LABORATORIO NACIONAL DE INFORMÁTICA
AVANZADA A. C.

Centro de Enseñanza LANIA

Uso de una Colonia de Hormigas para resolver Problemas de Programación de Horarios

Tesis que presenta:
Emanuel Téllez Enríquez

para obtener el grado de:
Maestro en Ciencias de la Computación

Directores de Tesis:
Dr. Carlos A. Coello Coello (CINVESTAV-IPN)
Dr. Efrén Mezura Montes (LANIA)

Xalapa Ver. 11 de enero del 2007.

Resumen

Los problemas de programación de horarios son muy importantes ya que en la vida real tienen una alta aplicación en diversos campos de la industria, en donde existe un conjunto de tareas que deben ser organizadas de tal forma que el orden con que éstas se desarrollen permitan ahorrar tiempo, dinero y esfuerzo.

Desde hace varios años la complejidad de estos problemas ha llamado la atención de ingenieros y científicos, interesados en encontrar métodos que ofrezcan en un tiempo razonable buenas soluciones. Tales métodos proponen formas de solucionar el problema desde diversos enfoques, pero hasta el momento no se conoce alguno que pueda dar una garantía de resolver cualquier instancia del problema antes mencionado. Entre estos métodos podemos encontrar las llamadas metaheurísticas, que tienen su inspiración en diversas áreas como la química, la física, la biología, entre otras.

Del área biológica podemos encontrar diversas formas de vida que sorprenden y cada vez enfatizan la tan conocida frase: *"la naturaleza es sabia"*, y en la cual, el hombre ha observado, analizado y descubierto los *"comportamientos inteligentes"* de diversas especies para aplicarlos en la resolución de problemas de la vida diaria.

En esta tesis se presenta una variante del algoritmo *Ant System* (AS) basado en el comportamiento de forrajeo de las hormigas. El algoritmo propuesto es desarrollado para resolver el problema de programación de horarios.

El manejo de las reglas de decibilidad y la aplicación de un nuevo conocimiento biológico sobre el comportamiento para localizar los caminos más cortos entre el nido de la colonia de hormigas y la fuente de alimentos son características que distinguen este trabajo.

En esta tesis se reportan las diversas variantes realizadas que dieron lugar a la generación del nuevo algoritmo. Para medir el desempeño de dicho algoritmo se compararon los resultados obtenidos de diversas pruebas contra otros algoritmos del estado del arte.

A mis abuelos:
Francisca Viveros García
Moises Téllez González
Paulina Emelia Alonso Hernández
Simón Enríquez Leos †

Agradezco la beca terminal de maestría
que se me otorgó por medio del proyecto CONACyT titulado
"Técnicas Avanzadas de Optimización Evolutiva Multiobjetivo" (Ref. 45683-Y),
cuyo responsable es el Dr. Carlos A. Coello Coello.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Existe un área de la ingeniería y las matemáticas dedicada a la búsqueda de la mejor solución a un problema de entre todo un conjunto de posibles soluciones, llamada Optimización. Dentro de esta área podemos encontrar una diversidad de problemas que podemos clasificar en dos grupos principales de acuerdo al tipo de solución que se trata de encontrar: Optimización Numérica y Optimización Combinatoria. Por ejemplo: asumuendo que las variables son continuas en las funciones, el problema se considera de optimización numérica. En cambio, si las variables son de naturaleza discreta, el problema de encontrar soluciones óptimas es conocido como optimización combinatoria.- En esta tesis, nos ocuparemos de un problema de optimización combinatoria, por lo cual nuestra discusión se centrará en este tema. La optimización combinatoria incluye un conjunto de problemas cuyo objetivo es localizar la permutación adecuada que minimice (o maximice) una función llamada función objetivo y que además la solución cumpla con las restricciones particulares del problema.

Los problemas de programación de horarios son un tipo de problema perteneciente al área de la optimización combinatoria que han atraído el interés de muchos investigadores por la alta complejidad de los mismos y la cantidad de recurso consumido en la búsqueda de buenas soluciones. Básicamente, estos problemas constan de un conjunto de actividades (a) que deben ser procesadas en un conjunto finito de recursos disponibles (r) (bajo ciertas restricciones), para lo cual, el tamaño de un problema está dado por $a \times r$, de tal forma que el número de soluciones al problema crece en forma exponencial a medida que se aumentan actividades, recursos o ambos.

Estos problemas han sido abordados desde la década de los sesenta a través de métodos exactos y métodos aproximados. Sin embargo, los métodos exactos han demostrado poca

ventaja ante problemas de tamaños muy grandes, en los cuales, realizar un análisis exhaustivo resultaría computacionalmente prohibitivo. Es precisamente en este tipo de problemas que los métodos aproximados han proporcionado un conjunto de técnicas heurísticas inspiradas en diversas áreas tales como las matemáticas, física, química, biología, entre otras. Entre estas técnicas podemos mencionar: Recocido Simulado, Búsqueda Tabú, GRASP, Algoritmos Genéticos, etc. Una técnica relativamente reciente llamada Optimización basada en Colonias de Hormigas *Ant Colony Optimization* (ACO) ha demostrado tener ventajas frente a otras técnicas del área en la resolución de problemas de optimización combinatoria.

Aunque el Ant System (algoritmo básico de ACO) ya ha sido aplicado previamente al problema de programación de horarios (o *Job Shop Scheduling Problem*) este no ha obtenido resultados favorables frente a otras heurísticas, nuestra hipótesis fue que, a partir de la observación de los buenos resultados que ha obtenido el Ant System frente a diversos problemas combinatorios, y valiéndonos de algunos cambios en la estructura básica del Ant System, podríamos hacer competitivo este motor de búsqueda para las diversas instancias del problema de programación de horarios. El objetivo de esta tesis fue desarrollar una variante del Ant System para el problema de programación de horarios que resultase competitiva desde el punto de vista de la calidad de los resultados y el número de evaluaciones de la función objetivo. La validación correspondiente se realizó utilizando archivos de prueba conocidos en el estado del arte.

Para el logro del objetivo proponemos implementar diversas variantes del AS para el problema de programación de horarios, realizando una comparativa entre cada versión generada a través de los resultados preliminares obtenidos con los archivos de prueba y determinando el mejor de éstos para poder, posteriormente, compararlo contra los mejores algoritmos del estado del arte: un sistema inmune artificial (SIA), un algoritmo cultural (CULT) y una búsqueda tabú (TS) de su nombre en inglés *Tabu Search*.

La organización de la tesis es la siguiente: en el capítulo 2 se presentará el problema específico a optimizar, las características y la importancia del mismo. En el capítulo 3 se porporcionará una introducción a la Computación Evolutiva y se describirá en qué consiste la metaheurística de ACO para finalmente abordar un algoritmo básico de esta metaheurística llamado Ant System, presentando su pseudocódigo y algunas variantes. En el capítulo 4 se presentan las versiones de AS desarrolladas en esta tesis para el problema de programación de horarios en forma detallada, los archivos de prueba utilizados y los resultados preliminares obtenidos. Así mismo, se presenta la selección de la versión final que se utilizará para comparar con las otras heurísticas. En el capítulo 5 se presentan los resultados obtenidos

en las pruebas realizadas y los cuadros comparativos con las otras heurísticas. Finalmente, el capítulo 6 proporciona una discusión de los resultados obtenidos, las conclusiones y el trabajo futuro.

Capítulo 2

El Job Shop Scheduling Problem

En este capítulo se presenta el problema de programación de horarios. Primeramente se hará una introducción al concepto de "optimización" y "optimización combinatoria". A continuación, se describirá el problema y se darán las características del mismo, se analizarán algunas de las formas más comunes de representación del problema y la clasificación de los posibles planes de trabajo. Para finalizar se hará una revisión del estado del arte y la importancia de estudio de dicho problema.

2.1. Introducción

El hombre, a través de la historia, ha tratado de alcanzar los mejores resultados en sus actividades, ya sean estas económicas, administrativas, políticas, militares, etc. afrontando para ello un numeroso grupo de problemas. En todas estas actividades subyace el término optimizar, que podemos entender como buscar lo mejor, o también se puede interpretar como obtener la máxima ganancia o tener la menor pérdida. Así también, como se explica más ampliamente en [36], un problema es una pregunta general que deseamos responder y que típicamente posee varios parámetros o variables cuyos valores no están especificados.

La optimización es una rama de estudio de la matemática aplicada llamada Investigación de Operaciones, en la cual existen muchos problemas que pueden ser representados de tal manera que optimicen una función objetivo que a su vez, depende de algunas variables de decisión, las cuales estarán definidas sobre una región (espacio de búsqueda) formada por un conjunto de restricciones particulares del problema a resolver [26]. Para introducir estas nociones a continuación se describe un ejemplo sencillo: Un grupo de ingenieros desea construir un puente al menor costo posible, para lo cual requerirá un conjunto de materiales

X, Y, Z, donde el número de trabajadores para tal construcción no deberá superar a 100 empleados.

Así tenemos que, el costo de la construcción del puente, dependerá de los materiales *X, Y, Z* utilizados y el número de trabajadores (éstas serán nuestras variables de decisión). Las restricciones del problema serán: el que deberán utilizarse los materiales *X, Y* y *Z*, cuidando que el número de trabajadores no exceda de 100.

Entonces un problema general de optimización se puede definir matemáticamente de la siguiente forma:

$$\begin{aligned} &\text{Optimizar } f(\bar{x}) \\ &\text{Sujeto a } g_i(\bar{x})\,\{\geq, =, \leq\}\, b_i; i = 1, \cdots, m; \\ &\qquad \bar{x} \geq 0 \end{aligned}$$

Donde $f(\bar{x})$ es una función objetivo que debe ser optimizada por la intersección de una familia de m restricciones $g_i(\cdot)$ y donde el vector $\bar{x}$, compuesto por n variables de decisión, toma valores mayores o iguales a cero. El problema de optimización puede ser de maximización o minimización.

Partiendo de la estructura general del problema de optimización descrita en el párrafo anterior, tenemos que existe una gran variedad de problemas conocidos y clasificados en la literatura [36] [37]. Por ejemplo: asumiendo que las variables son continuas en las funciones, el problema se considera de optimización numérica. En cambio, si las variables son de naturaleza discreta, el problema de encontrar soluciones óptimas es conocido como optimización combinatoria [26] [37]. Existe una gran variedad de problemas de optimización combinatoria como por ejemplo:

- Problemas de RUTEO: Problema de Orden Secuencial, Problema de Ruteo de Vehículos [40].
- Problemas de ASIGNACIÓN: Problema de Asignación Cuadrática, Problema de Asignación Generalizada, Problema de Asignación de Frecuencias, Problema de Coloreado de Gráficos, Problema de Organización de Cursos en la Universidad [41].
- Problemas de PROGRAMACIÓN: Flow Shop Problem, Open Shop Problem, Job Shop Scheduling Problem, Group Shop Scheduling Problem [41].

En este último grupo encontramos clasificado nuestro problema de estudio: el problema de programación de horarios.

2.2. Descripción del problema

En un ámbito más cercano, todo el mundo organiza sus tareas a realizar durante el día o la semana, haciéndolo de una manera muy sencilla, algunos recordando las actividades por realizar y otros quizás anotándolas. No obstante, en un ámbito empresarial resultará más complicado conocer cuál es el orden en el que deberán llevarse a cabo las actividades, según distintas prioridades tales como fechas de entrega, inventario en curso y otros. Así, la programación de horarios apoya a las principales áreas de una empresa. Sin embargo, aunque utilizaremos la terminología de la industria para describir el problema, ésta no es la única aplicación que se tiene sobre programación de horarios. Podemos encontrar la programación de horarios, por ejemplo en: Aeropuertos (en la planificación de los despegues y aterrizajes de aviones a través de sus pistas); en los hospitales (al distribuir el tiempo de los médicos y enfermeras en la atención de los pacientes [42] [38]), etc.

Los problemas de programación de horarios tienen sus orígenes en la segunda guerra mundial por la necesidad urgente y creciente de asignación de recursos en las operaciones militares [46]. Aquí podemos encontrar el Problema de Programación de Horarios conocido más ampliamente en la literatura como Job Shop Scheduling Problem [39], abreviado con las siglas: JSSP.

El JSSP es un problema de optimización combinatoria y consiste, de forma genérica, en la asignación de un número determinado de actividades en un número finito de recursos disponibles. El objetivo al optimizar este problema es encontrar una permutación de dichas actividades que minimice el tiempo de término de todas las actividades en conjunto.

De este problema podemos encontrar diversas variantes cuya principal diferencia estriba en las restricciones propias de cada problema en particular. El tipo de problema sobre el cual nos enfocaremos es conocido como *JSSP Clásico*, y a continuación lo describiremos:

> *El JSSP Clásico provee un conjunto finito de trabajos J, que deberán ser procesados en un conjunto finito M de máquinas. A cada trabajo que se procesa en cada máquina se le denomina operación (relación binaria de trabajo - máquina) y esta operación tendrá asignado un tiempo específico de procesamiento. Las operaciones correspondientes a un trabajo tendrán una secuencia tecnológica dada por el problema en particular y esta secuencia será inamovible.*

Las restricciones propias del JSSP clásico las enlistamos a continuación:

- Ninguna máquina podrá procesar dos operaciones al mismo tiempo.
- Todas las operaciones tendrán la misma prioridad de procesamiento.
- Todas las operaciones de un trabajo en particular deberán ser procesadas en su totalidad.
- Deberá respetarse la secuencia tecnológica de cada trabajo.
- Los trabajos deberán esperar a que la máquina siguiente a utilizar esté disponible para poder continuar su procesamiento, aunque haya que retardarse en tiempo.
- Sólo hay un tipo de máquina.
- Las máquinas podrán estar ociosas en cualquier momento del plan de trabajo.

El objetivo al optimizar el JSSP clásico es encontrar un plan de trabajo o programa (permutación de operaciones) donde todas las operaciones hayan sido concluidas en el menor tiempo posible.

Finalmente, el número de programas (soluciones) que pueden ser generados para un problema en particular, sabiendo que cada secuencia de operaciones puede ser permutada independientemente, sería de $(j!)^m$ donde j denota el número de trabajos y m el número de máquinas. Como ejemplo: si se tiene un total de 24 trabajos a ser procesados en una sola máquina, se tienen entonces: $24! = 6.20 \times 10^{23}$ posibles soluciones. Aunque se tuviera una máquina que pudiera obtener una solución por segundo, se tardaría poco más que la edad del universo, aproximadamente unos 20 mil millones de años en enumerar todas las soluciones [13]. Esto nos da una idea de lo complejo que resultaría resolver el problema por enumeración.

2.3. Definición formal del problema

Un JSSP Clásico está determinado por el tamaño $j \times m$ donde j es el número de trabajos $J = \{J_1, J_2, J_3, \ldots, J_j\}$ y m es el número de máquinas del problema $M = \{M_1, M_2, M_3, \ldots, M_m\}$. En el JSSP Clásico para todo elemento del conjunto trabajos J deberá existir su correspondiente relación con cada uno de los elementos del conjunto de máquinas M. La relación de cada trabajo con cada máquina conforma una relación binaria de $J_j \rightarrow M_m$ que es llamada *Operación(j,m)*. Ver figura 2.1

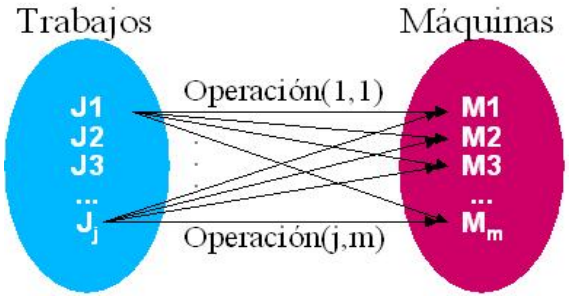

Figura 2.1: Relación trabajos - máquinas de un JSSP Clásico de tamaño $j \times m$.

El total de operaciones del problema es denotado por $O_{JM} = \{O_{1,1}, O_{1,2}, O_{1,3}, \ldots, O_{j,m}\}$. El conjunto de operaciones O correspondientes a un mismo trabajo tiene una secuencia de procesamiento llamada: *Secuencia tecnológica* S tal que $S = \{S_1, S_2, S_3, \ldots, S_m\}$ donde m es el número de máquinas del problema. Por lo tanto, para cada trabajo j tendremos la secuencia tecnológica: $O_{JS} = \{O_{j1}, O_{j2}, O_{j3}, \ldots, O_{js}\}$. Esto es debido a que en el JSSP Clásico se requiere que exista en cada secuencia tecnológica de trabajo una operación por cada máquina del problema.

Cada *operación (j,m)* tiene asociado un tiempo denominado: *tiempo de procesamiento*, y es denotado por la letra t. Este parámetro indica el tiempo requerido de ese trabajo j en la máquina m. Adicionalmente, se puede tener un *tiempo de liberación* antes de que otro trabajo pueda ser procesado y un *tiempo de preparación* que cada máquina requerirá antes de poder estar lista para atender otra operación. Estos tiempos son manejados en forma discreta (generalmente usando números enteros).

Una instancia de un problema, como se define en [36] es *"la formulación del mismo para unos datos concretos, es decir, la descripción de una asignación de valores concretos para cada uno de los parámetros del problema"*. Así tenemos que, una instancia de JSSP generalmente es definida mediante una matriz. Ver tabla 2.1.

La lectura de la matriz se realiza por filas y cada fila hace referencia a un trabajo

independiente, tal como lo muestra la primer columna de la matriz. Cada uno de los trabajos listados tiene especificada su secuencia tecnológica indicando la máquina en que debe ser procesada y el tiempo de procesamiento especificado entre paréntesis.

Al resolver el problema se obtiene un programa de trabajo que consiste de una permutación de las operaciones del problema. Este programa puede ser comprendido y visualizado a través de una gráfica de Gantt, ver figura 2.2.

trabajo	máquina (tiempo)		
1	1(3)	2(3)	3(3)
2	1(2)	3(3)	2(4)
3	2(3)	1(2)	3(1)

Tabla 2.1: Instancia del JSSP de tamaño 3×3.

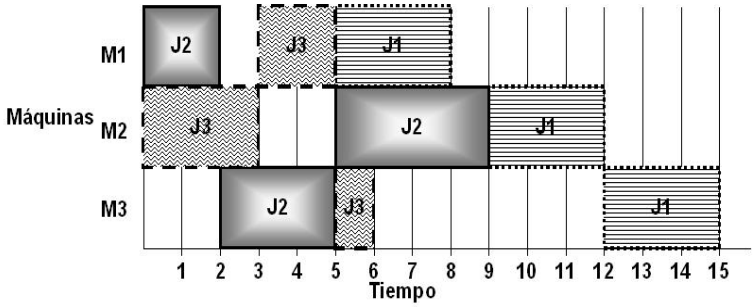

Figura 2.2: Gráfica de Gantt que muestra uno de los posibles programas de trabajo para el JSSP Clásico de tamaño 3×3 mostrado en la tabla 2.1.

Existen varios objetivos en el JSSP Clásico sujetos de optimizar, por lo cual será necesario aclarar la notación que se utilizará para describirlos.

- C_i: es el tiempo de término *(completion time)* de la última operación del trabajo J_i.
- F_i: es el tiempo de realización de todas las operaciones *(flow time)* del trabajo J_i desde el comienzo de la primera operación hasta la última.
- d_i: existen problemas para los que se especifica el tiempo comprometido *(due date)* que es el tiempo máximo de término de un trabajo J_i.
- L_i: dada por la ecuación $L_i = C_i - d_i$, representa el retraso *(lateness)* del trabajo J_i.
- T_i: dada por la ecuación $T_i = MAX\,(L_i, 0)$, representa la tardanza *(tardiness)* del trabajo J_i.

- E_i: dada por la ecuación $E_i = MAX\,(-L_i, 0)$, representa la puntualidad *(earliness)* del trabajo J_i.

A continuación listamos las medidas de desempeño más importantes al optimizar el JSSP:

- **Makespan:** es el tiempo mínimo para completar los trabajos, $C_{MAX} = MAX_{i\in\{1...n\}C_i}$
- **Total flow time:** es el tiempo consumido por todos los trabajos, $F_\Sigma = \Sigma_{i\in\{1...n\}}F_i$
- **Total lateness:** es la suma de todos los retrasos de los trabajos, $L_\Sigma = \Sigma_{i\in\{1...n\}}L_i$
- **Total tardiness:** es la suma de todas las tardanzas de los trabajos, $T_\Sigma = \Sigma_{i\in\{1...n\}}T_i$
- **Total earliness:** es la suma de todos los tiempos de terminación previos al comprometido de cada trabajo, $E_\Sigma = \Sigma_{i\in\{1...n\}}E_i$
- **Maximum lateness:** el mayor retraso de los trabajos, $L_{MAX} = MAX_{i\in\{1...n\}}L_i$
- **Maximum tardiness:** la mayor tardanza de los trabajos, $T_{MAX} = MAX_{i\in\{1...n\}}T_i$

En todas las medidas de desempeño antes mencionadas, el objetivo de optimización es minimizar.

Cabe señalar que en este trabajo sólo se tomará como objetivo único de optimización el makespan, debido a que es la medida de desempeño conocida y referida en los diversos estudios del JSSP, lo cual facilitará la comparación de nuestros resultados con respecto a algoritmos del estado del arte.

2.4. Representación de soluciones

Como en casi todos los problemas del mundo real, debemos encontrar una forma de abstraerlos para poder abordarlos, resolverlos e interpretar la solución. A esta forma de abstracción se le denomina representación de la solución. A lo largo del tiempo en que se ha abordado el JSSP se han creado diversas formas de abstracción o de representación que a continuación describimos. Para ello utilizaremos la instancia del problema 3×3 descrito en la tabla 2.1 de la sección anterior.

Representación en Grafo Disyuntivo [51][11]: El JSSP puede ser formalmente descrito por un grafo disyuntivo $G = (V, C \cup D)$, del cual describimos a continuación su construcción:

Se coloca un conjunto V de nodos representando las operaciones *(trabajo, máquina)*. Ver figura 2.3.

