, Neural Networks*)

  Una Red Neuronal es un conjunto de elementos (*neuronas*) conectados entre ellos. Las conexiones entre estos elementos tienen asociado un peso y un sentido. Estas conexiones proporcionan a cada neurona una señal de entrada que es función del resto de neuronas. Del mismo modo, cada neurona producirá una salida que será función de las entradas y de sus propios parámetros internos.

  A pesar de que cada neurona computa su estado de forma local e independiente, la Red evoluciona en sucesivas iteraciones, de acuerdo a una regla de transición, hasta un estado estable en el cual ya no se producen cambios. Este estado final corresponde a la solución final.

- Algoritmos de las hormigas (algoritmos *ACO*)

  Los algoritmos ACO fueron introducidos por primera vez en *Colorni, Dorigo y Maniezzo 1991* en el denominado AS (Ant System). Su principal idea consiste en simular el comportamiento de un conjunto de agentes que cooperan entre ellos con el objetivo de resolver un problema de optimización. La principal característica es su sistema de comunicación, inspirado en el comportamiento de las hormigas. En el capítulo 4 se explica de forma detallada todo lo referente a este tipo de meta-heurísticas.

Un punto importante que afecta al diseño de las heurísticas anteriores son dos parámetros que normalmente entrarán en conflicto, por lo que se deberá buscar un equilibrio entre ambos:

- El *grado de explotación*, que se define como la cantidad de esfuerzo dirigido hacia una zona concreta del espacio de exploración (si una región se considera prometedora, se debe intentar buscar a fondo dentro de ese sub-espacio).

- El *grado de exploración*, que se define como la cantidad de esfuerzo dedicada a la exploración en diferentes regiones del espacio (a veces es conveniente escoger soluciones en curso peores que la mejor encontrada hasta el momento para poder moverse a otro sub-espacio de exploración, y quizás así poder descubrir mejores soluciones).

Otra relación a tener en cuenta es entre el esfuerzo total (en tiempo o número de iteraciones), y la eficacia de la heurística (hallar la mejor solución posible).

En general, el funcionamiento de estos procedimientos heurísticos es bastante bueno, sin embargo, se ha de tener en cuenta uno de sus principales problemas: la fijación de los parámetros de control de estos algoritmos, ya que puede ocurrir que el funcionamiento de los algoritmos dependa fuertemente de estos parámetros. Siendo corriente, que se gaste más tiempo buscando los parámetros óptimos que en el diseño del propio algoritmo y/o que el funcionamiento del algoritmo dependa fuertemente de estos parámetros.

CAPÍTULO 4: ***ALGORITMOS ACO***

En este documento se describen los denominados algoritmos de hormigas para problemas de optimización discreta. En la primera parte se explican las principales características de la meta-heurística ACO (Ant Colony Optimization). A continuación, se describirá, a modo de ejemplo, la aplicación de estos algoritmos a la resolución del problema TSP. Y por último, se analizarán sus principales propiedades.

4.1 INTRODUCCIÓN

En este apartado se describen los denominados algoritmos ACO *(Ant Colony Optimization*) para problemas de optimización discreta. Sus principales características son: *(1)* la utilización de "*feed-back*" positivo (acelera el descubrimiento de soluciones de gran calidad), *(2) computación distribuida* (la estructura de estos algoritmos permite su paralelización de forma muy simple y natural), y *(3)* el *uso de heurísticas Greedy constructivas* (ayuda a encontrar soluciones aceptables en las primeras etapas del proceso de exploración).

Estos algoritmos están inspirados en el comportamiento de las colonias de hormigas, concretamente en su forma de resolver problemas mediante la colaboración multi-agente. Las hormigas reales son capaces de encontrar el camino más corto entre el hormiguero y sus fuentes de alimento pese a carecer del sentido de la vista. Además, también son capaces de adaptarse rápidamente a los cambios en su entorno.

Esto es posible gracias a un sistema de comunicación indirecto que les permite, mediante modificaciones en el entorno producidas por ellas misma (mediante una sustancia denominada feromona), modificar su entorno y poder tomar decisiones. Los algoritmos de las hormigas (*Ant System*, *Ant Colony System*, *Max-Min Ant System*, etc.) utilizan una analogía de estas habilidades de las hormigas reales para resolver de forma heurística problemas de optimización discreta.

En el apartado 4.2 se explican las principales características de la meta-heurística ACO (*Ant Colony Optimization*). En los siguiente apartados (4.3 y 4.4) se describirá, a modo de ejemplo, la aplicación de estos algoritmos a la resolución del problema TSP, así como sus propiedades. Y por último, en al apartado 4.5, se analizarán otras posibles aplicaciones de estas meta-heurísticas, tanto a problemas de optimización estáticos como dinámicos.

4.2 ALGORITMOS DE HORMIGAS

4.2.1 Descripción de los algoritmos de hormigas

Una heurística pertenece a los denominados “algoritmos de las hormigas” si está inspirado en el comportamiento real de dichos insectos. A continuación se explica dicha analogía para posteriormente describir el algoritmo de forma más detallada.

Las hormigas son insectos que viven en colonias. Su conducta está dirigida más hacia la supervivencia de la colonia en general, que a la de sus componentes individuales. Uno de los comportamientos más interesantes de estos insectos es la búsqueda de alimento, y, en particular, cómo las hormigas, pese a carecer del sentido de la vista, pueden establecer los caminos más cortos entre su hormiguero y la fuente de alimento.

La solución a este enigma es una sustancia denominada feromona. Mientras las hormigas caminan del nido a la fuente de alimento, depositan en el camino dicha sustancia, dejando en él un rastro de feromona. Las hormigas que vengan después, detectarán mediante el olfato los diferentes rastros de las hormigas precedentes y tenderán, probabilísticamente, a escoger aquél con un rastro más intenso.

Por lo tanto, estos rastros de feromona, no sólo permitirán a las otras hormigas llegar a la fuente de alimento, sino que también les indicará cual es el camino más corto, ya que estos serán utilizados con mayor frecuencia que el resto, de tal forma que el rastro de feromona en ellos será más intenso.

De forma general, se pueden definir los algoritmos de hormigas como aquel conjunto de algoritmos diseñados para resolver cualquier tipo de problema de la vida real (no sólo problemas de optimización), y que basan su funcionamiento en el comportamiento de dichos insectos. La principal característica de estos algoritmos es la cooperación entre diferentes agentes para resolver un problema. Esta cooperación es posible gracias a un sistema de comunicación indirecto basado en una modificación del entorno (rastros de feromona) donde se mueven los agentes individuales (hormigas).

En este apartado se pretende ofrecer una visión general de las características de estos algoritmos. Para ello, se comentarán, una a una, las diferentes analogías con las hormigas reales que utilizan estos algoritmos:

- **Cooperación entre una colonia de individuos**: los algoritmos de hormigas se componen de una población, o colonia, de entidades que cooperan entre ellas de forma concurrente y asíncrona con el objetivo de encontrar un buena solución a un problema dado. El objetivo de esta cooperación entre toda la colonia es obtener soluciones de gran calidad.

- **Rastro de feromona**: las hormigas, o entidades, modifican algunos aspectos (rastro de feromona) de su entorno. Dicha modificación en un algoritmo de hormigas viene dado por cambios en una información numérica localmente guardada en el estado (entorno)

del problema en un instante dado. Esta información puede ser leída y/o actualizada por cada individuo de la población, por lo tanto, es función del comportamiento actual y pasado de la colonia.

Esta forma de comunicación indirecta es muy importante, ya que juega un papel básico en la utilización del conocimiento colectivo por parte de cada individuo, y su principal función es cambiar el entorno que perciben los individuos en función de la información histórica de toda la colonia.

- **Exploración de los caminos más cortos**: la función de la colonia de individuos es buscar soluciones con el mínimo coste (caminos mínimos entre el hormiguero y la fuente de alimentos en el caso de hormigas reales).

- **Movimiento locales**: Otro aspecto importante es que los individuos de una colonia "se mueven paso a paso" en un terreno de estados adyacentes del problema. Hay que tener en cuenta que un estado de un problema queda definido por dos niveles: *(1)* las condiciones del problema (si el problema es dinámico no variarán), y *(2)* la situación en que se encuentran los individuos en un instante dado (e.g. se pueden construir soluciones forma progresiva, transformando una solución en otra vecina, etc., en ambos casos, el estado de la solución en cada instante será un estado del problema). Obviamente, la definición de un estado y su adyacente dependerá del problema específico.

- **Política de transición de estados estocástica**: los individuos construyen soluciones aplicando una política de decisión probabilística para moverse hacia estados adyacentes. Esta política usa sólo información local, tanto en espacio como en el tiempo. Además, la función de probabilidad es función de la información propia del problema en un instante (conocida a priori), y de las modificaciones locales del entorno (rastros de feromona) que aportaron otros individuos en el pasado.

A parte de estas similitudes con las hormigas reales, las hormigas "virtuales" poseen otras propiedades complementarias:

- Los movimientos de las hormigas se producen entre *estados discretos.*

- Las hormigas virtuales poseen *memoria* de su pasado.

- Las hormigas virtuales dejan una *cantidad de rastro proporcional a la calidad de la solución*.

- El *mecanismo de sincronización* de actualización del rastro depende del problema y muchas veces no refleja el verdadero comportamiento de las hormigas.

- Para mejorar la eficiencia del sistema se pueden añadir *procesos complementarios de mejora* (apartado 4.3.6).

4.2.2 Algoritmos ACO

Los algoritmos ACO lo forman todas aquellas aplicaciones que se basan en la meta-heurística ACO. La meta-heurística ACO es, por lo tanto, un conjunto de ideas, estructuradas en forma de heurística, que aplica la analogía del comportamiento de las hormigas de una forma concreta (aunque no única). Consecuentemente, la meta-heurística *ACO* pertenece a los denominados algoritmos de las hormigas, y se puede utilizar como patrón para el diseño de métodos heurísticos específicos para resolver problemas de optimización discreta. A continuación se exponen las principales ideas de este patrón, y se presenta el pseudo-código propuesto por *Dorigo, Di Caro y Gambardella (1999).*

En la meta-heurística *ACO* una colonia finita de hormigas artificiales con las características del apartado 4.2.1 buscan de forma colectiva soluciones de gran calidad para el problema de optimización considerado.

Cada hormiga construye una solución, o un componente de ésta, comenzando por un estado inicial que depende del problema y con un criterio escogido a priori. Por lo tanto, la forma de construir soluciones factibles por parte de las hormigas es incremental.

Mientras las hormigas construyen "sus" propias soluciones, cada hormiga recoge información de las características del problema y de su propio comportamiento, y la usa para modificar la representación del problema. Esta nueva representación del problema será la que verán las hormigas siguientes. Las hormigas pueden actuar de forma simultánea o independiente, pero siempre cooperando entre ellas, gracias a este sistema de comunicación indirecto (apartado 4.4).

Las soluciones de gran calidad (coste mínimo) sólo se encuentran como consecuencia de la cooperación global entre todos los agentes de la colonia que construyen diferentes soluciones de forma concurrente.

Según la definición de vecindario (que depende del problema), cada hormiga construye una solución moviéndose a través de una secuencia (finita) de estados vecinos. Los movimientos son seleccionados aplicando una política de exploración local estocástica dirigida por *(1)* información privada de la hormiga, *(2)* información local específica del problema, y *(3)* un rastro "público" de feromona disponible para toda la colonia.

El estado interno de la hormiga, o memoria, se utiliza normalmente para evitar soluciones no-factibles. La información local que utiliza la hormiga para moverse suele estar basada en algún criterio heurístico específico del problema. Por último, todo el conocimiento acumulado por todas las hormigas durante todo el proceso de exploración, se encuentra codificado en forma de rastro de feromona, y equivale a una memoria a largo plazo compartida por la colonia.

La decisión de cuándo "dejar el rastro" de feromona, así como qué cantidad, depende de las características del problema y del diseño de la implementación. Cómo ya se ha dicho, este sistema de cooperación o autocatálisis (apartado 4.4.4) juega un papel muy importante en el

funcionamiento de los algoritmos ACO, ya que cuantas más hormigas escojan un movimiento, más atrayente será éste para las siguientes hormigas. En general, la cantidad de feromona depositada por una hormiga es proporcional a la calidad de la solución que la hormiga ha construido (o está construyendo).

La *tabla_de_decisión_de_la_hormiga* es una tabla probabilidades usada por las hormigas para decidir su política de exploración a través de las regiones del espacio de soluciones más interesante. Los valores de esta tabla son función de los rastros de feromona disponibles en un instante determinado y de los valores heurísticos.

Para evitar una rápida desviación de toda la colonia hacia la misma parte del espacio de exploración se utilizan básicamente dos mecanismos: *(1)* un mecanismo de "evaporación" de feromona (similar a la evaporación real de la feromona) que permite a la colonia de hormigas "olvidar" lentamente su conocimiento pasado y poder dirigir la exploración hacia nuevas direcciones, y *(2)* la componente estocástica de la política de decisión a la hora de escoger un movimiento de una hormiga. Los diferentes niveles de importancia de la componente estocástica y del rastro determinan el balance entre la exploración y la explotación (apartado 3.4.3).

Una vez las hormigas han cumplido su tarea de construir una solución (o una parte) y depositar su rastro de feromona, la hormiga muere, es decir, es borrada del sistema.

Las principales actividades de los algoritmos ACO (*generación_hormigas_ y_actividad, evaporación_de_feromona* y *acciones_complementarias*) necesitan algún tipo de sincronización, que se representa en el pseudo-código como el *programa_actividades*. En general la estructura secuencial es adecuada para problemas no distribuidos. Si se pretende implementar el algoritmo en paralelo este punto deberá ser tenido en cuenta.

En el siguiente apartado de describe de forma la meta-heurística ACO que aparece en *Dorigo, Di Caro y Gambardella (1999).*

4.2.3 Descripción del algoritmo

```
1   proceso Meta-heurística_ACO()
2     mientras (no se cumple criterio de finalización )
3       programa_actividades
4         generación_hormigas_ y_actividad();
5         evaporación_de_feromona();
6         acciones_complementarias(); (opcional; apartado 4.3.6)
7       fin del programa_actividades
8     fin mientras
9   fin proceso

10  proceso generación de hormigas y de actividad()
11    mientras (recursos disponibles)
12      programar_de_creacion_de_una_hormiga();
13      nueva_hormiga_activa();
14    fin mientras
15  fin proceso


16  proceso nueva_hormiga_activa() {ciclo de vida de una hormiga}
17    inicializar_hormiga();
18    M= actualizacion_de_la_memoria_de_la_hormiga();
19    mientras (estado actual ≠ estado final deseado)
20      A=leer tabla de decisión de la hormiga();
21      P=calcular_probabilidades_de_transición(A, M, restricciones del problema);
22      siguiente_estado=aplicar_politica_de_decision_de_la_hormiga (P, restricciones del problema);
23      mover_hormiga_al_ siguiente estado(siguiente_estado);
24      si (actualizacion de rastro paso a paso)
25        depositar_feromona_en_el_arco_visitado();
26        actualizar_tabla_de_decisión_de_la_hormiga();
27      fin si
28      M=actualizar_estado_interno();
29    fin mientras
30    si (actualizacion de rastro al final de cada ciclo)
31      evaluar_solución();
32      depositar_feromona_en_todos_los_arcos_visitados();
33      actualizar_tabla_de_decisión_de_la_hormiga();
34    fin si
35    muerte_de_la_hormiga();
36  Fin proceso
```

4.3 APLICACIÓN AL PROBLEMA TSP

4.3.1 El problema TSP

En esta sección se introduce el *AS* (*Ant System*). Para entender mejor el funcionamiento del algoritmo, se describe su aplicación al problema del viajante de comercio (TSP), ya que la primera aplicación de un algoritmo ACO fue a este problema, *Dorigo, Maniezzo y Colorni (1996)*). El motivo principal es que el problema posee unas características intrínsecas que permiten aprovechar al máximo la analogía de las hormigas reales. Y aunque el modelo se vea influenciado por la estructura del problema, se da una idea general del proceso muy clara, que permite su adaptación a la resolución de otros problemas de optimización.

Dado un número *n* de ciudades, el TSP se define como el problema de hallar una longitud mínima en un trayecto cerrado que visite todas las ciudades una sola vez. Las distancias entre dos ciudades *i* y *j* (d_{ij}) vienen definidas por un grafo *(N,E)*, donde *N* es el conjunto de ciudades, y *E* es el conjunto de arcos entre ciudades.

4.3.2 Ant System (*AS*)

4.3.2.1 Ant-cycle

Sea $b_i(t)$ $(i=1,...,n)$ el número de hormigas en una ciudad i en el instante t, y $m = \sum_{i=1}^{n} b_i(t)$ el número total de hormigas. Cada hormiga es un agente individual con las siguiente características:

- puede escoger una ciudad para ir, con una probabilidad que será función de la distancia a dicha ciudad, y del rastro presente en los arcos conectados a ella.
- para forzar que una hormiga haga caminos legales, las transiciones a ciudades ya visitadas están prohibidas (esto implicará que las hormigas tengan "memoria").
- cuando una hormiga completa una trayecto o solución, deja una sustancia llamada “rastro” en los arcos *(i,j)* visitados.

Sea $\tau_{i,j}(t)$ la "intensidad del rastro del arco *(i,j)* en el instante *t*". Cada hormiga en cada instante *t* escoge la siguiente ciudad a la que dirigirse, y en el instante *t+1* dicha hormiga estará en esa ciudad. Por la tanto, se definirá una "iteración del algoritmo AS " como los *m* movimientos que deben hacer las *m* hormigas en el intervalo *(t, t+1)*, de tal forma, que después de *n* iteraciones del algoritmo (un ciclo), cada hormiga habrá completado un trayecto completo (solución factible). En este punto, la intensidad de los arcos se actualiza utilizando la fórmula siguiente:

$$\tau_{ij}(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij} \qquad (1)$$

donde ρ es un coeficiente es igual a la "evaporación" del rastro en el intervalo *(t, t+n)* en tanto por uno, el rastro total dejado en un arco *(i,j)* por las *m* hormigas en un ciclo es $\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$, donde:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & si\ la\ k-ésima\ hormiga\ pasa\ por\ el\ arco\ (i,j)\ en\ su\ trayecto(entre\ t\ y\ t+n) \\ 0 & en\ caso\ contrario \end{cases} \qquad (2)$$

El parámetro *Q* es una constante, y L_k es la distancia del trayecto de la *k-ésima* hormiga.

El coeficiente ρ debe ser menor que 1 para evitar una acumulación ilimitada de rastro (hay que evitar una convergencia de la solución final demasiado rápida). Y $\tau_{i,j}(0)$ es la cantidad de rastro incial, que debe de fijarse a un valor diferente de 0 para poder aplicar la formula que calcula la probabilidad de transición.

Para satisfacer las restricciones de que una hormiga visite las n diferentes ciudades, se asocia a cada una de ellas una estructura de datos llamado "*lista tabú*", que memoriza las ciudades que ya ha visitado hasta el instante *t*, y prohibe que una hormiga la vuelva a visitar antes de que un ciclo o trayecto (*n* iteraciones) haya sido completado.

Cuando se completa un trayecto, la *lista tabú* se usa para computar la solución en curso de la hormiga (distancia del trayecto seguido por la hormiga). Después, la *lista tabú* se vacía y la hormiga vuelve a ser “libre” otra vez (este proceso también recibe el nombre de muerte de una hormiga, siempre y cuando después haya un proceso de creación de hormigas para el siguiente ciclo). Por lo tanto, se define el vector $tabu_k$, como el vector dinámico creciente que contiene dicha lista tabú de la *k-ésima* hormiga.

Se llama visibilidad η_{ij} a la cantidad $1/d_{ij}$. Esta cantidad no se modifica durante el proceso del AS, al contrario que el rastro, que se actualiza en cada ciclo. La visibilidad es la componente heurística del algoritmo.

