IMPLEMENTACIÓN Y EVALUACIÓN DE UNA HEURÍSTICA DE COLONIA DE HORMIGAS EN LA PROGRAMACIÓN DE N TRABAJOS EN UNA MÁQUINA, CON EL OBJETIVO DE MINIMIZAR EL RETARDO MÁXIMO

OSCAR YECID BUITRAGO SUESCÚN

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL
MAESTRÍA EN INGENIERÍA INDUSTRIAL
BOGOTÁ D.C.
2004

IMPLEMENTACIÓN Y EVALUACIÓN DE UNA HEURÍSTICA DE COLONIA DE HORMIGAS EN LA PROGRAMACIÓN DE N TRABAJOS EN UNA MÁQUINA, CON EL OBJETIVO DE MINIMIZAR EL RETARDO MÁXIMO

OSCAR YECID BUITRAGO SUESCÚN

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

Asesor
Dr. EDGAR GONZALES BUTRÓN

UNIVERSIDAD DE LOS ANDES
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL
MAESTRÍA EN INGENIERÍA INDUSTRIAL
BOGOTÁ D.C.
2004

A:

Teresa Suescún Lozano
Héctor Suescún Lozano

AGRADECIMIENTOS

A Dios por darme la oportunidad de vivir esta experiencia.

A mi madre y mi familia por el apoyo prestado.

Al Doctor Edgar Gonzáles por su acertada asesoría y ayuda a lo largo del proyecto.

A Rodrigo Britto, Juan Pablo Caballero y Andrés Ramírez por su gran colaboración.

A mis compañeros de maestría.

A la Universidad de los Andes y al Departamento de Ingeniería Industrial.

CONTENIDO

LISTA DE TABLAS

Pág

INTRODUCCIÓN

El campo de la programación de la producción es muy importante desde el punto de vista investigativo y práctico ya que en el se modelan y plantean soluciones a diversas situaciones que son de interés para la industria y que a la vez constituyen verdaderos retos matemáticos y de investigación de operaciones. Dentro de este campo, un problema que se ha considerado como aspecto central en la planeación de operaciones y de la producción es la programación de máquinas [43].

A su vez es importante no separar demasiado el interés académico y práctico de los problemas ya que como se menciona en [41], ocurrió que en algún momento solo se concentro el esfuerzo en estudiar la complejidad de los problemas y en hallar procedimientos óptimos de solución. Esto hizo que los cursos de ingeniería se convirtieran en cursos de matemáticas y además centraran sus esfuerzos en la solución de problemas de poco interés para la industria. En realidad, en la mayoría de las situaciones prácticas se queda satisfecho con encontrar buenas soluciones a los problemas y no necesariamente la optima, ya que la búsqueda de esta última puede demandar cantidades ilimitadas de tiempo. Es por lo anterior que los enfoques metaheurísticos han cobrado el interés y la importancia que tienen.

Por otra parte, en este trabajo se decidió estudiar el desempeño del algoritmo de colonia de hormigas aplicado al problema de programación de una máquina, por dos causas fundamentales; es un algoritmo relativamente nuevo (fue introducido por [19] a principios de los noventa) y con poca difusión en nuestro medio y la segunda es que ha demostrado ser competitivo frente a otras técnicas de solución [2] [28], lo que lo hace una atractiva y prolifera fuente de trabajos de investigación.

El presente documento esta dividido en seis capítulos, en el primero se muestran la importancia del problema, los enfoques de solución de este, los objetivos general y específicos, los antecedentes y la metodología empleada.

En el segundo capítulo se hace una presentación del problema de secuenciar n trabajos en una máquina y los procedimientos de solución a este problema, también se incluye un breve resumen sobre diseño de experimentos ya que esta herramienta se uso en la determinación de los valores de los parámetros utilizados en el algoritmo implementado.

En la tercera parte se hace una completa exposición de la metaheurística de Colonia de Hormigas iniciando por una descripción del comportamiento biológico de las hormigas y culminando con una comparación entre colonia de hormigas y otras metaheurísticas.

En él capitulo cuatro se realiza una detallada descripción del algoritmo implementado, el diseño de experimentos, las mejoras que se proponen y de la forma en que se generaron los problemas analizados. En el quinto capitulo se presentan los resultados obtenidos y la respectiva discusión. Se presentan en el sexto y último capitulo las conclusiones y recomendaciones para futuras investigaciones.

1. CONSIDERACIONES GENERALES

1.1 PROGRAMACIÓN DE UNA MÁQUINA

El problema de programación de n trabajos en una máquina consiste en encontrar la forma de secuenciar los n trabajos, de manera que se minimice una función objetivo, por ejemplo tiempo de flujo, tardanza ò retardo máximo entre otros.

1.1.1 Importancia Del Problema. Este problema es importante ya que una máquina puede ser el cuello de botella en un ambiente de trabajo, porque ciertos ambientes de trabajo pueden ser modelados como una máquina [28] y además porque dicho problema debe ser solucionado en algoritmos que buscan programar la producción en ambientes de trabajo diferentes y más complejos tales como los talleres (como es el caso del algoritmo del cuello de botella móvil).

En la mayoría de los trabajos reportados en la literatura [2] [28] se supone que todos los trabajos que deben ser programados están disponibles en tiempo cero (ò al mismo tiempo), sin embargo esta suposición no se cumple en la mayoría de las situaciones reales ya que los trabajos provienen de diferentes estaciones que los liberan en diferentes tiempos, es decir que cada trabajo tiene un tiempo de inicio diferente en el que puede empezar a ser procesado en la máquina que se esta programando. En el estudio realizado en el presente trabajo se considera que los trabajos están disponibles en tiempos diferentes, aspecto que hace el problema más real e interesante.