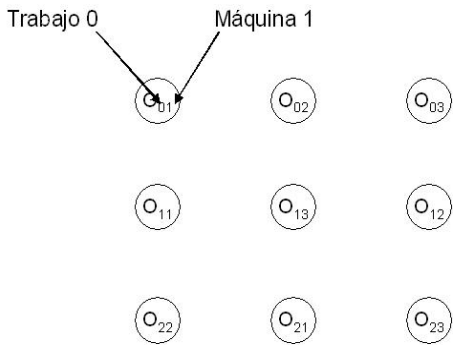

Figura 2.3: Construcción del grafo: Conjunto de V nodos.

Se añaden los tiempos de procesamiento de cada operación en la parte superior de cada una de ellas. Ver figura 2.4.

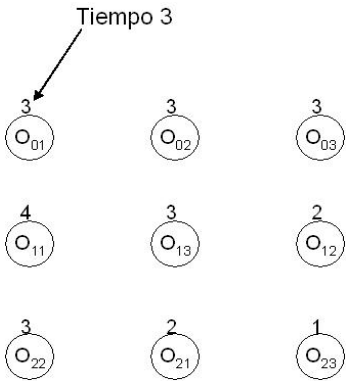

Figura 2.4: Construcción del grafo: Tiempos de cada operación.

Además se agregan dos nodos: primero y último, representando el inicio y término del grafo. Ver figura 2.5.

Se coloca el conjunto de arcos conjuntivos C que indican la secuencia tecnológica de las operaciones. Ver figura 2.6.

Finalmente se añade el conjunto de arcos disyuntivos D que indican los pares de operaciones que se ejecutan en una misma máquina. Ver figura 2.7.

La figura 2.7 es la resultante de la representación del problema JSSP en un grafo.

Representación Binaria [51]: Al convertir todos los arcos disyuntivos en conjuntivos de un JSSP representado en un grafo se obtiene un programa semiactivo (los programas semiactivos serán explicados más ampliamente en la siguiente sección). Para etiquetar la

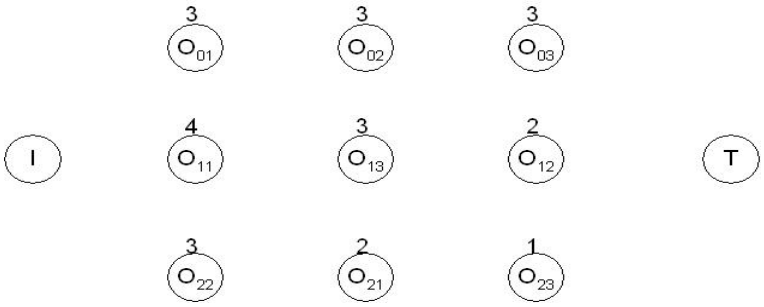

Figura 2.5: Construcción del grafo: Se agregan nodos inicial y final.

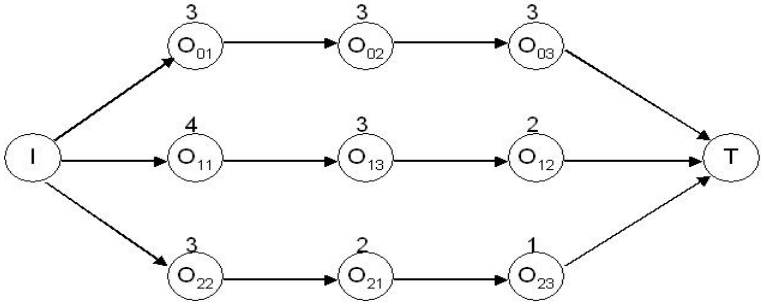

Figura 2.6: Construcción del grafo: Colocación de arcos conjuntivos.

dirección de cada uno de los arcos disyuntivos de un programa se puede utilizar una representación binaria indicando con 0 o 1 las direcciones de dichos arcos. Así, un programa puede ser representado por una cadena de tamaño $mj\,(j-1)\,/2$. La figura 2.8 muestra un ejemplo, donde un arco que conecta O_{ij} y $O_{kj}\,(i<k)$ es etiquetado como 1 si el arco es dirigido de O_{ij} a O_{kj} (entonces O_{ij} es procesada antes que O_{kj}) o 0, si es el otro caso.

Representación con Permutación [51]: Un programa puede ser representado por un conjunto de permutaciones de trabajos en cada máquina, de tal forma que existen m particiones de permutaciones de operaciones, donde m es el número de máquinas del problema. La cadena formada es llamada *Matriz de Secuencia de Trabajos*. Ver figura 2.9.

Existen muchas más representaciones utilizadas para el JSSP (ver por ejemplo [38]), pero sólo hemos incluido las más utilizadas en la literatura especializada.

2.5. Tipos de planes de trabajo

Tomando como referencia la figura 2.10 tenemos que una *selección* es un conjunto de arcos dirigidos seleccionados de los arcos disyuntivos. Así, por definición: será una *Selección Completa* si todas las disyunciones son seleccionadas. Y será una *Selección Completa*

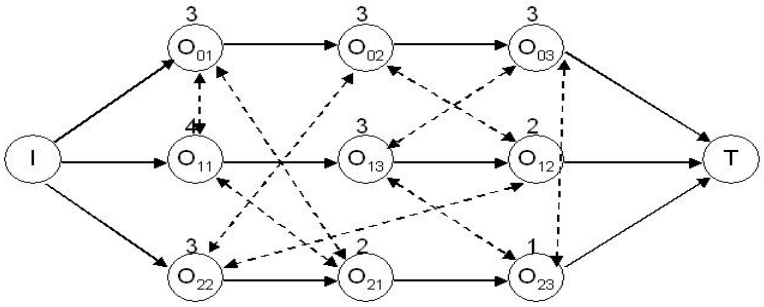

Figura 2.7: Construcción del grafo: Colocación de arcos disyuntivos.

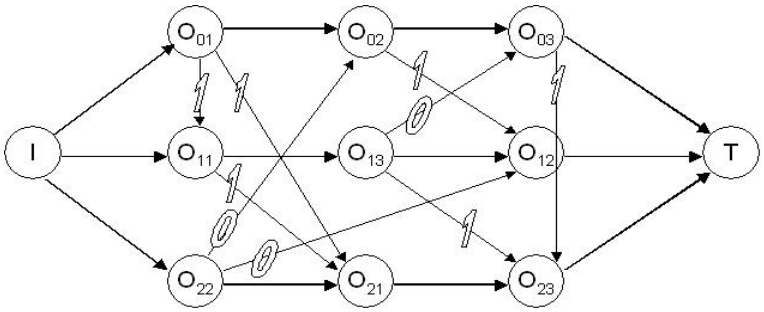

Figura 2.8: Representación binaria.

Consistente si además el grafo dirigido resultante es acíclico.

El total de posibles selecciones completas o programas que puede resolver un JSSP son clasificados de acuerdo a los siguientes criterios:

Un *Programa Semiactivo* es un programa obtenido de una selección completa consistente tan pronto como sea posible. En estos programas, ninguna operación podrá ser iniciada sin alterar la secuencia de las máquinas. Ver figura 2.11

A menudo el makespan de un programa semiactivo puede ser reducido con un simple cambio a la izquierda en alguna operación sin retrasar otros trabajos; tal cambio es llamado *Cambio Permisible a la Izquierda*. Un programa sin más cambios permisibles a la izquierda se denomina *Programa Activo*. En la figura 2.12 se muestra un cambio permisible y como éste, da como resultado un programa activo.

Un programa en el que ninguna máquina está nunca ociosa si una operación está lista para ser procesada en ésta, es llamado Programa sin Retardo [23].

M_1	M_2	M_3
1 2 3	3 1 2	2 1 3

Figura 2.9: Una matriz de secuencia de trabajos para un problema 3×3.

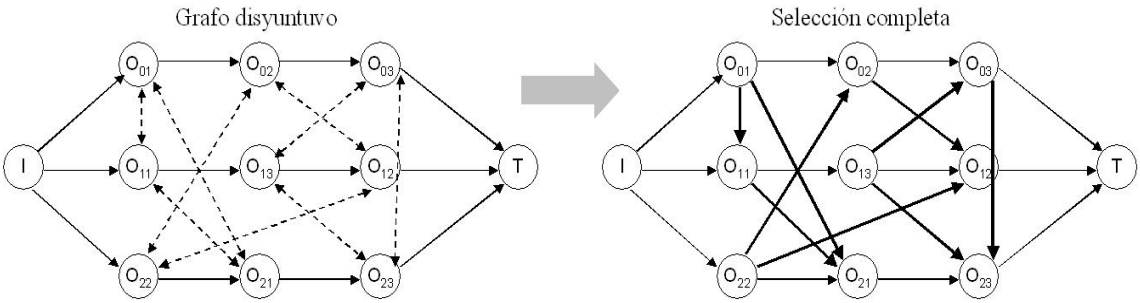

Figura 2.10: De un Grafo Disyuntivo a Selección Completa.

2.6. Estado del arte

Existen problemas de optimización combinatoria complejos en diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Sin embargo, a menudo estos problemas son muy difíciles de resolver en la práctica. El estudio de esta dificultad inherente para resolver dichos problemas tiene cabida en el campo de la teoría de las Ciencias de la Computación.

Muchos de los problemas de optimización combinatoria pueden ser clasificados como NP-completos. Ésta es una clase de problemas para los que no se conoce un algoritmo de tiempo polinomial que pueda resolverlos a optimalidad, aunque tampoco se ha demostrado que tal algoritmo no existe. Esto ha generado que muchos investigadores exploren el área implementando diversos métodos para abordarlos [26]. El Job Shop Scheduling no podría ser la excepción pues es un problema representativo del área de optimización combinatoria.

2.6.1. Métodos de solución del JSSP a lo largo la historia

Existe una amplia variedad de técnicas para resolver el JSSP. Los métodos con que ha sido abordado desde la década de los sesenta pueden ser clasificados en dos grupos: métodos exactos y métodos aproximados [46] [42].

- Los Métodos Exactos hacen exploraciones exhaustivas ya que intentan encontrar una solución óptima y demostrar que la solución obtenida es de hecho el óptimo global. Ejemplos de estos métodos son: backtracking, ramificación y poda (branch and bound).

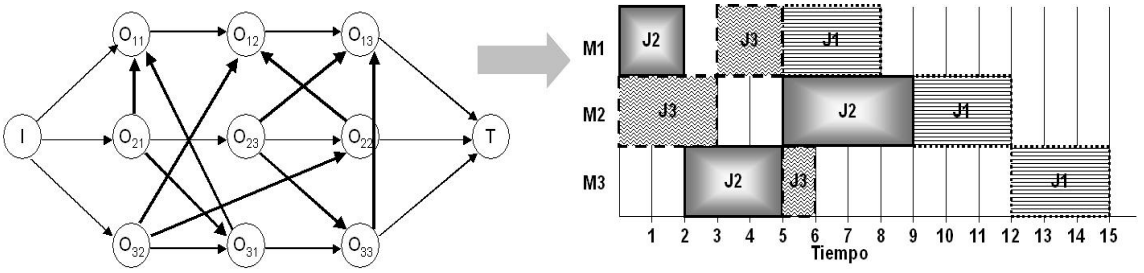

Figura 2.11: Una Selección Completa Consistente y su Programa Semiactivo correspondiente.

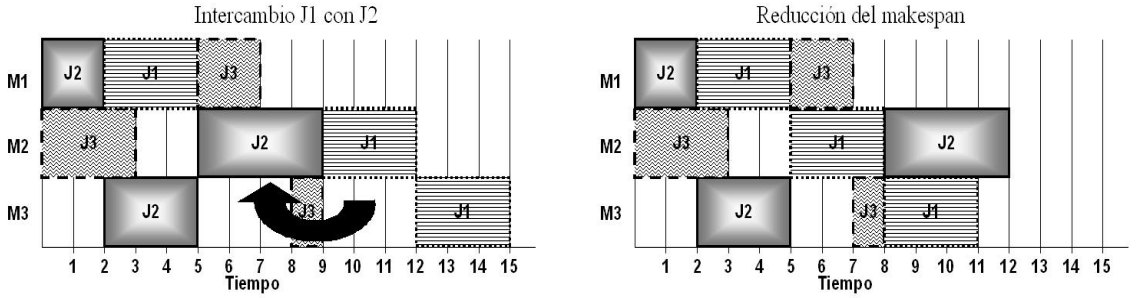

Figura 2.12: Cambio permisible a la izquierda y programa activo.

- Los Métodos Aproximados van eligiendo soluciones que cumplen determinados criterios. Ejemplos de estos métodos son: Algoritmos constructivos y de búsqueda local y Metaheurísticas.

Métodos Exactos.

Como se definió anteriormente, estos algoritmos se caracterizan por realizar exploraciones exhaustivas, sin embargo son de los algoritmos más difundidos en las aplicaciones industriales.

B&B (Branch and Bound) [50][7]: La desventaja de B&B radica en el hecho de que es necesario resolver un programa lineal completo en cada nodo. La figura 2.13 muestra un pequeño ejemplo del crecimiento del árbol de exploración.

SPT (Árbol de Caminos más Cortos) [50][7]: Genera distintas conexiones punto a punto, para producir una conexión multipunto. La figura 2.14 muestra un ejemplo del algoritmo.

A la fecha, los métodos de búsqueda y corte son los más efectivos para resolver programas enteros de tamaños prácticos, siendo generalmente mejor el de B&B. Los métodos exactos pretenden hallar un plan jerárquico único analizando todos los posibles ordenamientos de

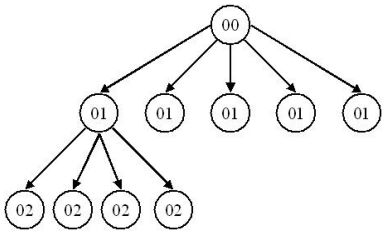

Figura 2.13: Búsqueda en un algoritmo B&B (Branch and Bound).

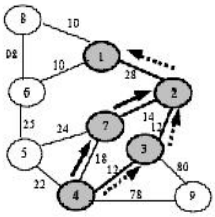

Figura 2.14: Búsqueda en un algoritmo SPT.

las tareas o procesos involucrados en la línea de producción (exploración exhaustiva). Sin embargo, una estrategia de búsqueda y ordenamiento que analice todas las combinaciones posibles es computacionalmente costosa y sólo funciona para algunos tipos (tamaños) de instancia.

Finalmente, hacemos notar algunos comentarios reportados en la literatura especializada: de [11] *"Los algoritmos exactos muestran un rendimiento pobre para muchos problemas."* y de [51] *"El algoritmo B&B para JSSP ... resuelve problemas JSP de hasta 15 * 14 (220 operaciones)."*. Esto nos da una idea de las limitaciones de estos algoritmos.

Métodos Aproximados.

Los métodos aproximados y las metaheurísticas incorporan conceptos de muchos y diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otras. Algunas metaheurísticas son las siguientes:

Recocido Simulado (Simulated Annealing SA) [1][31]: El recocido simulado se basa en una analogía con el proceso físico de enfriamiento, en el cual se forma una estructura reticular en un sólido calentándolo en un baño caliente hasta ablandarlo y luego enfriándolo pausadamente hasta solidificarse en un estado de baja energía.

Desde el punto de vista de la optimización combinatoria, el recocido simulado es un algoritmo estocástico de búsqueda local. Las soluciones vecinas de mejor costo son las que se prefieren, pero se aceptan también las soluciones de peor costo, aunque con una probabilidad

que decrece gradualmente durante el transcurso de la ejecución del algoritmo. Esta reducción de la tolerancia o probabilidad de aceptación es controlada por un conjunto de parámetros cuyos valores son determinados por un esquema de enfriamiento prescrito.

El recocido simulado ha sido vastamente aplicado con considerable éxito. Su naturaleza aleatoria permite convergencia asintótica a la solución óptima bajo condiciones moderadas. Desafortunadamente, la convergencia requiere típicamente de tiempo exponencial, convirtiendo el recocido simulado en impráctico como instrumento para procurarse soluciones óptimas. En cambio, como muchos algoritmos de búsqueda local, resulta eficiente como método de aproximación, para lo cual presenta una tasa de convergencia más aceptable.

Búsqueda tabú (Tabu Search TS) [24]: La búsqueda tabú combina un algoritmo determinístico de avance iterativo con la posibilidad de aceptar soluciones que incrementen el costo. De esta manera, la búsqueda es dirigida fuera de óptimos locales y pueden explorar otras partes del espacio de soluciones. Cuando una nueva solución es visitada, se considera un vecino legítimo de la solución actual aún cuando ésta empeore el costo.

El conjunto de vecinos queda representado por una lista tabú, cuyo objetivo es restringir la elección de soluciones para prevenir el regreso a puntos recientemente visitados. La lista tabú es actualizada dinámicamente durante la ejecución del algoritmo y define soluciones que no son aceptables en unas pocas iteraciones consecutivas. Sin embargo, una solución de la lista tabú puede ser aceptada si su calidad, en algún sentido elevada, lo justifica.

La búsqueda tabú ha sido aplicada en una gran variedad de problemas con considerable éxito ya que presenta un esquema de gran adaptabilidad que se ajusta a los detalles del problema considerado. Por otro lado, existen pocos conocimientos teóricos que guíen el proceso de ajuste que requiere la búsqueda tabú y cada usuario debe recurrir a la información práctica disponible y a su propia experiencia.

Búsqueda local iterativa (Iterated local search) [34]: En este algoritmo, los movimientos se realizan sólo si se mejora la solución, ver figura 2.15. Tiene la ventaja de ser flexible y aplicable con gran generalidad, pues sólo requiere de unas cuantas especificaciones como: representación de la solución, una función de evaluación y un método eficiente para explorar entornos. No obstante puede quedar estancada en óptimos locales fácilmente. En este sentido, se ha investigado mucho respecto del alcance de la exploración que provea soluciones de alta calidad en tiempos razonablemente bajos.

GRASP (Greedy Randomized Adaptative Search Procedures) [21][22]: Este enfoque, es goloso o voraz porque siempre escoge el mejor candidato para formar parte de la solución. Se sabe sin embargo, que la calidad de los algoritmos golosos se relaciona con

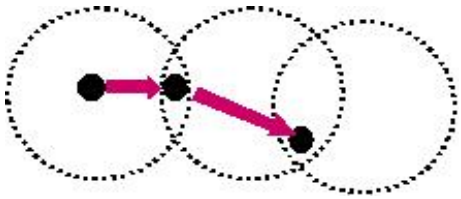

Figura 2.15: Movimientos en la búsqueda local iterativa.

las características de las instancias que pretenden resolver.

Algoritmos evolutivos [5][27]: Simulan el proceso evolutivo y se requiere: Codificar las estructuras que se replicarán, operaciones que afecten a los *individuos*, una función de aptitud y un mecanismo de selección. Los Algoritmos Genéticos son representativos de esta área, introducidos por Holland [28][29] y en los cuales utilizó conceptos de genética, población y teoría de la evolución para construir algoritmos que tratan de optimizar la aptitud de los individuos de una población a través del paso de generaciones.

El esquema de trabajo de los algoritmos genéticos ha sido aplicado a un gran número de problemas donde han aflorado buenos resultados, incluido el problema de programación de horarios. Los algoritmos genéticos conforman una extensa clase de la cual emergen muchos enfoques muy parecidos entre sí.

Dentro de las principales dificultades de estos algoritmos, como en casi todas las metaheurísticas, es que los parámetros iniciales de ejecución dependen en gran medida de pruebas empíricas por parte del usuario para llevar a buen éxito las soluciones.

De todas las metaheurísticas mencionadas la que ha reportado mejores resultados al optimizar el JSSP es la búsqueda tabú, frente a métodos exactos y metaheurísticas [39].

Dada la dificultad del problema de programación de horarios y su impacto económico en las plantas productivas, es de interés creciente en la investigación, desarrollándose para su solución heurísticas cada vez más precisas y veloces. Sin embargo, hasta el momento las técnicas que se conocen tienen aspectos desfavorables como el consumo de tiempo computacional para obtener una buena solución.

Una metaheurística relativamente reciente es la **Optimización basada en Colonias de Hormigas** la cual ha despertado el interés por la eficiencia con que puede resolver problemas de optimización combinatoria. Sobre esta técnica discutiremos más en el siguiente capítulo.

2.6.2. Breve descripción histórica

Durante la década pasada, un número creciente de procedimientos metaheurísticos han sido desarrollados para solucionar problemas de optimización difíciles como el JSSP. La figura 2.16 muestra brevemente los métodos que han abordado este problema.

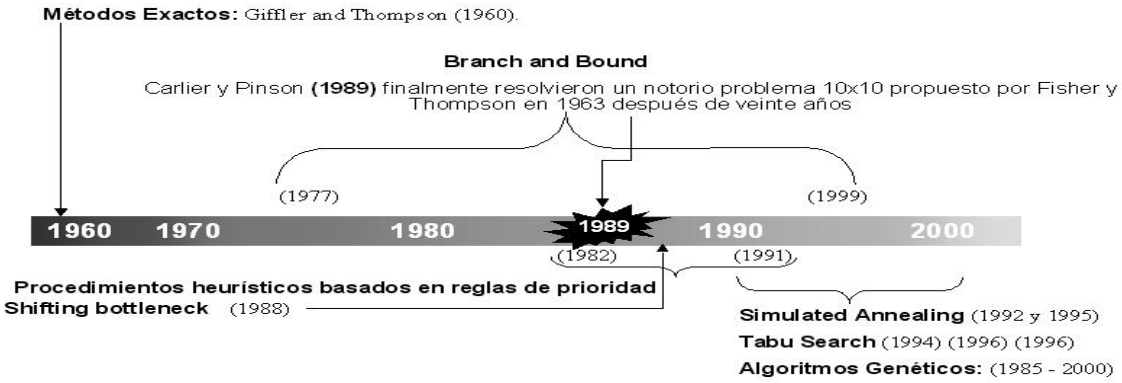

Figura 2.16: Diagrama de tiempo de métodos que han abordado el JSSP.