Finalmente, se define la probabilidad de transición de la ciudad *i* a la ciudad *j* para la hormiga *k-ésima* como:

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k\ factibles} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}]^{\beta}} & si\ \ j\ \ pertenece\ a\ la\ k-ésima\ ciudad\ \ permitida \\ o & en\ caso\ contrario \end{cases} \qquad (3)$$

Donde las ciudades permitidas son aquellas que no están en lista tabú ($tabu_k$), y donde α y β son parámetros que controlan la importancia del rastro y de la visibilidad, respectivamente. Por lo tanto, la probabilidad de transición es una compromiso entre la visibilidad (lo que

quiere decir que las ciudades cercanas deberían ser escogidas con mayor probabilidad según la heurística), y la intensidad del rastro en el instante *t* (lo que quiere decir que si el arco *(i,j)* ha sido visitado, entonces este arco tendrá una mayor probabilidad de ser escogido).

4.3.2.2 Ant-density y Ant-quantity

A parte del sistema presentado en el apartado anterior, existen otras posibilidades para actualizar el rastro denominadas *ant-density* y *ant-quantity*, *Colorni, Dorigo y Maniezzo (1992a), Dorigo, Maniezzo y Colorni (1996)*). La principal diferencia es que en estas dos versiones el rastro se actualiza después de cada iteración (movimiento de una ciudad a otra), sin esperar a que se acabe el ciclo. En el sistema *ant-density* la cantidad de rastro que una hormiga deja al pasar por un arco *(i,j)* es constante ($\Delta\tau_{i,j} = Q$), mientras que en el sistema *ant-quantity* las hormigas, al pasar de una ciudad *i* a una *j*, dejan en ese arco (*i*,*j*) un rastro inversamente proporcional a la distancia del mismo ($\Delta\tau_{i,j} = Q/d_{ij}$).

La ventaja del *ant-cycle* respecto a estos dos sistemas es que la información que se va almacenando en la memoria de las hormigas (*feed-back*) es global, es decir, el rastro que deja una hormiga en un trayecto es proporcional a la calidad de la solución final. Sin embargo en los sistemas *ant-quantity* y *ant-density* la información que dirige la búsqueda local, es decir, las hormigas sólo aportan al sistema de búsqueda información de una iteración, pero no de la solución final. Sin embargo, cuando la dimensión del problema aumenta la actualización del rastro paso a paso (*ant-density* o *ant-quantity*) agiliza el proceso de *feed-back.*

4.3.3 Ant Colony System (ACS)

Introduciendo algunas mejoras en el AS, se puede definir el ACS (Ant Colony System), pensado para la resolución de problemas mayores que los que era capaz de resolver el AS (*Dorigo, Maniezzo y Colorni (1996), Stützle y Dorigo (1999)*). Las diferencias entre ambos radican, básicamente, en cuatro puntos:

- *Actualización del rastro al final de cada ciclo.* Una vez completado un ciclo, se añade un rastro extra sólo en aquellos arcos por lo que ha pasado la hormiga con la mejor solución. Esta regla permite aumentar el grado de explotación del subespacio de soluciones vecinas a la mejor solución encontrada hasta el momento.

$$\tau_{ij}(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\Delta\tau_{ij}^{mejor} \qquad (4)$$

- *Actualización del rastro en cada iteración.* El papel de esta actualización es favorecer la exploración. Para conseguirlo, se modula la actualización del rastro en cada iteración de forma que las hormigas que vengan a continuación encuentren los arcos por los que ha pasado la hormiga precedente menos atractivos. De esta manera, las hormigas se dirigirán hacia nuevas soluciones. Esto se consigue reduciendo el rastro de dichos arcos mediante la fórmula siguiente:

$$\tau_{ij}(t+1) = (1-\varepsilon)\cdot\tau_{ij}(t) + \varepsilon\cdot\tau_0 \qquad (5)$$

donde ε, $0 < \varepsilon < 1$, y τ_0 son dos parámetros.

- *Capacidad de modular el grado de exploración y explotación del espacio de soluciones*. Mediante una regla de decisión pseudo-aleatoria , antes de cada iteración (pasar de una cuidad *i* a una ciudad *j*) una hormiga "elige" entre dos métodos de exploración: con una probabilidad q_o el método elegido es *(1)* ir a la ciudad con un producto $(\tau_{ij}(t))^{\alpha}\cdot(\eta_{ij})^{\beta}$ máximo (concentrarse en las mejores soluciones para converger de forma más rápida) , y con una probabilidad ($1- q_o$) el método utilizado es *(2)* utilizar una tabla de probabilidades de forma equivalente al AS (ecuación 1).

- *Información adicional mediante una lista de candidatas.* El objetivo es tener una lista de ciudades "favoritas", de forma que las hormigas las elijan antes que al resto de vecinas no visitadas.

4.3.4 Max-Min Ant System (*MMAS*)

Las principales diferencias de la versión *Min-Max-AS* (*Stützle y Hoos (1997a), Stützle y Hoos (1997b)*) respecto al *AS* básico son tres:

- El rastro de feromona es depositado por las *acciones_complementarias* (apartado 4.3.6), y sólo en los arcos que son utilizados por la hormiga con la mejor solución, similar a una estrategia elitista. La ecuación 1 se modificaría de la siguiente forma:

$$\tau_{ij}(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \Delta\tau_{ij}^{mejor} \qquad (6)$$

- Los rastros de feromona están restringidos a un intervalo [τ_{min} , τ_{max}], es decir, $\forall\tau_{ij}$ $\tau_{min} < \tau_{ij} < \tau_{max}$ .El objetivo es evitar el estancamiento de la solución debido a la utilización de hormigas elitistas. La imposición de límites al rastro indirectamente limita la probabilidad p_{ij} de seleccionar una ciudad *j* cuando una hormiga está en la ciudad *i* a un intervalo [p_{min} , p_{max}] con $0 < p_{min} < p_{ij} < p_{max} < 1$.

- Los rastros se inicializan a su valor máximo τ_{max}, de esta forma, la exploración de caminos al inicio de la exploración es mayor ya que las diferencias entre los rastros de feromona son menos pronunciados.

4.3.5 Versión "Rank-Based" del AS

Otra mejora sobre el AS es la denominada versión "Rank-Based" (*Bullnheimer, Hartl y Strauss (1997)*). Esta versión utiliza la misma estrategia elitista que la versión *Min-Max AS*: la hormiga con la mejor solución en un ciclo actualiza el rastro. Además, un número determinado de hormigas *m* con las mejores soluciones en esa iteración también pueden dejar un rastro de feromona. Con este propósito, esta hormigas se ordenan en función de la longitud de sus caminos ($L^1(t) \le L^2(t) \le ... \le L^r(t) \le ... \le L^m(t)$), y la cantidad de feromona que una hormiga deposita se pondera de acuerdo al rango *r* de la hormiga. Sólo las (ω-1) mejores

hormigas de cada iteración pueden depositar feromona. A la mejor solución global, que aportará la mayor parte de la información en forma de *feed-back*, se le da el peso ω. La *r-ésima* mejor hormiga de una iteración contribuye a la actualización de feromona con un peso dado por *max {0, ω - r}*. La regla final de actualización del rastro queda de la siguiente forma:

$$\tau_{ij}(t+n) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{r=1}^{\omega-1}(\omega - r)\cdot\Delta\tau_{ij}^{r} + \omega\cdot\Delta\tau_{ij}^{mejor} \qquad (7)$$

4.3.6 Técnicas adicionales

4.3.6.1 Back-tracking

La aplicación de este mecanismo a los algoritmos ACO (*Di Caro y Dorigo (1997)*) permite a una hormiga “retroceder" durante la construcción del trayecto y continuar por otra ciudad, así hasta encontrar una solución final.

4.3.6.2 Look-ahead

Este proceso sirve para que una hormiga cuando elige una ciudad a donde ir, intente prever como afectará dicha elección a las posteriores iteraciones, es decir, una exploración dirigida.

4.3.6.3 Mejora local

Una mejora local comienza desde una solución inicial y, de forma repetitiva, intenta mejorar la solución en curso mediante cambios locales. Si se considera la solución final a la que llega cada hormiga después de una iteración, se puede aplicar una mejora local a dicha solución, y actualizar el rastro de feromona con la mejor solución encontrada por la mejora local. Existen varios ejemplos donde se aplican procedimientos de mejora local a algoritmos ACO (*Stützle y Hoos (1997)*

4.3.6.4 Hormigas elitistas

La filosofía de las hormigas elitistas es dar mayor importancia a aquellas hormigas con mejores soluciones. Esta filosofía puede implementarse de múltiples maneras: la hormiga con la mejor solución después de cada ciclo añade una cantidad extra de rastro en los arcos que ha visitado, la única hormiga que añade rastro después de cada ciclo es la hormiga con la mejor solución (ACS), el peso de cada hormiga a la hora de actualizar el rastro (no confundir con la cantidad de rastro) se pondera en función de la calidad de la solución (versión “*Rank-Based*” del AS). Estos sistemas favorecen la convergencia de la solución, aunque puede producir un estancamiento prematuro.

4.3.6.5 Lista de candidatos

La construcción de un camino tiene una complejidad $O(n^2)$. Cuando aumente número de ciudades el tiempo de ejecución aumentará cuadráticamente. Para disminuir la complejidad

del problema en los ejemplos de gran tamaño se usará una lista de ciudades candidatas. Es decir, cuando una hormiga está construyendo un camino la siguiente ciudad se escogerá entre una lista de candidatas de esa ciudad, si es posible. Normalmente la lista de candidatas de una ciudad está formada por un número fijo de las ciudades vecinas más cercanas a dicha ciudad, y ordenadas de menor a mayor distancia.

4.3.6.6 Acelerar la actualización del rastro

Al aplicar los algoritmos ACO al problema TSP, los rastros de feromona se guardan en una matriz con $O(n^2)$ entradas (una para cada arco). Todas las entradas de la matriz deberían actualizarse en cada iteración debido a la evaporación del rastro de feromona. Esta operación es muy costosa en problemas con un número elevado de ciudades. Para evitarlo, la versión *MMAS* se puede aprovechar la existencia de las listas de candidatos, y aplicar el fenómeno de evaporación sólo a los arcos que unen la ciudad *i* con las ciudades que pertenecen a su lista de candidatos. Este mecanismo reduce la complejidad a $O(n)$.

4.4 PROPIEDADES DE LOS ALGORITMOS ACO

4.4.1 Parámetros de control

Los parámetros más importantes que controlan el buen funcionamiento de un algoritmo ACO son α, β, y ρ. Los parámetros α y β controlan el peso de la información histórica y de la regla heurística, respectivamente, ya que afectan directamente al cálculo de la probabilidad de decisión de las hormigas (apartado 4.3.2). Por lo tanto, son parámetros muy importantes, que determinarán el grado de exploración y de explotación del espacio de soluciones.

El parámetro ρ se puede entender como el porcentaje de "evaporación" del rastro de feromona en cada ciclo. Esta evaporación tiene como objetivo "hacer olvidar" a las hormigas, y evitar así la convergencia demasiado rápida del algoritmo. Esto sería muy negativo, ya que no se ha de perder de vista que fijado un número de ciclos (o cualquier otro criterio de finalización), el objetivo de la búsqueda no es otro que el de evaluar la mayor cantidad de soluciones posibles aprovechando la información histórica de las soluciones anteriores.

Como ya se ha comentado, los algoritmos ACO se basan en la interacción de diferentes agentes que cooperan sinérgicamente entre ellos. Esta sinergia relaciona el aumento del número de individuos (hasta un número de individuos límite) con el aumento de la calidad de las soluciones. Por lo tanto, otros parámetros de interés, y que influyen de forma significativa en los resultados del AS son el número de hormigas, así como las ciudades donde éstas se colocan.

4.4.2 Papel de la heurística

Como es sabido, las reglas Greedy sólo garantizan óptimos locales y generalmente conducen a resultados finales no muy buenos. Éstas reglas funcionan muy bien durante los primeros pasos de la construcción de la solución, pero los últimos pasos, al estar condicionados por los pasos iniciales, no son tan buenos, siendo el resultado global de poca calidad.

Como ya se ha comentado en apartados anteriores, uno de los papeles de la heurística en el AS es guiar las primeras etapas de exploración, cuando la experiencia en forma de rastro de feromona todavía no ha sido acumulada por la colonia. A medida que las hormigas ganan experiencia, la heurística va perdiendo importancia. No obstante, la heurística es una pieza importante durante todo el proceso, ya que contribuye a la exploración del espacio de soluciones, permitiendo en todo momento "saltar" de un espacio de soluciones a otro con un criterio "lógico".

4.4.3 Evaluación implícita de las soluciones

Esta propiedad proviene directamente de la analogía con las hormigas. La idea básica es que las soluciones con menor coste (menor "distancia" total) sean completadas antes que otras soluciones con mayor coste. Por tanto, las soluciones de más calidad recibirán su cantidad de feromona más rápidamente. Aunque esta propiedad no se utiliza en los ejemplos utilizados

para resolver el TSP descritos en apartados anteriores, es muy importante en problemas dinámicos como el encaminamiento de redes (apartado 0).

4.4.4 Autocatálisis

Ya se ha comentado que el sistema comunicación de los algoritmos ACO se basa un mecanismo de aceleración del proceso (feed-back positivo). Si un individuo tiene que elegir entre diferentes opciones en un instante concreto, y resulta que la opción que escoge es buena, entonces en el futuro esa elección será todavía más deseable que antes.

Este mecanismo combinado con la evaluación implícita de soluciones es un sistema muy potente en algoritmos de optimización basados en la generación de poblaciones de soluciones (e.g., en los algoritmos genéticos el mecanismo autocatalítico se implementa mediante un proceso de selección/reproducción). Su principal cualidad es que se favorecen rápidamente las mejores soluciones, y así poder dirigir el proceso de exploración.

No obstante, cuando se usa este mecanismo se debe evitar que la convergencia de la solución sea demasiado rápida (estancamiento de la solución). Esta situación contingente impide continuar la exploración, y puede llevarnos a un óptimo local, o incluso a soluciones de muy baja calidad si las primeras etapas de la exploración, debido al azar, se han concentrado en un espacio de soluciones con una calidad muy por debajo del resto de posibles soluciones.

4.4.5 Paralelización del algoritmo

La mayoría del trabajo que se ha desarrollado en el área de investigación de sistemas distribuidos (sistemas basados en la distribución de tareas simples entre diferentes agentes que trabajan en paralelo para conseguir un objetivo común, de forma que esa distribución les permita una eficiencia que individualmente no pueden alcanzar) así como en los algoritmos ACO está justificada por la creciente importancia de los sistemas de paralelización de ordenadores. El objetivo básico que se persigue con la paralelización o distribución es aumentar su velocidad de ejecución.

Un tema muy importante a la hora de implementar el algoritmo en paralelo es la definición de un protocolo de comunicaciones. En el AS un conjunto de hormigas se comunica gracias a las modificaciones de una estructura de datos global (entorno): después de cada solución el rastro dejado por cada hormiga hará cambiar la probabilidad con la que una misma decisión será tomada en el futuro. La forma más natural de paralelizar estos algoritmos es en función de los datos que componen el entorno, o del dominio de una población.

Existen múltiples modelos de protocolos de comunicación que se describen a continuación:

- Atribuir una unidad de proceso individual a cada hormiga.
- Dividir la colonia en *p* subcolonias, donde *p* es el número de procesadores en paralelo disponibles. Cada subcolonia actúa como una colonia completa implementando un algoritmo AS estándar. Después de que cada colonia haya completado una iteración

del algoritmo, se establece el proceso de comunicación entre las diferentes colonias, y toda la información de todas las hormigas de todas las colonias es compartida por todas las subcolonias.

- Dividir la colonia en *p* subcolonias igual que el modelo anterior, pero intercambiando la información entre las diferentes colonias sólo cada cierto número de iteraciones.

- Investiga qué información debería ser intercambiada por *m* subcolonias y cómo debería ser usada para actualizar el resto de rastros de feromona del resto de subcolonias. Estos autores proponen tres métodos diferentes: *(1)* cada subcolonia usa la mejor solución global para escoger donde añadir rastro de feromona; *(2)* cada subcolonia usa las mejores soluciones globales de cada subcolonia para escoger donde añadir rastro de feromona y *(3)* todas las subcolonias actualizan sus rastros utilizando el promedio del rastro que dejaría cada subcolonia.

4.5 OTRAS APLICACIONES

4.5.1 Problemas combinatorios estáticos

Los problemas combinatorios estáticos son aquellos cuyas características del problema se conocen a priori, cuando se define el problema, y no cambian mientras el problema se está resolviendo. La aplicación de algoritmos ACO a la resolución de este tipo de problemas es sencilla, básicamente han de definirse la forma incremental de construir la solución, la estructura del vecindario y la regla estocástica de transición utilizada para construir la solución.

En la tabla siguiente se presenta una lista de las aplicaciones de los algoritmos ACO a diferentes problemas estáticos (todas las referencias relativas a la lista de encuentran en *Dorigo, Di Caro y Gambardella (1999)*).

Problema	***Autores***	***Nombre del Algoritmo***
Problema del viajante de comercio (TSP)	Dorigo, Maniezzo y Colorni (1991)	AS
	Gambardella y Dorigo (1995)	Ant-*Q*
	Dorigo y Gambardella (1996)	ACS y ACS-3-opt
	Stützle y Hoos (1997)	MMAS
	Bullnheirmer, Hartl y Strauss (1997)	AS_{rank}
Problema de asignación cuadrática	Maniezzo, Colorni y Dorigo (1994)	AS-QAP
	Gambardella, Taillard y Dorigo (1997)	HAS-QAP[1]
	Stützle y Hoos (1998)	MMAS-QAP
	Maniezzo y Colorni (1998)	AS-QAP[2]
	Maniezzo (1998)	ANTS-QAP
Problema del taller mecánico (Job-Shop)	Colorni, Dorigo y Maniezzo (1994)	AS-JSP
Rutas de vehículos (VRP)	Bullnheirmer, Hartl y Strauss (1996)	AS-VRP
	Gambardella, Taillard y Agazzi (1999)	HAS-VRP
Orden secuencial	Gambardella y Dorigo (1997)	HAS-SOP

4.5.2 Problemas combinatorios dinámicos

Los algoritmos ACO, debido a su naturaleza concurrente y adaptativa, son particularmente apropiados para la resolución de problemas donde la presencia de fuentes exógenas determina una falta de estacionalidad en el problema.

Este tipo de problemas son los denominados problemas dinámicos, y están definidos como función de unos parámetros definidos por la dinámica del problema de un sistema subyacente. Es decir, el problema cambia durante la ejecución del algoritmo de optimización, por lo que dicho algoritmo debe ser capaz de adaptarse a dichos cambios durante su ejecución.

Muchos problemas relacionados con las redes de comunicaciones o de transporte son intrínsecamente distribuidos y no estacionarios, y normalmente no se pueden resolver de

forma óptima. En la tabla siguiente se resumen las principales aplicaciones de los algoritmos ACO a problemas combinatorios dinámicos (todas las referencias relativas a la lista de encuentran en *Dorigo, Di Caro y Gambardella (1999)*):

Problema	***Autores***	***Nombre del Algoritmo***
Enrutamiento de redes orientadas a conexión	Schoonderwoerd, Holland, Bruten y Rothkrantz (1996)	ABC
	White, Pagurek, Oppacher (1998)	ASGA
	Di Caro y Dorigo (1998)	Ant-Net-FS
	Bonabeau, Henaux, Guérin, Snyers, Kuntz y Théraulaz (1998)	ABC-smart ants
Enrutamiento de redes no-orientadas a conexión	Di Caro y Dorigo (1997)	Ant –Net y Ant-Net-FA
	Subramanian, Druschel y Chen (1997)	Regular ants
	Heusse, Guérin, Snyners y Kuntz (1998)	CAF
	Van der Put y Rothkrnatz (1998)	ABC-backward

CAPÍTULO 5: ***APLICACIÓN DE LOS ALGORITMOS ACO A LA RESOLUCIÓN DEL SALBP-E***

En este capítulo se proponen dos métodos heurísticos para la resolución del SALBP-E que basan su funcionamiento en los algoritmos de hormigas presentados en el capítulo 4. El primero es una adaptación de la heurística ACS (Ant Colony System) al problema del SALBP-E. El segundo método es una versión mejorada del primero, ya que se le añade un procedimiento de mejora local interno, y que se describe en este capítulo.