1.1.2 Enfoques Para Solucionar El Problema. El problema de programación de una máquina cuando los tiempos de inicio son diferentes ha sido clasificado como de tipo NP – Hard [45] y para solucionarlo se pueden implementar procedimientos de ramificación y acotación ó procedimientos heurísticos. Los primeros garantizan encontrar la solución óptima, pero para problemas de tamaño aun no tan grande, se tornan restrictivos desde el

punto de vista computacional. Los segundos por su parte, aunque no garantizan encontrar la solución optima si permiten encontrar buenas soluciones en un tiempo moderado.

Las heurísticas más importantes que se han implementado a la solución del problema son la búsqueda tabú, recocido simulado y los algoritmos genéticos. Todos estos tienen en común que son procedimientos de mejoramiento local, es decir que parten de una solución inicial factible y la mejoran iterativamente.

Por otra parte existen algoritmos que se basan en una filosofía diferente, la cual consiste en ir construyendo paso a paso una solución sin necesidad de una solución inicial, estos procedimientos se conocen como de tipo constructivo y dentro de ellos se encuentra la metaheurística de optimización con colonia de hormigas.

Para considerar una situación real, en este trabajo se soluciona un problema de secuenciación de una máquina en la cual los trabajos a ser programados se encuentran disponibles en tiempos diferentes. Dicho problema se soluciono mediante la implementación de optimización con colonia de hormigas (ACO), es importante anotar que uno de los aspectos que motivo la realización de este trabajo es que cuando se revisa la literatura sobre el tema, la mayoría de las veces se encuentran pasos de mejoramiento local en la parte final de los algoritmos de colonia de hormigas, con lo que queda el vacío del comportamiento real del algoritmo y de si los buenos resultados son virtud de la colonia de hormigas ó de los pasos de mejoramiento local mencionados.

1.2 ANTECEDENTES

El problema de programación de una máquina es atractivo para la investigación y ha sido estudiado ampliamente bajo diversos enfoques, entre los que se encuentran la programación dinámica [44], procedimientos de ramificación y acotación [14], teoremas y reglas de dominancia para reducir el esfuerzo de los procedimientos de ramificación y acotación [24], métodos Lagrangianos [26] y teoremas de descomposición [53].

Como se dijo, la literatura es generosa en problemas de programación de una máquina, sin embargo cuando el objetivo de la programación es minimizar el retardo máximo de n trabajos con tiempos de inicio diferentes, el número de artículos disponibles se reduce. Por otra parte, [45] demostró que este problema es NP – Hard.

Aunque en sus inicios se concentró en el problema del agente viajero [20][21][32], la metaheurística de colonia de hormigas también ha sido utilizada en la resolución de problemas de producción, en especial secuenciamiento de máquinas, en el trabajo de [2] se estudia en la minimización de la tardanza total en la programación de n trabajos, mientras [23] desarrolla un algoritmo de colonia de hormigas para solucionar el mismo problema pero con tiempos de alistamiento dependientes de la secuencia. Es importante anotar que en estos trabajos se incluyen procedimientos de mejoramiento local a las soluciones obtenidas y no se reportan los resultados obtenidos sin este mejoramiento. Entre los objetivos de este trabajo esta el evaluar el comportamiento de colonia de hormigas en sí, es decir sin pasos de mejoramiento local.

1.3 METODOLOGÍA

El trabajo se realizo en etapas, la primera consiste en una revisión bibliografía sobre los temas de programación de una máquina, algoritmos de solución y metaheuristica de colonia de hormigas. En la segunda etapa se definió el problema a trabajar y cual seria el enfoque algorítmico que se debía implementar, se considero que un aspecto importante en el desempeño del algoritmo es la forma en que se asignan en cada iteración los trabajos iniciales para cada secuencia. Una estrategia comúnmente utilizada es la de tomar como trabajo inicial el último trabajo de la secuencia anterior construida por la hormiga en consideración [23], la justificación de este procedimiento es que con el se mantiene la continuidad en el "rastro" dejado por la colonia de hormigas. Para evaluar este aspecto, en este trabajo se propone otra forma de asignar los trabajos iniciales. También se realizan propuestas sobre la forma de escoger los trabajos cuando se están construyendo las soluciones.

Para determinar el valor de los parámetros y de los exponentes de los factores de visibilidad utilizados en el algoritmo, se realizo un diseño de experimentos con el fin de determinar cuales de ellos son significativos y obtener una superficie de respuesta que pudiera ser optimizada, ya que en los reportes de la literatura no se halla una explicación clara de cómo se obtuvieron los valores que usan. En los trabajos de [2] [23] utilizan como valores de los exponentes y parámetros los recomendados por [20] ó los obtienen haciendo pruebas de ajuste.

Posteriormente se desarrollo un programa codificado en lenguaje C, con el que se realizaron las corridas para obtener los resultados que se analizaron. En esta etapa también se generaron los problemas que se correrían con el programa que se elaboró.

Por último se realizo una fase de análisis y comparación de resultados, para ello los problemas generados se solucionaron mediante la regla de despacho EDD y por medio de las metaheurísticas de recocido simulado y búsqueda Tabú disponibles en el software LISA.

1.4 OBJETIVO GENERAL

Implementar una metaheurística de colonia de hormigas para solucionar el problema de secuenciación en una máquina de n trabajos con tiempos de inicio diferentes, con el objetivo de minimizar el retardo máximo.

1.5 OBJETIVOS ESPECÍFICOS

Presentar la metaheurística de colonia de hormigas para que sea considerada como alternativa en la solución de problemas de optimización combinatoria.

Encontrar factores de visibilidad que le ayuden a las hormigas a construir mejores soluciones.

Determinar el valor de los exponentes para el rastro de feromonas y los factores de visibilidad por medio del diseño de experimentos.

Analizar el desempeño del algoritmo sin incluir pasos de mejoramiento local a las soluciones halladas por las hormigas.