Como se puede observar en la figura 2.16, el problema se ha tratado de resolver desde los años sesenta con métodos exactos y, desde entonces, ante la ineficiencia de dichos métodos para instancias representativas del problema, se han aplicado diversas heurísticas hasta nuestros tiempos. Esto ha sido motivado por la alta complejidad del problema, tal como puede observarse en el cronograma de la figura 2.16 donde se aprecia que no fue sino hasta el año 1989 que pudo resolverse una instancia de 10×10 planteada 26 años antes por Fisher y Thompson.

2.6.3. Conclusiones

Podemos finalizar mencionando que:

- Los problemas de 15 x 15 son considerados más allá del alcance de los métodos exactos de hoy.
- Entre las técnicas que han dado mejores resultados al resolver el problema del Job Shop Scheduling destacan las de aproximación [39].
- De todas las técnicas que han sido probadas con el JSSP, la búsqueda tabú ha tenido mayor robustez y calidad en los resultados.
- Hasta el momento no existe una técnica específica que resuelva eficientemente todo tipo de instancias del problema JSSP.

2.7. Importancia y justificación de estudio

El Job Shop Scheduling Problem es, dentro de una gran variedad de problemas de planeación de recursos, uno de los que han generado un mayor número de estudios. Esto se debe principalmente a que:

- Es un problema de aplicación práctica y que podemos encontrar en la vida real.
- Como todos los problemas de programación, es fácil de comprender. Sin embargo, encontrar la solución óptima resulta altamente costoso debido a que, con cada recurso o actividad agregada al problema, el tamaño del espacio de búsqueda tiende a crecer en forma exponencial, porque el número de permutaciones posibles será cada vez mayor.
- Hasta el momento no se conoce algoritmo alguno que sea capaz de resolver todas las instancias de este problema, aunque tampoco se ha demostrado que éste no existe.

Capítulo 3

El Ant System (AS)

En este capítulo se introduce el concepto de "heurística", se da una introducción a la computación evolutiva y a las heurísticas evolutivas y bioinspiradas, para posteriormente explicar el comportamiento real de las colonias de hormigas, que sirvieron de inspiración para construir la metaheurística de Ant Colony Optimization (ACO). Posteriormente, daremos paso a la presentación del algoritmo más básico perteneciente a esta clase: el Ant System. De este algoritmo se describirá el pseudocódigo y algunas variantes.

3.1. Introducción

Como se definió en el capítulo anterior, existen diversas técnicas para resolver problemas de optimización; tales técnicas pueden ser clasificadas como técnicas exactas o técnicas aproximadas. Sin embargo, existen problemas para los cuales no se conoce un algoritmo que pueda resolver todas sus instancias en tiempo polinomial puesto que tienen espacios de búsqueda muy grandes, lo que hace pertinente el uso de técnicas aproximadas que suelen ser conocidas como heurísticas. La palabra *heurística* se deriva del griego *heuriskein*, que significa "encontrar" o "descubrir". En [11], heurística se define como:

> *"A un proceso que puede resolver un cierto problema, pero que no ofrece ninguna garantía de lograrlo, se le denomina heurística para ese problema".*

Sabemos que estas técnicas retoman su inspiración de diversas áreas de estudio tales como: matemáticas, física, química, biología, etc. De este último campo, podemos mencionar las heurísticas inspiradas en el principio de selección natural, las cuales se conocen en conjunto como *computación evolutiva* o *algoritmos evolutivos*.

3.2. Heurísticas Evolutivas y Bioinspiradas

En las *heurísticas evolutivas* básicamente se simula el proceso evolutivo que actualmente conocemos como *teoría evolutiva* propuesta originalmente por Charles Darwin [11]. Las heurísticas evolutivas requieren de al menos los siguientes componentes básicos:

- Codificar las estructuras que se replicarán.
- Operaciones que afectarán a los individuos.
- Una función de aptitud.
- Un mecanismo de selección.

Un ejemplo de estas heurísticas evolutivas son los Algoritmos Genéticos (AG) [28][29] desarrollados por John Holland a principios de los sesentas, y que originalmente fueron llamados "planes reproductivos genéticos". En estos algoritmos se enfatiza la importancia de la cruza sexual sobre el de la mutación.

Las *heurísticas bioinspiradas* consideran el conjunto de algoritmos que simulan un proceso "inteligente" de los animales que viven en sociedades como las aves, las abejas, las termitas y las hormigas. En estas heurísticas no necesariamente se simula un proceso de evolución.

Ejemplos de estas heurísticas bioinspiradas son:

Particle Swarm Optimization (PSO) [20][30] es una técnica de optimización estocástica basada en poblaciones desarrollada por Russell Eberhart y James Kennedy en 1995; esta técnica está inspirada en el comportamiento social de las aves que vuelan en grupo.

Ant Colony Optimization (ACO) [14], que se describirá más ampliamente en la siguiente sección.

3.3. Ant Colony Optimization (ACO)

Una metaheurística es un conjunto de conceptos algorítmicos que pueden ser usados para definir métodos heurísticos aplicables a un conjunto de problemas diferentes.

Ant Colony Optimization (ACO) es una metaheurística que engloba un conjunto de técnicas de optimización inspiradas en el comportamiento colectivo de forrajeo de las hormigas, las cuales son capaces de encontrar un camino corto entre el nido y la fuente de alimento; esta técnica nació con la tesis doctoral de Marco Dorigo en Milán, Italia en

1992 [14][19][16][17][18]. En las siguientes secciones se describe el comportamiento natural de las colonias de hormigas y la adaptación que dio origen al primer algoritmo de esta clase.

3.3.1. Inspiración biológica

El comportamiento "inteligente" de diversos animales sociales como hormigas, abejas, termitas, aves y peces, ha motivado su investigación por parte de científicos quienes están interesados en entender las interacciones sutiles que existen entre estos animales que crean grandes y elaboradas comunidades. Además de los biólogos, los ingenieros esperan aplicar el conocimiento obtenido por los científicos para resolver problemas intrincados en la ciencias de la computación, las redes de comunicaciones y la robótica, entre muchas otras áreas.

Veamos ahora hechos biológicos:

- Se sabe que cerca del 2 % de todos los insectos son sociales.
- Alrededor del 50 % de todos los insectos sociales son hormigas.
- Las hormigas son insectos que habitan nuestro planeta desde hace 100,000,000 de años, los humanos sólo lo habitamos desde hace 50,000 años.

Algunas propiedades de estas sociedades son que constan de un conjunto de entidades simples, en las que no existe un control global y la organización se puede dar de dos formas:

1. Comunicación directa: que puede ser en forma visual entre cada individuo de la sociedad.
2. Comunicación indirecta: aquí Grassé introdujo el término *Stigmergy* en 1959 [25], en el cual se habla de la comunicación entre individuos a través de otros medios tales como los elementos químicos.

El resultado de aplicar estas propiedades en conjunto es que tareas complejas pueden ser realizadas en forma cooperativa.

Ejemplos naturales de estas sociedades:

1. Comportamiento en la construcción de nidos de avispas, ver figura 3.1.
2. Comportamiento en la construcción de nidos de hormigas tejedoras, ver figura 3.2.
3. Comportamiento de forrajeo de las hormigas, ver figura 3.3.

Figura 3.1: Nidos de avispas y Modelo visual.

Es de este último ejemplo sobre el que describiremos a continuación el comportamiento natural de forrajeo de las colonias de hormigas. Éste es un proceso que resulta interesante pues las hormigas, siendo insectos casi ciegos, pueden determinar el camino más corto entre su nido y la fuente de alimento. Como la mayoría de las especies de hormigas, las cosechadoras rojas se comunican mediante el tacto y el olfato. Pero en vez de oler el aire, utilizan sus sensibles antenas para detectar olores en sus alrededores. Con las antenas son capaces de detectar olores muy sutiles producidos por hidrocarburos. La superficie exterior del cuerpo de una hormiga contiene unos 25 hidrocarburos diferentes, que emiten aromas ligeramente diferentes, imperceptibles para los humanos. Los hidrocarburos son moléculas simples de hidrógeno y carbono. Pero cambios limitados en su concentración pueden provocar modificaciones de comportamiento muy importantes entre las hormigas. Estos hidrocarburos que sirven para comunicarse se denomina feromona. A continuación se describe el proceso mediante el cual, utilizando la feromona, las hormigas son capaces de construir la ruta más corta entre el nido y el alimento.

Inicialmente, las hormigas desconocen el camino más corto entre el nido y la comida para lo cual, al inicio tendrán una distribución azaroza por todo el espacio explorado, ver figura 3.4 (a). Al cabo de cierto tiempo, y sabiendo que las hormigas se mueven a una misma velocidad constante, la cantidad de feromona se hará más persistente en aquellos lugares que son más transitados y evitará su rápida evaporación. De tal forma, las distancias más

Figura 3.2: Construcción de nidos de hormigas tejedoras.

Figura 3.3: Comportamiento de forrajeo de las hormigas.

cortas entre el nido y la fuente de alimento tenderán a ser transitadas más frecuentemente por las hormigas, lo que permitirá hacer más fuerte el rastro de los caminos más cortos y éstas tenderán a abandonar con el tiempo los caminos largos, ver figura 3.4 (b).

Si esta ruta se pierde por el bloqueo de algún objeto o por los cambios ambientales, ver figura 3.4 (c), una vez que las hormigas se encuentren en el punto que se ha perdido el rastro volverán a elegir aleatoriamente un camino, ver figura 3.4 (d), algunas hacia arriba y otras hacia abajo. Sin embargo, aquellas que elijan ir por la parte de arriba que es el camino más corto, alcanzarán el otro extremo con mayor rapidez que las que eligieron el camino más largo, de tal forma que, transcurrido cierto tiempo (por retroalimentación de la feromona), volverán a reestablecer el camino corto y terminarán abandonando el camino largo, ver figura 3.4 (e).

De este modo, las colonias de hormigas pueden establecer rutas cortas utilizando la feromona como medio de intercambio de información, lo que se conoce como *stigmergy* [25].

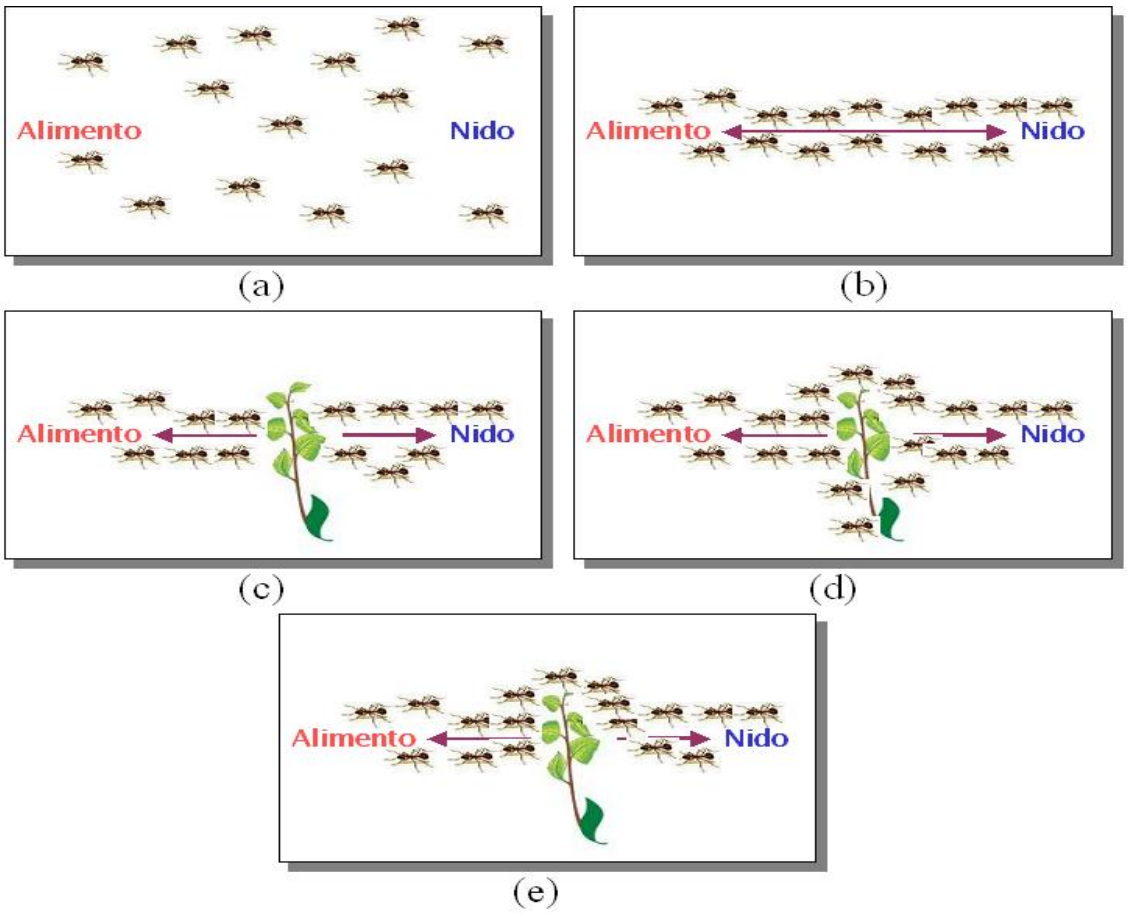

Figura 3.4: Comportamiento adaptativo de las hormigas.

De tal forma, aquellos rastros que contengan mayor cantidad de feromona serán seguidos con mayor seguridad por las hormigas, a diferencia de los rastros que son pobres en cantidades de este químico.

3.3.2. Descripción de la metaheurística ACO

Informalmente, un algoritmo de ACO puede ser visto como la interrelación de tres procedimientos [19]: Construcción de soluciones por hormigas, actualización de la feromona y un Servidor de acciones. La figura 3.5 muestra el pseudocódigo de esta metaheurística.

La *construcción de soluciones* administra una colonia de hormigas que visitan estados adyacentes de un problema considerado (previamente modelado). Las hormigas pueden moverse aplicando una política de decisión estocástica usando la información de los rastros de

```
Procedimiento ACOMetaheuristic
    Actividades programadas
        Construcción de soluciones por hormigas
        Actualización de feromona
        Servidor de acciones
    Fin-Actividades programadas
Fin-Procedimiento
```

Figura 3.5: Pseudocódigo de la metaheurística ACO.

feromona y la información heurística. De esta forma, las hormigas construyen incrementalmente una solución al problema.

La *actualización de la feromona* es el proceso mediante el cual los rastros de feromona son modificados. El valor del rastro puede incrementarse debido a que las hormigas depositan feromona en cada uno de los componentes o conexiones que usan para moverse de un nodo a otro del problema. Y el valor del rastro también puede decrementarse por medio de la simulación de la evaporación de feromona, lo que evita una convergencia prematura del algoritmo.

El *servidor de acciones* es un procedimiento utilizado para implementar acciones centralizadas las cuales no pueden ser desarrolladas por las hormigas en forma individual. Un ejemplo de estas acciones puede ser la activación de un procedimiento de optimización local o la compilación de información global que puede ser usada para tomar decisiones que modifiquen el comportamiento del algoritmo en forma general o parcial.

3.4. Ant System Clásico

El Ant System (AS) es el primer algoritmo perteneciente a la metaheurística de ACO. Fue desarrollado por Dorigo en 1992 y su primera aplicación fue en el problema del agente viajero [14][19][16][17][18].

3.4.1. Descripción

Para describir el mecanismo del Ant System nos basaremos en el problema del vendedor viajero, el cual se presenta a continuación [14]:

> *"Sean C un conjunto de N ciudades de un territorio. La distancia entre cada ciudad viene dada por la matriz $D = N \times N$, donde $d[x, y]$ representa la distancia que hay entre la ciudad x y la ciudad y. El objetivo es encontrar una ruta que, comenzando y terminando en una ciudad concreta, pase una sola vez por cada una de las ciudades y minimice la distancia recorrida por el viajante. Es decir, encontrar una permutación $P = \{c_0, c_2, ..., c_{n-1}\}$ tal que $d_P = \sum_{i=0}^{N-1} d\left[c_i, c_{i+1 mod(N)}\right]$ sea mínima."*

Considerando que C es el conjunto de ciudades a ser visitadas una sola vez con el objeto de encontrar la longitud mínima de recorrido, se tiene que el número de hormigas (artificiales) utilizadas para el problema será:

$$MAXH = \sum_{i=0}^{N-1} C_i \tag{3.1}$$

Para el conjunto de distancias entre las ciudades se define la *matriz de distancias* $D = \{d_{ij}, distancia\ entre\ las\ ciudades\ i, j\}$, a partir de la cual se calcula la *visibilidad* $\eta_{ij} = \frac{1}{d_{ij}}$. Así mismo, el AS utilizará una *matriz de feromonas* para almacenar la información de los caminos recorridos por las hormigas depositando una cantidad de feromona por cada par de ciudades (i, j); dicha matriz está dada por $\tau = \tau_{ij}$.

$\tau(i, j)$ especifica la intensidad del rastro de las feromonas en el arco (i, j), y se actualiza según:

$$\tau(i, j) = \rho \times \tau_{ij} + \Delta\tau_{ij} \tag{3.2}$$

donde ρ es el *coeficiente de persistencia* de las feromonas, de forma tal que $(1 - \rho)$ representa la *evaporación* de la feromona para el arco (i, j), mientras que la cantidad de feromona depositada en un arco (i, j), está dada por:

$$\Delta\tau_{ij} = \sum_{k=1}^{MAXH} \Delta\tau_{ij}^{k} \tag{3.3}$$

con $\Delta\tau_{ij}^{k}$ representando la cantidad de feromona depositada en el arco (i, j) por la hormiga k.

Para satisfacer la restricción de que cada hormiga visite todas las ciudades una sola vez, se asocia a cada hormiga k una estructura de datos llamada *lista tabú* que guarda las ciudades ya visitadas por dicha hormiga. Una vez que todas las ciudades han sido visitadas, el trayecto (ciclo) es completado y la lista tabú se almacena en espera de ser evaluado el costo total del recorrido.

Durante la ejecución del algoritmo Ant System, cada hormiga elige en forma probabilística la próxima ciudad a visitar, realizando un cálculo de probabilidad que está en función de la distancia y la cantidad de feromona depositada en el arco que une a las ciudades origen i con las ciudades destino j, esto es:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \times [\eta_{ij}]^\beta}{\sum_{j \notin Tabu_k} [\tau_{ij}]^\alpha \times [\eta_{ij}]^\beta}, si\ j \notin Tabu_k \tag{3.4}$$

donde α y β son constantes que expresan la importancia relativa del sendero de feromonas y la distancia entre las ciudades respectivamente. Así, un alto valor de α significa que el sendero de feromonas en muy importante y que las hormigas tienden a elegir caminos por los cuales otras hormigas ya pasaron. Si por el contrario, el valor de β es muy alto, las hormigas tienden a elegir la ciudad más cercana.

Mientras no hayan completado la ruta, las hormigas seguirán eligiendo ciudades hasta llenar su lista Tabú. Una vez que todas las hormigas han completado sus recorridos se procede a actualizar la matriz de feromonas con las ecuaciones antes descritas. Para esto, se debe calcular la longitud del trayecto realizado por cada hormiga.

La cantidad de feromona que se deposite en cada arco es proporcional a la distancia del recorrido completo encontrado por cada hormiga y por lo tanto, el cálculo se realiza de la siguiente manera:

$$\Delta\tau_{ij}^k = \frac{Q}{L_k} \text{ si la hormiga } k \text{ camina por el arco } (i,j) \tag{3.5}$$

donde Q es una constante y L_k es la longitud del recorrido completo realizado por la hormiga k.

Este proceso se repite iterativamente hasta que se cumpla algun criterio de término.

3.4.2. Pseudocódigo

A continuación se presenta el pseudocódigo del algoritmo secuencial de Ant System en su versión más utilizada [18], ver figura 3.6.

3.5. Algunas variantes del AS

El *Ant System* desde su aparición en 1992, ha despertado el interés de muchos investigadores por sus características que lo hacen particularmente adecuado para los problemas

1 **Fase de inicialización**
  Inicializar contador de ciclos NC
  Para cada arco (i, j):
    valor inicial de $\tau_{ij} = c$; donde c es una constante pequeña positiva
    $\Delta\tau_{ij} = 0$
  Colocar **MAXH** hormigas en **N** ciudades
2 **Colocar la ciudad origen en lista** $tabú_k$ **de cada hormiga**
3 **Repetir hasta llenar** $tabú_k$
  Para cada hormiga:
    Elegir próxima ciudad a ser visitada según ecuación 3.4
    Mover la hormiga a la próxima ciudad
    Insertar la ciudad en $tabú_k$
4 **Repetir para cada hormiga** k
  Regresar a la ciudad origen
  Calcular la longitud L_k del ciclo
  Guardar el camino más corto hasta el ciclo $NC : L_0^{NC} = min\left\{L_0^{NC-1}\ min_k\left\{L_k\right\}\right\}$
  Para cada arco (i, j)
  Calcular $\Delta\tau_{ij}$ según ecuación 3.3
5 **Para cada arco** (i, j)
  Actualizar τ_{ij} según ecuación 3.2
  $\Delta\tau_{ij} = 0$
6 **Aumentar el contador de ciclos** NC
  Si $NC < Nc_{max}$ **entonces**
    Vaciar $tabú_k$
    Ir a la fase 2
  Sino
    Imprimir camino más corto L_0^{NC}
  Fin

Figura 3.6: Ant System Secuencial de Dorigo et al. [18], donde el número a la izquierda indica el número de fase.

de optimización combinatoria. De estas aplicaciones se han derivado diversas variantes del AS que a continuación describimos:

Elitist Ant System (EAS)[14][16][17]: Fue la primera mejora planteada al AS inicial, introducida por el mismo Dorigo. La idea consiste en reforzar la mejor ruta encontrada depositando mayor cantidad de feromona; esta ruta es denotada como T^{bs} *best-so-far-ant*. El reforzamiento adicional de la ruta T^{bs} está dado por la ecuación $\frac{e}{C^{bs}}$, donde e es un parámetro que define el peso dado para la mejor ruta T^{bs}, y C^{bs} es su longitud del recorrido. La ecuación de actualización de la feromona del AS es modificada de la siguiente manera:

$$\tau(i,j) = \tau(i,j) + \sum_{k=1}^{MAXH} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}$$

La evaporación de la feromona es implementada como en el AS original. Los resultados presentados por Dorigo sugieren que el uso de *elitist strategy* con un apropiado valor para el parámetro e permite al AS encontrar buenas rutas y localizarlas en un número bajo de evaluaciones.