5.1 META-HEURÍSTICA ACO PARA EL SALBP-E

5.1.1 Introducción

La meta-heurística propuesta en este trabajo se basa en gran medida en el ACS (Ant Colony System) propuesto por *Dorigo y Gambardella 1997*. Dado un grafo *G* que defina un juego de datos del problema SALB-E (ver apartado 2.2) se pueden hacer las siguientes analogías con el problema TSP:

- Cada tarea del SALBP-E equivale a una ciudad en el TSP.
- Las restricciones de precedencias entre tareas estarán representadas por los arcos del grafo G al igual que en el problema del viajante de comercio TSP en su versión asimétrica. Es decir, se deberá tener en cuenta en sentido de los arcos. Además, debido a las restricciones de precedencias del problema SALB, existirá un menor número de candidatos durante la construcción de las soluciones (sobre todo al inicio de la solución).
- Un trayecto en el TSP (camino cerrado que recorre todas las ciudades sin repetir ninguna) tendrá su analogía en el problema del equilibrado en una secuencia de tareas donde están todas incluidas, no se repite ninguna, y cumplen las precedencias.
- La construcción de una solución se hará paso a paso, al igual que en el TSP. Cada hormiga comienza en un nodo inicial (tarea inicial), y su objetivo es construir una solución factible, es decir, una secuencia de tareas. Una secuencia se construye nodo a nodo (los nodos son los vértices del grafo G del SALBP-E): cuando una hormiga k se encuentra en un nodo (tarea) i, debe escoger, entre una lista de candidatos, un nodo j al cual moverse. La lista de candidatos dependerá del nodo en el cual se encuentra la hormiga y los nodos (tareas) ya asignados a la secuencia en curso, de tal forma que se garantice la factibilidad de las soluciones.
- Dada una hormiga situada en un nodo i y una lista de tareas candidatas, la elección de la siguiente tarea que debe asignarse a la secuencia de tareas se hace mediante una regla de transición probabilística similar a la que aparece en el apartado 4.3.2.
- En el TSP la calidad de una solución o trayecto es proporcional al inverso de su función objetivo, es decir, el inverso de la distancia del trayecto solución. Para medir la calidad del SALBP-E primero se evaluarán las secuencias de tareas probando todos los tiempo de ciclo posibles sobre cada secuencia de tareas, y escogiendo la mejor función objetivo (mínima) de entre todas las combinaciones que dichos tiempos de ciclo permiten. De esta forma, la calidad del SALBP-E será proporcional al inverso de esta función objetivo.
- Cuando se actualiza el rastro en el TSP, la "feromona" se deposita en los arcos del grafo que han sido utilizados por las hormigas para generar las soluciones del SALBP-E. En el SALBP-E esta feromona se deposita entre subsecuencias (parejas) de tareas de la secuencia.

- La idea principal de esta actualización, tanto para el TSP como para el SALBP-E, es depositar más rastro en aquellos arcos que pertenecen a soluciones de más calidad. Por lo tanto, en ambos problemas se dejará una cantidad de rastro de feromona inversamente proporcional a la calidad de sus respectivas funciones objetivo.

- Mientras en el TSP se utiliza una única heurística (distancia menor a la siguiente ciudad), para el SALBP-E se utilizarán 13 criterios heurísticos específicos del equilibrado de líneas.

5.1.2 Función objetivo del SALBP-E

En los problemas de equilibrado de líneas de montaje, se entiende por *solución* la distribución de las tareas en las diferentes estaciones de trabajo, sin tener en consideración el orden en el que se desarrollan dichas tareas dentro de la estación. Una vez establecida la relación tarea-estación, el orden de las tareas dentro de cada estación puede variar, siempre y cuando se cumplan las restricciones de precedencias.

La función objetivo del problema SALB-E es el producto del número de estaciones por el tiempo de ciclo. Para solucionar este problema sin tener que fijar ninguno de los dos parámetros a priori, se introduce el concepto de *secuencia de tareas*. Una secuencia de tareas es la solución que se obtendría si se asignase cada tarea a una única estación de trabajo, o lo que es lo mismo, el orden en que se deberían ejecutar las diferentes tareas sin tener en cuenta a que estación pertenecen. De esta forma, se disocia el concepto tiempo de ciclo y número de estaciones de la secuencia de tareas. Una secuencia de tareas posee entidad por sí misma y a partir de ella, se obtendrá una solución para cada combinación de tiempo de ciclo y número de estaciones posibles. Al procedimiento para encontrar la mejor solución para cada secuencia de tareas se le llamará *evaluación.*

5.1.3 Funcionamiento general: una colonia de hormigas

5.1.3.1 Esquema básico

El esquema básico de funcionamiento de éste algoritmo es el que se muestra en la figura 5.1.3.1

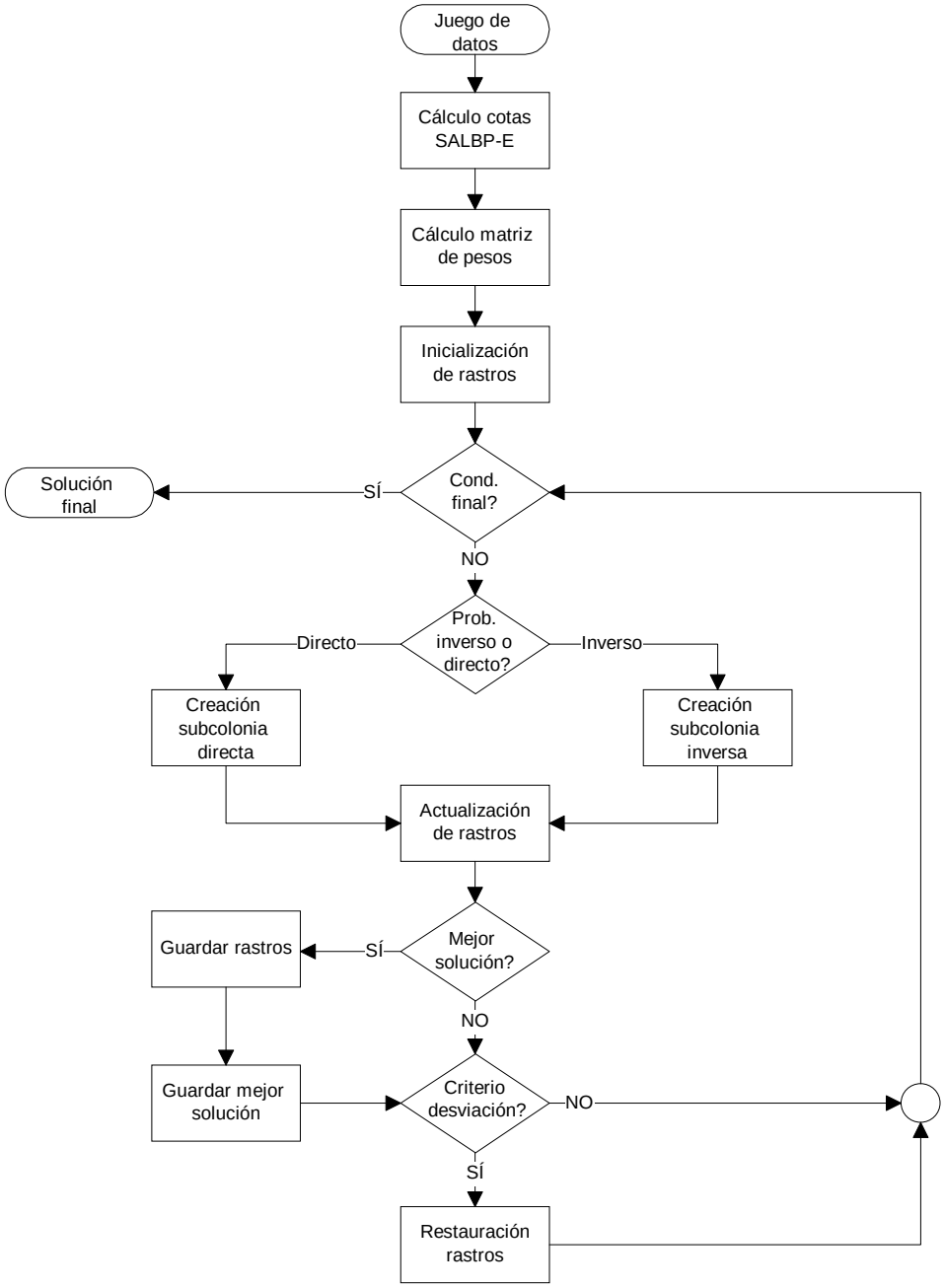


Figura 5.1.3.1

La idea consiste en crear un registro con información histórica (el rastro) e ir creando y enviando subcolonias (conjuntos) de hormigas que se nutran de él y a la vez lo realimenten.

Por lo tanto, el esquema básico sería:

1. Inicializar la matriz de rastros
2. Crear subcolonia de hormigas mientras no se cumpla un criterio de finalización
3. Las subcolonias crean secuencias de tareas utilizando la información del rastro.
4. Actualizar el rastro con las nuevas secuencias de tareas
5. Volver al paso 2

5.1.3.2 Matriz de rastros

Como ya se ha comentado, el rastro se deja entre parejas de tareas. Además, para que la regla de transición pueda ser aplicada también a la primera y la última tarea de la secuencia (este segundo caso es necesario si se resuelve el problema inverso y directo con la misma estructura de rastros), se crean dos tareas ficticias: α.y ϖ. La primera tarea α tendrá arcos a todas las tareas sin precedencias en el grafo, y por otro lado, todas las tareas que no tengan ningún sucesor inmediato se unirán mediante arcos a la tarea ϖ. Por lo tanto, para poder guardar toda esta información será necesario una matriz *(n+2)x(n+2)*, donde *n* es el número de tareas. La componente de rastro τ_{ij} representará la cantidad de rastro de feromona de la subsecuencia *tarea$_i$-tarea$_j$*.

La inicialización de la matriz de rastros se puede hacer de múltiples maneras. En la meta-heurística propuesta se han probado básicamente tres formas para inicializar la matriz de rastros: (1) rellenar sus elementos de forma aleatoria, (2) fijar todos sus elementos a una cantidad constante muy elevada y, finalmente, (3) fijar todos sus elementos a una cantidad constante prácticamente nula.

Las dos primeras tienen como objetivo permitir una mayor exploración durante las primeras etapas, mientras en que la última opción las hormigas iniciales tendrán una mayor influencia a la hora de guiar al resto de la colonia. En las pruebas realizadas se ha observado que tanto la forma aleatoria como este último caso, aumenta la probabilidad de estancarse en soluciones iniciales de no muy buena calidad. Mientras que en el primer caso, la matriz evoluciona de tal manera que el grado de exploración va disminuyendo a medida que se encuentran mejores soluciones.

5.1.3.3 Actualización del rastro

La actualización de la matriz de rastros se realiza después de que cada subcolonia de hormigas haya construido y evaluado todas las secuencias construidas por sus hormigas, utilizándose solamente la secuencia de tareas con mejor función objetivo (OBJ_{mejor}) para dejar rastro. Además de esta actualización cíclica, cuando alguna subcolonia mejora la mejor función objetivo encontrada hasta en momento, se vuelve a actualizar la matriz de rastros, depositando una cantidad extra de feromona sobre las subsecuencias de tareas de dicha solución.

La idea básica es que se “memoricen” aquellas subsecuencias de tareas que proporcionan buenas soluciones, y que esta memoria sea utilizada por el resto de hormigas para generar soluciones en ese mismo vecindario. La fórmula usada es:

$$\tau_{ij} \leftarrow (1-\rho)\cdot\tau_{ij} + \frac{1}{(d(\%)^{mejor})^2}$$

donde d(%) es la discrepancia de la función objetivo respecto a la mejor cota hallada para el SALBP-E:

$$d^{mejor}(\%) = \frac{OBJ_{mejor} - COTA_{\text{SALBP-E}}}{\text{COTA}_{\text{SALBP-E}}} \cdot 100$$

Para homogenizar la cantidad de rastro que se deja sin tener independientemente del juego de datos particular, se necesita una referencia que no dependa del rango de la función objetivo. Por este motivo, se usa como medida de calidad de la solución el inverso de la discrepancia a la cota, en lugar del inverso de la función objetivo, ya que ésta varía de forma sensible según el juego de datos.

Se ha observado que en algunas ocasiones, debido al rango de la función objetivo del juego de datos, se necesita un refuerzo puntual de algunos rastros para poder profundizar en el mejor espacio de soluciones.

Es decir, para garantizar que siempre que se encuentre una buena solución el resto de hormigas puedan seguirla, cuando se mejora la mejor solución hasta el momento, se utiliza otra fórmula que asegura que las subsecuencias de esa nueva mejor solución tengan más rastro que las subsecuencias con mayor rastro hasta el momento:

$$\tau_{ij} \leftarrow k \cdot \max_{\forall k} \{\tau_{ik}\} \quad \forall (i, j) \in subsecuencia\ de\ la\ mejor\ solución\ encontrada$$

Es decir, dada la secuencia (i,j) perteneciente a la mejor solución encontrada hasta el momento, se busca en la matriz de rastros el máximo rastro entre la tarea i y el resto, y se actualiza el valor de τ_{ij} con una cantidad k veces superior a ese valor máximo.

Este proceso, esquematizado en forma de diagrama, sería el mostrado en la figura 5.1.3.3:

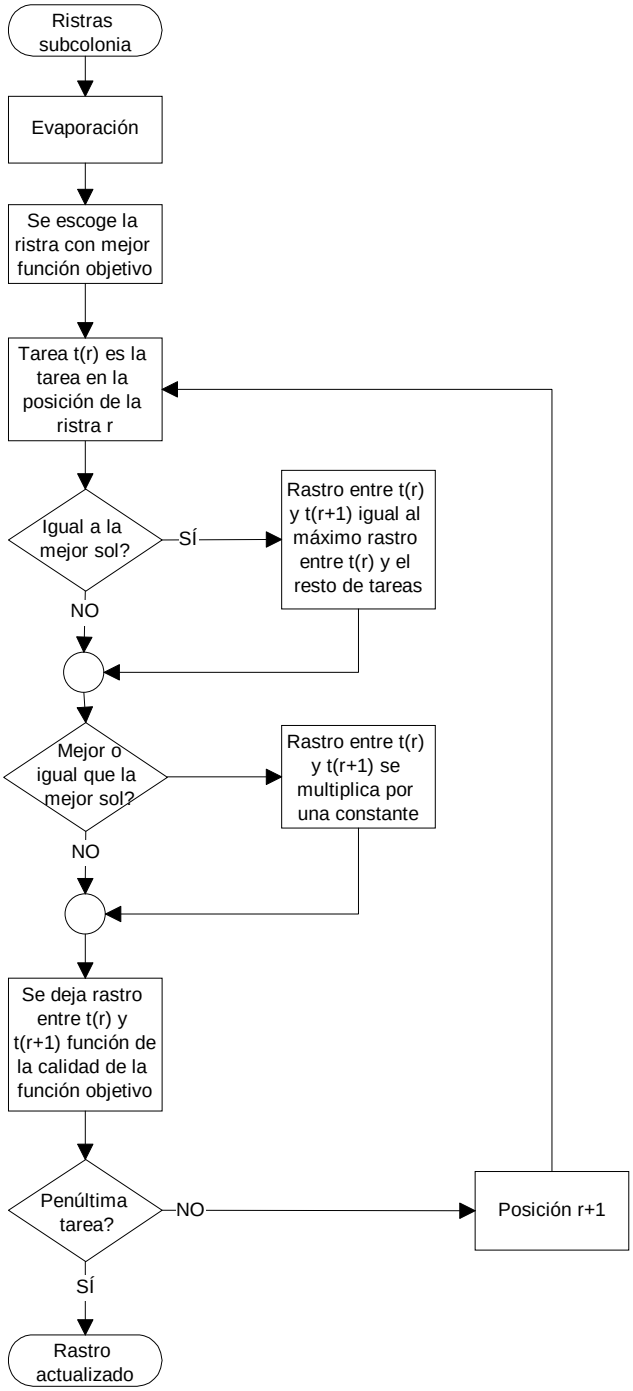


Figura 5.1.3.3

5.1.3.4 Backtracking de la matriz de rastros

Debido a la evaporación del rastro, que es necesaria para garantizar una buena exploración del espacio de soluciones, puede ocurrir que las hormigas olviden demasiada información y exploren espacios de soluciones no deseados, esto ocurrirá sobre todo después de un cierto número de etapas sin mejorar la mejor función objetivo encontrada hasta el momento.

Para evitar este comportamiento, se consolidará la matriz de rastros cada vez que se mejore la mejor solución hasta el momento (justo después de actualizarla), y, si durante la evolución de proceso se cumple una condición de desviación impuesta a priori, la matriz de rastros de feromona en curso se inicializará de nuevo con los valores de la "mejor matriz de rastros encontrada hasta el momento".

Para definir el criterio de desviación, se fija un valor mínimo porcentual del número de repeticiones de las soluciones (que se caracterizan por su función objetivo) cada *n* subcolonias. Es decir, cada *n* subcolonias de deberá cumplir que al menos *p* de ellas tengan como mejor solución objetivo la mejor solución objetivo encontrada hasta en momento. Con ello se garantiza que se está explorando dicho subespacio de soluciones.

Siguiendo el mismo esquema que en el apartado 5.1.6, se ha fijado *n* en 100 y *p* en 35.

5.1.3.5 Exploración del problema directo e inverso

Con el objetivo de aprovechar la inversibilidad del problema, la meta-heurística propuesta lanza subcolonias de hormigas que resuelven el problema directo, y subcolonias que resuelven su versión inversa, de forma alternativa.

No obstante, no son totalmente independientes, ya que utilizan la misma matriz de rastros (esto se consigue simplemente trasponiendo la matriz cada vez que se cambia el "sentido" del problema).

Esto, lejos de ser un problema, hace que la calidad de la información de la matriz de rastros sea más rica, pues contiene la información del problema desde la óptica inversa y directa. Es decir, las hormigas que construyen soluciones del problema directo aprovechan información histórica del problema inverso, y viceversa.

5.1.4 Funcionamiento interno: la subcolonia

5.1.4.1 Esquema básico

La unidad funcional de la subcolonia es la hormiga. Cada hormiga es independiente de las demás y es capaz de crear una secuencia de tareas propia. Las herramientas que usa son:

- Información histórica (rastros)
- Criterio heurístico intrínseco a cada hormiga

Durante el proceso de creación de la secuencia, ambas se combinan formando los *índices de decisión*, que darán la información última a la hormiga para poder crear la secuencia.

Por lo tanto, el esquema básico sería:

1. Crear una hormiga con un criterio heurístico propio
2. Crear una lista de tareas candidatas para ser añadidas a la secuencia de tareas
3. Calcular los índices de decisión utilizando el criterio heurístico propio y el rastro entre la última tarea asignada y las candidatas
4. Asignar la tarea en función de los índices
5. Actualizar las tareas candidatas. Ir al paso 3 hasta que ya no queden candidatas.
6. Evaluar la secuencia de tareas

Las hormigas se integran dentro de subcolonias. Cada subcolonia consta de *n* hormigas con diferentes criterios heurísticos. En este trabajo, se han utilizado subcolonias de catorce hormigas con catorce criterios diferentes. Los trece de la tabla más el azar:

Nomenclatura:

i,j *Índice de tarea*
N *Número de tareas*
C *Tiempo de ciclo*
T_i *Duración de la tarea i*
IS_i *Conjunto de tareas siguientes inmediatas a la tarea i*
S_i *Conjunto de tareas siguientes a la tarea i*
TP_i *Conjunto de tareas precedentes a la tarea i*
L_i *Nivel de la tarea i en el grafo de precedencias*

Seleccionar i* tal que $v(i^*) = \max_i \{v(i)\}$.