Realizar comparaciones con soluciones obtenidas mediante otros procedimientos para el mismo problema.

2. MARCO TEÓRICO

2.1 FORMULACIÓN DEL PROBLEMA DE PROGRAMACIÓN DE N TRABAJOS EN UNA MAQUÍNA

La programación de trabajos en una máquina es un aspecto de mucha importancia en la planeación y programación de la producción, debido a que muchos ambientes de producción están conformados por una sola máquina (especialmente en las pequeñas industrias), como ejemplo considérese una empresa de elaboración de tubos de PVC en la que se cuenta con sola una máquina de extrusión. Por otra parte en numerosas ocasiones ambientes de trabajo más complejos se pueden modelar como una sola máquina [23]. Cuando se tiene un sistema de varias máquinas, la secuenciación de los trabajos en la máquina cuello de botella determina el desempeño de todo el sistema, por ello se programa primero dicha máquina y luego las demás, en esta situación el problema original se reduce a un problema de una sola máquina.

El problema de programación de una sola máquina también es importante en los procedimientos de descomposición que buscan solucionar ambientes de trabajo complicados, en este tipo de enfoques se soluciona repetidamente el problema de programación de una máquina, tal es el caso del algoritmo del cuello de botella móvil, con el que se busca programar la producción en talleres (job shop).

El problema considerado en este trabajo se define de la siguiente manera: Se tiene una máquina que debe usarse para procesar n trabajos de manera tal que se minimice el retardo máximo. Cada uno de los trabajos tiene las siguientes características:

Tiempo de proceso (p_j). Representa el tiempo que el trabajo j permanece procesándose en la máquina una vez que ingresa a ella.

Tiempo de liberación (r_j). Es el tiempo en el que el trabajo j ingresa al sistema, es decir el tiempo en el que trabajo esta disponible para empezar a ser procesado. En muchas ocasiones todos los trabajos están disponibles a tiempo cero [ó al mismo tiempo], sin embargo en situaciones más reales, cuando los trabajos llegan a la máquina en consideración, provienen de diferentes estaciones las cuales los liberan en tiempos diferentes.

Este ingrediente le añade mas complejidad al problema de secuenciación de una máquina, ya que aún si todos los trabajos están disponibles a tiempo cero, dependiendo de la función objetivo el problema es NP- Hard [45].

Tiempo de entrega (d_j). Es el tiempo máximo en el que se debe terminar de procesar el trabajo j, es decir el tiempo en el que se le prometió al cliente [interno o externo] que se le entregaría el trabajo. Si el trabajo es entregado después de la fecha de entrega, se incurre en una penalización por incumplimiento, estas sanciones no son siempre del tipo multas o similares, se puede llegar a perder el cliente y este costo es mas elevado y difícil de cuantificar.

Tiempo de terminación (C_j). Es el tiempo en el que el trabajo j termina de ser procesado en la máquina, se calcula como el tiempo en el que el trabajo empezó a ser procesado en la máquina más el tiempo de proceso. En el problema que se está considerando, debido a que se tienen tiempos de liberación diferentes para cada trabajo, se debe calcular el tiempo de terminación como el tiempo de proceso más el máximo entre el tiempo actual del sistema y el tiempo de liberación. El tiempo actual del sistema es el tiempo en que se termina de procesar el último trabajo de la secuencia parcial.

No se permiten interrupciones en los trabajos, es decir que una vez se introduce un trabajo a al máquina se debe terminar su proceso.

En los problemas de programación de la producción existen varios objetivos que se pueden considerar; minimizar el retardo máximo, minimizar la tardanza total, minimizar la

tardanza ponderada, minimizar la tardanza máxima, entre otros. En este trabajo se tiene como objetivo encontrar la secuencia en que se deben programar n trabajos con el fin de minimizar el retardo máximo.

El retardo para un trabajo esta definido como:

$$L_j = C_j - d_j \qquad (1)$$

El retardo máximo esta dado por:

$$L_{max} = \max(L_1, L_2, \ldots, L_n) \qquad (2)$$

2.1.1 Representación Gráfica Del Problema. El problema de secuenciación de n trabajos en una máquina se puede representar por medio de un grafo acíclico dirigido con un nodo de origen y un nodo de terminación [2]. Es suficiente un nodo para representar el subconjunto de trabajos ya programados debido a que el nodo contiene cuales trabajos han sido programados pero no en que orden. Se puede decir entonces que los nodos representan conjuntos de trabajos y los arcos representan el trabajo que se añade a este conjunto.

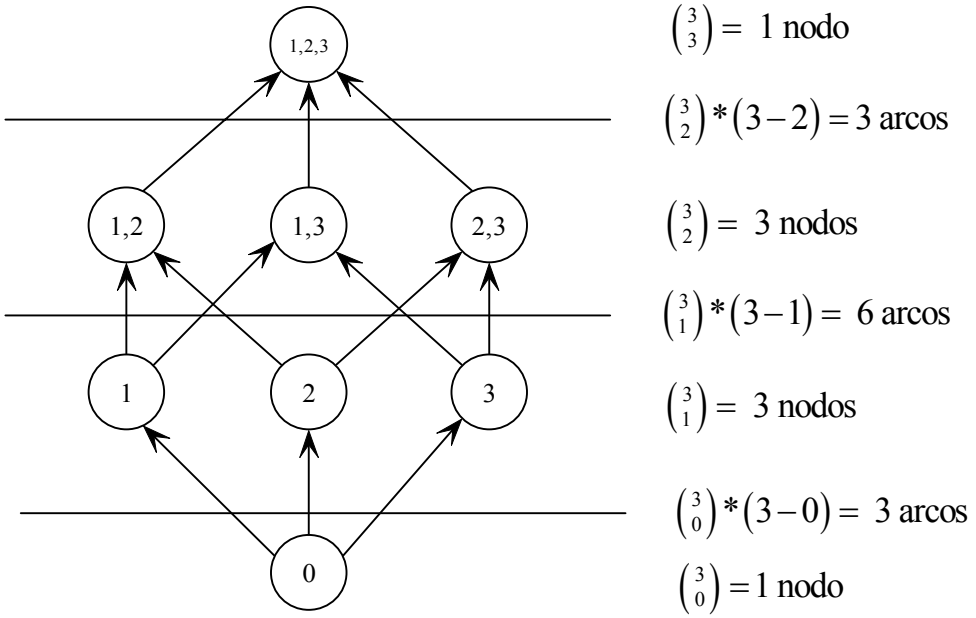
El nodo fuente esta en el primer nivel y representa el inicio de la secuencia cuando ningún trabajo ha sido programado (el conjunto correspondiente es vació), de este nodo salen n arcos, cada arco representa el tiempo de terminación del correspondiente trabajo.
En el segundo nivel se tienen n nodos cada uno de los cuales tiene un arco de entrada y n-1 arcos de salida. En el tercer nivel cada nodo tiene dos arcos de entrada y n-2 de salida y así sucesivamente. El nodo de terminación representa el conjunto de todos los trabajos y tiene n arcos de entrada y ninguno de salida.