Aunque ésta parece una buena estrategia, se puede argumentar que la velocidad de convergencia se acelera y al mismo tiempo se pierde capacidad de exploración pues las hormigas pronto tenderán a circular por un camino que podría ser un óptimo local.

Rank-Based Ant System (AS_{rank})[8]: Otra mejora al AS propuesta por Bullnheimer es llamada AS_{rank} en la cual cada hormiga deposita una cantidad de feromona que es decrementada de acuerdo a un nivel previo marcado para ella. Adicionalmente, como en EAS el *best-so-far-ant* siempre se deposita la mayor cantidad de feromona en cada iteración.

Antes de la actualización de los niveles de feromona, las hormigas son ordenadas en forma decreciente de acuerdo a la longitud del camino encontrado y la cantidad de feromona que una hormiga k deposita, es medida de acuerdo a su nivel r alcanzado en el reordenamiento. En cada iteración, sólo las $(w-1)$ mejores hormigas y la hormiga que obtuvo el mejor recorrido tendrán el derecho de actualización según la siguiente ecuación:

$$\tau(i,j) = \tau(i,j) + \sum_{r=1}^{w-1} (w-r)\,\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{bs}$$

donde la contribución de la hormiga está dada por $\Delta\tau_{ij}^{r} = \frac{1}{C^{r}}$ y $\Delta\tau_{ij}^{bs} = \frac{1}{C^{bs}}$. Los resultados de la evaluación experimental sugieren que el AS_{rank} tiene ligeramente un mejor desempeño que EAS y mucho mejor que AS. Al igual que EAS, en este algoritmo se puede presentar una convergencia prematura al reforzar sólo las mejores rutas localizadas.

MAX-MIN **Ant System (***MMAS***)**[43][44][45]: La contribución realizada con este algoritmo es que los niveles de feromona son controlados a través de un intervalo $[\tau_{min}, \tau_{max}]$. Con este mecanismo se pretende que los niveles de feromona se mantengan estables para evitar una pronta convergencia del algoritmo y ampliar las capacidades de exploración a diferencia de los algoritmos antes mencionados que refuerzan los niveles de feromona en cada ciclo.

Ant Colony System (*ACS***)**[15]: ACS difiere de AS en tres principales aspectos. Primero, el ACS realiza una explotación mayor sobre la experiencia acumulada a través del uso de reglas de selección de acciones, consideradas agresivas. Segundo, la evaporación de la feromona y la actualización de la misma sólo ocurre en los arcos pertenecientes a la mejor ruta encontrada y tercero, cada vez que una hormiga transita un arco (i, j) moviéndose de la ciudad i a una ciudad j, va disminuyendo la feromona del arco transitado para incrementar la exploración de rutas alternativas.

Como puede observarse, esta técnica provee al Ant System de diversas herramientas que le ayudan a no estancarse en un óptimo local y ayuda a la exploración de nuevas rutas. Sin embargo, las decisiones agresivas pueden provocar cambios bruscos en la información compartida que evita que la colonia converja satisfactoriamente en muchos problemas combinatorios.

En esta sección se presentaron algunos de los algoritmos derivados del AS de mayor aplicabilidad. Aunque éstos han demostrado robustez para muchos problemas conocidos del área, aún queda mucho por investigar ante los diversos problemas combinatorios en que hay cabida para poder hacer investigación con AS. Tal es el caso de los problemas de programación de horarios que han sido abordados por diversas heurísticas incluyendo ACO, sin aún alcanzar un grado que garantice en un tiempo relativamente razonable soluciones satisfactorias para cualquier tipo de instancia, por grande que ésta sea.

Capítulo 4

Adaptaciones del AS al JSSP

En este capítulo se presentan las diversas variantes de AS para el JSSP que fueron probadas antes de obtener la técnica definitiva que constituye la propuesta principal de esta tesis. Estas variantes son incluidas en el trabajo como referencia sobre los resultados obtenidos ante cambios básicos en el algoritmo y su respuesta. Se comenzará con el desarrollo del algoritmo original propuesto por Dorigo para el JSSP [12]. Posteriormente se describirá cada una de las técnicas implementadas y, finalmente, se proporcionará un cuadro comparativo que resume los resultados preliminares obtenidos y que dio lugar a la selección de la técnica final que se usará para el estudio comparativo presentado en el siguiente capítulo.

4.1. Primeras pruebas desarrolladas

Como punto de partida se realizó la implementación del AS básico de Dorigo para el problema JSSP [12], el cual describe el funcionamiento de la colonia de hormigas sobre un grafo utilizado para representar el problema.

4.1.1. AS para JSSP básico

A continuación describiremos gráficamente el uso de la colonia de hormigas en un grafo disyuntivo representando el problema JSSP para posteriormente dar una descripción del pseudocódigo utilizado. Como primer paso colocaremos en el punto de partida I a toda la colonia de hormigas, ver figura 4.1.

Por cada hormiga, se debe seleccionar una operación en forma aleatoria e insertarla en su memoria tabú, ver figura 4.2.

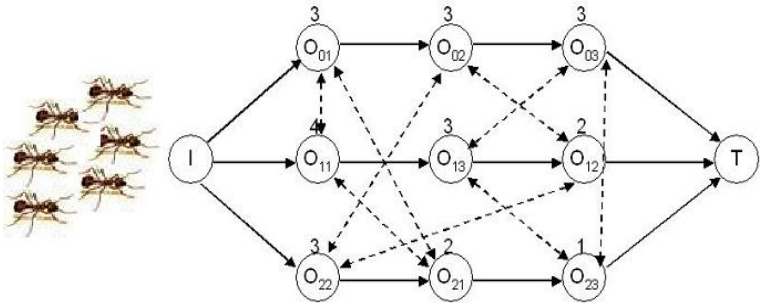

Figura 4.1: Colocación de la colonia de hormigas en el punto inicial I.

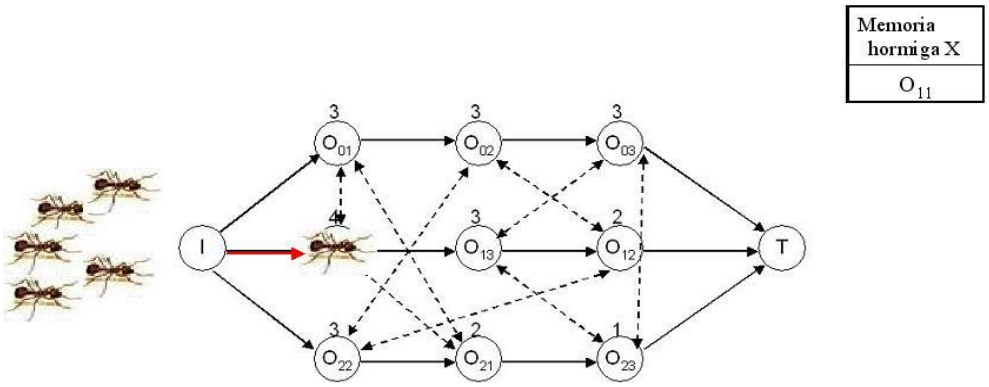

Figura 4.2: Selección aleatoria de operación inicial I e inserción en memoria Tabú.

Luego, se debe ir eligiendo la próxima operación a visitar en función de la ecuación de probabilidad:

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{j \notin Tabu_k} [\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}, si\ j \notin Tabu_k \tag{4.1}$$

donde τ_{ij} es la cantidad de *feromona* depositada en el arco (i, j) que une la operación i a la operación j, y la *decibilidad* está dada por η_{ij}. Esta decibilidad es determinada por la regla LPT *(Longest Processing Time)*, es decir, la operación de tiempo más largo.

En la figura 4.3 se observa cómo la hormiga elige la siguiente operación a ser visitada y la anota en su memoria tabú.

De esta misma manera termina el recorrido llenando la memoria tabú con todas las operaciones y el orden en que fueron visitadas, ver figura 4.4.

Posteriormente, se hacen circular una a una todas las hormigas de la colonia hasta que todas hayan completado su propia ruta, ver figura 4.5.

Con la memoria tabú de cada una de las hormigas se procede a realizar la evaluación del programa para determinar el *makespan* de cada una. Después, se realiza la actualización de

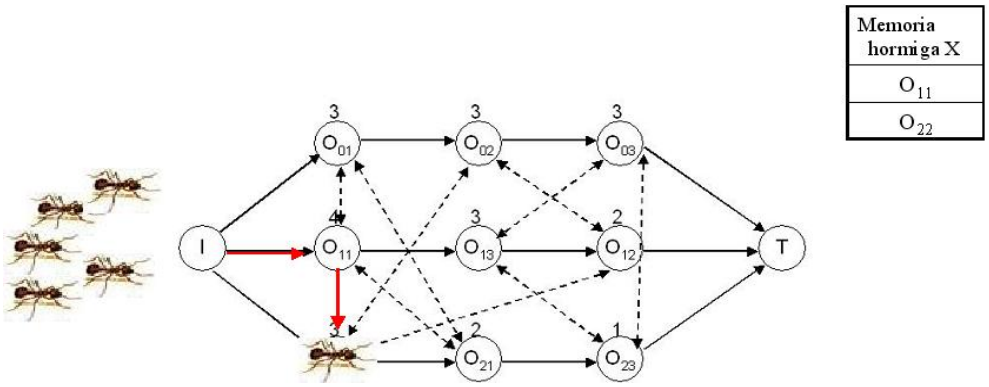

Figura 4.3: Selección de la siguiente operación y la inserción en lista Tabú.

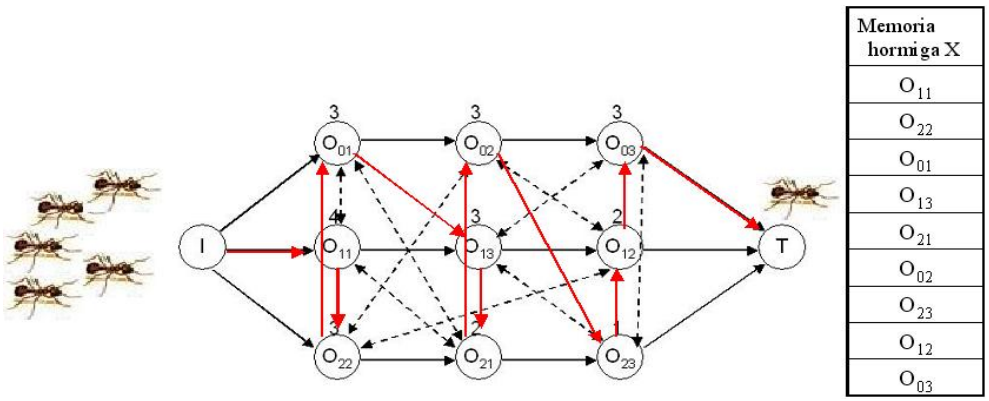

Figura 4.4: Ruta construida por la hormiga ×.

la matriz de feromonas utilizando las ecuaciones ya revisadas en el capítulo anterior y que a continuación se enumeran:

$$\tau\left(i,j\right) = \rho \times \tau_{ij} + \Delta\tau_{ij} \tag{4.2}$$

donde ρ es el *coeficiente de persistencia* de las feromonas, de forma tal que $(1-\rho)$ representa la *evaporación* de la feromona para el arco (i,j), mientras que la cantidad de feromona depositada en un arco (i,j), está dada por:

$$\Delta\tau_{ij} = \sum_{k=1}^{MAXH} \Delta\tau_{ij}^{k} \tag{4.3}$$

La cantidad de feromona que se deposite en cada arco es proporcional al makespan encontrado por cada hormiga y, por lo tanto, el cálculo se realiza de la siguiente manera:

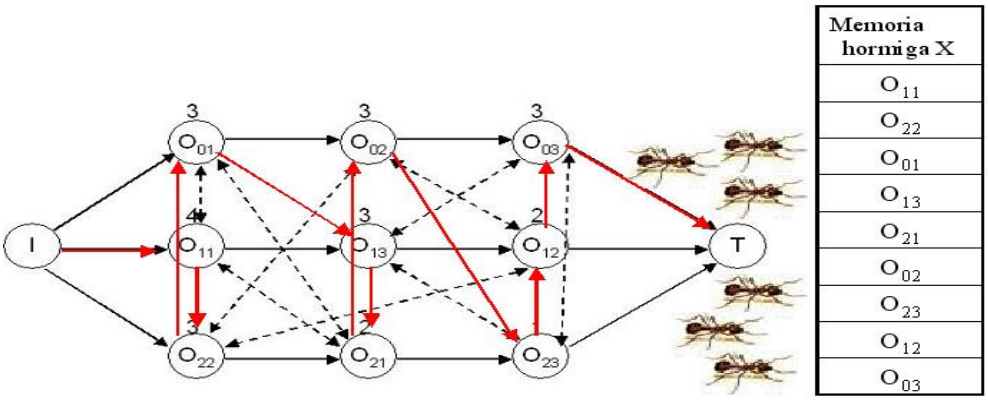

Figura 4.5: Todas las hormigas han circulado por el grafo y han generado sus programas.

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \text{ si la hormiga } k \text{ camina por el arco } (i,j) \tag{4.4}$$

donde Q es una constante y L_k es el *makespan* del programa realizado por la hormiga k.

El número de hormigas (artificiales) utilizadas para el problema está dada por:

$$MAXH = \sum_{i=0}^{J-1} J_i \tag{4.5}$$

donde J representa el número de trabajos de la instancia del problema.

El pseudocódigo descrito en [12] se muestra en la figura 4.6 utilizando las ecuaciones descritas anteriormente.

A partir de este modelo básico se generaron diversas versiones del AS para JSSP. A continuación se detallan cada una de estas técnicas. Los parámetros y detalles de las evaluaciones realizadas se describen hacia el final del capítulo.

Cabe mencionar que para todas estas técnicas **no se utilizaron mecanismos de reparación de los planes de trabajo generados**.

Nota: *Un mecanismo de reparación es un método determinístico incluido en la técnica heurística que ayuda a mejorar las soluciones obtenidas por el motor de búsqueda. Este mecanismo genera planes activos a partir de la solución otorgada por la heurística. En la mayoría de los reportes no se incluye o no se hace referencia a este costo computacional que se encuentra implícito en la ejecución del algoritmo.*

1 **Fase de inicialización**
  Inicializar contador de ciclos NC
  Para cada arco (i, j):
    valor inicial de $\tau_{ij} = c$; donde c es una constante positiva pequeña
    $\Delta\tau_{ij} = 0$
  Colocar **MAXH** hormigas en **N** operaciones
2 **Colocar la primera operación en lista** $tabú_k$ **de cada hormiga**
3 **Repetir hasta llenar** $tabú_k$
  Para cada hormiga:
    Elegir próxima operación a ser visitada según ecuación (4.1)
    Mover la hormiga a la próxima operación
    Insertar la operación en $tabú_k$
4 **Repetir para cada hormiga** k
  Calcular el makespan L_k del programa generado
  Guardar el programa de makespan más chico hasta el ciclo $NC : L_0^{NC} = min\left\{L_0^{NC-1}\ min_k\left\{L_k\right\}\right\}$
Para cada arco (i, j)
  Calcular $\Delta\tau_{ij}$ según ecuación (4.3)
5 **Para cada arco** (i, j)
  Actualizar τ_{ij} según ecuación (4.2)
  $\Delta\tau_{ij} = 0$
6 **Aumentar el contador de ciclos** NC
  Si $NC < Nc_{max}$ **entonces**
    Vaciar $tabú_k$
    Ir a la fase 2
  Sino
    Imprimir programa más corto L_0^{NC}
  Fin

Figura 4.6: Ant System Secuencial de Dorigo et al. [12] para el JSSP.

4.1.2. AS para JSSP con arcos individuales

En la primera propuesta de AS para JSSP el planteamiento original que une dos operaciones es a través del uso de un solo arco, de tal forma que si una hormiga transita desde la operación i hasta la operación j o viceversa, la actualización de la feromona se realiza sobre el mismo arco sin indicar la dirección en que se transitó. Por lo tanto, después de varios recorridos, no es posible determinar si la cantidad de hormigas que han transitado por ahí, lo han hecho desde el nodo i a j o de j a i. En otras palabras, no es posible determinar la precendencia elegida entre dos operaciones.

Como ejemplo, en la figura 4.7 se muestra cómo de la operación i a la operación j transita sólo una hormiga y de la operación i a la operación k transitan dos hormigas en el ciclo NC. Sin embargo, en ese mismo ciclo transitaron tres hormigas de la operación j a la operación i.

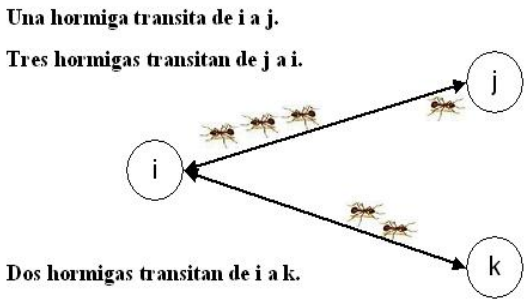

Figura 4.7: Tránsito de las hormigas por los arcos que unen operaciones.

Así, después de realizar la actualización de la feromona, el arco (i, j) tendrá una mayor contribución de feromona que el arco (i, k) de tal manera que, en el ciclo $NC + 1$, si una hormiga se localiza en la operación i, cuando realice el cálculo probabilístico, podría elegir una operación en la cual, *"engañosamente"*, parecería que fueron cuatro hormigas las que transitaron de la operación i a la operación j. Ver figura 4.8.

Bajo esta observación se deriva la primera modificación, la cual consiste en colocar un arco individual de ida y otro de venida por cada par de operaciones (i, j) que permita diferenciar exactamente la cantidad de feromona depositada en las operaciones recorridas en cada sentido de los arcos. El modelo lo podemos ver gráficamente en la figura 4.9.

Lo que se espera es que el desempeño del algoritmo mejore ante la diferenciación realizada entre arcos. Los resultados son discutidos al final del capítulo.

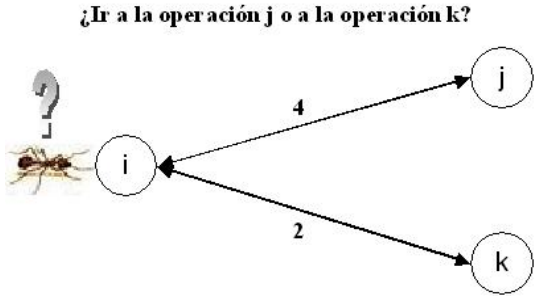

Figura 4.8: Actualización de feromona en arcos.

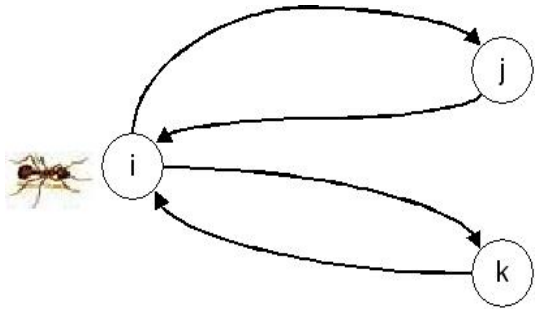

Figura 4.9: Propuesta de arcos separados por el sentido entre las operaciones (i, j).

4.1.3. AS para JSSP con exploración aleatoria del 10 % de los ciclos

En el AS original, desde el primer ciclo existe una cantidad mínima de feromona que es igual en todos los arcos. Sin embargo, de la observación natural sabemos que cuando las hormigas se distribuyen en forma aleatoria lo hacen para reconocer un camino sobre el cual, a cada paso, van depositando una cantidad de feromona que en un principio no existe. Entonces, la propuesta es que las hormigas incialmente exploren en forma aleatoria el grafo del problema en un intervalo del 1 al 10 porciento de los ciclos. Sólo los mejores makespan localizados serán depositados en la matriz de feromonas para que posteriormente esa información sea la que guíe la búsqueda en el resto de los ciclos.

De los experimientos realizados, se observó que un 10 porciento de los ciclos de exploración aleatoria proveían mejores resultados. Con esto se buscó generar una mayor diversidad de planes de trabajo que redistribuyera la feromona a cada ciclo, puesto que es el único medio de comunicación entre la colonia de hormigas.

4.1.4. AS para JSSP con cálculo de Distancia de Hamming

La siguiente versión fue una variante del AS clásico con la intención de verificar la diversidad de los planes de trabajo y, de este modo, proponer nuevos métodos o modificaciones a las ideas generadas.

Para medir la diferencia de planes de trabajo generados se propuso hacer uso de la distancia de Hamming entre dos planes generados de la siguiente manera:

Se recorrían ambos planes generados por cada par de hormigas $x1$ y $x2$, y las diferencias encontradas en las posiciones (o secuencia de los trabajos) marcaría las diferencias que serían medidas. En la figura 4.10 se muestran dos planes de trabajo y las diferencias encontradas.

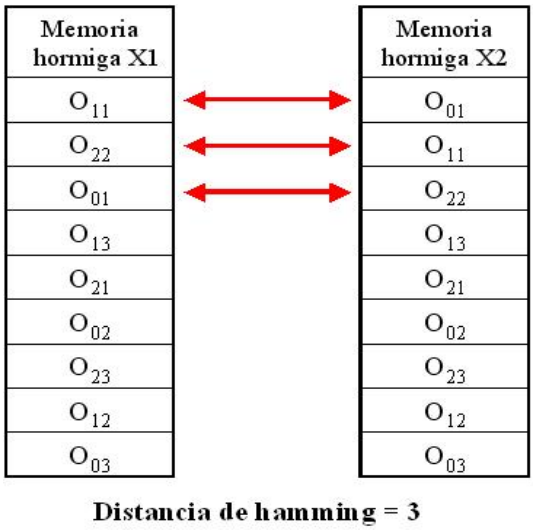

Figura 4.10: Cálculo de la distancia de hamming.