Nombre	***Peso***
1. Longest Processing Time	$v(i) = t_i$
2. Greatest number of Immediate Successors	$v(i) = \lvert IS_i \rvert$
3. Greatest number of Successors	$v(i) = \lvert S_i \rvert$
4. Greatest Ranked Positional Weight	$v(i) = t_i + \sum t_j \ (j \in S_i)$
5. Greatest Average Ranked Positional Weight	$v(i) = \left(t_i + \sum t_j \ (j \in S_i)\right) / \ \left(\lvert S_i \rvert + 1\right)$
6. Smallest Upper Bound	$v(i) = -UB_i = -N - 1 + \left[\left(t_i + \sum t_j \ (j \in S_i)\right) / \ C\right]^+$
7. Smallest Upper Bound / Number of Successors	$v(i) = -UB_i \ / \ \left(\lvert S_i \rvert + 1\right)$
8. Greatest Processing Time / Upper Bound	$v(i) = t_i / UB_i$
9. Smallest Lower Bound	$v(i) = -LB_i = -\left[\left(t_i + \sum t_j \ (j \in TP_i)\right) / \ C\right]^+$
10. Minimum Slack	$v(i) = -\left(UB_i - LB_i\right)$
11. Maximum Number of Successors / Slack	$v(i) = \lvert S_i \rvert / \left(UB_i - LB_i\right)$
12. Bhattcharjee & Sahu	$v(i) = t_i + \lvert S_i \rvert$
13. Etiquetas Kilbridge & Webster	$v(i) = t - L_i$

Para homogeneizar los valores de los diferentes pesos de un juego de datos concreto, así como entre los diferentes juegos de datos, pesos poder utilizarse en las diferentes hormigas y juegos de datos, se ha estandarizado el valor de *v(i)* para que todos los criterios tengan el mismo rango en función del número de tareas del juego de datos. En este trabajo, se han transformado a unos valores entre 1 y el número de tareas del juego de datos, con valores siempre enteros.

La representación gráfica del proceso sería la que muestra la figura 5.1.4.1:

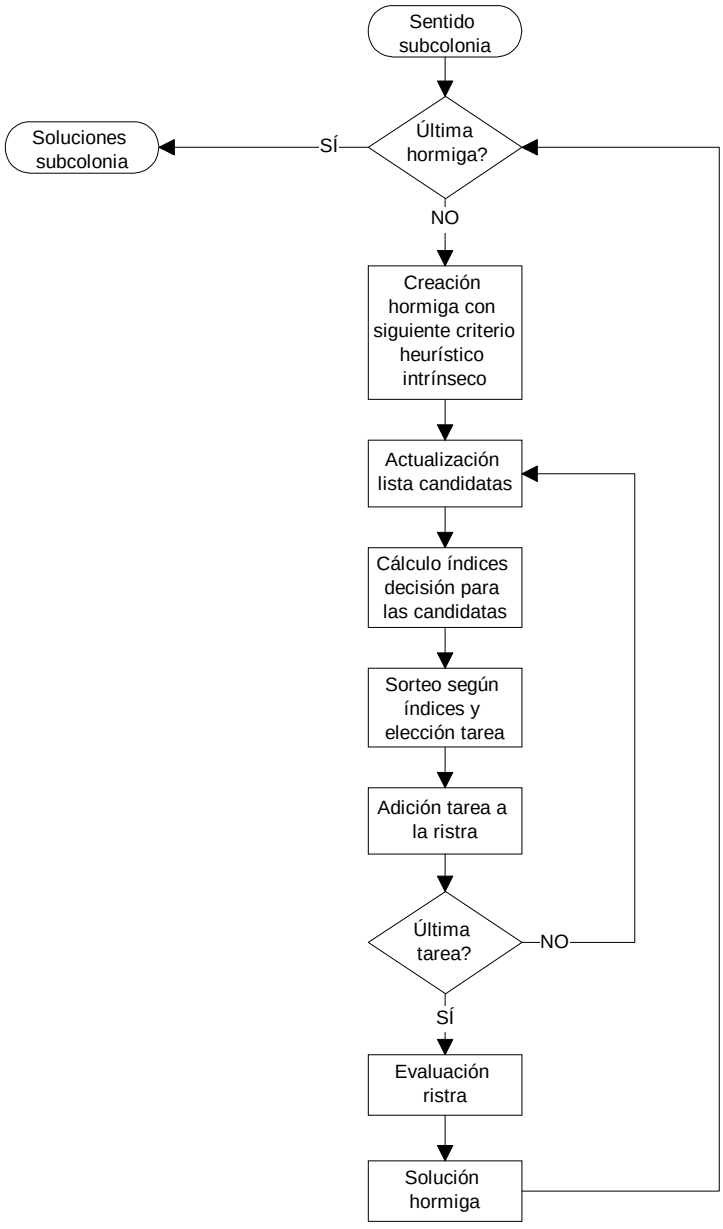


Figura 5.1.4.1

5.1.4.2 Índices de decisión

Cada vez que una hormiga asigna una nueva tarea a la secuencia en curso, actualiza la lista de candidatas y calcula para cada una de ellas un índice de probabilidad que le servirá para escoger la siguiente tarea.

Éste índice combina el conocimiento acumulado en la matriz de rastros y la inteligencia propia de la hormiga debido su criterio heurístico intrínseco. Se calculará un índice para cada tarea *j* del conjunto de tareas candidatas *C*, siendo *i* la última tarea asignada a la secuencia de tareas:

$$u_{ij} = [\tau_{ij}]^{\alpha} \cdot [\eta_j]^{\beta} \quad \forall j \in C$$

donde τ_{ij} es el rastro histórico entre la tarea *i* y la *j*, y η_j el peso de la tarea *j* según el criterio heurístico intrínseco de la hormiga.

Los parámetros α y β controlan la importancia del rastro y la inteligencia heurística. Por lo tanto, la probabilidad de transición es un compromiso entre ambos.

Para poder controlar en todo momento los valores de τ_{ij} y η_j, se usan valores ponderados con el resto de τ_{ij} y η_j de las tareas candidatas *C*. De esta forma, se consigue que la mayor importancia de uno u otro se maneje mediante los parámetros α y β, y no se vea afectado por otros parámetros del problema como el número de tareas o la calidad de la cota.

El índice para cada tarea se obtiene de la ponderación entre todos los índices de las tareas candidatas:

$$p_{ij} = \frac{u_{ij}}{\sum_{k \in J} u_{ik}} \quad \forall j \in C$$

5.1.4.3 Construcción de secuencias de tareas

Cada hormiga comienza en el nodo (tarea) α del grafo, y va añadiendo nuevos nodos, o tareas, de forma iterativa, hasta llegar al nodo ϖ (en el caso del problema inverso, se comenzará en el nodo ϖ y finalizará en el nodo α). En cada iteración, existe una lista de nodos candidatos, que depende de las restricciones de precedencias y de los nodos ya asignados a la secuencia, garantizándose en todo momento la factibilidad de las secuencias finales.

Para elegir el siguiente nodo, se calculan los índices de decisión de cada candidato, y se escoge aleatoriamente uno de ellos según el índice de probabilidad marcado por los índices de

decisión. Una vez asignado el nodo a la secuencia de tareas, se actualiza la lista de candidatos y se repite el proceso hasta llegar a la última tarea ϖ.

5.1.4.4 Evaluación de la secuencia

El procedimiento para generar una solución a partir de una secuencia de tareas consiste en, fijado un tiempo de ciclo, ir llenando estaciones con ese tiempo de ciclo siguiendo el orden de la secuencia de tareas, tal y como muestra la figura 5.1.4.4 (a):

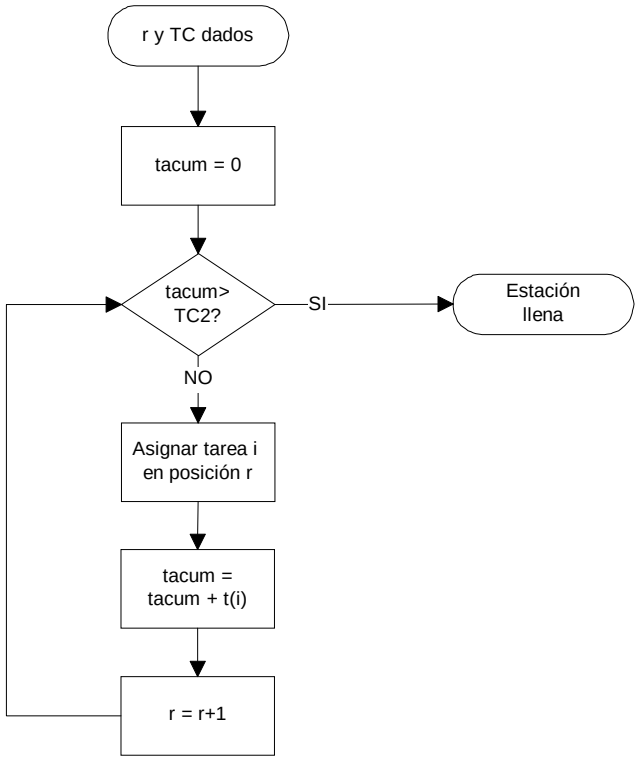


Figura 5.1.4.4 (a)

Una vez asignadas todas las tareas, se obtiene el número de estaciones que corresponde al tiempo de ciclo de fijado. Si se probasen todos los tiempos de ciclo posibles, se podría obtener la mejor combinación de tiempo de ciclo y número de estaciones de la secuencia de tareas.

La idea es probar todos aquellos tiempos de ciclo necesarios para obtener todas las soluciones posibles sin que se repita ninguna. Una vez probados, se escoge la mejor solución como solución óptima asociada a la secuencia de tareas.

La determinación de estos tiempos de ciclo se lleva a cabo mediante tres parámetros: cota mínima de tiempo de ciclo, cota máxima y tiempos de ciclo inertes.

Cota mínima de tiempo de ciclo: $max\left\{max\{t(i)\}; \frac{\sum_i t(i)}{n_{max}}\right\}$ donde n_{max} es el número de estaciones máximo definido en el problema.

Cota máxima de tiempo de ciclo: no existe una buena cota para problema el SALB-E, pero se puede empezar con la suma de los tiempos de todas las tareas. Para cada secuencia de tareas, se tomará como cota máxima de tiempo de ciclo el primer tiempo que proporcione un número de estaciones igual al mínimo. Esta cota se irá actualizando con los sucesivos tiempos probados sobre una ristra.

Por lo tanto, se probarán los tiempos de ciclo comprendidos entre las cotas máxima y mínimas. Es importante tener en cuenta que estas cotas se han de recalcular para cada secuencia de tareas, ya que cada secuencia es independiente de las otras.

Tiempo de ciclo inertes: teniendo en cuenta que la evaluación se basa en ir fijando tiempos de ciclo desde el mínimo hasta el máximo, mediante un aumento gradual entre ambos límites, cuantos menos tiempos de ciclo se prueben, más rápido será el proceso.

El procedimiento más simple sería ir aumentando de uno en uno el tiempo de ciclo a probar desde el mismo hasta el máximo. Sin embargo, existen tiempos de ciclo que a priori se sabe que no van a cambiar la función objetivo. Son los tiempo de ciclo que se denominarán inertes.

La idea consiste en aprovechar la información que se obtiene al fijar el tiempo de ciclo anterior. Al cerrar una estación, se puede saber en cuanto habría de aumentar el tiempo de ciclo para que dicha estación incluyese a la siguiente tarea. Si se sabe para cada estación y se escoge el mínimo de todos estos tiempos de ciclo, se tendrá el siguiente tiempo de ciclo que modificaría la distribución de tareas.

Gráficamente, este concepto se puede apreciar en la figura 5.1.4.4 (b):

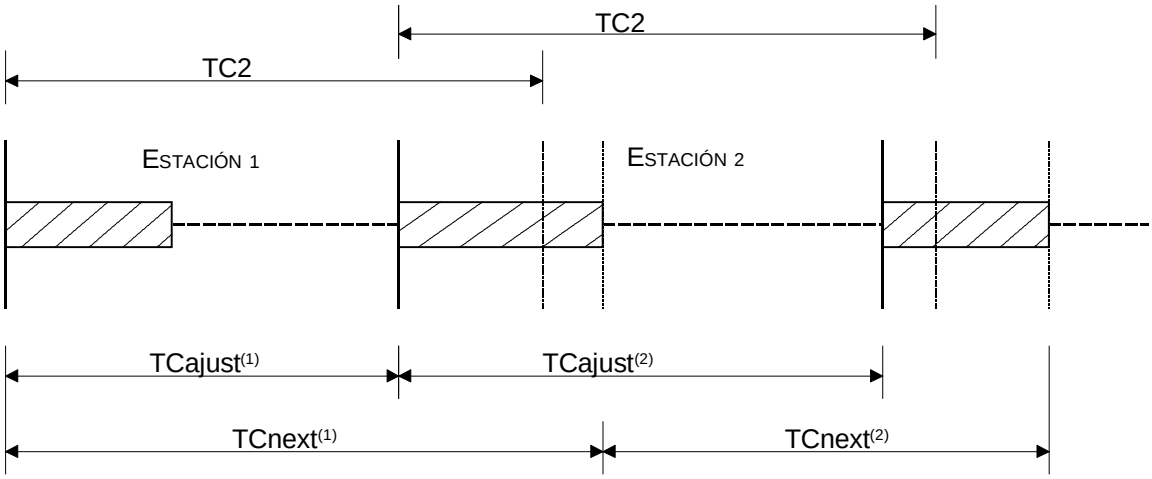


Figura 5.1.4.4 (b)

El tiempo de ciclo fijado es *TC2* y el siguiente tiempo de ciclo que se debería probar *TCnext*. El menor *TCnext* es el (1). Si *TC2* no aumenta como mínimo hasta $TCnext^{(1)}$ la distribución de tareas no variará. Por lo tanto, como el número de tareas será el mismo y el tiempo de ciclo habrá aumentado, la función objetivo obtenida será peor. Esta forma de calcular el siguiente tiempo de ciclo a probar tiene la ventaja de que no se pierde ninguna solución.

Otra idea que se extrae de la figura 5.1.4.4 (b), es el concepto de *tiempo de ciclo ajustado*. Aunque el tiempo de ciclo que se esté probando sea *TC2*, al existir tiempos muertos al final de las estaciones, se puede ajustar el tiempo de ciclo al que realmente existiría en la línea: el de la estación más lenta. Si se calcula la suma del tiempo de las tareas de cada estación ($TCajust^{(i)}$) y se escoge el mayor, se obtendrá el tiempo de ciclo ajustado de la solución (en este caso $TCajust^{(2)}$).

De esta forma, la función objetivo estará más ajustada: $TCajust \cdot n$. Es importante destacar que durante la prueba de un tiempo de ciclo, su tiempo de ciclo ajustado sólo puede aumentar, aunque, obviamente, siempre será menor que *TC2*. Esta propiedad es fundamental para el uso de las cotas de los apartados siguientes.

5.1.5 Criterio de finalización

Existen diferentes criterios de finalización. El objetivo es abortar el proceso si se considera que no se va a poder mejorar la solución obtenida hasta el momento.

Esta situación se ha de definir antes del inicio de la resolución del problema, y normalmente viene dada por un estancamiento de las soluciones obtenidas por las hormigas.

Un posible método para controlar estos fenómenos es el cálculo de la desviación tipo de las hormigas de una subcolonia. Si esta desviación es menor que un parámetro fijado a priori durante un número determinado de iteraciones, se podrá concluir que las hormigas se han estancado en un óptimo local.

Por otro lado, también se puede controlar de forma similar la desviación estándar de las últimas *n* mejores hormigas de cada subcolonia, y parar el proceso si se encuentra por debajo de cierto valor predeterminado.

Para la pruebas realizadas, se han utilizado, unos criterios más básicos, como son el tiempo de ejecución y el número de subcolonias. Como se verá más adelante, con la configuración de parámetros de control utilizada, la variabilidad de ambas desviaciones tipo es siempre elevada, por lo que servirían como control de estancamiento, según las pruebas, si su valor fuese cercano a cero.

5.1.6 Configuración de los parámetros básicos

Los parámetros más importantes, de los cuales depende el buen funcionamiento de la meta-heurística, son α, β, ρ y k (ver apartado 4.4.1). Para su configuración se ha llevado a cabo una experiencia computacional que se resume a continuación.

- 400 combinaciones con un juego de datos de más de 100 tareas (2 pruebas).
- Repetición con las 25 mejores combinaciones anteriores se ha lanzado el juego de datos 5 veces más.
- Con las 5 mejores combinaciones anteriores, se han probado 10 juegos de datos diferentes (5 pruebas).

Los parámetros elegidos para la meta-heurística han sido α=0.75, β=0.25, ρ=0.01 y k=10.

5.2 META-HEURISTICA ACO PARA EL SALBP-E COMBINADA CON UNA MEJORA LOCAL

5.2.1 Objetivos de la mejora local dentro del algoritmo de hormigas

Después de ajustar los parámetros básicos de funcionamiento, se puede observar que las hormigas son capaces de explorar diferentes espacios de soluciones a lo largo del proceso, convergiendo hacia una solución final.

Esta capacidad de exploración, sin embargo, limita en muchas ocasiones la explotación del subespacio de soluciones, haciendo que el proceso, a veces, sea largo o simplemente no se encuentre el óptimo local, quedándose muy próximo a él.

Para solucionar este problema, se ha añadido una mejora local durante el proceso a ciertas hormigas con el objetivo de:

- Mejorar la calidad de las soluciones (y por lo tanto del rastro).
- Acelerar el proceso general.

Para cumplir este objetivo, la mejora local se puede aplicar en diferentes hormigas clave. En este trabajo, se ha probado aplicando la mejora local a todas las hormigas de la subcolonia que mejora o iguala la mejor solución en curso.

Sin embargo, este proceso es demasiado costoso en tiempo para el rendimiento global que se obtenía, ya que se invierten demasiados recursos en la explotación del espacio de soluciones.

Por ello, se ha decidido aplicar la mejora local solamente a las hormigas que igualen o mejoren la mejor solución hallada hasta el momento. Además, para evitar un posible estancamiento en un óptimo local, cada *k* subcolonias, se escoge una hormiga al azar, y se le aplica una mejora local.