En resumen, el grafo tiene n+1 niveles y n transiciones, en cada transición exactamente un trabajo es añadido. En el nivel i ($i = (0, \ldots n)$ hay $\binom{n}{i}$ nodos. El numero de transiciones

entre el nivel y el nivel i+1 esta dado por $\binom{n}{i}$ * [n-i] arcos. En el grafo completo el numero de arcos es $\sum_{i=0}^{n-1}\left[\binom{n}{i}*(n-i)\right] = 2^n \frac{n}{2}$ y el numero de nodos es $\sum_{i=0}^{n}\binom{n}{i} = 2^n$

En la figura 1 se representa el grafo para un problema de secuenciación de tres trabajos.

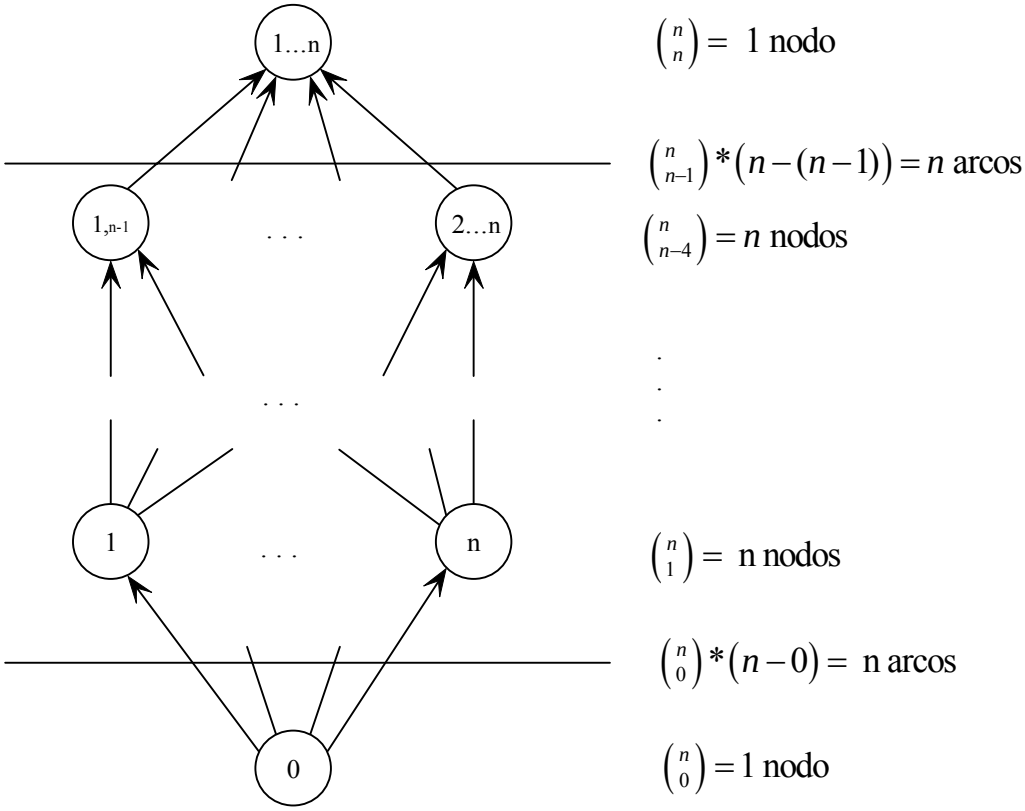
Figura 1. Grafo para un problema de secuenciación de tres trabajos en una máquina.



Una ruta entre el nodo fuente y el nodo terminal representa una solución factible al problema, la secuencia óptima es la ruta ò secuencia de trabajos que minimiza el retardo máximo.

En la figura 2 se muestra la representación para el caso de n trabajos [2].

Figura 2. Grafo para el caso de n trabajos



2.2 MÉTODOS DE SOLUCIÓN

El problema considerado ha sido atacado básicamente por tres enfoques; técnicas de ramificación y acotación, reglas de despacho y métodos metaheurísticos.

2.2.1 Ramificación y Acotación. Los procedimientos de ramificación y acotación son esquemas de enumeración en los que algunas soluciones (programaciones) o familias de soluciones son descartadas debido a que sus valores objetivos son superiores a una cota inferior calculada con anterioridad y esta cota es a su vez superior al valor de la función objetivo de una solución hallada antes. La principal ventaja de estos procedimientos es que garantizan encontrar la solución óptima mientras que su desventaja es que consumen mucho tiempo de computador, hasta el punto de resultar prohibitivos para problemas de tamaño relativamente pequeño, 30 trabajos, [52].