Obsérvese que en los planes de la figura 4.10, a partir de la operación $(1, 3)$ la ruta construida por ambas hormigas es la misma. Esta observación pudo verificarse en todas las pruebas de cálculo desarrolladas. Esto nos permitió concluir que, llegadas dos hormigas a una misma operación, el cálculo es exactamente el mismo puesto que la cantidad de feromona depositada y las distancias son las mismas. En otras palabras, la información que las hormigas encuentran para realizar el cálculo de probabilidad es la misma. Por lo tanto, la diversidad de los planes de trabajo se pierde y, peor aún, si dos o más hormigas en un mismo ciclo comienzan desde la misma operación, realizarán el mismo programa de trabajo. En la figura 4.11 se muestra cómo dos hormigas que iniciaron de operaciones diferentes, una vez llegadas a una misma operación, eligen seguir por el mismo camino.

Esta observación dio origen a nuevas ideas que se incluyeron también en esta tesis y que

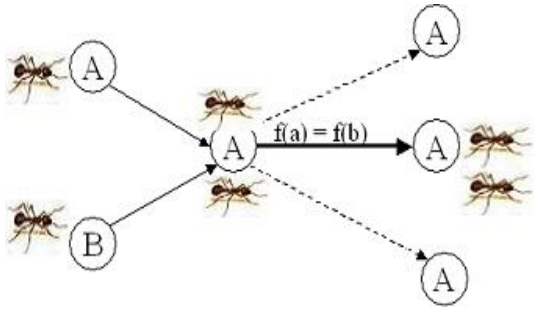

Figura 4.11: Dos hormigas eligiendo el mismo camino.

a continuación se mencionan.

4.1.5. AS para JSSP usando distribucion inversa

Los arcos más recorridos tendrán mayores cantidades de feromona que aquellos menos explorados (ver figura 4.12), lo que hace que la diversidad de programas se pierda por dejar de explorar otras posibilidades en los programas. Para ello se plantea el uso de algo que hemos denominado *distribución inversa* en esta tesis.

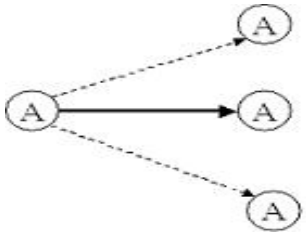

Figura 4.12: Los arcos más explorados contienen mayor cantidad de feromona.

La *distribución inversa* es un procedimiento que consiste en la disminución de la feromona en arcos más explorados y hacer más intenso el rastro de feromona en aquellos que no han sido tan explorados. Esto propiciará que las hormigas seleccionen rutas poco transitadas. La fórmula de cálculo de la distribución de la feromona en los arcos es la siguiente:

$$\frac{1}{\text{Valor de la feromona en el arco}}$$

La distribución inversa sólo se aplicará a N nodos, donde N es un parámetro suministrado por el usuario y no podrá ser mayor al número de trabajos de la instancia del problema. Esto es con la intención de que el algoritmo pueda llegar a converger con todos sus individuos y sólo se permita un número de variación controlable. Aplicando la distribución inversa

se hacen más atractivos los arcos menos explorados, ver figura 4.13.

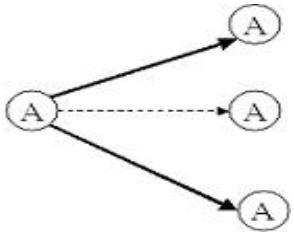

Figura 4.13: Arcos después de aplicar la distribución inversa.

4.1.6. AS para JSSP con cambio de parámetros (α, β)

Una forma de inducir programas diferentes por las hormigas es manipular en forma aleatoria los parámetros α y β. Así, aunque dos hormigas lleguen a un mismo punto y se localice la misma información sobre la feromona y los tiempos de procesamiento, ambas serán portadoras de diferente influencia de α y β que son los parámetros encargados de regular la importancia que la hormiga dará a la feromona o a los tiempos de procesamiento. Ver figura 4.14

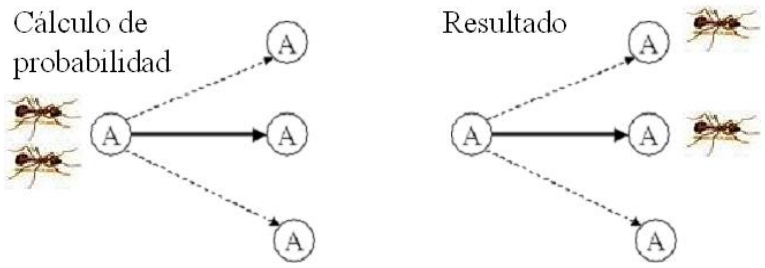

Figura 4.14: Hormigas con diferente influencia de feromona elegirán operaciones distintas.

Los cambios de la influencia de α y β se realizan en cada ciclo para que la información que porte la hormiga sea diversa. Esto no interfiere con la convergencia del algoritmo puesto que la concentración de feromona sigue siendo la misma para toda la ejecución.

4.1.7. AS para JSSP con cambio de parámetro de persistencia (ρ)

La matriz de feromonas es el medio de comunicación entre las hormigas de la colonia. Por lo tanto, se propone que el parámetro ρ, encargado de determinar la parte de feromona que permanece en cada ciclo y por ende, la cantidad que debe evaporarse, varíe a lo largo del

programa para que la cantidad de feromona que queda depositada en cada ciclo sea distinta y así generar planes diferentes.

4.1.8. AS para JSSP con actualización de un porcentaje de hormigas

En este método se propone que sea sólo un porcentaje de las hormigas de la colonia el que tenga el derecho de actualización. El porcentaje es un parámetro que el usuario debe introducir. El método no pretende ser elitista, por lo cual entre las hormigas que pueden ser elegidas para actualizar podrán encontrarse incluso, las que obtuvieron los peores resultados. Si son 10 hormigas y sólo se pretende que se actualice un 70 por ciento, entonces se seleccionarán 7 hormigas aleatoriamente para actualizar la matriz de feromona. La figura 4.15 muestra la selección de este porcentaje de hormigas.

Figura 4.15: Selección de un porcentaje de hormigas para actualizar la matriz de feromona.

La selección de buenos y malos elementos realiza cambios en la comunicación y diversifica los programas para nuevos recorridos.

4.1.9. AS para JSSP con exploración usando reglas de prioridad aleatorias

En [12] se definieron un conjunto de reglas de prioridad que son propuestas en la literatura para el problema del JSSP y que a continuación describimos:

- SPT: Seleccionar la operación con tiempo de procesamiento más corto.
- LPT: Seleccionar la operación con tiempo de procesamiento más largo.

La decibilidad η en AS es calculada bajo la regla LPT, que es constante durante toda la ejecución del algoritmo, para lo cual se propuso utilizar reglas de prioridad aleatoria entre SPT y LPT. El procedimiento es el siguiente:

Cuando la hormiga llegue a un punto donde tiene que evaluar la próxima operación a seguir, se determinará en forma aleatoria la regla de prioridad que deberá usar para el cálculo. Así tenemos que las reglas pueden ser:

$$\text{Regla de tiempos largos: } \eta_{ij} = TP_{ij}$$

$$\text{Regla de tiempos cortos: } \eta_{ij} = \frac{1}{TP_{ij}}$$

Donde TP_{ij} es el *tiempo de procesamiento* de la operación (i, j). En la figura 4.16 se observa cómo dos hormigas con reglas de prioridad diferentes eligen caminos diferentes.

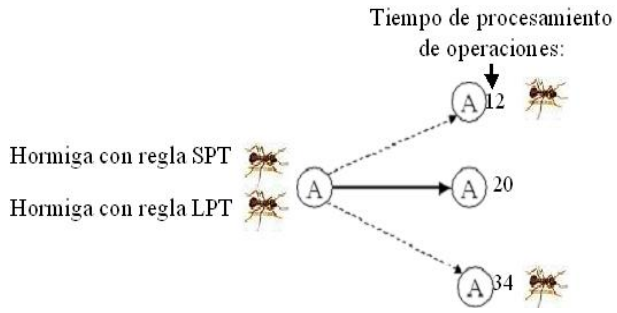

Figura 4.16: Selección de operación con reglas de prioridad diferentes.

Esto permite ampliar la exploración puesto que, aunque dos hormigas tengan la misma información para cálculo de decisión, ésta puede cambiar por la regla de prioridad utilizada, con lo cual se elimina el problema observado en la sección donde se explicó el cálculo de la distancia de Hamming en los programas donde dos hormigas llegadas a una misma operación construían el mismo plan de trabajo.

4.1.10. AS para JSSP calculando desperdicio de tiempo

Cuando existe una construcción gráfica de un plan de trabajo, por ejemplo, usando gráficas de Gantt, se hacen visibles los *"huecos"*, mejor conocidos como *tiempos ociosos* o *muertos* en que una máquina queda sin tarea alguna esperando a que otro trabajo pueda ser procesado en ella; esto lo podemos observar en la figura 4.17.

De esta observación, se propone que después de realizar la actualización de la feromona, se realice un cálculo de aquella precedencia de operaciones que ocasionaron un desperdicio de tiempo y disminuir la feromona en el arco (i, j) en proporción al tiempo desperdiciado, de tal forma que mayores tiempos ociosos generarán una disminución mayor en forma porcentual. Es decir, si el desperdicio generado fue de 6 tiempos, la disminución de la cantidad de

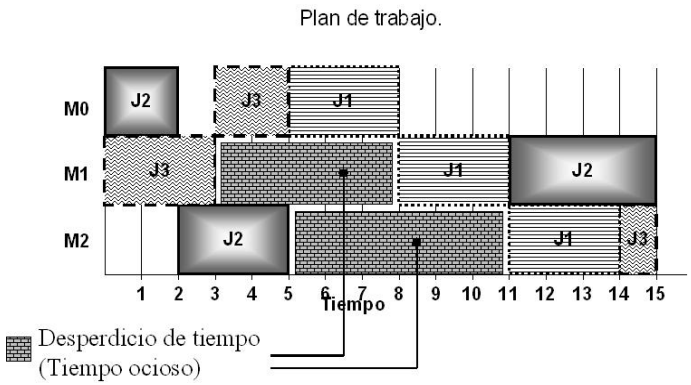

Figura 4.17: Desperdicio de tiempo.

feromona depositada se disminuirá en 6 por ciento, el porcentaje máximo de disminución será de 99 por ciento con la finalidad de que se garantice siempre la existencia mínima de feromona en el arco (i, j).

A continuación se detalla un cálculo que muestra este decremento de feromona, para lo cual tomaremos de referencia la figura 4.17. Supondremos que después de la actualización de feromona todos los arcos contienen un valor de 50 que indica el nivel de feromona.

En la figura 4.17 se observa que de la operación $(0, 1)$ que precede a la operación $(1, 1)$ se genera un tiempo ocioso de 5 unidades entre los trabajos $J3$ y $J1$, lo cual indica que no fueron buenas elecciones en el programa y se disminuirá la feromona en 5 por ciento de lo que hay depositado entre los arcos $(0, 1)$ y $(1, 1)$; así se tiene que el cálculo es de acuerdo a la siguiente fórmula:

$$\nabla\tau_{ij} = \tau_{ij} \times \frac{to}{100} \text{ donde } to \leq 100$$

$$\nabla\tau_{O_{01}O_{11}} = 50 \times \frac{5}{100}$$

$$\nabla\tau_{O_{01}O_{11}} = 2{,}5$$

La nueva cantidad de feromona será:

$$\tau_{ij} = \tau_{ij} - \nabla\tau_{ij}$$

$$\tau_{O_{01}O_{11}} = 50 - 2{,}5$$

$$\tau_{O_{01}O_{11}} = 47{,}5$$

De este modo, cuando el tiempo ocioso de una máquina es mayor, la cantidad de feromona que se disminuirá entre las operaciones que lo generan será mayor, con lo cual se espera que las hormigas no decidan ir por las mismas rutas que generan tiempos ociosos grandes.

4.1.11. AS para JSSP con exclusión de la operación de trabajo similar a la operación actual en la que se localiza la hormiga

De igual modo que en la explicación de la sección anterior, cuando las hormigas construyen una ruta, si se grafica la solución podemos observar que en ocasiones se genera un efecto de escalonamiento de operaciones. Esto impide que operaciones que vienen después se puedan procesar con antelación aunque la máquina esté disponible u ociosa. En la figura 4.17 podemos observar un escalonamiento de las operaciones $(0,1)$, $(1,1)$ y $(2,1)$, de tal manera que la hormiga eligió operaciones del mismo trabajo una tras otra. En este caso, todas las operaciones del trabajo 1 (ver figura 4.18); para lo cual no siempre es deseable por las consecuencias que tiene sobre el makespan generado y el bloqueo de operaciones que pueden procesarse con prioridad.

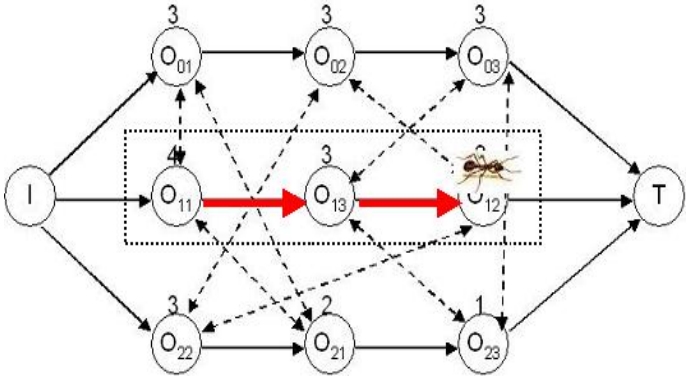

Figura 4.18: Hormiga eligiendo operaciones de un mismo trabajo.

Para que la hormiga pueda moverse de una operación a otra, antes de realizar el cálculo probabilístico, determina primero las posibles operaciones que puede procesar, a lo cual se le conoce como conjunto S. En la figura 4.19 se observa cómo la hormiga se localiza en la operación O_{11} y se calculan las posibles operaciones a las cuales puede moverse, que en este caso son: $S = \{O_{01}, O_{13}, O_{22}\}$ y que en la figura 4.19 se señalan en un recuadro.

Si la hormiga elige ir a la operación del mismo trabajo en que se encuentra actualmente ocasionará un efecto de escalonamiento como se observó anteriormente en la figura 4.17. La propuesta consiste en eliminar del conjunto S la operación de trabajo similar a la operación

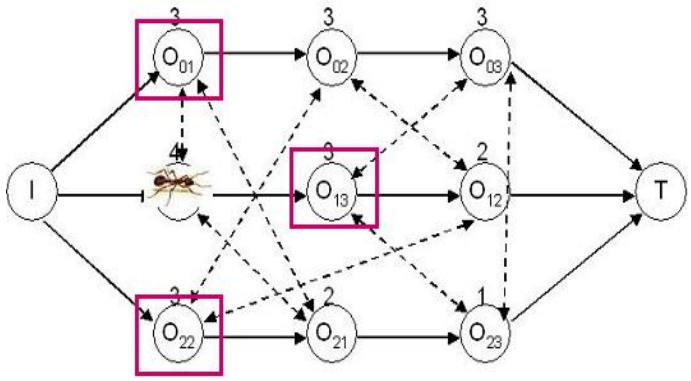

Figura 4.19: Una hormiga determinando el conjunto S.

actual en la que se localiza la hormiga. En nuestro ejemplo debido a que la hormiga se localiza en la operación O_{11}, es decir en el trabajo 1, se procederá a eliminar del conjunto S la operación O_{13}, ver figura 4.20.

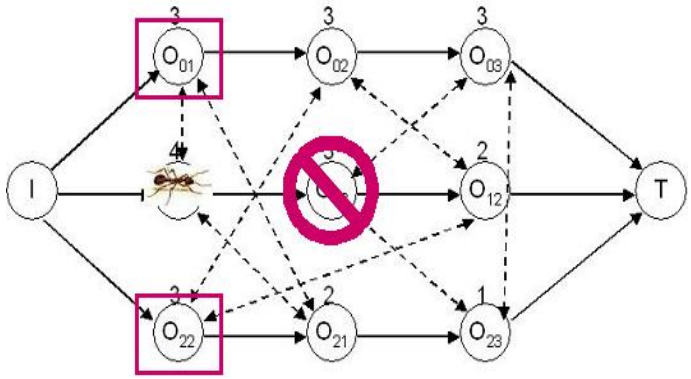

Figura 4.20: Eliminación del conjunto S la operación similar al trabajo actual de la hormiga.

Así se tendrá la posibilidad de generar un plan de trabajo en el que no se bloquee la oportunidad de que otros trabajos puedan procesarse con antelación si es que la máquina está disponible en ese momento.

4.1.12. AS para JSSP una nueva inspiración

Por varios años los científicos han propuesto varias teorías sobre cómo las hormigas pueden encontrar el camino a casa. Una de esas teorías es que lo hacen como las abejas y que pueden recordar señales visuales, pero experimientos realizados con hormigas revelan que pueden andar en la obscuridad y aún con los *"ojos vendados"*. Otra teoría refutada es que las hormigas pueden medir el tiempo que les lleva colocarse desde un extremo a otro.

Y otros estudios han mostrado que una vez que las hormigas encuentran una buena fuente de alimento enseñan a otras hormigas cómo encontrarlo. En el caso de las hormigas del desierto, en que el terreno es cambiante a cada momento por lo estéril de éste y los cambios arenosos con los vientos, las hormigas usan señales *"celestes"* para poder orientarse en la dirección correcta una vez que regresan al nido. Así, han existido diversas teorías sobre su comportamiento.

Pero con pocas señales en el terreno, los científicos se han preguntado: ¿Cómo estos insectos toman siempre el camino más corto y saben exactamente la dirección correcta?

Un nuevo estudio revela que las hormigas usan un podómetro biológico interno y que contar los pasos es una parte crucial en el esquema de búsqueda *nido - alimento - nido* [49]. Esta técnica del podómetro interno fue primeramente propuesta en 1904 pero no fue aprobada sino hasta recientemente [9].

Para corroborar esto, los científicos realizaron un experimento el cual consistió en una *cirugía estética*. Pegaron extensiones parecidas a unos zancos a las patas de unas hormigas para alargar el tamaño de su paso. Y a otras hormigas les acortaron el paso recortando sus patas, así redujeron sus patas prácticamente a tacones. Ver figuras 4.21 y 4.22.

Figura 4.21: Hormigas con zancos.

Las hormigas con el tamaño de paso más grande caminaban el número de pasos adecuado pero como la zancada era más grande, pronto dejaban atrás el objetivo y las hormigas con las patas recortadas no alcanzaban el alimento. Después de acostumbrarse a sus nuevas patas eran capaces de volver a ajustar su podómetro al nuevo número de pasos que tenían que dar con su tamaño de paso.

De estas nuevas investigaciones, se propone un Ant System basado en el *"conteo de pasos"* por cada una de las hormigas, es decir llevar un *"podómetro"* en la estructura de datos y el conteo de pasos equiparable al tiempo total realizado por cada operación que

Figura 4.22: Hormigas con patas recortadas.

agregan al programa. Para ello será necesario realizar un cambio en la estructura de la lista Tabú para poder llevar un mejor control del programa realizado y obtener los datos necesarios para el cálculo. Adicionalmente el dato que mide la "decibilidad" en la ecuación de probabilidad se cambiará por el término "factibilidad" que indicará la prontitud con la que la hormiga podrá reanudar su camino dependiendo de la operación en que se ubique. A continuación se explica el método con un ejemplo:

Como en el Ant System original, la primera operación se coloca en forma aleatoria. Para el ejemplo suponemos que una hormiga parte en la operación $(2, 2)$, ver figura 4.23.

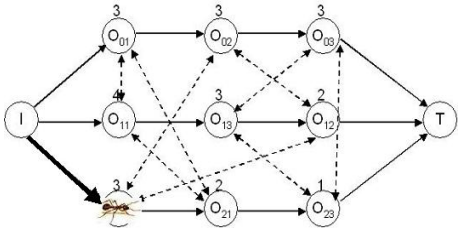

Figura 4.23: Colocación de la primera operación en forma aleatoria.

A partir de la posición actual, se determina el conjunto S de posibles operaciones que pueden procesarse posteriormente, en nuestro caso son las operaciones $S = \{(0, 1), (1, 1), (2, 1)\}$, ver figura 4.24.

El procedimiento consta de tres pasos:

Paso 1. Con las operaciones del conjunto S se calcula el tiempo de inicio de cada una de ellas tomando como base los siguientes puntos:

- Tomar en cuenta el recurso a utilizar y determinar el momento en que puede iniciar la operación, de acuerdo al plan que lleva generado la hormiga.

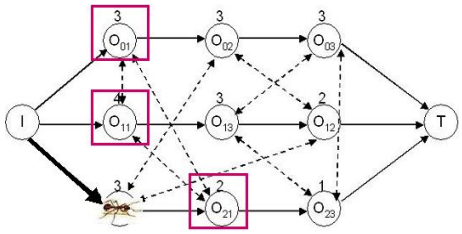

Figura 4.24: Cálculo del conjunto S en procedimiento.

- La restricción de inicio de su operación precedente.
- Si en el tiempo que se determina para que inicie la operación, al menos existe una operación con tiempo de inicio cero, se deberá sumar la unidad a todos los tiempos de inicio.

Para explicar este cálculo en el ejemplo, tomaremos como referencia la gráfica de Gant mostrada en la figura 4.25.