Gráficamente, el proceso sería el descrito en la figura 5.2.1:

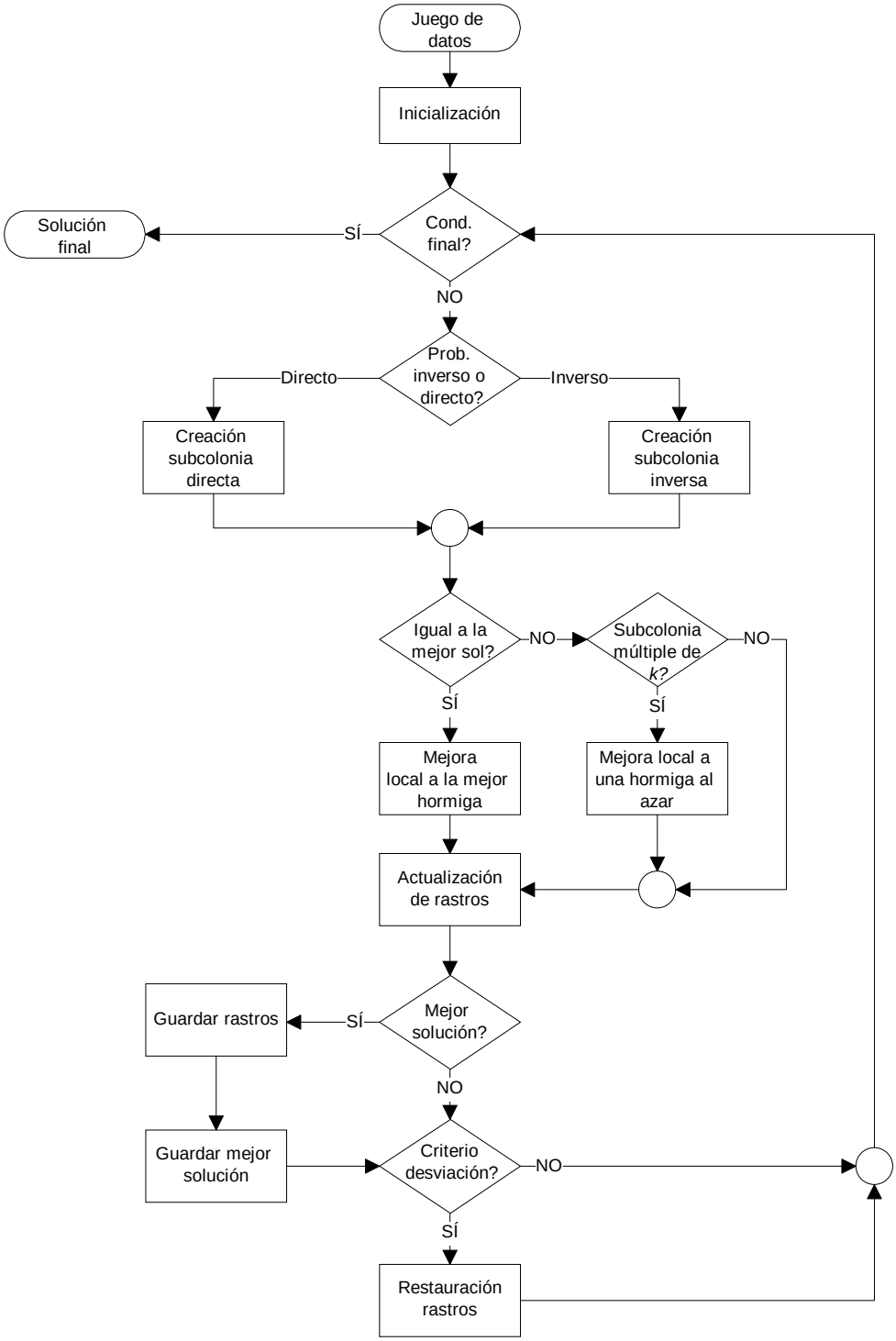


Figura 5.2.1

5.2.2 Funcionamiento de la mejora local

El funcionamiento general de la mejora local consistirá en, a partir de la secuencia de tareas dada por una hormiga, generar nuevas secuencias de tareas mediante intercambios lineales de vecinas, obteniendo la combinación local óptima de tiempo de ciclo y número de estaciones

para cada secuencia de tareas generada. Se escogerá como solución final de la mejora la mejor solución encontrada.

De forma general, consistirá en un proceso iterativo de la forma:

1. Elegir secuencia de tareas inicial
2. Evaluar la secuencia de tareas inicial
3. Generar nueva secuencia de tareas mediante un intercambio
4. Evaluar la nueva secuencia de tareas
5. Guardar la mejor secuencia de tareas encontrada hasta el momento
6. Si se cumple la condición final, acabar. En caso contrario, ir al paso 3.

Este proceso se explica en forma de diagrama en la figura 5.2.2.

Para formalizar el esquema de la mejora local que se empleará, se concretan los cuatros puntos enumerados en el apartado 3.4.3:

- *Entorno o vecindario*: conjunto de secuencias de tareas que se puedan generar a partir de intercambios binarios entre tareas consecutivas de la secuencia de tareas precedente.

- *Elección de la secuencia de tareas inicial*: secuencia de tareas generada por la hormiga a la que se aplica la mejora local.

- *Elección de la nueva secuencia de tareas en curso*: se aceptará una nueva secuencia de tareas como secuencia de tareas en curso si cumple la relación de precedencias y al ser evaluada se obtiene una función objetivo mejor o igual a la mejor encontrada hasta el momento.

- *Detención del proceso*: al superarse un número de intercambios fijado, al conseguir una solución con tiempo muerto cero o al haber hecho una pasada de intercambios sin aceptar ninguno.

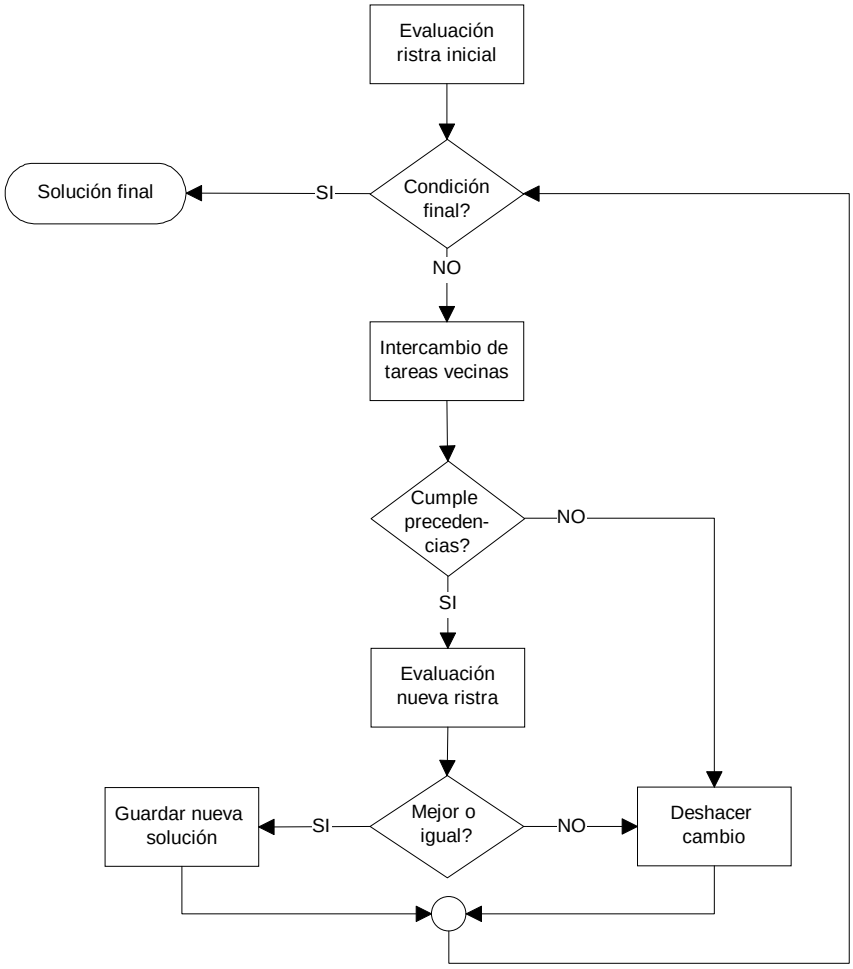


Figura 5.2.2

5.2.3 Generación de nuevas secuencias de tareas durante la mejora local

Las nuevas secuencia de tareas a evaluar se generan a partir de intercambios binarios entre *tareas consecutivas* de la última secuencia de tareas aceptada. La elección de la pareja de tareas a intercambiar se efectúa mediante un contador que recorre la secuencia de tareas de extremo a extremo y al llegar al final vuelve de nuevo al extremo inicial. La tarea que se encuentre en la misma posición que el marcador se intercambiará con su tarea consecutiva en la secuencia de tareas (la de la derecha en caso que se hagan intercambios de izquierda a derecha; la de la izquierda en caso de intercambios de derecha a izquierda).

El avance de este contador de posición es independiente de la aceptación o no de la secuencia de tareas. Cada intercambio que se haga, supone el avance del contador en una posición. Al llegar a la penúltima posición de la secuencia de tareas, el contador volverá de nuevo a la primera posición si no se cumple ninguna condición de final.

Se podría hacer que el contador volviese al inicio cada vez que se acepta una solución, pero si se independiza de este hecho, se consigue que las tareas avancen todo lo que puedan y salten, por decirlo así, a la siguiente estación.

Realmente, no existen estaciones, pues se trabaja con secuencia de tareas. Sin embargo, puesto que se trata de una mejora local, la mejor solución entre una secuencia de tareas y la siguiente (después de un intercambio binario) no puede diferir excesivamente en su número de estaciones.

Por lo tanto, al intercambiar dos tareas consecutivas, es probable que se muevan dentro de la misma estación de la mejor solución hallada para ambas, y sin mejorar de esta forma la función objetivo. Si se permite que el siguiente intercambio se haga sobre la misma tarea y su consecutiva, se podrá desplazar hasta saltar de estación, abriendo la posibilidad al cambio en la función objetivo.

5.2.4 Condición de final

Existen cuatro parámetros que controlan la detención del proceso: número de intercambios máximo, función objetivo igual a la cota mínima del SALBP-E e inmovilidad de la ristra. De los tres, los dos últimos detienen el proceso porque ya no es posible mejorar más y el primero es un parámetro que se define antes de empezar.

Número de intercambios: antes de comenzar la mejora, se fija el número de intercambios aceptados, o sea, el número de secuencias de tareas a evaluar. Para este trabajo, se ha fijado el número de intercambios a 2000.

Función objetivo igual a la cota mínima del SALBP-E: si durante el proceso de mejora se obtiene una solución igual a una cota mínima de problema *SALB-E* se habrá minimizado la función objetivo hasta el óptimo global y por lo tanto, no se podrá mejorar más, con lo que se detendrá el proceso.

Inmovilidad de la ristra: si no se acepta ningún cambio desde que el contador de posición está al inicio hasta que llega a la penúltima posición, se detendrá el proceso, pues en la siguiente pasada de cambios sucederá lo mismo. Sin embargo, esta detención no afecta al proceso total, como las dos anteriores, sino que detiene la mejora que se está efectuando hasta el momento y pasa a la siguiente dirección de intercambio o a la siguiente ristra inicial, según corresponda.

5.2.5 Evaluación de la solución

5.2.5.1 Funcionamiento general de la evaluación

En el apartado 5.1.4.4 “Evaluación de la secuencia”, se ha comentado el proceso de selección de los tiempos de ciclo a probar, así como el uso de cotas máximas y mínimas de tiempo para determinar el rango en el que se deben probar.

En las meta-heurísticas propuesta, cada secuencia de tareas es independiente de las otras. Por lo tanto, las cotas son únicas para cada una de ellas. En la mejora local, sin embargo, sólo se tienen en cuenta aquellas secuencias en las que se pueda obtener una función objetivo mejor o igual a la mejor hallada hasta el momento.

Por lo tanto, en cada una de las secuencias generadas, se tiene una función objetivo de referencia que se debe igualar o mejorar. Si durante la prueba de un tiempo de ciclo, se puede asegurar que la solución obtenida con el mismo no va a igualar o mejorar la mejor función objetivo hallada hasta el momento, se puede abortar la prueba y pasar al siguiente tiempo de ciclo.

Además, al tener una función objetivo de referencia para todas las secuencias, se puede ir ajustando la cota de tiempo de ciclo máximo.

5.2.5.2 Cotas dinámicas de fin de prueba de tiempo de ciclo

Una vez se ha decidido que tiempo de ciclo se fija, se puede abortar el proceso de evaluación de ese tiempo de ciclo y pasar al siguiente mediante el uso de unas cotas que se van ajustando a lo largo de la mejora (dinámicas).

La idea consiste en aprovechar que la función objetivo va mejorando a lo largo del proceso. Al probar un tiempo de ciclo, este puede abortarse mediante el uso de dos cotas: *cota de función objetivo absoluta y cota de función objetivo estimada*.

Cota de función objetivo absoluta: durante el proceso de prueba de un tiempo de ciclo, se van llenando estaciones. Cuando una estación está llena, se abre la siguiente. En ese momento, si el número de estaciones abiertas por el tiempo de ciclo ajustado mayor hasta el momento es superior a la función objetivo mejor que se ha encontrado en todo el proceso de mejora hasta el momento, se puede afirmar que el tiempo de ciclo que se está probando en la ristra no va a mejorar ni igualar la función objetivo, pues el número de estaciones y el tiempo de ciclo ajustado serán iguales o mayores que los actuales.

Cota de función objetivo estimada: si en la cota anterior se aprovechaba que ya se sabía cuantas estaciones estaban abiertas, en ésta, la idea es calcular una cota mínima de estaciones que aun quedan por abrir cuando se sabe las que ya están abiertas.

Si se han abierto *k+1* estaciones, entonces hay *k* estaciones llenas. Si $t_{restante}$ es la suma del tiempo de las tareas que aun no han sido asignadas, el número mínimo de estaciones que aún se deben abrir (sin tener en cuenta la que está abierta y vacía) es $n_{est} = \frac{t_{restante}}{TC2} + k$. Se usa *TC2* y no el tiempo ajustado, ya que el ajustado puede aumentar. Por lo tanto, al estar dividiendo, se elige el mayor, o sea *TC2*, que puede ser considerada una cota superior de *Tcajust* (apartado 5.1.4.4).

De esta forma, se puede calcular la cota mínima de la función objetivo:

$$n_{est} \cdot TCajust$$

Se usa *TCajust* y no *TC2* pues al estar multiplicando se debe escoger la menor. Si en el transcurso de la evaluación de una ristra con un tiempo de ciclo fijado, la cota mínima de la

función objetivo es mayor que la mejor función objetivo hallada hasta el momento, se puede afirmar que ese tiempo de ciclo con esa ristra no mejorará la función objetivo y por lo tanto no es necesario seguir probándolo, con lo que se puede pasar al siguiente tiempo de ciclo.

El esquema de la evaluación en forma de diagrama se encuentra en la figura 5.2.5.2, donde *obj0* es la mejor función objetivo hallada hasta el momento, y *obj2* la función objetivo que se obtiene con *TC2*.

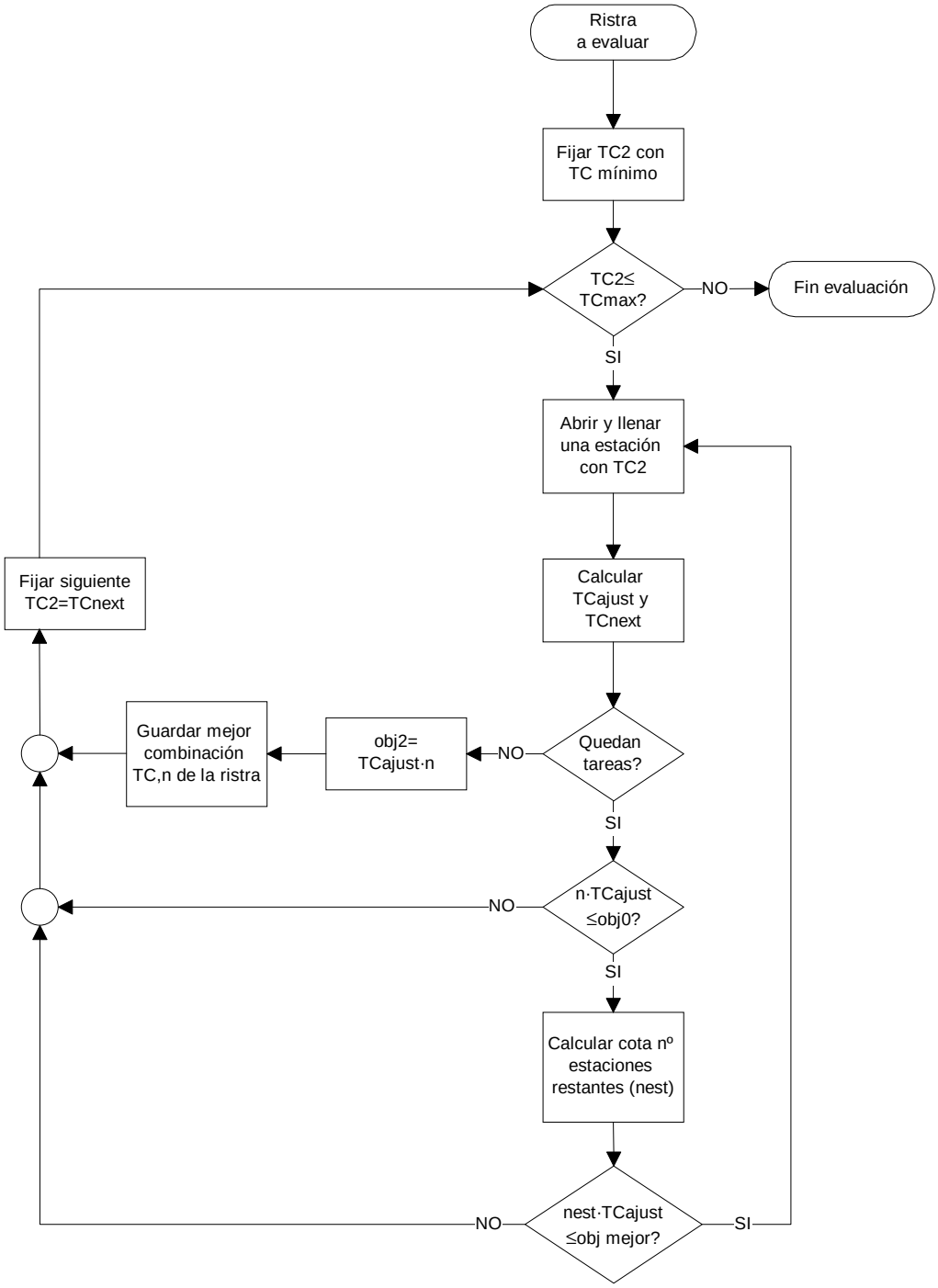


Figura 5.2.5.2

5.2.5.3 Ajuste del tiempo de ciclo máximo y condiciones de fin de evaluación

Las cotas dinámicas abortan la prueba de un tiempo de ciclo para pasar al siguiente. Al margen del uso de estas cotas, hay condiciones en que se puede finalizar la evaluación de una secuencia de tareas, y que, además sirven para mejorar la cota de tiempo de ciclo máximo.

La idea es detener la evaluación de la secuencia de tareas en el momento en el que un tiempo de ciclo obtenga un número de estaciones igual o inferior al mínimo. Dado que los tiempos de ciclo van aumentando, si se alcanza una solución con un número de estaciones inferior al mínimo, si se sigue aumentando el tiempo de ciclo no se conseguirá ninguna solución de al menos un número de estaciones mínimo.

Por otro lado, si se obtiene una solución con un número de estaciones igual al mínimo, como ya no se puede reducir más, no tiene sentido probar tiempos de ciclo mayores, y se detiene la evaluación. Pero ésta no es su única función.

Aprovechando que se tiene el menor tiempo de ciclo que da el número de estaciones mínimo para la secuencia de tareas, éste se puede tomar como nueva cota de tiempo de ciclo máximo si es menor que la que se tenía. De este modo, se va actualizando a medida que avanza el proceso.