2.2.2 Reglas De Despacho. Son reglas que dan un orden de prioridad a los trabajos que están a la espera de ser procesados, este orden se asigna teniendo en cuenta los atributos de los trabajos y/ó de las máquinas que los deben procesar. Es decir que cuando la máquina queda libre el siguiente trabajo en ser procesado es aquel que tiene una prioridad más alta. Estas reglas han sido estudiadas por décadas y se pueden encontrar numerosos reportes en la literatura [52]. El principal atractivo de estas reglas es su sencillez y que para problemas de ciertas características garantizan la solución optima. Por ejemplo si se quiere minimizar la tardanza ponderada y tanto los tiempos de entrega como de liberación de todos los trabajos son cero, la regla de menor tiempo de proceso ponderado (WSPT) garantiza la solución optima [52].

Una forma de clasificar las reglas de despacho es en estáticas y dinámicas, ya que sean independientes o dependientes del tiempo. Existen muchas reglas de despacho entre las que se encuentran las llamadas reglas de despacho compuestas, las cuales son un tanto más complejas. A continuación se presentan algunas reglas de despacho

Menor tiempo de liberación primero (ERD). Esta regla estática es equivalente a la conocida regla primero en llegar primero en ser servido.

Menor tiempo de entrega primero (EDD). Cuando la máquina esta disponible se empieza a procesar el trabajo que debe entregarse primero.

Mínimo tiempo de holgura primero. Es la variante dinámica de la regla EDD. Cuando la máquina se libera en el tiempo t, se calcula máx $(d_j - p_j - t, 0)$, el trabajo con el menor valor es el que se procesa. Es interesante notar que al comparar dos trabajos i y j bajo esta regla, en un determinado tiempo el trabajo i puede tener mayor prioridad que j, pero en un tiempo posterior ambos trabajos pueden tener la misma prioridad.

Menor tiempo de proceso ponderado primero (WSPT). Cuando la máquina esta libre, se calcula la relación w_j/p_j donde w_j es un factor que tiene en cuenta la importancia del trabajo, el trabajo con el mayor valor de la relación calculada es el siguiente en ser procesado.

Mayor tiempo de proceso primero (LPT). Se ordenan y procesan los trabajos en orden decreciente de sus tiempos de proceso.

2.2.3 Procedimientos Metaheurísticos. Debido a los inconvenientes presentados con los dos enfoques anteriores, se han desarrollado procedimientos metaheurísticos para la solución de problemas de tipo NP-Hard. Estos procedimientos han sido ampliamente estudiados y aunque no garantizan encontrar la solución optima, en ocasiones lo logran y en otras pueden proporcionar una buena solución en tiempo razonable. Entre las metaheurísticas mas conocidas y que han sido aplicadas al problema de secuenciación de una máquina se encuentran; recocido simulado, búsqueda tabú y algoritmos genéticos. A continuación se hace una pequeña descripción de cada uno de ellos.

2.2.3.1 Recocido simulado [42]. Este procedimiento se origino en la ciencia de materiales, al hacer analogías entre la termodinámica del proceso de recocido (que se usa para eliminar tensiones internas de metales) y los problemas de optimización combinatoria. Es un procedimiento de búsqueda local por lo que se necesita una solución inicial que se va mejorando iterativamente. Se presenta una breve descripción del método [52], para lo cual a continuación se hace una aclaración de la nomenclatura usada.

En una iteración k del procedimiento se tiene una secuencia (solución) actual S_k y la mejor secuencia encontrada hasta el momento se denota por S_0. $G(S_k)$ representa el valor de la función objetivo para una secuencia dada. El algoritmo en cada iteración busca una nueva secuencia en la vecindad de S_k. Un candidato S_c es una secuencia que se secciona de entre los vecinos de S_k. $\beta_1 \geq \beta_2 \geq ... > 0$ son parámetros de control que en ocasiones son

conocidos como parámetros de enfriamiento o temperaturas. Si $G(S_c) \geq G(S_k)$, se hace un movimiento hacia S_c con probabilidad.

$$P(S_k, S_c) = \exp\left(\frac{G(S_k) - G(S_c)}{\beta_k}\right). \quad (3)$$

Paso 1.

k = 1 y seleccione β_{-1}.

Seleccione una solución [secuencia] inicial, usando alguna heurística.

$S_0 = S_1$

Paso 2.

Seleccione una secuencia candidata del vecindario de S_k.

Si $G(S_0) < G(S_c) < G(S_k)$, haga $G(S_{k+1}) \geq G(S_c)$ y vaya al paso tres.

Si $G(S_c) < G(S_0)$, haga $S_0 = S_{k+1} = S_c$ y vaya al paso tres.

Si $G(S_c) > G(S_k)$, genere un número aleatorio U_k en el intervalo [0, 1].

Si $U_k \leq P(S_k, S_c)$, haga $S_{k+1} = S_c$, $S_{k+1} = S_k$ vaya al paso tres.

Paso 3.

Seleccione $\beta_{k+1} \leq \beta_k$.

Incremente k en una unidad.

Si k = N termine, en caso contrario vaya al paso 2.

2.2.3.2 Búsqueda Tabú. El nombre y la metodología de este procedimiento fueron introducidos por [35], el algoritmo de búsqueda tabú es parecido al de recocido simulado en cuanto a que es de búsqueda local y define la vecindad de manera similar. En términos generales el método de búsqueda tabú puede esbozarse de la siguiente manera:

Partiendo de una solución inicial factible del problema considerado, se realizan movimientos paso a paso hacia una solución que proporcione el valor mínimo de la función objetivo G. Para esto se puede representar a cada solución por medio de un punto s (en

algún espacio) y se define una vecindad N(s) para cada punto s como un conjunto de soluciones adyacentes a la solución s.