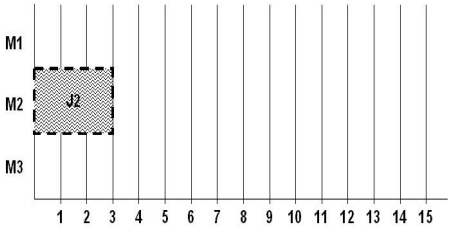

Figura 4.25: Gráfica de Gantt con la primera operación insertada.

De acuerdo a la gráfica podemos observar que el tiempo de inicio para la operación $(0, 1)$ es cero (ver figura 4.26(a)), para la operación $(1, 1)$ es cero (ver figura 4.26(b)) y para la operación $(2, 1)$ el tiempo de inicio más próximo es tres, debido a que la operación que le precede terminará de procesarse hasta el tiempo número tres (ver figura 4.26(c)).

Para concluir el paso, si al menos existe una operación en que el tiempo de inicio fue cero, se sumará la unidad al tiempo de inicio de todas las operaciones; de lo contrario, el tiempo de inicio queda igual. La tabla 4.1 muestra en resumen el cálculo de este primer paso.

Paso 2. Con el tiempo de inicio (TI) calculado para cada operación, se procede a calcular la *factibilidad* η de cada operación dividiendo una constante Q entre el tiempo de inicio de cada operación:

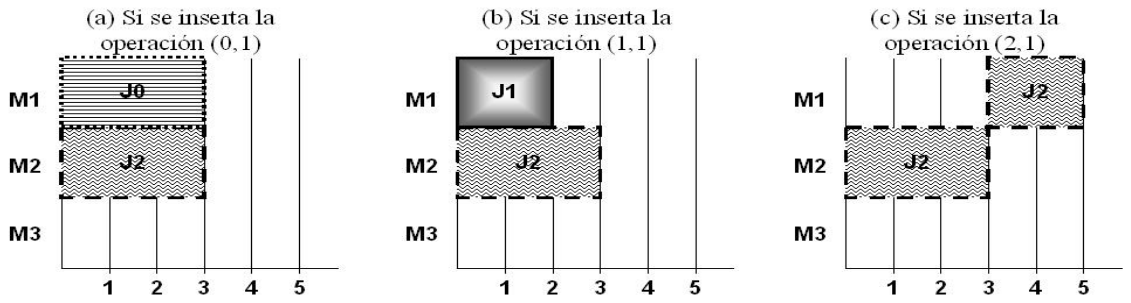

Figura 4.26: Cálculo del tiempo de inicio próximo para cada operación del conjunto S.

trabajo	máquina	tiempo	Tiempo de inicio(TI)	TI+1
0	0	3	0	1
1	0	2	0	1
2	0	2	3	4

Tabla 4.1: Cálculo del tiempo de inicio de cada operación.

$$\eta_{ij} = \frac{Q}{TI_{ij}}$$

La tabla 4.2 muestra en la columna seis este cálculo. Como puede observarse, tienen una mayor factibilidad aquellas operaciones que tienen un tiempo de inicio más próximo. Así podemos saber la prontitud con que puede empezar a *"caminar"* una hormiga y se puede conocer el número de pasos que están ahorrándose al elegir una u otra operación.

Paso 3. Ahora se utiliza la factibilidad η para realizar el cálculo utilizando la ecuación de probabilidad ya conocida.

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}{\sum_{j \notin Tabu_k} [\tau_{ij}]^{\alpha} \times [\eta_{ij}]^{\beta}}, si\ j \notin Tabu_k$$

Suponiendo que α, β y τ, en este caso tomarán el valor de uno, se puede observar que

trabajo	máquina	tiempo	Tiempo de inicio(TI)	TI+1	η
0	0	3	0	1	1
1	0	2	0	1	1
2	0	2	3	4	0.2

Tabla 4.2: Cálculo de factibilidad η.

trabajo	máquina	tiempo	TI	TI+1	η	τ	$\tau^{\alpha}\eta^{\beta}$	p_{ij}
0	0	3	0	1	1	1	1	0.45
1	0	2	0	1	1	1	1	0.45
2	0	2	3	4	0.2	1	0.2	0.09

Tabla 4.3: Cálculo de probabilidad.

tendrán mayor probabilidad de ser procesadas las operaciones que se procesarán con mayor prontitud (ver tabla 4.3).

En el ejemplo anterior se muestra la ventaja de la decibilidad a través del *"conteo de pasos"*. Sin embargo, a esta técnica se agregaron dos procedimientos como apoyo adicional con base en las observaciones realizadas de técnicas anteriores y pruebas experimentales durante el desarrollo de esta técnica. La primera de ellas es que el valor de α y β no son constantes dentro del sistema, es decir, estos valores cambian para cada hormiga al momento de construir una ruta, permitiendo que la importancia que se deba dar a la feromona y a la decibilidad distribuya la búsqueda en ambos sentidos para evitar que dos o más hormigas construyan un mismo plan al portar la misma información. Los valores de α y β son controlados de tal manera que la suma de ambos parámetros sea un 100 % y que en ningún caso, alguno de los parámetros su valor sea dominante o totalitario, es decir, que tome el valor de 0 ó de 100. Para cumplir con estas condiciones antes de que la hormiga comience a construir el programa, se le asignan los valores de α y β con base en el siguiente procedimiento:

$$\alpha = flip\,(0{,}01, 0{,}99)$$

$$\beta = 1{,}0 - \alpha$$

Un segundo procedimiento agregado fue tener las reglas de decisión en forma aleatoria haciendo uso de la decibilidad y el tiempo de procesamiento de la operación. Como se pudo observar en el ejemplo anterior, al calcular la decibilidad se puede dar el caso en el que dos o más operaciones contengan el mismo valor de decibilidad, lo que bien podría arreglarse con una selección aleatoria. Sin embargo, podemos hacer que se utilice una regla de decibilidad con base al tiempo de procesamiento de la operación; para ello la decibilidad se calculará de acuerdo a las fórmulas siguientes:

Regla de decibilidad de operaciones con tiempo de procesamiento largo:

$$\eta_{ij} = \frac{Q}{TI_{ij}} \times TP_{ij}$$

Regla de decibilidad de operaciones con tiempo de procesamiento corto:

$$\eta_{ij} = \frac{Q}{TI_{ij}} \times \frac{1}{TP_{ij}}$$

donde Q es una constante, TI_{ij} es el tiempo de inicio de la operación (i, j) y TP_{ij} es el tiempo de procesamiento de la operación (i, j).

La aplicación de estas reglas se realizará aplicando un flip con el 50 por ciento antes que la hormiga comience a construir la ruta.

4.2. Resultados preliminares

En esta sección detallamos los resultados que se obtuvieron en las pruebas preliminares realizadas para definir la propuesta final.

4.2.1. Diseño experimental

Las técnicas fueron programadas en lenguaje C y las pruebas preliminares fueron desarrolladas en PC's con sistema operativo Windows XP, procesador Intel de 2.00 GHz y 512 MB de memoria RAM.

Conforme se determinaron los mejores parámetros para cada una de las técnicas antes presentadas, los experimentos preliminares consistieron en la prueba de cinco corridas sobre un conjunto limitado y representativo de instancias de JSSP que son referidas en la mayoría de la literatura y que sirven como estándar para verificar el desempeño de los algoritmos.

Estos problemas consistieron de una selección sistemática, tomando un problema cada dos en el intervalo de problemas de la clase LA entre el primer y vigésimo problemas, ya que, como se explicará más detalladamente en el capítulo cinco, es una de las clases de archivos de prueba con mayor diversidad en complejidad y tamaños. El conjunto de problemas definidos para estas pruebas preliminares fueron: Problemas de prueba = { $LA01$, $LA03$, $LA05$, $LA07$, $LA09$, $LA11$, $LA13$, $LA15$, $LA17$, $LA19$ }.

Los parámetros para cada una de las técnicas se detallan a continuación:

El número de ciclos se ajustó a 3000 que es el número de iteraciones que se encuentran reportadas en la literatura, al igual que el número de hormigas utilizadas que fue de $MAXH = \sum_{i=0}^{J-1} j$ [12][48].

1. *Ant System para JSSP básico*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

2. *Ant System para JSSP con arcos individuales*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

3. *Ant System para JSSP con exploración aleatoria del 10 % de ciclos*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

4. *Ant System para JSSP usando distribución inversa*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia, nodos de cambio = N_j donde N_j es el número de trabajos de la instancia y un porcentaje de cambio del 30 %.

5. *Ant System para JSSP con cambio de parámetros (α,β)*: Número de ciclos 3000, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

6. *Ant System para JSSP con cambio de parámetro de persistencia ρ*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

7. *Ant System para JSSP con actualización de un porcentaje de hormigas*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$, porcentaje de actualización = 30 % y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

8. *Ant System para JSSP con exploración usando reglas de prioridad aleatorias*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

9. *Ant System para JSSP calculando desperdicio de tiempo*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

10. *Ant System para JSSP con exclusión de la operación de trabajo similar a la operación actual en que se localiza la hormiga*: Número de ciclos 3000, $\alpha = 0{,}85$, $\beta = 0{,}15$, $\rho = 0{,}7$ y $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia.

11. *Ant System para JSSP una nueva inspiración*: Número de ciclos 1000, persistencia de la feromona $\rho = 0{,}7$ y

$$MAXH = \frac{\sum_{i=0}^{J-1} j_i}{2}$$

donde J es el total de trabajos de la instancia.

4.2.2. Resultados

Los resultados se detallan en la tabla 4.4 y sólo muestran el mejor de cada una de las cinco corridas realizadas en cada experimento.

Como se puede observar en las técnicas de la 1 a la 10, aunque algunas técnicas como el uso de la distribución inversa (técnica 4), el cambio de parámetro de persistencia (técnica 6), cálculo del desperdicio de tiempo (técnica 9) y la exclusión de operaciones de un mismo trabajo (técnica 10) mostraron mejoras respecto de los otros métodos, los resultados aún quedan muy lejos del mejor conocido. Para ellos se requerirá de un mecanismo de reparación determinístico que permita mejorar los resultados obtenidos, como en la mayoría de las técnicas heurísticas.

Sin embargo, podemos observar que en las pruebas practicadas con la técnica 11, en la que se retoma el *"conteo de pasos"*, los resultados se acercan por mucho al mejor conocido. De hecho, para algunas instancias, el mejor conocido es alcanzado tal como se puede observar para los problemas LA01, LA05, LA09, LA11 y LA17.

De estos resultados preliminares se toma como método final la técnica 11, en la que se realiza el *"conteo de pasos"* y que se detalla en el pseudocódigo en la siguiente sección.

4.2.3. Pseudocódigo de la propuesta final AS_{cp}

En las pruebas preliminares, el método de *"conteo de pasos"* muestra que puede mejorar el desempeño del AS Clásico y es por ello que fue seleccionado como punto de comparación con otros algoritmos del estado del arte y que se detallan en el siguiente capítulo.

Como fue mencionado anteriormente, esta variante de AS incrementa tres pasos antes del cálculo probabilístico que la hormiga realiza para poder decidir qué operación visitar. En la figura 4.27 se muestra el pseudocódigo completo de la variante de AS y que denominamos: AS_{cp} que se define como *Ant System con Conteo de Pasos*.

En el pseudocódigo de la figura las partes que fueron modificadas para el cálculo son las que están marcadas en el paso tres como 3.a, 3.b y 3.c, en las cuales se añadieron los pasos descritos en la sección 4.1.12.

1 **Fase de inicialización**
  Inicializar contador de ciclos NC
  Para cada arco (i, j):
    valor inicial de $\tau_{ij} = c$; donde c es una constante pequeña positiva
    $\Delta\tau_{ij} = 0$
  Colocar **MAXH** hormigas en **N** operaciones
2 **Colocar la primera operación en lista** $tabú_k$ **de cada hormiga y asignar** α, β **y regla de decibilidad**
3 **Repetir hasta llenar** $tabú_k$
  Para cada hormiga:
3.a Determinar conjunto de operaciones posibles a ser visitadas S
3.b Calcular tiempo de inicio de cada operación y determinar factibilidad η
3.c Elegir próxima operación a ser visitada según ecuación 4.1
    Mover la hormiga a la próxima operación
    Insertar la operación en $tabú_k$
4 **Repetir para cada hormiga** k
  Calcular el makespan L_k del programa generado
  Guardar el programa de makespan más chico hasta el ciclo $NC : L_0^{NC} = min\left\{L_0^{NC-1}\ min_k\{L_k\}\right\}$
  Para cada arco (i, j)
  Calcular $\Delta\tau_{ij}$ según ecuación 4.3
5 **Para cada arco** (i, j)
  Actualizar τ_{ij} según ecuación 4.2
  $\Delta\tau_{ij} = 0$
6 **Aumentar el contador de ciclos** NC
  Si $NC < Nc_{max}$ **entonces**
    Vaciar $tabú_k$
    Ir a la fase 2
  Sino
    Imprimir programa más corto L_0^{NC}
  Fin

Figura 4.27: Pseudocódigo de Ant System con conteo de pasos (AS_{cp}).

Problema	Mejor conocido	AS para JSSP										
		1	2	3	4	5	6	7	8	9	10	11
LA01	666	1772	1772	2261	958	1009	852	1025	1762	1045	978	666
LA03	597	1255	1255	1722	807	859	832	840	1245	948	938	601
LA05	593	1606	1524	1826	723	813	745	813	1606	1112	879	593
LA07	890	2374	2370	2554	1231	1273	1220	1380	2374	1347	1149	899
LA09	951	2453	2510	2744	1403	1347	1286	1511	1561	1476	1283	951
LA11	1222	3325	3559	2748	1836	1892	1836	2077	3538	1674	1519	1222
LA13	1150	2428	2697	2694	1999	1778	1999	2103	2428	1631	1472	1174
LA15	1207	2894	2983	3015	1843	1860	1856	2941	2894	1544	1379	1214
LA17	784	2761	2813	2911	1660	1660	1651	2780	2761	1006	941	784
LA19	842	2648	2155	3514	1435	1567	1435	1941	2648	1128	1014	846

Tabla 4.4: Resultados preliminares de cada una de las técnicas.

Capítulo 5

Pruebas y análisis de resultados

En este capítulo se presentan los resultados obtenidos en las pruebas realizadas al AS_{cp} y la comparación de estos resultados con otros algoritmos del estado del arte. Se comienza indicando las medidas que se utilizarán para medir el desempeño del algoritmo; posteriormente se indican los problemas de prueba seleccionados y las características de cada uno de ellos. Después se detallan los resultados de cada una de las ejecuciones del algoritmo y finalmente se comparan los resultados obtenidos con los publicados por otros algoritmos del estado del arte y se realiza un análisis de los mismos.

5.1. Medidas de desempeño

Para medir el desempeño del algoritmo se realizaron comparaciones del número de evaluaciones de la función objetivo, que en el caso del algoritmo AS_{cp} es el cálculo del makespan de acuerdo a la permutación de operaciones generada por la hormiga y que se almacena en la lista tabú. Para medir la calidad de las soluciones, se realizó una comparación directa del makespan obtenido por el algoritmo propuesto en cada una de las instancias y se comparó con el mejor conocido.

En el caso de las pruebas realizadas a AS_{cp}, se presentan los datos estadísticos obtenidos de todas las ejecuciones realizadas. En las comparaciones que se realizaron del algoritmo con otras técnicas del estado del arte se utilizaron los resultados que los autores muestran en sus publicaciones.

5.2. Selección de problemas

Las funciones utilizadas para la prueba del algoritmo constan de 40 problemas pertenecientes a la clase LA. Esta clase maneja 8 tamaños diferentes de problemas propuestos por Lawrence [33]: 10×5, 15×5, 20×5, 10×10, 15×10, 20×10, 30×10 y 15×15; a cada uno de estos tamaños Lawrence los llamó $F1-5$, $G1-5$, $H1-5$, $A1-5$, $B1-5$, $C1-5$, $D1-5$ y $I1-5$, respectivamente. Sin embargo, el nombre de LA fue dado por Applegate y Cook [4] y es uno de los más utilizados. Los tiempos de procesamiento fueron generados con un intervalo, es decir, que cada tiempo de procesamiento incluido en los problemas tiene asignado un valor que puede ir desde 5 a 99.

Los detalles de las instancias fueron tomados de [6] y en la tabla 5.1 se resume cada una de las principales características de los problemas: el nombre del problema (Problema), tamaño del problema (j, m), límite inferior (LB - Lower Bound) y límite superior (UB - Upper Bound), la desviación que existe entre el límite inferior y el valor mínimo del makespan calculado teóricamente, la fecha en que se publicó y quién lo reporta.

Los problemas de la clase LA son de los más famosos en la literatura, ya que son muy variados en cuanto a tamaño y complejidad; por esta razón se tomaron como funciones de prueba en este trabajo.

5.3. Resultados

En esta sección se presentan los resultados obtenidos por el algoritmo AS_{cp} para cada uno de los problemas descritos en la sección anterior. Las pruebas que se realizaron con cada uno de los problemas consta de 20 ejecuciones del algoritmo; al final se resume cada uno de los resultados obtenidos, para este caso el makespan, la evaluación en que se encontró y las estadísticas de todas las ejecuciones. Estas medidas estadísticas son el mejor, el peor, el promedio, la mediana y la desviación estándar.

5.3.1. Detalle de parámetros

Los parámetros que se utilizaron para las pruebas son los siguientes: El número de hormigas $MAXH = \sum_{i=0}^{J-1} j_i$ donde J es el total de trabajos de la instancia, el número de ciclos 1000 y un valor de persistencia $\rho = 0{,}7$.

5.3.2. Entorno computacional

Las técnicas fueron programadas en lenguaje C y las pruebas se desarrollaron en PCs con sistema operativo Windows XP, procesador Intel de 2.00 GHz y 512 MB de memoria RAM. Se considera que el tamaño del programa al ejecutarse no rebasa la memoria disponible en las PCs y que por lo tanto no se requirió de mayores consideraciones.

5.3.3. Detalle de resultados

En las tablas 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 y 5.9 se muestran los resultados obtenidos por cada una de las 20 ejecuciones independientes del algoritmo. Cada tabla agrupa 5 problemas de la clase LA y se muestra en cada una de ellas: el número de prueba realizada (indicada con la columna *Prueba*), el nombre del problema (etiquetado como *Prob*), el número de hormigas utilizadas por cada prueba (etiquetado como *Ants*), y las columnas *Ciclo* indica el número de ciclo en que se encontró el mejor makespan, *#Eval* indica el número de evaluaciones realizadas a la función objetivo hasta el ciclo en que se localizó, es decir, $Ciclo \times Ants$ y la columna C_{max} indica el mejor makespan localizado.

Como podemos observar en la tabla 5.2, en los problemas de tamaño 10×5 nuestro AS_{cp} no tiene dificultades para encontrar los mejores resultados conocidos, a excepción del problema la04 en el que se queda a sólo 5 unidades. Este desempeño permanece para la mayoría de los problemas aún cuando el tamaño crezca como en los problemas de tamaños 15×5 y 20×5 para los cuales alcanza el mejor resultado conocido en 3 de 5 problemas de tales tamaños.

Si bien el AS_{cp} tuvo algunas dificultades para alcanzar el mejor makespan conocido en los problemas 10×10, 15×10, 20×10 y 30×10 se puede observar que el número de evaluaciones de la función objetivo no incrementa en gran medida y aún así los resultados mantienen cierta calidad pues la diferencia respecto del mejor conocido varía en muy pocas unidades.

Para los problemas 15×15 tuvo un buen desempeño como se observó en los problemas de 10×5, 15×5 y 20×5, sin aumentar el número de evaluaciones a la función objetivo.

5.4. Algoritmos de comparación

A continuación se describen de manera breve los algoritmos seleccionados contra los cuales se realiza una comparación en cuanto al número de evaluaciones y la mejor solución obtenida. Estos algoritmos fueron seleccionados ya que se dispone de la información necesaria

reportada en la literatura como para tener la información más precisa de las evaluaciones realizadas, ya que es un dato que muy pocos artículos reportan y además son algoritmos que han obtenido un buen desempeño contra otros algoritmos del estado del arte.

Los algoritmos contra los cuales fue comparado el AS_{cp} son los siguientes:

- **TS** *Busqueda Tabú* [39]: Es una metaheurística designada para encontrar una solución vecina óptima en problemas de optimización combinatoria. Esta técnica ha reportado hasta el momento los mejores resultados para el JSSP y por tanto, se tomó en cuenta como punto de referencia.
- **AIS** *Sistema Inmune Artificial* [13]: Un algoritmo que está basado en el funcionamiento del sistema inmunológico y que en una comparativa con algunos otros algoritmos como un Algoritmo Genético Simple, el Algoritmo Genético Híbrido y GRASP, obtuvo buenos resultados que nos permite seleccionarlo como algoritmo de comparación.
- **CULT** *Algoritmo Cultural* [32]: Basado en teorías sociales y arqueológicas las cuales tratan de modelar la evolución cultural, este algoritmo fue seleccionado ya que frente algoritmos genéticos y GRASP pudo obtener buenos resultados logrando disminuir el número de evaluaciones realizadas en comparación, al menos contra un algoritmo GRASP.

En todos los casos, los resultados obtenidos para la comparación fueron tomados de sus respectivas publicaciones [39][13][32].

5.5. Comparación de resultados

Los resultados de la comparación se muestran en la tabla 5.10 en las cuales se presentan los siguientes datos: Problema, tamaño, mejor solución conocida (MSC) y los algoritmos AS_{cp} que se refiere a nuestro Ant System con Conteo de pasos, SIA un Sistema Inmune Artificial, $CULT$ un Algoritmo Cultural, $INSA$ heurística utilizada por Tabú Search [39] para la construcción de la solución inicial y TS Tabú Search. En todos los algoritmos se detalla el C_{max} (makespan reportado) de cada problema y #Eval que es el número de evaluaciones realizadas tomadas de cada una de las referencias. Este último dato no se presenta para el caso del algoritmo INSA puesto que no se encuentra reportado en el artículo.