Para simplificar su representación, su diagrama de funcionamiento se ha representado sin tener en cuenta la interacción con el uso de las cotas dinámicas en la figura 5.2.5.3

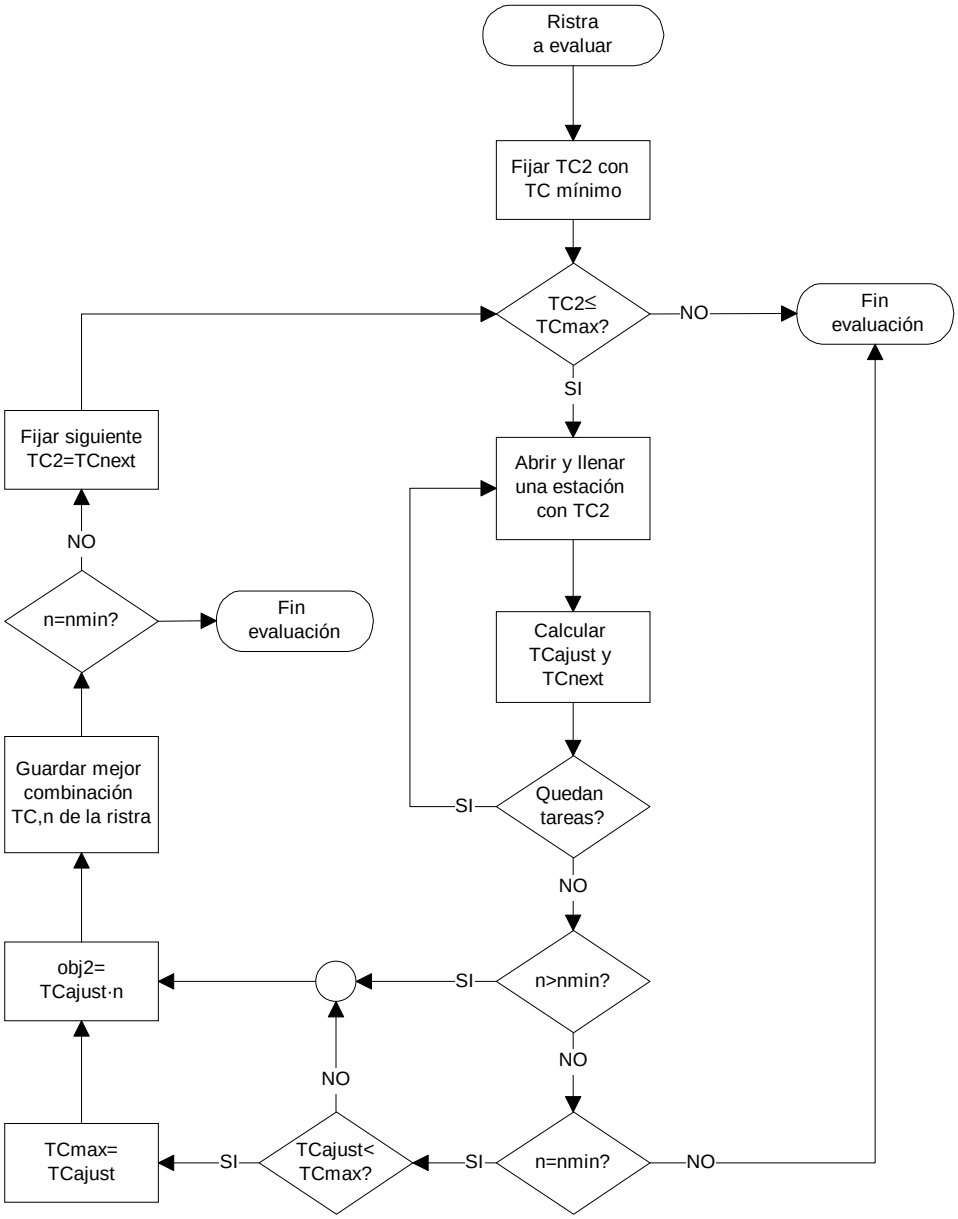


Figura 5.2.5.3

5.2.5.4 Uso conjunto de condiciones de fin de tiempo de ciclo y fin de evaluación

Si se usan las condiciones de fin de evaluación y fin de tiempo de ciclo conjuntamente, surge el problema de que al abortar la prueba de un tiempo de ciclo no se obtiene el número de estaciones que le corresponden y por lo tanto se complica el uso de las condiciones de fin de evaluación, pues se basan en el número de estaciones.

Si no se aborta la prueba del tiempo de ciclo (se asignan todas las tareas), simplemente se decide si seguir evaluando o no según lo visto en el apartado 5.2.5.3. En caso de continuar con la evaluación, se prueba el siguiente tiempo de ciclo tal y como se describe en el apartado 5.2.5.2.

Si se aborta la prueba de tiempo de ciclo, pueden surgir dos opciones: el número de estaciones sea igual al mínimo o diferente.

Número de estaciones igual a n_{min}: como el proceso se ha abortado, no se sabe si el número de estaciones abiertas iba a aumentar. Se comprueba si la suma de los tiempos de las estaciones restantes es superior al tiempo de ciclo. En caso de no ser superior, el número de estaciones será igual a n y por lo tanto, se podrá actualizar la cota de tiempo de ciclo máximo, el tiempo de ciclo ajustado y finalizar la evaluación. Si es superior, se aumenta el tiempo de ciclo.

Número de estaciones diferente a n_{min}: si es superior a n_{min}, significa que, al menos, ya habían abiertas más estaciones que las mínimas y aun se puede aumentar el tiempo de ciclo. Si es inferior a n_{min}, como no se puede saber cuantas habrían en realidad también se aumenta el tiempo de ciclo.

El procedimiento se puede apreciar en forma de diagrama en la figura 5.2.5.4

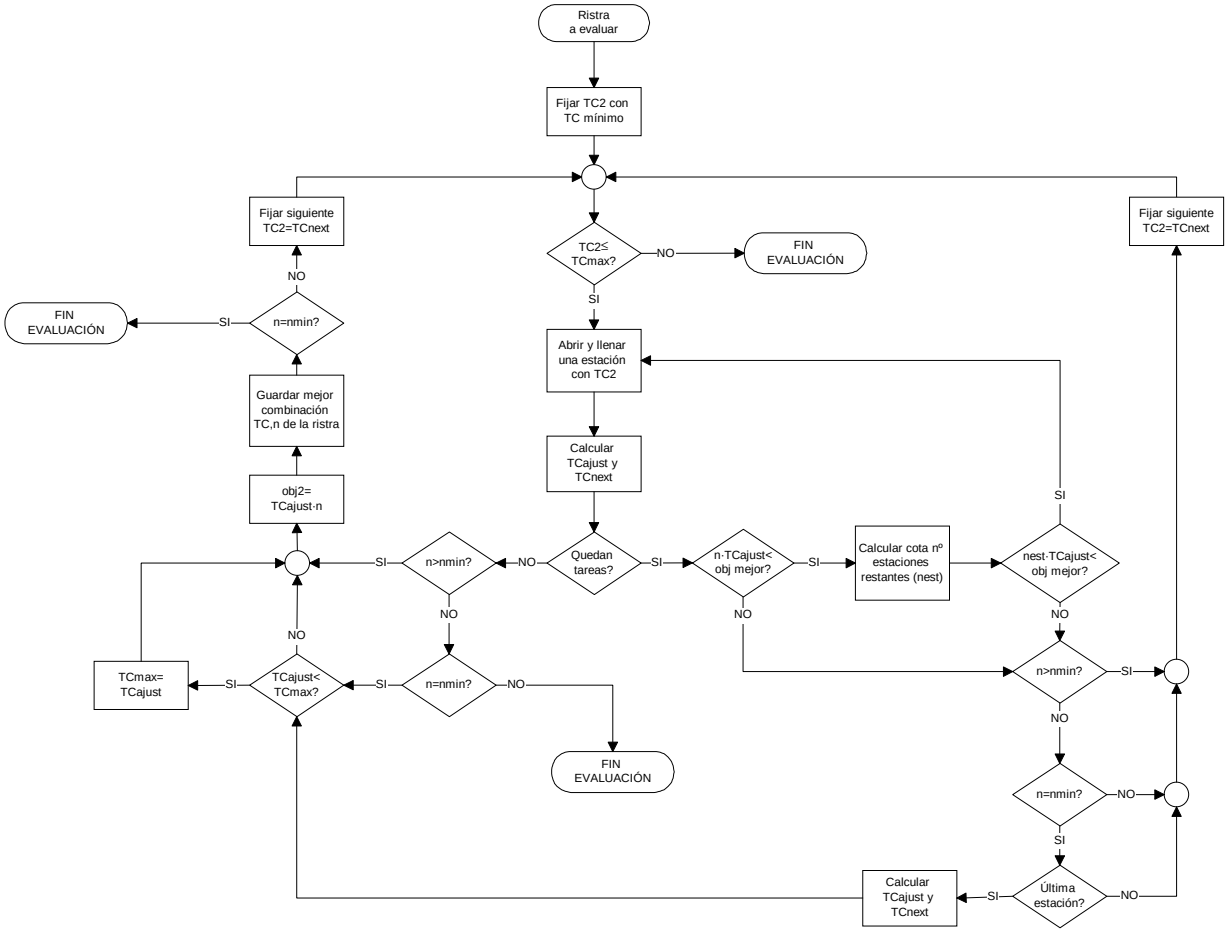


Figura 5.2.5.4

CAPÍTULO 6: ***EXPERIENCIA COMPUTACIONAL***

En este capítulo se describe la experiencia computacional realizada para evaluar la eficiencia de los métodos propuestos en el capítulo 5. El capítulo consta de tres partes, en la primera se describe la experiencia computacional, los objetivos que se persiguen, así como un resumen de los resultados obtenidos. En la segunda parte se describe el entorno computacional. Y finalmente, en la tercera parte se resume la experiencia computacional realizada con el juego de datos reales de la empresa Rieju.

6.1 DESCRIPCION DE LAS PRUEBAS

6.1.1 Introducción

En este capítulo se describe la experiencia computacional llevada a cabo para estudiar el comportamiento de los dos métodos heurísticos propuestos para la resolución del SALBP-E. Para poder contrastar los resultados, se ha diseñado un método de resolución de referencia con el que poder comparar los resultados obtenidos.

Dicho método es un procedimiento *multi-start* que utiliza como método de exploración de entornos la mejora local del apartado 5.2.2. Básicamente, su funcionamiento consiste en construir secuencias de tareas de forma aleatoria a las cuales se le aplica una mejora local mediante intercambios lineales. La evaluación de las secuencias es la misma que utilizan los dos algoritmos de hormigas propuestos en el capítulo 5.

La experiencia computacional se ha dividido en dos partes. En el apartado 6.1.2 se resumen los resultados obtenidos fijando, para los tres métodos heurísticos, el número de ristras a evaluar. Con esta experiencia se pretende comprobar la eficacia de los métodos basados el la meta-heurística ACO. En el siguiente apartado se fija el tiempo disponible por las heurísticas para estudiar su eficiencia. Finalmente, en el apartado 6.2 se describe tanto el lenguaje de programación utilizado, Java, así como las máquinas que se han utilizado para la experiencia.

6.1.2 Pruebas con el número de soluciones fijado

La experiencia computacional fijando el número de soluciones ha constado de 256 juegos de datos con diferentes grafos de precedencia y/o diferentes combinaciones del número mínimo de estaciones (n_{min}) y número máximo de estaciones (n_{max}). Cada uno de estos juegos de datos han sido solucionados 10 veces, lo cual suma un total de 2560 pruebas independientes. El número de soluciones que se ha fijado es de 84000.

Los resultados obtenidos para cada juego de datos se encuentran en los anexos I y II. En la tabla 6.1.2 se resumen estos resultados. El número de óptimos en tanto por cien se calcula respecto al total de pruebas individuales, cada una de las 10 soluciones obtenidas para un mismo juego de problema se considerarán independientes para calcular este porcentaje. En todos los valores de la tabla se considera el valor medio de todos los problemas pertenecientes a su grupo.

Grupo de juego de datos	Tipo de prueba	nº óptimos [%]	% discrepancia al óptimo			Tiempo medio [s]
			Min	Media	Max	
Grupo I	Multi-Start	70,7	0,14	0,31	0,73	23
	Hormigas	72,4	0,06	0,32	0,55	26
	Hormigas+Mejora	79,3	0,06	0,20	0,42	23
Grupo II	Multi-Start	45,4	0,85	1,24	1,72	40
	Hormigas	44,6	0,76	1,35	1,82	68
	Hormigas+Mejora	54,1	0,49	0,91	1,43	59
Grupo III	Multi-Start	13,6	1,31	1,91	2,46	67
	Hormigas	16,7	1,34	2,07	2,74	159
	Hormigas+Mejora	21,0	0,83	1,28	1,81	109
Grupo IV	Multi-Start	0,8	2,71	3,32	3,90	119
	Hormigas	2,2	2,64	3,49	4,24	392
	Hormigas+Mejora	2,4	1,78	2,54	3,27	208
Grupo V	Multi-Start	0,0	2,04	2,46	2,82	371
	Hormigas	0,0	1,83	2,30	2,78	884
	Hormigas+Mejora	0,0	1,36	1,96	2,46	675

Tabla 6.1.2

6.1.3 Pruebas con el tiempo de resolución fijado

La experiencia computacional fijando el tiempo disponible para cada grupo de problemas (ver apartado 6.2.3) ha constado del mismo juego de datos compuesto por 256 juegos de datos. Cada uno de estos juegos de datos han sido solucionados 10 veces, lo cual suma un total de 2560 pruebas independientes. Los tiempos que se han fijado han sido de 1 minuto para el grupo I, 2 para el grupo II, 4 para el grupo III, y 8 minutos para el grupo IV y V.

Los resultados obtenidos para cada juego de datos se encuentran en los anexos I y II. En la tabla 6.1.3 se resumen estos resultados fijado el tiempo de resolución. La tabla es equivalente a la que aparece en el apartado anterior.

Grupo de juego de datos	Tipo de prueba	nº óptimos [%]	% discrepancia al óptimo			Tiempo medio [s]
			Min	Media	Max	
Grupo I	Multi-Start	78,1	0,11	0,17	0,30	44
	Hormigas	70,5	0,19	0,35	0,47	20
	Hormigas+Mejora	76,4	0,06	0,24	0,42	48
Grupo II	Multi-Start	51,3	0,76	1,04	1,35	97
	Hormigas	45,9	0,69	1,31	1,86	68
	Hormigas+Mejora	53,1	0,48	0,88	1,36	106
Grupo III	Multi-Start	19,2	1,14	1,49	1,97	216
	Hormigas	17,0	1,26	2,02	2,93	208
	Hormigas+Mejora	20,7	0,82	1,21	1,70	238
Grupo IV	Multi-Start	1,0	2,46	2,98	3,42	475
	Hormigas	2,0	2,67	3,48	4,26	472
	Hormigas+Mejora	1,4	1,65	2,41	3,10	479
Grupo V	Multi-Start	0,0	1,72	2,14	2,48	480
	Hormigas	0,0	1,68	2,08	2,48	480
	Hormigas+Mejora	0,0	1,30	1,79	2,27	480

Tabla 6.1.3

6.2 ENTORNO COMPUTACIONAL

6.2.1 El lenguaje Java

Para programar las dos meta-heurísticas propuestas en el capítulo 5, así como la heurística *multi-start* se ha utilizado en lenguaje de programación Java, usando como compilador y máquina virtual la versión 1.2 de las JDK de Sun.

Las características principales que ofrece Java respecto a otros lenguajes de programación, son:

- Java ofrece toda la *funcionalidad* de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos (aritmética de punteros, no existen referencias, registros, necesidad de liberar memoria, etc.).

- Java es un lenguaje *orientado a objetos*, trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

- Java es un leguaje *distribuido* y permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

- El lenguaje Java es *interpretado*. Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.

6.2.2 Maquinas utilizadas

El ordenador utilizado para llevar a cabo la experiencia computacional es una máquina Sun Ultra Enterprise 450 cuyas características técnicas son: 4 procesadores SuperSparc de 400MHz y 1 GB de memoria RAM con el sistema operativo Unix SUN/SOLARIS 2.7.

6.2.3 Juego de datos

El juego de datos que se ha utilizado es una colección de grafos de precedencias que aparece en *Scholl (1999)*. La descripción exacta de cada uno de los grafos, así como toda la información referente a las soluciones se puede encontrar en la siguiente dirección de Internet:

http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/alb/salbgdat.htm

El juego de datos completo está formado por 24 grafos de precedencias diferentes , no obstante, para cada grafo se pueden fijar diferentes combinaciones del número de estaciones

mínimo (n_{min}) y máximo (n_{max}), con lo que el total de juegos de datos que se prueban es 256. Las combinaciones de n_{max} y n_{min} que se utilizan ya están resueltas de forma óptima en *Scholl (1999)* o se conoce la mejor solución encontrada hasta el momento. De esta forma se tiene una referencia que permite contrastar la calidad de los resultados obtenidos mediante los procedimientos expuestos en este trabajo.

En la tabla 7.1 se describen las características básicas de los 24 grafos especificando el número de tareas, el tiempo de tarea mínimo (t_{min}) y máximo (t_{max}), y por último la suma total de los tiempos de todas las tareas (t_{sum}).

Nombre	***n***	t_{min}	t_{max}	t_{sum}
Arcus1	83	233	3691	75707
Arcus2	111	10	5689	150399
Barthold	148	3	383	5634
Barthol2	148	1	83	4234
Bowman	8	3	17	75
Buxey	29	1	25	324
Gunther	35	1	40	483
Hahn	53	40	1775	14026
Heskiaoff	28	1	108	1024
Jackson	11	1	7	46
Kilbridge	45	3	55	552
Lutz1	32	100	1400	14140
Lutz2	89	1	10	485
Lutz3	89	1	74	1644
Mansoor	11	2	45	185
Mertens	7	1	6	29
Mitchell	21	1	13	105
Mukherje	94	8	171	4208
Roszieg	25	1	13	125
Sawyer	30	1	25	324
Scholl	297	5	1386	69655
Tonge	70	1	156	3510
Warnecke	58	7	53	1548
Wee-Mag	75	2	27	1499

Para la realización de la experiencia computacional se han dividido los 24 grafos de precedencias en 5 grupos en función del número de tareas. En la tabla siguiente se describen estos grupos:

Grupo	***Tareas***	***Juego de datos***
I	7-30	Mertens-Sawyer
II	32-58	Lutz1-Warnecke
III	70-89	Tonge-Lutz3
IV	94-148	Arcus2-Barthol2
V	297	Scholl

6.3 RIEJU

6.3.1 Características del juego de datos

Para evaluar el funcionamiento de la meta-heurística propuesta en este trabajo, se utilizará el juego de datos real utilizado por *Riera (1997)* y *Carreras-Candi y Domingo (1997)* en sus respectivos proyectos final de carrera. Los datos pertenecen a la línea de montaje del modelo de ciclomotor *First* de la fábrica Rieju, S.A. en Figueras.

El problema de la factoría Rieju posee varias características que hacen que no se pueda enmarcar completamente dentro del problema SALB-E. Concretamente, existen incompatibilidades entre tareas, así como la necesidad de minimizar el número de giros en la línea, es decir, minimizar el número de veces que se ha de cambiar la posición del producto para poder realizar la tarea correspondiente.

Para adaptar el juego de datos a las características del problema que se estudia en este trabajo, se han tenido que relajar algunas de las restricciones anteriores. De esta forma, se han mantenido todas las restricciones de precedencia, los tiempos de ejecución, número de operarios máximo y mínimo, límites del tiempo de ciclo impuestos por la tasa de producción requerida. Sin embargo, se han eliminado las incompatibilidades entre tareas y la minimización del número de giros dentro de una estación.

En el anexo III se encuentra más información sobre la empresa, la descripción de las tareas, la simplificación llevada a cabo por *Plans (ver Ferrer (2000))* y los resultados obtenidos.

6.3.2 Resultados

A continuación se resumen los resultados obtenidos fijando el número de estaciones mínimo y máximo a 9 y 12, respectivamente. De la misma forma que en el resto de la experiencia computacional, el número de pruebas por juego de datos y tipo de prueba ha sido de 10 con un tiempo de ejecución máximo de 30 minutos.

Tipo de juego de datos	Tipo de prueba	% discrepancia a la cota			Tiempo medio [s]
		Min	Media	Max	
Com-pleto	Multi-Start	0,286	0,423	0,519	1800
	Hormigas	0,231	0,426	0,646	1800
	Hormigas+Mejora	0,176	0,342	0,529	1800
Simpli-ficado	Multi-Start	0,218	0,347	0,461	1800
	Hormigas	0,235	0,399	0,552	1800
	Hormigas+Mejora	0,144	0,279	0,416	1800

CAPÍTULO 7: ***CONCLUSIONES***

En este capítulo se exponen las conclusiones del trabajo realizado, y que se han dividido en dos partes. La primera parte contiene las conclusiones que se pueden extraer de la aplicación de los algoritmos de hormigas a la resolución del problema de equilibrado SALBP-E. Y la segunda parte se centra en los resultados de la experiencia computacional, así como la comparativa de los métodos heurísticos probados.

7.1 LOS ALGORITMOS ACO

Los algoritmos ACO (*Ant Colony Optimization*) se presentan como un campo de investigación novedoso y prometedora. Como ya se ha comentado, la meta-heurística ACO pertenece al grupo de las denominadas meta-heurísticas estocásticas como pueden ser los Algoritmos Genéticos, el Recocido Simulado, la Búsqueda Tabú o las Redes Neuronales, etc. que se caracterizan por utilizar analogías o principios básicos de fenómenos naturales.