El paso básico del procedimiento consiste en empezar desde un punto factible s y generar un conjunto de soluciones en N(s), de estas se elige la mejor (s*) y se ubica en este nuevo punto ya sea que G(s*) tenga o no mejor valor que G(s). Hasta este punto el procedimiento es semejante a otras técnicas de mejoramiento local a excepción del hecho de que se puede mover desde s hacia una solución peor s*.

Existen varias formas de definir el entorno reducido de una solución. La más sencilla consiste en etiquetar como tabú las soluciones previamente visitadas en un pasado cercano, esta se conoce como memoria a corto plazo y se basa en guardar en una lista tabú (T) las soluciones visitadas recientemente. Así en una iteración determinada, el entorno reducido de una solución se obtendría como el entorno usual, eliminando las soluciones pertenecientes a la lista tabú. El objetivo principal de marcar las soluciones visitadas es evitar que la búsqueda se vuelva cíclica. Por ello se considera que tras un cierto número de iteraciones la búsqueda está en una región distinta y puede liberarse de la condición de tabú (pertenencia a T) a las soluciones antiguas. De esta forma se reduce el esfuerzo computacional de calcular el entorno reducido en cada iteración. En los orígenes de la búsqueda tabú se sugerían listas de tamaño pequeño, actualmente se considera que las listas pueden ajustarse dinámicamente según la estrategia que se esté utilizando [35].

2.2.3.3 Algoritmos genéticos. Los algoritmos genéticos son mas generales y abstractos que los procedimientos de recocido simulado y búsqueda tabú y estos de cierta forma pueden ser vistos como casos especiales de los algoritmos genéticos [52].

Cuando se aplican los algoritmos genéticos a la programación de la producción, las secuencias o programaciones se ven como individuos o miembros de una población. Cada individuo se caracteriza por un factor de adaptación (fitness) que se mide por el valor asociado de la función objetivo. El proceso se hace iterativamente y cada iteración es una generación. La población de una generación consiste de los individuos existentes de la generación previa más las nuevas programaciones o hijos de la generación previa. El

tamaño de la población generalmente permanece constante de una generación a otra. Los hijos son generados a través de la reproducción y mutación de los individuos que hicieron parte de la generación previa (padres). Los individuos también son conocidos como cromosomas.
En un ambiente multimáquinas, un cromosoma puede estar formado por subcromosomas, cada uno con la información correspondiente a la secuencia de trabajos en una máquina. Una mutación en un cromosoma padre pude ser equivalente un intercambio adyacente en la correspondiente secuencia. En cada generación los individuos mas aptos se reproducen mientras los menos aptos mueren. Los procesos de nacimiento, muerte y reproducción que determinan la composición de la siguiente generación pueden ser complejos y usualmente dependen de los niveles de adaptación de los individuos de la actual generación.

Un algoritmo genético como proceso de búsqueda difiere en un importante aspecto del recocido simulado y la búsqueda tabú, en cada paso iterativo un número diferente de programaciones son generadas y llevadas al siguiente paso, en tanto que en recocido simulado y búsqueda tabú solo una programación es llevada de una iteración a la siguiente, debido a esto es que el recocido simulado y la búsqueda tabú pueden ser vistos como casos específicos de los algoritmos genéticos con un tamaño de población igual a 1 [52].

El esquema de diversificación mencionado es una característica importante de los algoritmos genéticos. Por lo tanto en los algoritmos genéticos el concepto de vecindad no se basa en una sola programación, sino mas bien un conjunto de programaciones. El diseño de la vecindad de la secuencias de la población actual se basa en técnicas mas generales que las utilizadas en recocido simulado y búsqueda tabú. Una nueva programación puede ser construida combinando partes de diferentes programaciones con la población actual, a este procedimiento se le conoce como cruzamiento o crossover.

2.3 DISEÑO DE EXPERIMENTOS

Para determinar cuales de los parámetros que se usan en el algoritmo son los más significativos y obtener los valores que se utilizarían para estos, se recurrió al diseño de experimentos. Esta metodología se aplica en estudios experimentales en los que se realizan cálculos sobre la información muestral, seguidos de la extracción de inferencias acerca de la población que se estudia.

Con frecuencia existen características del experimento que pueden ser controladas por el experimentador tales como el número de factores a estudiar, el número de niveles de los factores, combinaciones de los tratamientos a utilizar, entre otros**.** El diseño de la forma en que se decide obtener la información experimental para probar las hipótesis de interés o para determinar cuales de los factores estudiados son significativos, es determinante en el análisis de los datos y en la calidad de la información obtenida. En este trabajo se recurrió al diseño factorial en tres niveles por cada factor.

2.3.1 Diseños factoriales. En un diseño factorial se investiga simultáneamente los efectos de cierto número de diferentes factores, donde los tratamientos constan de todas las combinaciones que se pueden formar entre los diferentes factores. Cuando se asume que los efectos de los factores estudiados son independientes, el diseño factorial permite estimar cada efecto principal con la misma precisión que si todo el diseño se hubiese dedicado a ese solo factor [13].

La experimentación factorial puede ser adecuada en las siguientes situaciones [2]:

- En trabajos de exploración, donde el objetivo es determinar rápidamente los efectos de de cada uno de cierto número de factores.

- En investigaciones de las interacciones de los efectos de varios factores

- En experimentos diseñados para poder llegar a recomendaciones que deben aplicarse a una gran variedad de condiciones.

- Investigaciones dirigidas a encontrar la combinación de niveles de factores que producen una respuesta máxima ó mínima.

Cuando en un diseño factorial se tienen k factores de interés los cuales se estudian a tres niveles cada uno, se habla de un diseño factorial 3^k.

La principal desventaja del diseño factorial estriba en su tamaño y complejidad cuando los factores que se requiere estudiar son numerosos. Sin embargo es posible obtener una buena cantidad de información realizando solo una fracción del diseño completo, a estos diseños se les conoce como factoriales fraccionados. La intención de los diseños fraccionados es reducir el tamaño del experimento total en tanto se retenga información importante con relación a los factores [13].