Para facilitar la comprensión de los resultados en la tabla 5.10 se remarcan los mejores resultados obtenidos por cada algoritmo en cuanto el makespan C_{max} y el número de evaluaciones a la función objetivo (#Eval). En el caso del algoritmo de Búsqueda Tabú TS no

se comparó el número de evaluaciones realizadas, ya que este algoritmo funciona como un mecanismo de reparación sobre el trabajo realizado por el algoritmo $INSA$, y es por ello que en algunos campos del #Eval en TS el valor reportado es de cero evaluaciones, debido a que, el algoritmo $INSA$ por sí solo alcanzó la mejor solución conocida.

5.6. Análisis de resultados

En la tabla 5.10 se muestra en resumen los resultados obtenidos por nuestro algoritmo y los de comparación. Como se aclaró en el capítulo anterior, nuestro algoritmo AS_{cp} no cuenta con un mecanismo de resparación de las soluciones; por lo tanto, estaremos comparando los resultados obtenidos por el motor de búsqueda. La discusión se centrará en dos puntos principales: la calidad de las soluciones y el número de evaluaciones realizadas de la función objetivo.

5.6.1. Calidad de las soluciones

Podemos observar que con respecto a los algoritmos SIA, $CULT$ y TS nuestro AS_{cp} no alcanzó en todas las pruebas los mejores makespan conocidos, donde los resultados obtenidos por AS_{cp} para problemas de tamaño práctico (10×5, 15×5 y 20×5) no obtuvo muchas dificultades en resolverlos, en los cuales para problemas de tamaño 10×10, 15×10, 20×10 y 30×10 frente a los algoritmos de comparación no logró alcanzar los mejores resultados conocidos para todas las instancias, aunque las diferencias son relativamente mínimas.

Por otra parte, podemos ver que para los problemas de tamaño 15×15 nuestro AS_{cp} obtuvo ciertas ventajas sobre los algoritmos SIA y $CULT$, puesto que logró resolver instancias que no pudieron resolver ninguno de ellos, y que para tal caso sólo el algoritmo de TS pudo resolver también 3 de las 5 instancias propuestas de este tamaño.

Si comparamos nuestro AS_{cp} con el algoritmo $INSA$ que es utilizado por TS para generar una solución inicial y el cual no cuenta con mecanismos de reparación, como una forma más equiparable en el motor de búsqueda, podemos mencionar que AS_{cp} tiene muchas ventajas al encontrar mejores soluciones que $INSA$ el cual sólo logra resolver 4 de las 40 instancias y AS_{cp} resolvió 22 de los 40 problemas propuestos.

5.6.2. Número de evaluaciones a la función objetivo

Con respecto a los algortimos SIA, $CULT$ y TS, podemos observar que la calidad de las soluciones puede ser mejor. Sin embargo, el costo computacional que les implica

es muy alto ya que aunque nuestro algoritmo en algunos problemas no alcanzó la mejor solución conocida el resultado obtenido de la ejecución es bueno si tomamos en cuenta que el costo computacional, supera en mucho a los otros algoritmos en los cuales el número de evaluaciones crece conforme a los tamaños de los problemas.

Para el caso de nuestro algoritmo el desempeño resulta ser bueno en cuanto al ahorro de evaluaciones de la función objetivo y como puede observarse en los problemas de tamaño 15×15 el AS_{cp} obtuvo mejores resultados respecto de los demás tanto en la solución del problema como el número de evaluaciones, que para el caso de SIA y $CULT$ resultó difícil de alcanzar, incluso para la búsqueda tabú aún después de partir de una buena solución encontrada por $INSA$ ya que el número de evaluaciones creció sin tener muy buenos resultados.

Respecto al algoritmo $INSA$ (motor de búsqueda utilizado por TS), no pudimos realizar comparaciones del número de evaluaciones realizadas debido a que este dato no se reporta en el artículo [39].

Si vemos sólo el promedio de evaluaciones de nuestro algoritmo AS_{cp} podemos verificar que aunque el tamaño del problema crece, el número de evaluaciones a la función objetivo (que representan el costo de obtener la solución) no se incrementa en forma exponencial como lo realizan los otros algoritmos de comparación, puesto que existe en promedio de alrededor de 3,564 evaluaciones, tal como puede observarse en la tabla 5.11. Además, podemos verificar la diferencia que existe en el número de evaluaciones promedio realizadas entre nuestro algoritmo y los demás de comparación, donde el algoritmo cultural, el sistema inmune artificial y la búsqueda tabú quedan muy por arriba.

Problema	j	m	LB	UB	Desv	Fecha	Referencia
la01	10	5	666	666	0.00	1988	[3]
la02	10	5	655	655	0.00	1988	[3]
la03	10	5	597	597	0.00	1991	[4]
la04	10	5	590	590	0.00	1991	[4]
la05	10	5	593	593	0.00	1988	[3]
la06	15	5	926	926	0.00	1988	[3]
la07	15	5	890	890	0.00	1988	[3]
la08	15	5	863	863	0.00	1988	[3]
la09	15	5	951	951	0.00	1988	[3]
la10	15	5	958	958	0.00	1988	[3]
la11	20	5	1222	1222	0.00	1988	[3]
la12	20	5	1039	1039	0.00	1988	[3]
la13	20	5	1150	1150	0.00	1988	[3]
la14	20	5	1292	1292	0.00	1988	[3]
la15	20	5	1207	1207	0.00	1988	[3]
la16	10	10	945	945	0.00	1990	[10]
la17	10	10	784	784	0.00	1990	[10]
la18	10	10	848	848	0.00	1988	[4]
la19	10	10	842	842	0.00	1988	[4]
la20	10	10	902	902	0.00	1988	[4]
la21	15	10	1046	1046	0.00	1996	[2]
la22	15	10	927	927	0.00	1988	[4]
la23	15	10	1032	1032	0.00	1988	[3]
la24	15	10	935	935	0.00	1991	[4]
la25	15	10	977	977	0.00	1991	[4]
la26	20	10	1218	1218	0.00	1988	[3]
la27	20	10	1235	1235	0.00	1988	[3]
la28	20	10	1216	1216	0.00	1988	[3]
la29	20	10	1152	1152	0.00	1996	[35]
la30	20	10	1355	1355	0.00	1988	[3]
la31	30	10	1784	1784	0.00	1988	[3]
la32	30	10	1850	1850	0.00	1988	[3]
la33	30	10	1719	1719	0.00	1988	[3]
la34	30	10	1721	1721	0.00	1988	[3]
la35	30	10	1888	1888	0.00	1988	[3]
la36	15	15	1268	1268	0.00	1990	[10]
la37	15	15	1397	1397	0.00	1990	[10]
la38	15	15	1196	1196	0.00	1996	[39]
la39	15	15	1233	1233	0.00	1991	[4]
la40	15	15	1222	1222	0.00	1991	[4]

Tabla 5.1: Detalle de problemas de la clase LA.

Prueba	Prob: Ants: Ciclo:	la01 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la02 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la03 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la04 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la05 5 #Eval	C_{max}
1	94	62604	666	204	133620	655	445	265665	597	154	91630	595	47	27871	593
2	122	81252	666	169	110695	655	129	77013	597	162	96390	597	26	15418	593
3	146	97236	666	308	201740	655	388	231636	597	189	112455	595	155	91915	593
4	122	81252	666	122	79910	655	126	75222	597	135	80325	595	41	24313	593
5	138	91908	666	344	225320	655	223	133131	597	146	86870	595	49	29057	593
6	127	84582	666	146	95630	655	348	207756	597	126	74970	595	68	40324	593
7	168	111888	666	49	32095	655	105	62685	597	186	110670	595	27	16011	593
8	95	63270	666	742	486010	655	409	244173	597	204	121380	595	112	66416	593
9	125	83250	666	234	153270	655	164	97908	597	68	40460	595	47	27871	593
10	24	15984	666	197	129035	655	138	82386	597	233	138635	595	103	61079	593
11	12	7992	666	184	120520	655	148	88356	597	187	111265	597	25	14825	593
12	65	43290	666	116	75980	655	364	217308	597	194	115430	595	95	56335	593
13	128	85248	666	233	152615	655	155	92535	597	135	80325	595	48	28464	593
14	204	135864	666	198	129690	655	164	97908	597	204	121380	595	47	27871	593
15	105	69930	666	249	163095	655	192	114624	597	104	61880	595	36	21348	593
16	241	160506	666	694	454570	655	249	148653	597	605	359975	595	76	45068	593
17	113	75258	666	49	32095	655	849	506853	597	129	76755	595	148	87764	593
18	91	60606	666	272	178160	655	97	57909	597	148	88060	595	28	16604	593
19	134	89244	666	134	87770	655	769	459093	597	143	85085	595	194	115042	593
20	29	19314	666	497	325535	655	486	290142	597	187	111265	595	174	103182	593
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		60	666		245	655		485	597		340	595		125	593
Peor		1205	666		3710	655		4245	597		3025	597		970	593
Promedio		570	666		1285	655		1487	597		909	595.2		386	593
Mediana		610	666		1005	655		1037.5	597		790	595		242.5	593
Desviación		279.24	0		939.21	0		1069.42	0		534.79	0.62		266.65	0

Tabla 5.2: Resultados de los problemas de clase LA de 01 a 05.

Prueba	Prob: Ants: Ciclo:	la06 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la07 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la08 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la09 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la10 7 #Eval	C_{max}
1	462	3234	928	168	1176	890	68	476	870	98	686	951	501	3507	970
2	384	2688	928	294	2058	890	47	329	870	182	1274	951	68	476	970
3	199	1393	928	168	1176	890	156	1092	870	144	1008	951	401	2807	970
4	694	4858	928	729	5103	890	684	4788	870	87	609	951	165	1155	970
5	158	1106	928	496	3472	890	146	1022	863	9	63	951	155	1085	970
6	475	3325	938	944	6608	890	198	1386	870	163	1141	951	167	1169	970
7	869	6083	928	168	1176	890	689	4823	870	142	994	956	193	1351	970
8	694	4858	928	83	581	890	86	602	863	172	1204	978	124	868	970
9	129	903	940	136	952	890	94	658	870	184	1288	951	188	1316	965
10	73	511	928	74	518	890	168	1176	870	142	994	951	214	1498	970
11	948	6636	938	194	1358	890	276	1932	870	169	1183	966	147	1029	965
12	429	3003	928	168	1176	890	197	1379	870	138	966	951	26	182	970
13	648	4536	928	185	1295	890	168	1176	870	195	1365	951	324	2268	970
14	236	1652	928	72	504	890	496	3472	870	142	994	951	168	1176	970
15	647	4529	928	195	1365	890	135	945	863	136	952	951	459	3213	970
16	692	4844	943	203	1421	890	52	364	870	155	1085	966	166	1162	965
17	168	1176	928	86	602	890	486	3402	870	168	1176	951	29	203	970
18	742	5194	928	168	1176	890	48	336	872	155	1085	951	48	336	970
19	963	6741	928	195	1365	890	79	553	870	164	1148	951	729	5103	970
20	486	3402	928	49	343	890	84	588	870	121	847	951	85	595	970
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		511	928		343	890		329	863		63	951		182	965
Peor		6741	943		6608	890		4823	872		1365	978		5103	970
Promedio		3533	930.35		1671	890		1524	869.05		1003	954.1		1524	969.25
Mediana		3363.5	928		1176	890		1057	870		1046.5	951		1165.5	970
Desviación		1962.4	4.91		1598.7	0		1428.7	2.65		291.1	7.3		1260.1	1.83

Tabla 5.3: Resultados de los problemas de clase LA de 06 a 10.

Prueba	Prob: Ants: Ciclo:	la11 10 #Eval	C_{max}	Prob: Ants: Ciclo:	la12 10 #Eval	C_{max}	Prob: Ants: Ciclo:	la13 10 #Eval	C_{max}	Prob: Ants: Ciclo:	la14 10 #Eval	C_{max}	Prob: Ants: Ciclo:	la15 10 #Eval	C_{max}
1	125	1250	1222	168	1680	1040	214	2140	1150	204	2040	1292	263	2630	1212
2	148	1480	1230	144	1440	1040	205	2050	1150	304	3040	1301	168	1680	1212
3	17	170	1222	94	940	1040	306	3060	1150	196	1960	1292	428	4280	1210
4	308	3080	1222	208	2080	1040	142	1420	1150	203	2030	1301	12	120	1210
5	59	590	1222	429	4290	1040	198	1980	1150	108	1080	1292	823	8230	1210
6	146	1460	1222	167	1670	1040	306	3060	1150	94	940	1292	47	470	1210
7	126	1260	1235	302	3020	1040	140	1400	1150	264	2640	1292	429	4290	1210
8	96	960	1222	158	1580	1040	38	380	1150	506	5060	1301	423	4230	1210
9	148	1480	1222	176	1760	1045	106	1060	1150	136	1360	1292	159	1590	1210
10	231	2310	1235	184	1840	1040	102	1020	1150	486	4860	1292	629	6290	1212
11	156	1560	1222	196	1960	1040	63	630	1150	102	1020	1292	125	1250	1210
12	139	1390	1222	147	1470	1040	76	760	1150	172	1720	1292	622	6220	1212
13	8	80	1222	135	1350	1045	422	4220	1150	38	380	1292	138	1380	1210
14	92	920	1222	152	1520	1040	84	840	1150	126	1260	1292	472	4720	1212
15	112	1120	1222	187	1870	1040	35	350	1150	37	370	1292	158	1580	1210
16	154	1540	1235	136	1360	1045	49	490	1150	105	1050	1292	8	80	1212
17	12	120	1222	247	2470	1040	763	7630	1150	120	1200	1292	704	7040	1210
18	369	3690	1230	184	1840	1040	125	1250	1150	41	410	1292	122	1220	1212
19	482	4820	1222	176	1760	1040	146	1460	1150	1	10	1292	745	7450	1212
20	156	1560	1222	864	8640	1040	47	470	1150	752	7520	1292	125	1250	1210
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		80	1222		940	1040		350	1150		10	1292		80	1210
Peor		4820	1235		8640	1045		7630	1150		7520	1301		8230	1212
Promedio		1542	1224		2227	1040		1783	1150		1997	1293		3300	1210.8
Mediana		1425	1222		1760	1040		1325	1150		1310	1292		2155	1210
Desviación		1178.1	5.05		1668.9	1.83		1718	0		1874.9	3.3		2632.5	1.01

Tabla 5.4: Resultados de los problemas de clase LA de 11 a 15.

Prueba	Prob: Ants: Ciclo:	la16 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la17 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la18 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la19 5 #Eval	C_{max}	Prob: Ants: Ciclo:	la20 5 #Eval	C_{max}
1	245	1225	970	482	2410	784	642	3210	872	135	675	856	494	2470	908
2	315	1575	971	368	1840	784	398	1990	855	164	820	856	597	2985	908
3	395	1975	946	712	3560	784	126	630	855	125	625	856	321	1605	908
4	745	3725	970	426	2130	784	423	2115	855	139	695	842	168	840	908
5	215	1075	970	829	4145	784	814	4070	855	604	3020	850	706	3530	908
6	628	3140	970	476	2380	784	12	60	872	704	3520	850	74	370	908
7	153	765	970	125	625	784	824	4120	855	566	2830	842	168	840	908
8	142	710	970	196	980	784	613	3065	855	315	1575	842	924	4620	908
9	22	110	970	436	2180	784	124	620	855	482	2410	842	824	4120	908
10	856	4280	970	635	3175	784	131	655	855	236	1180	842	163	815	908
11	452	2260	970	286	1430	784	3	15	872	406	2030	842	128	640	908
12	426	2130	946	741	3705	784	869	4345	855	87	435	842	712	3560	908
13	395	1975	970	125	625	784	147	735	855	164	820	842	428	2140	908
14	682	3410	946	485	2425	784	852	4260	855	169	845	842	639	3195	908
15	142	710	970	269	1345	784	423	2115	855	412	2060	850	726	3630	908
16	762	3810	970	129	645	784	901	4505	855	136	680	850	146	730	908
17	135	675	970	36	180	784	11	55	855	482	2410	842	129	645	908
18	84	420	946	847	4235	784	159	795	855	172	860	842	716	3580	908
19	648	3240	970	469	2345	784	485	2425	855	648	3240	856	138	690	908
20	256	1280	970	847	4235	784	123	615	855	482	2410	842	633	3165	908
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		110	946		180	784		15	855		435	842		370	908
Peor		4280	971		4235	784		4505	872		3520	856		4620	908
Promedio		1924	965.25		2229	784		2020	857.55		1657	846.4		2208	908
Mediana		1775	970		2262.5	784		2052.5	855		1377.5	842		2305	908
Desviación		1278.9	9.88		1291	0		1630.4	6.23		1014.2	5.86		1420.4	0

Tabla 5.5: Resultados de los problemas de clase LA de 16 a 20.

Prueba	Prob: Ants: Ciclo:	la21 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la22 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la23 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la24 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la25 7 #Eval	C_{max}
1	743	5201	1060	426	2982	927	982	6874	1047	436	3052	941	9	63	977
2	726	5082	1060	128	896	927	136	952	1047	652	4564	941	853	5971	977
3	795	5565	1060	428	2996	927	145	1015	1047	986	6902	943	421	2947	977
4	762	5334	1060	5	35	927	823	5761	1047	426	2982	941	695	4865	977
5	482	3374	1060	128	896	927	142	994	1060	385	2695	941	36	252	977
6	693	4851	1060	486	3402	927	723	5061	1047	421	2947	941	749	5243	977
7	153	1071	1060	925	6475	927	421	2947	1060	12	84	941	862	6034	977
8	840	5880	1060	368	2576	927	125	875	1047	862	6034	941	763	5341	977
9	125	875	1060	712	4984	930	638	4466	1047	135	945	941	128	896	980
10	932	6524	1055	368	2576	927	682	4774	1060	752	5264	941	746	5222	977
11	422	2954	1060	856	5992	927	176	1232	1047	726	5082	941	163	1141	977
12	452	3164	1060	965	6755	927	33	231	1047	842	5894	943	5	35	977
13	698	4886	1060	731	5117	927	896	6272	1047	963	6741	941	872	6104	980
14	126	882	1055	405	2835	930	714	4998	1047	125	875	943	635	4445	977
15	358	2506	1060	136	952	927	125	875	1047	169	1183	941	911	6377	977
16	746	5222	1055	426	2982	927	138	966	1047	34	238	941	412	2884	977
17	852	5964	1055	135	945	930	871	6097	1047	412	2884	943	125	875	977
18	126	882	1055	384	2688	927	963	6741	1047	196	1372	941	136	952	977
19	935	6545	1060	128	896	927	428	2996	1047	485	3395	941	128	896	977
20	428	2996	1060	694	4858	927	964	6748	1047	423	2961	941	187	1309	977
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		875	1055		35	927		231	1047		84	941		35	977
Peor		6545	1060		6755	930		6874	1060		6902	943		6377	980
Promedio		3987	1058.75		3091	927.45		3543	1048.95		3304	941.4		3092	977.3
Mediana		4868.5	1060		2908.5	927		3731	1047		2971.5	941		2915.5	977
Desviación		1964	2.22		2025.7	1.1		2457.2	4.76		2151.4	0.82		2392.7	0.92

Tabla 5.6: Resultados de los problemas de clase LA de 21 a 25.

Prueba	Prob:	la26		Prob:	la27		Prob:	la28		Prob:	la29		Prob:	la30	
	Ants:	10		Ants:	10		Ants:	10		Ants:	10		Ants:	10	
	Ciclo:	#Eval	C_{max}	Ciclo:	#Eval	C_{max}	Ciclo:	#Eval	C_{max}	Ciclo:	#Eval	C_{max}	Ciclo:	#Eval	C_{max}
1	253	2530	1218	526	5260	1240	486	4860	1216	854	8540	1164	124	1240	1355
2	368	3680	1218	421	4210	1246	624	6240	1216	256	2560	1167	365	3650	1355
3	426	4260	1218	385	3850	1240	123	1230	1216	963	9630	1164	105	1050	1355
4	152	1520	1218	746	7460	1240	98	980	1216	74	740	1164	428	4280	1355
5	368	3680	1218	935	9350	1240	742	7420	1216	125	1250	1164	624	6240	1355
6	124	1240	1218	628	6280	1248	449	4490	1216	421	4210	1164	188	1880	1355
7	752	7520	1218	126	1260	1240	214	2140	1216	233	2330	1167	142	1420	1355
8	125	1250	1218	153	1530	1240	105	1050	1216	142	1420	1164	136	1360	1355
9	126	1260	1220	144	1440	1260	296	2960	1218	156	1560	1164	124	1240	1360
10	368	3680	1218	399	3990	1240	741	7410	1216	485	4850	1164	186	1860	1355
11	913	9130	1218	865	8650	1240	31	310	1216	126	1260	1167	172	1720	1355
12	102	1020	1218	662	6620	1240	415	4150	1216	955	9550	1164	122	1220	1355
13	3	30	1218	185	1850	1242	125	1250	1216	213	2130	1164	123	1230	1355
14	458	4580	1218	472	4720	1240	310	3100	1216	123	1230	1164	168	1680	1355
15	423	4230	1218	236	2360	1248	412	4120	1218	145	1450	1170	244	2440	1355
16	855	8550	1218	854	8540	1248	69	690	1216	358	3580	1164	168	1680	1360
17	126	1260	1218	2	20	1248	125	1250	1216	948	9480	1164	943	9430	1355
18	138	1380	1218	124	1240	1240	147	1470	1216	712	7120	1164	15	150	1355
19	944	9440	1218	842	8420	1240	125	1250	1216	103	1030	1164	147	1470	1355
20	128	1280	1218	964	9640	1240	963	9630	1216	14	140	1164	12	120	1355
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		30	1218		20	1240		310	1216		140	1164		120	1355
Peor		9440	1220		9640	1260		9630	1218		9630	1170		9430	1360
Promedio		3576	1218.1		4834	1243		3300	1216.2		3703	1164.75		2268	1355.5
Mediana		3105	1218		4465	1240		2550	1216		2230	1164		1575	1355
Desviación		2934.2	0.45		3123.9	5.21		2673.1	0.62		3297.7	1.65		2193.5	1.54

Tabla 5.7: Resultados de los problemas de clase LA de 26 a 30.