Este nuevo método de búsqueda se basa en un proceso autocatalítico y distribuido. La idea general de este tipo de algoritmos es guiar una población de agentes mediante un proceso autocatalítico dirigido por un criterio heurístico. Si los individuos no interaccionan entre ellos, el proceso autocatalítico y el criterio heurístico hacen que el individuo converja hacia soluciones subóptimas en tiempo exponencial. Cuando los agentes interaccionan entre ellos, el criterio heurístico encamina el proceso autocatalítico, y facilita la convergencia hacia soluciones de gran calidad sin "caer" en óptimos locales. Ninguna solución se excluye completamente, aunque la probabilidad de explorar subespacios de soluciones de poca calidad se reduce.

Las principales contribuciones de estos algoritmos son:

- La simulación de diferentes aspectos de sistemas naturales como la *selección* (correspondiente a la optimización) y la *mutación* (correspondiente a la búsqueda aleatoria).
- El empleo de *feed-back* positivo, o autocatálisis, como herramienta para resolver problemas de optimización.
- La demostración de que la sinergia puede ser muy útil en sistemas distribuidos. La efectividad de la exploración que llevan a cabo los algoritmos ACO, donde los individuos interaccionan entre ellos, es superior a una búsqueda llevada a cabo por el mismo número de individuos independientes entre sí.
- Al igual que otras meta-heurísticas, como pueden ser los algoritmos genéticos, se explota al máximo la capacidad de utilizar/aprovechar las mejores soluciones encontradas en el pasado.
- A parte de su eficiencia, una de las principales ventajas de estos algoritmos es su fácil implementación para la resolución de muchos problemas de optimización diferentes, ya que se basan en una idea general y sencilla de adaptar a las diferentes características del problema.

7.2 CONCLUSIONES DE LA EXPERIENCIA COMPUTACIONAL

7.2.1 Heurística de referencia

La mayoría de métodos heurísticos y exactos que se encuentran en la literatura dedicados al estudio de los problemas SALB se centran sobre todo en sus versiones 1 y 2. No es difícil encontrar gran cantidad de procedimientos y tablas de resultados con las que poder comparar nuevos métodos.

Desgraciadamente, el problema SALB-E no tiene el mismo nivel de documentación, por lo que se hace difícil encontrar métodos heurísticos contrastados con los que poder comparar los métodos propuestos en este trabajo.

Existen algunos métodos constructivos greedy basados en la fijación de diferentes combinaciones de tiempo de ciclo y número de estaciones, como en *Scholl (1999)*, pero no se ha podido disponer de ningún estudio basado en la exploración de entornos para el SALBP-E.

Por este motivo, se ha decidido implementar un procedimiento heurístico de referencia con el cual poder comparar el funcionamiento de los algoritmos de hormigas adaptados al SALBP-E.

El algoritmo escogido es un procedimiento *multi-start*, ya que se trata de un método heurístico sencillo de implementar, rápido y eficaz para la resolución de problemas combinatorios. Para hacer más equitativa la comparación, el funcionamiento del *multi-start* utiliza las mismas herramientas de evaluación y mejora local que las heurísticas propuestas.

Por lo tanto, el método de referencia empleado es por sí mismo una heurística de considerable potencia.

7.2.2 Evaluación de la solución

Tal y como se describe en el apartado 5.2.5, el método de evaluación empleado está pensado para poder explorar cualquier combinación posible de tiempo de ciclo y número de estaciones, dada una secuencia de tareas.

La idea es poder desligar la resolución del problema de un número de estaciones o tiempo de ciclo concretos. De esta forma, se hace sencillo el intercambio de tareas dentro de la mejora local y se facilita la transformación de una solución a rastro, pues primero se construye la secuencia de tareas y después se busca la combinación de tiempo de ciclo y número de estaciones óptima en lugar de fijar alguno de los dos parámetros y construir a partir de él una solución.

Por lo tanto, el método de evaluación propuesto es apropiado para la exploración de entornos y permite una exploración completa de posibles combinaciones factibles dada una secuencia de tareas.

7.2.3 Optimización de los parámetros y backtracking

Habitualmente, el tiempo requerido para ajustar y optimizar los parámetros de un algoritmo es mayor que el tiempo dedicado a su implementación, sobre todo en aquellas meta-heurísticas (tales como recocido simulado, tabu search, algoritmos genéticos, etc.) en las que la fijación de parámetros influye de manera notable en su funcionamiento.

Los algoritmos de hormigas pertenecen a este tipo de meta-heurísticas. Dependiendo de los parámetros de control definidos a lo largo del capítulo 5, se ha comprobado que su comportamiento evolutivo varía considerablemente, por lo que hay que ajustar dichos parámetros garantizar un comportamiento adecuado.

Estudiando la mejor solución de cada subcolonia (que es la que deja el rastro definitivamente), se han observado dos problemas en evolución de la solución en curso:

- *Excesiva exploración en detrimento del rastro*: se caracteriza por el no aprovechamiento de la información histórica de la colonia. Como se puede ver en el gráfico 7.2.3 (a) cuando una hormiga mejora la mejor solución global las siguientes hormigas no son capaces de seguir su rastro.
- *Excesiva explotación en detrimento de la exploración*: se caracteriza por un estancamiento prematuro de la mejor solución global con un valor de la función objetivo de no muy buena calidad. Como se observa en el gráfico 7.2.3 (b), ya no hay exploración desde las primeras iteraciones.

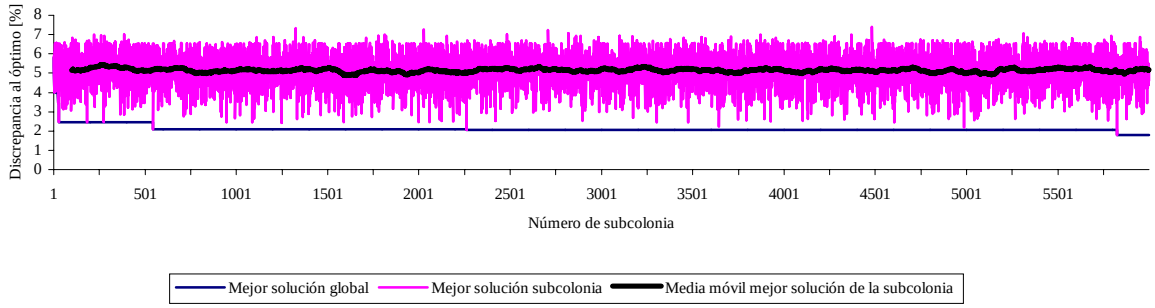


Gráfico 7.2.3 (a)

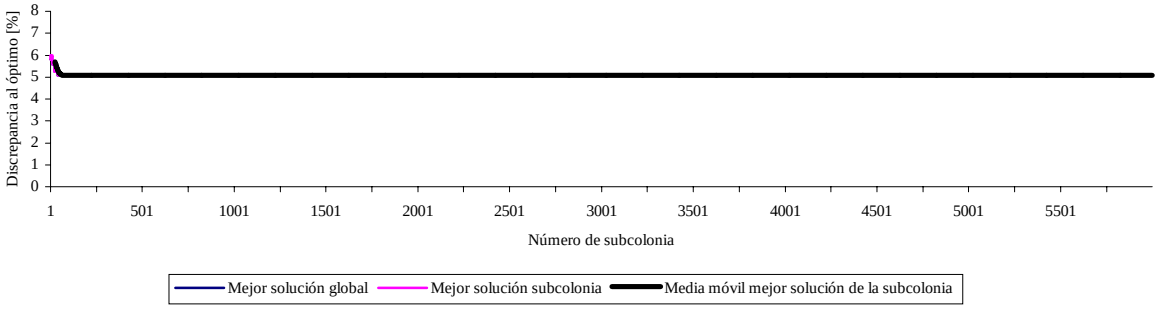


Gráfico 7.2.3 (b)

Las pruebas realizadas con más de 500 combinaciones de parámetros (ver apartado 5.1.6) han proporcionado unos valores empíricos que garantizan un buen funcionamiento para la gran mayoría de los juegos de datos probados.

El objetivo no es buscar la combinación óptima, sino encontrar un rango en el que las hormigas se comporten de forma robusta y eficiente independientemente del juego de datos que se emplea.

Una vez se han encontrado los parámetros que proporcionan un equilibrio entre exploración y explotación, se ha detectado en algunas ocasiones un comportamiento no deseado a largo plazo (apartado 5.1.3.3):

- *Pérdida de rastro*: cuando no se mejora la mejor solución global obtenida hasta el momento durante un número de iteraciones elevado, y debido a la evaporación acumulada del rastro, las hormigas pueden llegar salir del mejor espacio de soluciones. Como muestra el gráfico 7.2.3 (c), el comportamiento es el deseado mientras se producen mejoras en la solución global. Sin embargo, llegado a un punto, las hormigas progresivamente van abandonando el espacio de soluciones donde se encuentra la mejor solución hallada.

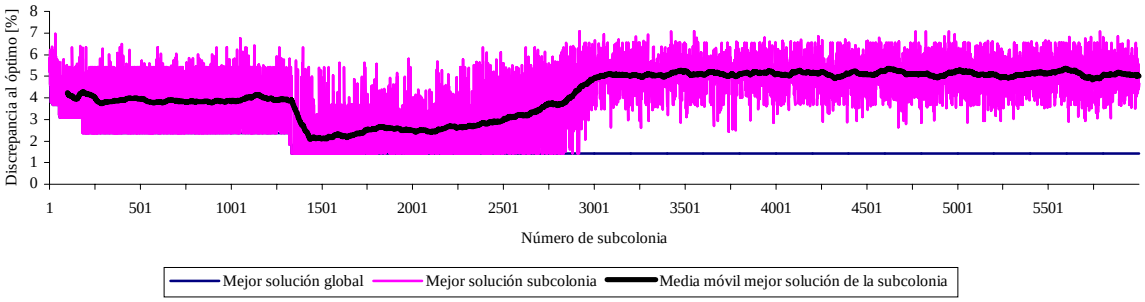


Gráfico 7.2.3 (c)

Para solucionar este problema, se ha optado por un procedimiento de *backtracking* (ver apartado 5.1.3.4) que garantiza que, en caso de una pérdida como la comentada, se regenera la matriz de rastros, de forma que las hormigas vuelven al mejor espacio de soluciones encontrado hasta el momento, tal y como se puede ver en el gráfico 7.2.3 (d).

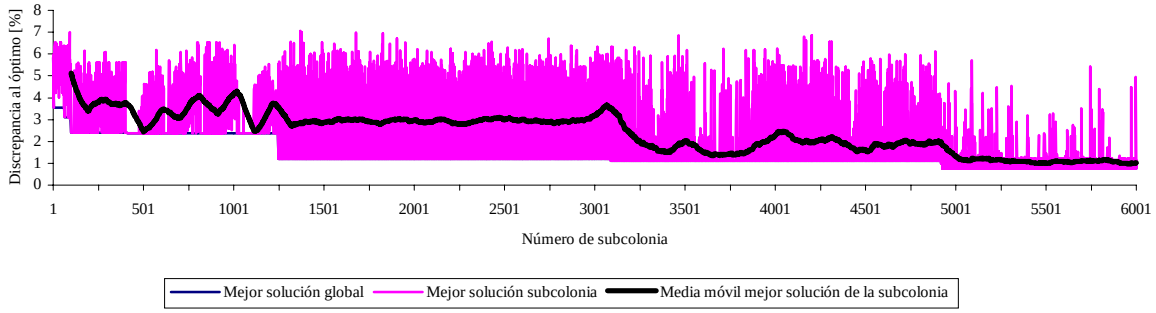


Gráfico 7.2.3 (d)

7.2.4 Comparación de métodos heurísticos

7.2.4.1 Pruebas con número de secuencias a evaluar fijado

Fijando el número de secuencias a evaluar, se pretende comparar los diferentes métodos heurísticos sin tener en cuenta su rapidez, sino su eficacia. Los tres métodos heurísticos han dispuesto de 84000 secuencias de tareas a evaluar (ver apartado 6.1.2).

Si se analizan los resultados de la tabla 6.1.2, se puede observar que el algoritmo de hormigas para el SALBP-E sin mejora local obtiene, de forma general, unos resultados similares a los de la heurística *multi-start* de referencia.

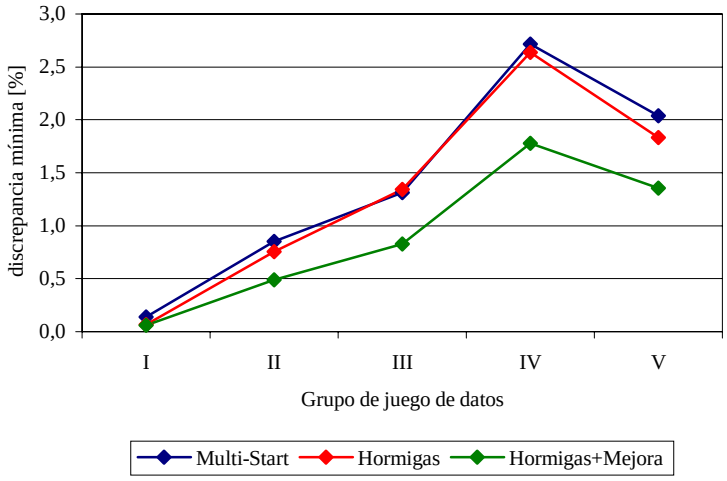
El algoritmo de hormigas sin mejora local basa su capacidad de optimización en un equilibrio entre exploración y explotación. En cambio, el *multi-start* tiene una capacidad de explotación superior, y se beneficia de una solución inicial que no es totalmente aleatoria, pues está condicionada por el grafo de precedencias.

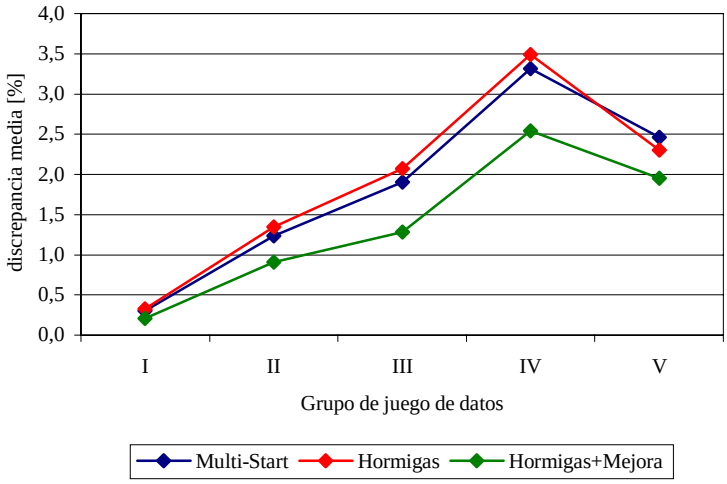
Comparando estos resultados con los obtenidos en otros trabajos sobre algoritmos de hormigas aplicados a problemas combinatorios, y donde también se ha utilizado como algoritmo de referencia una heurística *multi-start* (*Gambardella et al (1999)*), se puede concluir que el algoritmo de hormigas sin mejora local propuesto para el SALBP-E tiene unos buenos resultados.

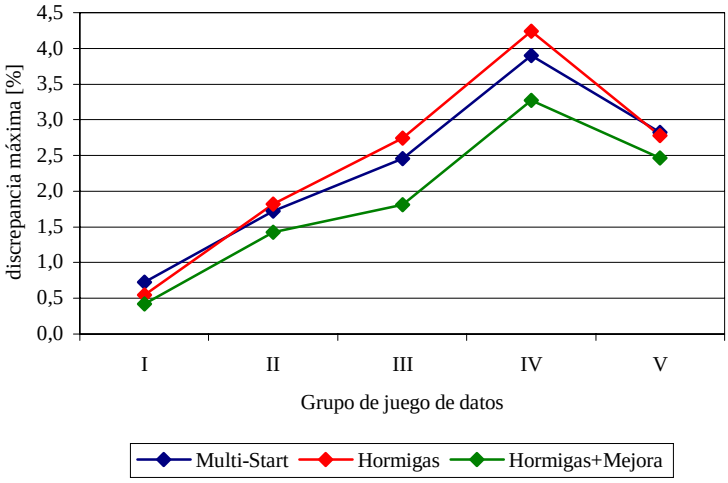
Para compensar la falta de explotación del algoritmo de hormigas sin alterar el equilibrio necesario para su funcionamiento, se ha añadido una mejora local puntual en algunas de las hormigas (ver apartado 5.2.2).

En todas la pruebas realizadas, ésta ha sido la mejor heurística de las tres, tanto en número de óptimos, como en discrepancia mínima, media y máxima respecto al óptimo.

En los gráficos 7.2.4.1 (a), (b), (c) y (d) se muestran gráficamente los resultados de la tabla 6.1.2:







Gráficos 7.2.4.1 (a), (b) y (c)

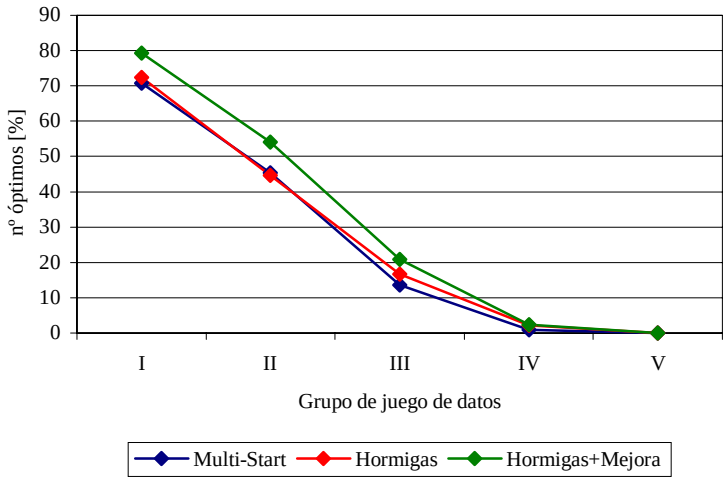


Gráfico 7.2.4.1 (d)

En el gráfico 7.2.4.1 (e), se compara el tiempo de ejecución de los tres métodos heurísticos:

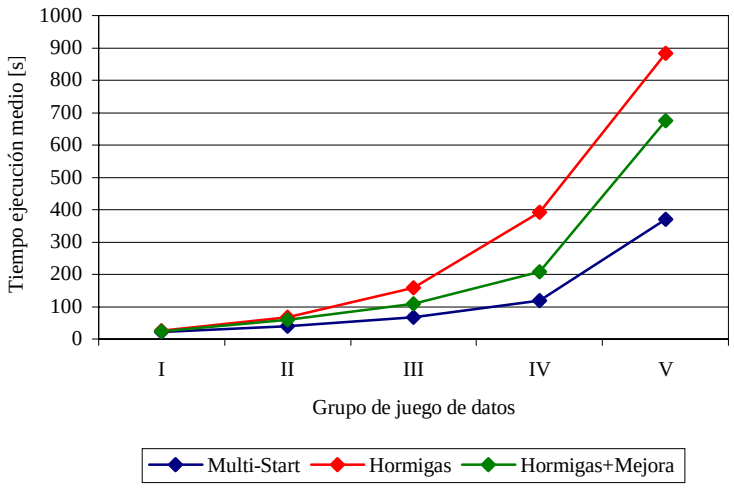


Gráfico 7.2.4.1 (e)

Como era de esperar, el tiempo de ejecución crece de forma exponencial al número de tareas del juego de datos. Además, para cada grupo de problemas, el método más rápido, fijado el número de secuencias, es el *multi-start*. Esto es debido a que gran parte del coste temporal al ejecutar el programa implementado se consume en el procesos aleatorios.