2.3.2.1 Diseños fraccionados. Como se menciono, el número de corridas requeridas en los diseños factoriales crece rápidamente a medida que se incrementa el número de factores. Por esta razón en muchas ocasiones se requiere realizar tan solo una fracción del diseño completo, en general se puede construir una fracción $(\frac{1}{3})^p$ del diseño 3^k completo, siendo p< k y donde cada bloque tiene 3^{k-p} pruebas. El diseño obtenido es denominado 3^{k-p} fraccional.

El procedimiento para construir un diseño de este tipo es seleccionar p componentes de interacción y utilizarlos para dividir las 3^k combinaciones de tratamientos en 3^p bloques, cada uno de los bloques obtenidos es un diseño factorial fraccionado 3^{k-p}.Los contrastes definición consisten en los p efectos seleccionados inicialmente y sus $(3^p - 2p - 1)/2$

interacciones generalizadas. Los alias de los efectos principales se obtienen multiplicándolos (modulo tres) por los contrastes definición y sus cuadrados.

Sin embargo la disminución del número de pruebas no se obtiene sin sacrificar otros aspectos, los resultados están propensos a una mala interpretación que no se presentaría con el diseño completo. En los diseños factoriales fraccionados es importante conocer la resolución del diseño con el fin de poder realizar una interpretación correcta de los resultados obtenidos. Las características de resolución de los diseños más utilizados son:

Diseños de resolución III. En estos diseños no se confunden efectos principales entre si, pero algún efecto principal esta confundido con interacciones de dos factores y estos a su vez pueden estar confundidos con otras interacciones dobles.

Diseños de resolución IV. Los efectos principales no se confunden entre si ni con interacciones dobles, pero las interacciones dobles si se encuentran confundidas entre si.

Diseños de resolución V. Ni los efectos principales ni las interacciones dobles están confundidas entre si, pero las interacciones de dos factores se encuentran confundidas con interacciones triples.

2.3.2 Métodos de superficies de respuesta. Los métodos de superficie de respuesta son un conjunto de técnicas matemáticas y estadísticas útiles para modelar y analizar problemas en los que una respuesta de interés recibe influencia de varias variables.

En la mayoría de las situaciones la relación funcional entre la respuesta y las variables independientes es desconocida, por lo tanto se debe primero encontrar una aproximación adecuada de la verdadera forma funcional. Por lo general se emplea un polinomio de orden inferior en alguna región de las variables independientes, esto con el fin de conducir al investigador de manera rápida y eficiente hacia la región general del óptimo. Una vez se encuentra esta región, se puede emplear un modelo más elaborado para encontrar el óptimo.

La finalidad de los métodos de superficie de respuesta es determinar las condiciones de operación óptimas ó determinar una región del espacio de los factores en la que se satisfagan los requerimientos de operación. Análisis más detallados se pueden encontrar en [13] y [49].

El ajuste y análisis de superficies de respuesta se facilita en gran medida con la elección apropiada del diseño experimental. Para ajustar diseños de primer orden se usan diseños de primer orden ortogonales, dentro de los cuales están incluidos los diseños factoriales 2^k. Para diseños de segundo orden se usan diseños 3^k, fracciones de diseños 3^k ó diseños especiales para este fin [49]. En el caso de este estudio se realizo un diseño 3^k fraccionado.

3 METAHEURISTICA DE COLONIA DE HORMIGAS

3.1 ORIGEN Y BASES DE LA METAHEURISTICA DE COLONIA DE HORMIGAS

El algoritmo de colonia de hormigas fue propuesto inicialmente por Dorigo [19] para la solución de problemas combinatorios tales como el del agente viajero (TSP) y el de asignación cuadrática (QAP). Tiene sus bases en la observación de hormigas reales ya que estas son insectos sociales, es decir que viven en colonias, que tienen características de comportamiento que han llamado la atención de varios investigadores. Uno de estos aspectos en particular es como la colonia tiene la capacidad de encontrar la ruta más corta entre el hormiguero y las fuentes de alimento.

Mientras recorren el camino entre el hormiguero y la fuente de alimento y viceversa, las hormigas depositan en la tierra una sustancia química llamada feromona, formando en el camino un rastro de feromona. Las hormigas pueden percibir la feromona y cuando tienen que escoger una ruta se deciden por aquella que tenga el rastro más fuerte de feromonas [mayor concentración de la sustancia]. El rastro de feromonas también les permite encontrar el camino de regreso al hormiguero (ó a la fuente de alimento) y transmitir esta información a otras hormigas.

Se ha observado y también demostrado experimentalmente [15] que el rastro de feromonas le sirve a la colonia de hormigas para encontrar la ruta mas corta del hormiguero a la fuente de alimento cuando existen rutas alternativas. Uno de estos experimentos es el desarrollado por [15] en el cual se uso una colonia de hormigas de la especie Linephitema humile, cuyo hormiguero se separo de una fuente de alimento por medio de un puente de dos ramas de igual longitud tal como se ilustra en la figura, las hormigas se dejan en libertad y se mide en el tiempo el porcentaje de hormigas que escogen cada una de las

ramas. El resultado muestra que después de una fase inicial de transición en la que pueden existir oscilaciones, las hormigas tienden a converger a la misma ruta.