Prueba	Prob: Ants: Ciclo:	la31 15 #Eval	C_{max}	Prob: Ants: Ciclo:	la32 15 #Eval	C_{max}	Prob: Ants: Ciclo:	la33 15 #Eval	C_{max}	Prob: Ants: Ciclo:	la34 15 #Eval	C_{max}	Prob: Ants: Ciclo:	la35 15 #Eval	C_{max}
1	542	8130	1791	948	14220	1850	875	13125	1754	876	13140	1721	658	9870	1984
2	638	9570	1791	751	11265	1850	685	10275	1754	856	12840	1721	143	2145	1973
3	624	9360	1791	873	13095	1850	854	12810	1754	965	14475	1721	957	14355	1973
4	715	10725	1791	398	5970	1850	965	14475	1772	826	12390	1721	685	10275	1973
5	256	3840	1791	744	11160	1850	472	7080	1754	485	7275	1721	742	11130	1973
6	485	7275	1791	698	10470	1850	586	8790	1754	963	14445	1841	684	10260	1941
7	967	14505	1791	849	12735	1850	633	9495	1772	412	6180	1721	428	6420	1973
8	886	13290	1842	987	14805	1850	846	12690	1754	368	5520	1721	632	9480	1973
9	486	7290	1791	746	11190	1850	954	14310	1754	764	11460	1721	145	2175	1973
10	912	13680	1791	852	12780	1850	628	9420	1754	449	6735	1802	856	12840	1973
11	358	5370	1842	369	5535	1912	716	10740	1772	922	13830	1721	964	14460	1941
12	476	7140	1791	688	10320	1850	854	12810	1754	876	13140	1721	125	1875	1973
13	558	8370	1791	477	7155	1850	871	13065	1754	894	13410	1721	744	11160	1973
14	621	9315	1791	586	8790	1850	126	1890	1754	874	13110	1721	697	10455	1973
15	648	9720	1842	956	14340	1850	964	14460	1754	863	12945	1721	857	12855	1973
16	942	14130	1844	476	7140	1850	826	12390	1772	814	12210	1721	872	13080	1941
17	713	10695	1791	158	2370	1850	795	11925	1754	689	10335	1721	844	12660	1973
18	248	3720	1791	476	7140	1850	948	14220	1754	599	8985	1721	837	12555	1973
19	796	11940	1842	958	14370	1850	936	14040	1754	358	5370	1721	942	14130	1973
20	841	12615	1791	746	11190	1850	910	13650	1754	486	7290	1721	574	8610	1973
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		3720	1791		2370	1850		1890	1754		5370	1721		1875	1941
Peor		14505	1844		14805	1912		14475	1772		14475	1841		14460	1984
Promedio		9534	1803		10302	1853.1		11583	1757.6		10754	1731		10039	1968.75
Mediana		9465	1791		11175	1850		12750	1754		12300	1721		10792.5	1973
Desviación		3228.6	22.84		3465.7	13.86		3115.4	7.39		3213.7	31.57		3987.8	12.21

Tabla 5.8: Resultados de los problemas de clase LA de 31 a 35.

Prueba	Prob: Ants: Ciclo:	la36 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la37 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la38 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la39 7 #Eval	C_{max}	Prob: Ants: Ciclo:	la40 7 #Eval	C_{max}
1	758	5306	1270	475	3325	1406	142	994	1198	856	5992	1233	856	5992	1222
2	968	6776	1270	855	5985	1406	597	4179	1196	748	5236	1233	742	5194	1222
3	847	5929	1355	596	4172	1406	694	4858	1196	569	3983	1233	965	6755	1222
4	826	5782	1270	472	3304	1406	859	6013	1196	985	6895	1233	125	875	1222
5	698	4886	1270	369	2583	1406	658	4606	1196	475	3325	1233	684	4788	1222
6	742	5194	1270	586	4102	1406	485	3395	1196	455	3185	1236	326	2282	1222
7	368	2576	1270	741	5187	1406	964	6748	1196	426	2982	1233	749	5243	1222
8	142	994	1270	258	1806	1450	258	1806	1196	842	5894	1233	842	5894	1222
9	485	3395	1355	638	4466	1406	756	5292	1198	724	5068	1233	199	1393	1222
10	658	4606	1270	428	2996	1406	741	5187	1196	684	4788	1236	748	5236	1222
11	695	4865	1270	425	2975	1406	723	5061	1196	746	5222	1233	556	3892	1222
12	856	5992	1291	993	6951	1406	782	5474	1196	853	5971	1233	143	1001	1222
13	485	3395	1270	715	5005	1406	746	5222	1196	125	875	1233	941	6587	1222
14	158	1106	1270	742	5194	1450	149	1043	1196	986	6902	1233	825	5775	1222
15	475	3325	1291	125	875	1406	759	5313	1196	746	5222	1233	855	5985	1222
16	148	1036	1270	368	2576	1406	458	3206	1198	258	1806	1233	846	5922	1222
17	639	4473	1270	147	1029	1406	236	1652	1196	639	4473	1233	831	5817	1222
18	485	3395	1270	963	6741	1406	124	868	1196	428	2996	1233	876	6132	1222
19	486	3402	1270	125	875	1406	786	5502	1196	125	875	1233	712	4984	1222
20	472	3304	1270	365	2555	1406	774	5418	1196	872	6104	1233	851	5957	1222
		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}		#Eval	C_{max}
Mejor		994	1270		875	1406		868	1196		875	1233		875	1222
Peor		6776	1355		6951	1450		6748	1198		6902	1236		6755	1222
Promedio		3986	1280.6		3635	1410.4		4091	1196.3		4389	1233.3		4785	1222
Mediana		3937.5	1270		3314.5	1406		4959.5	1196		4928	1233		5509	1222
Desviación		1692.8	26.24		1833.8	13.54		1854.5	0.73		1830.2	0.92		1871.9	0

Tabla 5.9: Resultados de los problemas de clase LA de 36 a 40.

Problema	Tamaño	MSC	Algoritmos									
			AS_{cp}		SIA		$CULT$		$INSA$		TS	
			C_{max}	#Eval	C_{max}	#Eval	C_{max}	#Eval	C_{max}	#Eval	C_{max}	#Eval
La01	10 × 5	666	**666**	**570**	**666**	2345	**666**	4000	**666**	-	**666**	0
La02	10 × 5	655	**655**	**1285**	**655**	171095	**655**	20000	722	-	**655**	5353
La03	10 × 5	597	**597**	**1487**	**597**	545876	603	8000	681	-	**597**	7546
La04	10 × 5	590	595	**909**	**590**	4826	**590**	700000	659	-	**590**	21
La05	10 × 5	593	**593**	386	**593**	**84**	**593**	2000	**593**	-	**593**	0
La06	15 × 5	926	928	3533	**926**	**135**	**926**	2000	950	-	**926**	6
La07	15 × 5	890	**890**	1671	**890**	**1664**	**890**	4000	976	-	**890**	23
La08	15 × 5	863	**863**	1524	**863**	**1468**	**863**	4000	868	-	**863**	1
La09	15 × 5	951	**951**	1003	**951**	**548**	**951**	2000	**951**	-	**951**	0
La10	15 × 5	958	965	**1524**	**958**	2624	**958**	2000	**958**	-	**958**	0
La11	20 × 5	1222	**1222**	1542	**1222**	**157**	**1222**	2000	1293	-	**1222**	11
La12	20 × 5	1039	1040	2227	**1039**	10521	**1039**	**2000**	1044	-	**1039**	1
La13	20 × 5	1150	**1150**	1783	**1150**	**880**	**1150**	2000	1154	-	**1150**	1
La14	20 × 5	1292	**1292**	1997	**1292**	**27**	**1292**	2000	1328	-	**1292**	1
La15	20 × 5	1207	1210	**3300**	**1207**	7838	**1207**	8000	1323	-	**1207**	90
La16	10 × 10	945	946	**1924**	**945**	26451	946	25000	1077	-	**945**	19695
La17	10 × 10	784	**784**	**2229**	**784**	41266	**784**	25000	821	-	**784**	1031
La18	10 × 10	848	855	**2020**	**848**	18636	**848**	140000	926	-	**848**	18023
La19	10 × 10	842	**842**	**1657**	**842**	33531	**842**	32000	971	-	**842**	15039
La20	10 × 10	902	908	**2208**	907	29171	907	100000	1003	-	**902**	37959
La21	15 × 10	1046	1055	**3987**	**1046**	328405	1089	1700000	1179	-	1047	3732
La22	15 × 10	927	**927**	**3091**	**927**	458947	945	1700000	1032	-	**927**	2246
La23	15 × 10	1032	1047	**3543**	**1032**	42285	**1032**	200000	1132	-	**1032**	208
La24	15 × 10	935	941	**3304**	**935**	473731	964	1000000	1021	-	939	30001
La25	15 × 10	977	**977**	**3092**	979	407427	993	1000000	1147	-	**977**	27677
La26	20 × 10	1218	**1218**	**3576**	**1218**	478017	**1218**	1700000	1397	-	**1218**	2106
La27	20 × 10	1235	1240	**4834**	1240	476269	1269	200000	1466	-	1236	8971
La28	20 × 10	1216	**1216**	**3300**	**1216**	484701	1241	1800000	1485	-	**1216**	16630
La29	20 × 10	1157	1164	**3703**	1170	600328	1189	1800000	1385	-	1160	63469
La30	20 × 10	1355	**1355**	**2268**	**1355**	113932	**1355**	200000	1463	-	**1355**	282
La31	30 × 10	1784	1791	9534	**1784**	**6880**	**1784**	15000	1966	-	**1784**	84
La32	30 × 10	1850	**1850**	10302	**1850**	**8401**	**1850**	40000	1982	-	**1850**	135
La33	30 × 10	1719	1754	11583	**1719**	**6039**	**1719**	20000	1767	-	**1719**	9
La34	30 × 10	1721	**1721**	**10754**	**1721**	21603	**1721**	160000	1844	-	**1721**	302
La35	30 × 10	1888	1941	10039	**1888**	**7849**	**1888**	160000	1967	-	**1888**	48
La36	15 × 15	1268	1270	**3986**	1281	272736	1292	1000000	1445	-	**1268**	64473
La37	15 × 15	1397	1406	**3635**	1408	497661	1451	1400000	1726	-	1407	41183
La38	15 × 15	1196	**1196**	**4091**	1204	506151	1276	1800000	1307	-	**1196**	15579
La39	15 × 15	1233	**1233**	**4389**	1249	460554	1266	200000	1393	-	**1233**	31991
La40	15 × 15	1222	**1222**	**4785**	1228	451273	1265	1000000	1387	-	1229	30417

Tabla 5.10: Comparación de resultados con otros algoritmos del estado del arte.

Algoritmo	**Promedio de #Eval**	**Diferencia de #Eval con AS_{cp}**
AS_{cp}	**3,564**	0
SIA	175,058	171,494
$CULT$	454,525	450,961
TS	11,108	7,544

Tabla 5.11: Promedios de evaluaciones realizadas a la función objetivo por cada uno de los algoritmos comparados.

Capítulo 6

Conclusiones

En esta tesis se propuso una variante del Ant System para resolver problemas de programación de horarios. Este algoritmo pertenece a la clase de algoritmos Ant Colony Optimization(ACO), cuya inspiración es retomada del comportamiento de forrajeo de las hormigas. El algoritmo propuesto es el resultado de diversos cambios básicos realizados al Ant System original propuesto por Dorigo, en el cual se utiliza la feromona como principal mecanismo de comunicación entre las hormigas. En nuestro algoritmo, además de utilizar la feromona como mecanismo de comunicación indirecta (o lo que biológicamente es conocido como *stigmargy*), se propone utilizar un *conteo de pasos* basado en nuevas investigaciones biológicas que han concluido que las hormigas pueden contar el número de zancadas que hay entre el nido y la fuente de alimento (como si las hormigas tuvieran un *podómetro biológico* incluido). Además, se hicieron modificaciones a las reglas de selección de operaciones con base en los tiempos más próximos de cada operación.

En el capítulo 5, se presentaron los resultados de veinte ejecuciones independientes realizadas a nuestro algoritmo y en ellos se puede observar que, con pocas evaluaciones, la calidad de los resultados es buena aún cuando el tamaño del problema crezca.

Además, los resultados fueron comparados con otros algoritmos del estado del arte: un sistema inmune artificial (SIA), un algoritmo cultural (CULT) y el algoritmo utilizado por la búsqueda tabú que genera la solución inicial (INSA) y la misma búsqueda Tabú. La comparación de resultados se realiza desde el punto de vista de la calidad de la soluciones, es decir encontrar la mejor solución conocida y el número de evaluaciones realizadas de la función objetivo. Los algoritmos seleccionados fueron tomados en cuenta por contener la información necesaria para la comparación, a excepción de la búsqueda tabú que es considerado uno de los mejores algoritmos conocidos para resolver el JSSP.

Tomando en cuenta que nuestro algoritmo no posee ningún mecanismo de reparación y que sólo el motor búsqueda ha logrado obtener buenos resultados, podemos concluir que las modificaciones realizadas al AS original mejoraron la calidad de los resultados, haciéndolo competitivo en el ahorro de evaluaciones de la función objetivo, sin perder calidad en la soluciones, aún cuando los problemas crezcan en el número de recursos.

Nuestro algoritmo resulta ser una buena opción para resolver problemas de programación de horarios, cuando el costo computacional es una prioridad.

6.1. Trabajo futuro

Como trabajo futuro se plantea:

- Realizar pruebas en cuanto a mejorar las soluciones obtenidas, esto puede lograrse a través de un mecanismo de reparación de programas a partir de la solución generada por cada una de las hormigas.
- También se requiere seguir realizando exploraciones en el comportamiento del algoritmo añadiendo a la propuesta final AS_{cp} mecanismos que fueron previamente probados en forma individual tales como:
  - El uso de la distribución inversa.
  - Cambios controlados en el parámetro de persistencia de la feromona durante la ejecución del algoritmo, tal como se realizó con los parámetros de influencia de α y β.
  - La actualización de un solo porcentaje de hormigas.
  - Cálculo de desperdicio de tiempo.
  - Exclusión de la operación de trabajo similar a la operación actual en la que se localiza la hormiga.

Bibliografía

[1] E. H. L. Aarts, J. H. M. Korst, and P. J. M. van Laarhoven. Simulated annealing. *E. H. L. Aarts y J.K. Lenstra, editores, Local Search in Combinatorial Optimization John Wiley & Sons, Chichester*, pages 91–120, 1997.

[2] E. H. L. Aarts, R. J. M. Vaessens, and J. K. Lenstra. Job shop scheduling by local search. In *INFORMS J. Comput.*, volume 8(3), pages 302–317, 1996.

[3] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. In *Management Science*, volume 34(3), pages 391–401, 1988.

[4] D. Applegate and W. Cook. A computational study of the job-shop scheduling instance. In *ORSA Journal on Computing*, volume 3, pages 149–156, 1991.

[5] T. Bäck. Evolutionary algorithms in theory and practice. *Oxford University Press, New York, NY*, 1996.

[6] J. E. Beasley. Or-library: Distributing test problems by electronic mail. In *Journal of the Operations Research Society*, volume 41(11), pages 1069–1072, 1990.

[7] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, Englewood Cliffs, NJ, 1996.

[8] B. Bullnheimer, R. F. Hartl, and C. Strauss. A new rank-based version of ant system: A computational study. *Central European Journal for Operations Research and Economics*, 7(1):25–38, 1999.

[9] B. Carey. When ants go marching, they count their steps, 2006. http://www.livescience.com/animalworld/060629_ant_pedometers.html.

[10] J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. In *Annal of Operation Research*, volume 26, pages 269–287, 1990.

[11] C. Coello. *Introducción a la Computación Evolutiva - Notas de Curso*. CINVESTAV-IPN Departamento de Ingeniería Eléctrica, Sección de Computación, Mayo, 2003.

[12] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL-Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[13] D. Cortés. Un sistema inmune artificial para resolver el problema del job shop scheduling. Master's thesis, CINVESTAV, 2004. MSc in Electrical Engineering.

[14] M. Dorigo. *Optimization, learning and natural algorithms [in Italian].* PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, 1992.

[15] M. Dorigo and L. M. Gambardella. Ant colonies for the travelling salesman problem. *BioSystems*, 43(2):73–81, 1997.

[16] M. Dorigo, V. Maniezzo, and A. Colorni. Positive feedback as a search strategy. *Technical report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan*, 1991a.

[17] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: An autocatalytic optimizing process. *Technical report 91-016 revised, Dipartimento di Elettronica, Politecnico di Milano, Milan*, 1991b.

[18] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man. and Cybernetics–Part B*, 26(1):29–41, 1996.

[19] M. Dorigo and T. Stützle. *Ant Colony Optimization.* Asco Typesetters, Hong Kong, 2004.

[20] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan.*, pages 39–43, 1995.

[21] T. A. Feo and M. G. C. Resende. Greedy randomized adaptative search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[22] P. Festa and M. G. C. Resende. Grasp: An annotated bibliography. *P. Hansen y C. C. Ribeiro, editores, Essays and Surveys on Metaheuristics. Kluwer Academic Publishers*, 2001.

[23] S. French. Sequencing and scheduling – an introduction to the mathematics of the job-shop. *Chichester: Ellis Horwood*, 1982.

[24] F. Glover and M. Laguna. Tabu search. *Kluwer Academic Publishers, Boston, MA*, 1997.

[25] P. P. Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. *La théorie de la Stigmergie: essai d interpretation du Comportement des Termites Constructeurs*, pages 41–81, 1959.

[26] H. Hancock. Theory of maxims and minims. *Dover Publications, Inc., Nueva York*, 1960.

[27] J. Holland. Adaptation in natural and artificial systems. *University of Michigan Press, Ann Arbor*, 1975.

[28] J. H. Holland. Concerning efficient adaptative systems. *In M. C. Yovits. G. T. Jacobi, and G. D. Goldstein, editors, Self-Organizing Systems. Spartan Books, Washington, D.C.*, pages 215–230, 1962.

[29] J. H. Holland. Outline for a logical theory of adaptative systems. *Journal of the Association for Computing Machinery*, 9:297–314, 1962.

[30] J. Kennedy and R. C. Eberhart. Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ.*, pages 1942–1948, 1995.

[31] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science, 220*, pages 671–680, 1983.

[32] R. Landa and C. A. Coello. A cultural algorithm for solving the job shop scheduling problem. In Yaochu Jin, editor, *Knowledge Incorporation in Evolutionary Computation*, pages 37–55. Springer, 2005.

[33] S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). In *Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania*, 1984.

[34] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. *F. Glover and G. Kochenberger, editores, Handbook of Metaheuristics. Kluwer Academic Publishers*, pages 321–353, 2003.

[35] P. D. Martin. *A Time-Oriented Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem.* PhD thesis, School of Operations Research & Industrial Engineering, Cornell University, Ithaca, New York, August 1996. Ph.D. Thesis.

[36] A. Marzal, M. Castro, and P.Albar. Apuntes de algorítmica (capítulo 10) introducción a la complejidad de problemas, 2003, 2004, 2005. http://aulavirtual.uji.es/mod/resource/view.php?id=15747.

[37] R. Moraga, G. Whitehouse, and G. Depuy. Meta-raps: Un enfoque de solución eficaz para problemas combinatorios. *Revista Ingeniería Industrial Año 2. No.1*, Segundo semestre 2003.

[38] T. E. Morton and D. W. Pentico. Scheduling job shops: Basic methods. In Portland State University Dundar F. Kocaoglu, editor, *Heuristic Scheduling System with applications to production systems and project management*, chapter 15, pages 359–421. Wiley-IEEE, 1993.

[39] E.Ñowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[40] D. Vigo P. Toth. The vehicle routing problem. *Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia.*, 2001.

[41] S. K. Sahni and T. Gonzalez. P-complete approximation problems. In *Journal of the Assoc Comput.*, volume 3, pages 555–565, 1976.

[42] P. Sánchez and S. López. Programación de tareas, un reto diario en la empresa. *Anales de mecánica y electricidad*, pages 24–30, Mayo – junio 2005.

[43] T. Stützle and H. H. Hoos. Improving tha ant system: a detailed report on the max-min ant system. *Technical report AIDA-96-12, FG Intellektik, FB Informatik, TU Darmstadt, Germany*, 1996.

[44] T. Stützle and H. H. Hoos. The max-min ant system and local search for the traveling salesman problem. *In T. Bäck, Z. Michalewicz, and X. Yao (Eds.), Procedings of the 1997 IEEE Intertational Conference on Evolutionary Computation (ICEC'97). Piscataway, NJ, IEEE Press*, pages 309–314, 1997.

[45] T. Stützle and H. H. Hoos. Max-min ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[46] M. Tupia and David Mauricio. Un algoritmo voraz para resolver el problema de la programación de tareas. *Revista de Investigación de Sistemas e Informática (RISI)*, 1(1), julio-diciembre 2004.

[47] Juan Angel Alvarez Vázquez. Un algoritmo branch&bound para el problema de job shop scheduling. Master's thesis, ITESM, Campus Cuernavaca, Mayo, 2000.

[48] M. Ventresca and B. M. Ombuki. Ant colony optimization for job shop scheduling problem. Technical report, Department of Computer Science, Brock University, February 2004.

[49] M. Wittlinger, R. Wehner, and H. Wolf. The ant odometer: Stepping on stilts and stumps. In *Department of Neurobiology, University of Ulm, D-89069 Ulm, Germany*, volume 312(5782), pages 1965–1967, 2006.

[50] C. H. Papadimitriou y K. Steiglitz. *Combinatorial Optimization - Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[51] Takeshi Yamada and Ryohei Nakano. Job-shop scheduling. In P. J. Fleming A. M. S. Zalzala, editor, *Genetic Algorithms in Engineering Systems*, chapter 7, pages 134–160. IET, 1999.