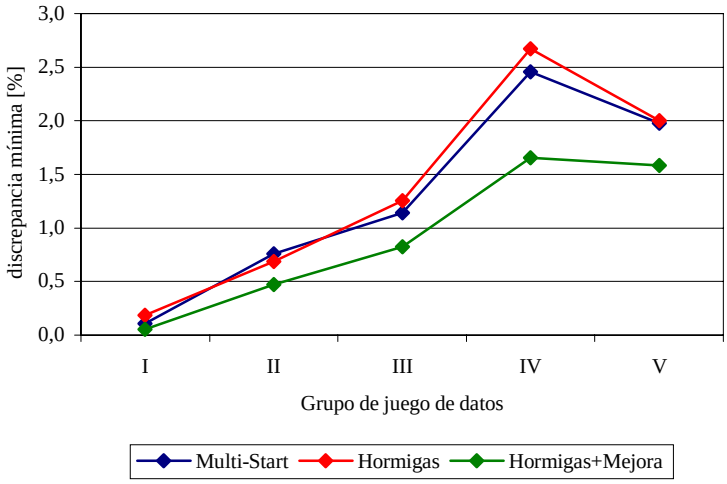
El algoritmo de hormigas sin mejora local es quien más procesos aleatorios utiliza para generar una secuencia de tareas y por tanto, el más lento. En este sentido, el algoritmo de hormigas con mejora local, dado que un parte de las secuencias se generan mediante intercambios y no procedimientos aleatorios, es más rápido que el algoritmo de hormigas sin mejora local, aunque más lento que el *multi-start*.

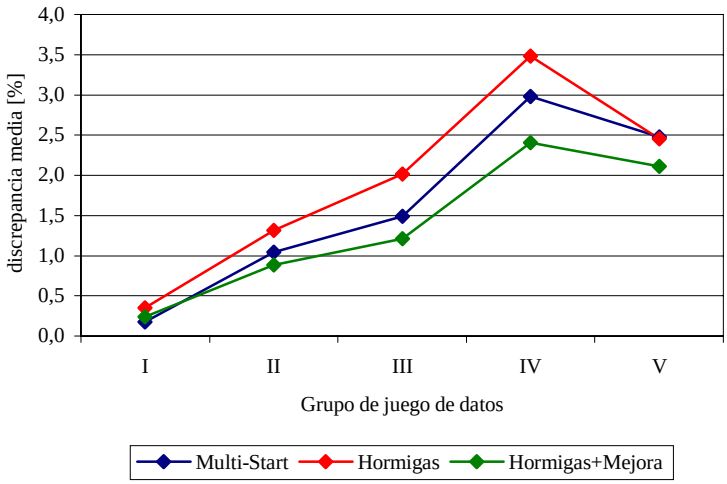
7.2.4.2 Pruebas con tiempo de ejecución fijado

En la prueba anterior, se han comparado los tres métodos fijando el número de secuencias evaluadas hasta el rango en que las hormigas llegan a estancarse. En esta prueba, se ha fijado el tiempo de ejecución y no el número de secuencias, dejando además un tiempo suficiente para que las hormigas sobrepasen el rango de estancamiento anterior.

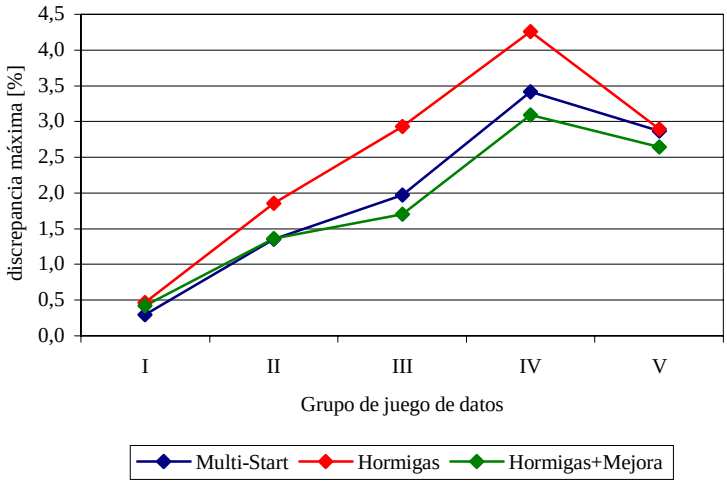
De esta forma, se puede garantizar que los algoritmos de hormigas llegan a su máxima capacidad de exploración antes de finalizar la prueba, mientras que el multi-start, dada su filosofía de funcionamiento, no tiene esta limitación.

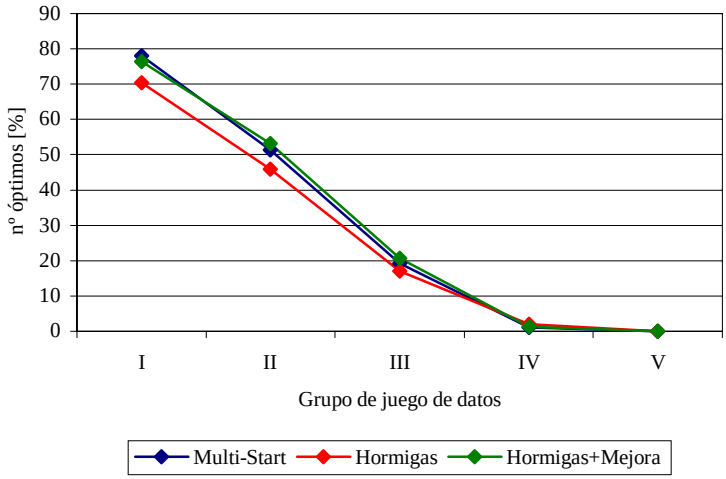
En los gráficos 7.2.4.2 (a), (b), (c) y (d) se pueden observar los resultados de la experiencia:





Gráficos 7.2.4.2 (a), (b)





Gráficos 7.2.4.2 (c), (d)

Como puede apreciarse en los gráficos, los tres métodos experimentan una mejoría respecto a las pruebas con el número de secuencias fijado, ya que disponen de más tiempo de ejecución y por lo tanto de más secuencias a evaluar.

La heurística que experimenta un mayor descenso en sus discrepancias al óptimo es el *multi-start*. Esto es debido a que, como ya se ha comentado, las hormigas han llegado al límite de exploración, mientras que el *multi-start*, debido a que parte de soluciones aleatorias, puede seguir mejorando si dispone de más tiempo.

Esto queda patente en la diferencia que se observa entre el algoritmo de hormigas sin mejora y el *multi-start*, que en la anterior prueba obtenían resultados similares, y ahora el segundo obtiene aproximadamente una discrepancia media al óptimo un 20% inferior que el primero.

Aun así, el algoritmo de hormigas con mejora local sigue siendo superior al resto en todos los resultados. Concretamente, su discrepancia media al óptimo es un 16% mejor que la obtenida por el *multi-start*.

Una vez analizado todo, se puede concluir que de los tres métodos analizados, el algoritmo que mejor resultados tiene, tanto en número de óptimos porcentual, como en discrepancias al óptimos mínima, media y máxima es el algoritmo de hormigas adaptado al SALBP-E con mejora local.

7.3 APLICACIÓN AL CASO RIEJU

Si se comparan los resultados obtenidos por los tres métodos heurísticos, se puede concluir que, tal como indicaban las pruebas de experiencia computacional, el algoritmo de hormigas con mejora local es el que menos discrepancia mínima, media y máxima respecto a la cota mínima obtiene (ver tabla 6.3.2).

En el gráfico 7.3 se puede observar el comportamiento de los tres métodos para el problema completo y simplificado.

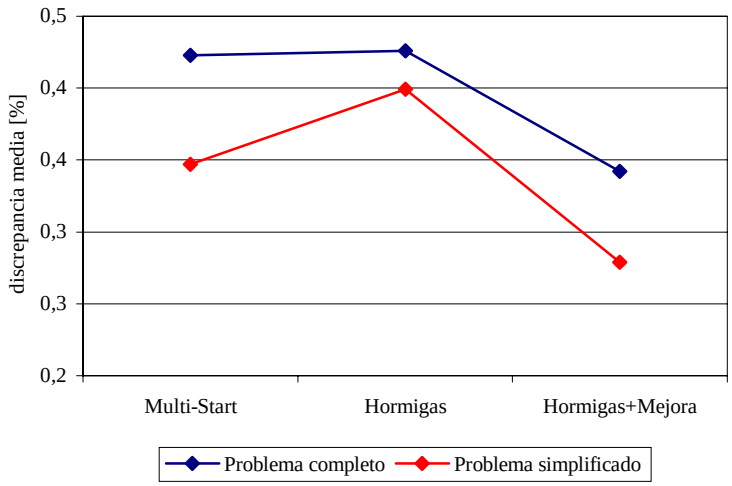


Gráfico 7.3

Como era de esperar, la simplificación del problema, al reducir el número de tareas, permite obtener un equilibrado de la línea más ajustado. Por lo tanto, la simplificación es muy útil, ya que facilita la búqueda de buenas soluciones sin alterar la solución final del problema..

Además, también se puede concluir que la aplicación de las heurística propuestas a un juego de datos reales se ha realizado con exito, ya que el comportamiento eficiente de los métodos basados en hormigas no se han visto alterados en ningún momento por las caracterísiticas propias de los juegos reales.

CAPÍTULO 8: ***LÍNEAS FUTURAS DE INVESTIGACIÓN***

A partir de todo lo comentado en los capítulos anteriores, en este capítulo se proponen diferentes líneas de investigación para el futuro.

Debido a que estos algoritmos son novedosos y se encuentran aún en fase de desarrollo, existen múltiples líneas futuras de investigación como pueden ser:

- La formulación de una teoría matemática del modelo propuesto, y en general de los algoritmos de hormigas (todo lo descrito hasta el momento se ha deducido de la experimentación computacional).

- Aplicación de este tipo de algoritmos a otros problemas de equilibrado (SALBP-1, SALBP-2, U-ALBP, etc.) y secuenciación (flow-shop, ORV, etc.).

- Estudiar como afectaría a la meta-heurística propuesta la incorporación de nuevas restricciones presentes en las necesidades industriales.

- Mejorar la rapidez del método mediante su paralelización.

- La combinación de los algoritmos de hormigas adaptados al SALBP-E con otras técnicas de exploración de entornos (tabu search , hill climbing, recocido simulado, etc.).

- La posibilidad de controlar los parámetros básicos de control del algoritmo (α, β, ρ y k) durante el proceso. Es decir, poder adaptar/variar estos parámetros según la etapa en que se encuentre. De esta forma se puede regular el grado de exploración y explotación a lo largo de todo el proceso.

CAPÍTULO 9: ***PRESUPUESTO***

En este capítulo se detalla en presupuesto del proyecto en sus dos partidas de coste básicas: los costes derivados de la investigación y desarrollo del proyecto y los costes asociados a la puesta en marcha en la factoría de Rieju. Se analizarán ambas por separado, para poder dar finalmente un presupuesto global para el desarrollo y puesta en marcha definitiva.

9.1 INVESTIGACIÓN Y DESARROLLO

9.1.1 Costes de personal

En los costes de personal se incluye el coste humano necesario para llevar a cabo desde el estudio previo del problema, hasta experimentación con un programa informático.

En la siguiente tabla se desglosan los costes relacionados con esta partida:

Actividad	Nº de horas	Coste [Ptas/hora]	Coste [Ptas]
Estudio previo	300	4.500	1.350.000
Desarrollo	300	4.500	1.350.000
Implementación	200	2.500	500.000
Depuración	50	2.500	125.000
Optimización del algoritmo	300	4.500	1.350.000
Experimentación	200	2.500	500.000
Documentación	175	4.500	787.500
Total	**1525**		**5.962.500**

9.1.2 Costes de material

Los costes de material los forman básicamente las máquinas utilizadas para la implementación y la documentación (PC's P-III) y una máquina Sun para la experimentación.

El coste en software es nulo, pues las JDK así como el entorno de programación utilizados son de libre licencia (freeware).

Para calcular el coste de material asociado al proyecto, se ha tenido en cuenta una amortización de los PC's de tres años y cinco para la máquina Sun. Los PC's son de uso exclusivo para el proyecto, por lo que se ha asociado un coste proporcional de los tres años de amortización. La máquina Sun se comparte con cinco usuarios, con lo que el coste sería una quinta parte del coste de amortización durante su uso.

Se considera que tanto los PC's como la máquina Sun han sido utilizados durante los seis meses que ha durado el proyecto.

Actividad	Unidades	Precio [Ptas/u]	Coste [Ptas]
PC's P-III 600, 64 MB	2	200.000	66.667
Sun UltraEnterprise 450	1	6.500.000	130.000
Total			**196.667**

9.1.3 Otros costes

Incluye el coste de material auxiliar, servicios utilizados y material de oficina. Se estima en **35.000 Ptas.**

9.1.4 Resumen costes

Actividad	**Coste [Ptas]**
Coste de personal	5.962.500
Coste de material	196.667
Otros costes	35.000

Total	**6.194.167**

9.2 APLICACIÓN AL CASO RIEJU

9.2.1 Costes de puesta en marcha

La puesta en marcha del programa informático en la factoría Rieju consta de tres etapas básicas: implantación del programa informático en los ordenadores que deban realizar los cálculos así como enseñar su manejo al personal; un control de su funcionamiento durante la primera etapa para detectar posibles correcciones; y por último hacer las últimas correcciones necesarias para que el programa funcione correctamente.

Actividad	Nº de horas	Coste [Ptas/hora]	Coste [Ptas]
Implantación	50	4.500	225.000
Control	50	4.500	225.000
Correcciones	50	4.500	225.000
Total	**150**		**675.000**

9.2.2 Costes de investigación

Se imputan directamente a Rieju, pues se considera que el producto va ser exclusivamente para su factoría. Por lo tanto será de 6.200.000 Ptas.

9.2.3 Costes totales de la aplicación al caso Rieju

Actividad	Coste [Ptas]
Coste de puesta en marcha	675.000
Coste de investigación	6.200.000
Total	**7.875.000**

BIBLIOGRAFIA Y REFERENCIAS

ADENSO D. ET AL (1996). "Optimización heurística y redes neuronales en Dirección de Operaciones e Ingeniería". Editorial Paraninfo, Madrid.

ALONSO M. (1998). "El lenguaje de modelización GAMS". Document Intern de Treball - DIT 98/1-, DOE, ETSEIB-UPC.

BAUTISTA J., COMPANYS R. y COROMINAS A. (1995). "Sequenciació d'unitats en context JIT". Colección TOE nº 9, Edicions UPC.

BAUTISTA J., MATEO M., FERRER R., PEREIRA J. y COMPANYS R. (2000). "The assembly line balancing problem solved by hybrid heuristic procedures and driven exploration".POMS, Sevilla 2000.

BAUTISTA J., SUÁREZ R., MATEO M. y COMPANYS R. (2000). "Local Search Heuristics for the Assembly Line Balancing Problem with Incompatibilities Between Tasks". Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000, San Francisco, CA, USA, April 24-28, 2000, pp. 2404-2409.

BISHOP J. (1999). "Java. Fundamentos de programación". Ed. Addison-Wesley. Madrid.

BRETÓN J., FERNÁNDEZ J.A. y BAUTISTA J. (2000). "Resolución del problema SALBP-E mediante procedimientos heurísticos y la programación lineal entera". DIT-OE-ETSEIB-UPC.

BROOKE A., KENDRICK D., MEERAUS A. y RAMAN R. (1998). "GAMS a user's guide". GAMS Development Corporation.

BULLNHEIMER B., HARTL R. F. y STRAUSS C. (1997). "A new Ranked Based Version of the Ant System-Computational Study". Technical report, University of Viena, Institute of Management Science.

CARRERAS-CANDI M. Y DOMINGO E. (1997). "Una aplicación de los algoritmos GRASP y genéticos al problema de equilibrado de líneas de montaje. Resolución de un caso de empresa". PFC ETSEIB-UPC director J. Bautista.

COLORNI A., DORIGO M. y MANIEZZO V. (1992a). "Distributed Optimization by Ant Colonies". Proceedings of the First European Conference on Artificial Life, Paris, France, F.Varela and P.Bourgine (Eds.), Elsevier Publishing, 134-142.

COLORNI A., DORIGO M. y MANIEZZO V. (1992b). "An Investigation of Some Properties of an Ant Algorithm. Proceedings of the Parallel Problem Solving from Nature Conference" (PPSN 92), Brussels, Belgium, R.Männer and B.Manderick (Eds.), Elsevier Publishing, 509-520.

COLORNI A., DORIGO M., MAFFIOLI F., MANIEZZO V., RIGHINI G., TRUBIAN M. (1996). "Heuristics from Nature for Hard Combinatorial Problems". International Transactions in Operational Research, 3(1):1-21.

COMPANYS R., COROMINAS A. (1994). "Organización de la producción II. Dirección de operaciones 4". Edicions UPC, colección Aula ETSEIB.

CPLEX (1998). "Using CPLEX callable Library". CPLEX Divison.

DI CARO G. y DORIGO M. (1997). "Ant Net: A mobile agents approach to adaptative routing".Technical Report 97-12, IRIDIA, Université Libre de Bruxelles.

DOMINGUEZ J.A., ALVAREZ J.M, GARCIA S., DOMINGUEZ M.A. y RUIZ A. (1995). "Dirección de operaciones. Aspectos estratégicos en la producción y los servicios". Ediciones McGrawHill.

DORIGO M. y GAMBARDELLA L.M. (1997). "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem". IEEE Transactions on Evolutionary Computation, 1(1):53-66.

DORIGO M., DI CARO G. y GAMBARDELLA L.M. (1999). "Ant Algorithms for Discrete Optimization". Artificial Life, 5(2):137-172.

DORIGO M., MANIEZZO V. y COLORNI A. (1996). "The Ant System: Optimization by a Colony of Cooperating Agents". IEEE Transactions on Systems, Man, y Cybernetics-Part B, 26(1):29-4.

ECKEL B. (1998). "Thinking in Java". Ed. Prentice Hall. USA.

FERRER R. (2000). "Resolución del problema de equilibrado de líneas de montaje mediante procedimientos híbridos con algoritmos heurísticos y de exploración dirigida". PFC. ETSEIB-UPC, director J. Bautista.

FERRER R., BAUTISTA J. (1999). "Modelización y resolución de problemas de equilibrado de líneas de montaje con PLE". Document Intern de Treball DT-I-1999/02-, IOC, ETSEIB-UPC.

GAMBARDELLA L.C. y DORIGO M. (1999). "An ant colony system hybridized with a new local search for the sequential ordering problem". Aceptado para publicación en Informs Journal on Computing, marzo 2000.

GAMS (1999). "GAMS, the solver manual". GAMS Development Corporation.

GAMS (1999)."GAMS/CPLEX 6.5 User Notes". GAMS Development Corporation. *http://www1.gams.com/solvers/cplex/cplexman.htm*

PLANS J. (1999). ”Classificació, modelització i resolució dels problemes de disseny i asssignació de tasques en linies de producció”. Tesis doctoral DOE-UPC.

PLANS J., COROMINAS A. (2000). ” Solving assembly line problems by means of MILP and fix & relax” IOC-DT-P-2000-22. Institut d'Organització i Control de Sistemes Industrials, Barcelona.

RIERA J. (1997). “Anàlisis del procés actual i obtenció d’una solució pel problema del muntatge en una cadena de ciclomotors tipus *Scooter*. Aplicació a l’empresa Rieju, S.A.”. PFC-DOE-UPC. Director J. Bautista.

SÁNCHEZ M., ALONSO V. “El lenguaje de programacion Java™”. e-Paper: http://www.disca.upv.es/misan/publicat.htm

SCHOLL A. (1999). “Balancing and Sequencing of Assembly Lines”. Heidelberg Physica-Verlag cop.

STÜTZLE T. y DORIGO M. (1999). "ACO Algorithms for the Traveling Salesman Problem". In K. Miettinen, M. Makela, P. Neittaanmaki, J. Periaux, editors, Evolutionary Algorithms in Engineering and Computer Science, Wiley, 1999.

STÜTZLE T. y HOOS H. (1997). "Improvements on the ant system: Introducing *MAX-MIN* ant system”. In Proceding of the International Conference on Artificial Neural Networks and Genetic Algorithms,pag. 245-249. Springer Verlag, Wien.

STÜTZLE T. y HOOS H. (1997). "The *MAX-MIN* ant system and local search for the Traveling Salesman Problem". In T. Baeck, Z. Michalewicz, y X. Yao, editors, Proceedings of IEEE-ICEC-EPS’97, IEEE International Conference on Evolutionary Computation and Programming Conference, pag. 309-314. IEEEE Press.

VALERO J. (1991). “Modelos y algoritmos de PLE para el diseño y equilibrado de líneas de producción y montaje”. PFC- ETSEIB-UPC, director A. Corominas.