Figura 3. Experimento de Denebourg [15]

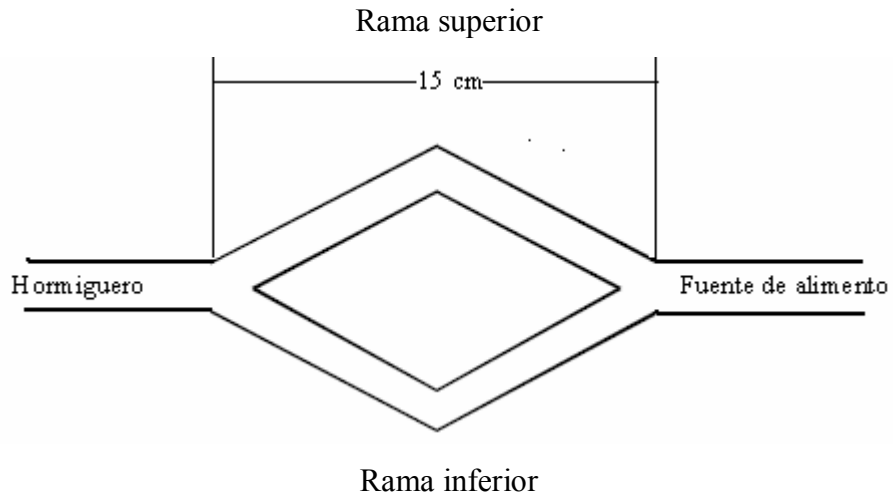
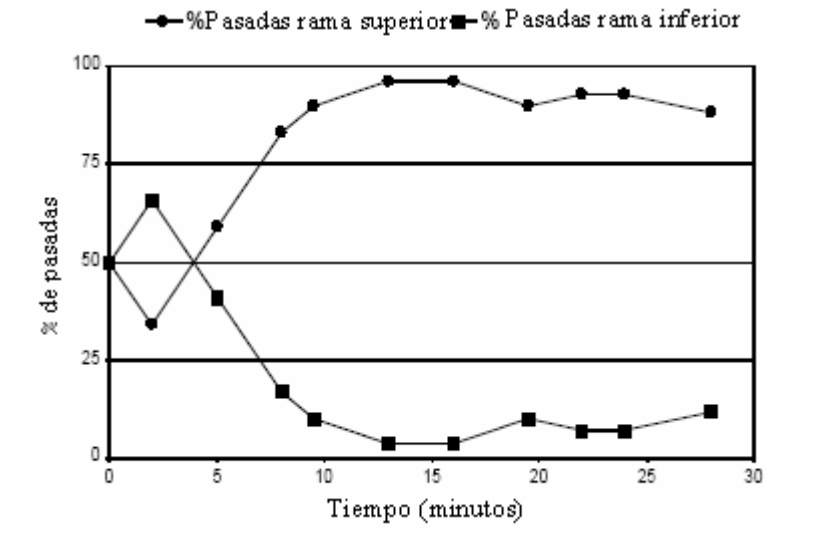


Figura 4. Resultados del experimento.



En este experimento inicialmente no hay rastro de feromonas en el puente, por lo que las hormigas pueden escoger cualquiera de las dos ramas con la misma probabilidad, sin embargo debido a la aleatoriedad de la selección, en algún momento un porcentaje mayor de hormigas habrá escogido aleatoriamente alguno de los dos caminos, como las hormigas

depositan feromonas, este camino quedara con un rastro más fuerte con lo que se incita a las hormigas que vienen atrás a escoger este camino.

El modelo probabilístico que describe este fenómeno es el descrito por [37]. Primero se hace la suposición que la cantidad de feromona en un camino es proporcional al numero de hormigas que han pasado por allí, esto implica que no se tienen en cuenta los efectos de la evaporación de la feromona, la justificación del supuesto es que como la prueba usualmente se hace durante un periodo de tiempo de una hora, la cantidad de feromona evaporada es despreciable. En el modelo, la probabilidad de escoger una de las ramas en determinado tiempo depende de la cantidad de feromona en cada una de las ramas.

Para mayor precisión sea S_m e I_m la cantidad de hormigas que escogieron la rama superior e inferior respectivamente, después que una cantidad m de hormigas han cruzado el puente. Dado que m = $S_m + I_m$, la probabilidad $\boldsymbol{P_S}(m)$ con que la (m+1)- ésima hormiga escogerá la rama superior es:

$$P_S(m) = \frac{(S_m + k)^h}{(S_m + k)^h + (I_m + k)^h} \quad (4)$$

$$P_I(m) = 1 - P_s(m) \quad (5)$$

Esta forma funcional de la probabilidad de elección de uno de los dos caminos, se obtuvo de experimentos en los que se realizo seguimiento al rastro de feromonas [51], los parámetros h y k son parámetros que indican el ajuste del modelo a los datos experimentales. El dinamismo de la elección es seguido representado por la siguiente expresión.

$$S_{(m+1)} = S_m + 1 \text{ si } \psi \le P_S \quad (6)$$

$$S_{(m+1)} = S_m \text{ en caso contrario (7)}$$

Donde ψ es una variable aleatoria uniformemente distribuida en el intervalo (0, 1).

Se han realizado simulaciones de Montecarlo para verificar la correspondencia del modelo con los resultados experimentales. Los resultados de la simulación fueron acordes con los datos obtenidos mediante experimentación con hormigas reales, cuando los valores de los parámetros son $h \approx 2$, $k \approx 20$ [51].

Se puede modificar el experimento anteriormente descrito, haciendo cada una de las ramas del puente de longitud diferente [37]. El modelo de la ecuación (6) puede describir esta nueva situación ya que el mecanismo de segregación de feromonas no ha cambiado.

Los resultados indican que la gran mayoría de las veces se escogió la ruta más corta, ya que las primera hormigas en llegar a la fuente de alimento fueron las que escogieron la ruta mas corta, estas al regresar incrementaban el rastro de feromonas sobre la rama mas corta del puente, con lo que estimulaban a las demás hormigas a escoger esta ruta. En este caso las fluctuaciones iniciales son mucho menos importantes que en el caso en que las dos ramas tiene la misma longitud. En la figura 5 se representa el montaje de esta situación y se muestran sus resultados.

Figura 5. Montaje del experimento de Goss [37]

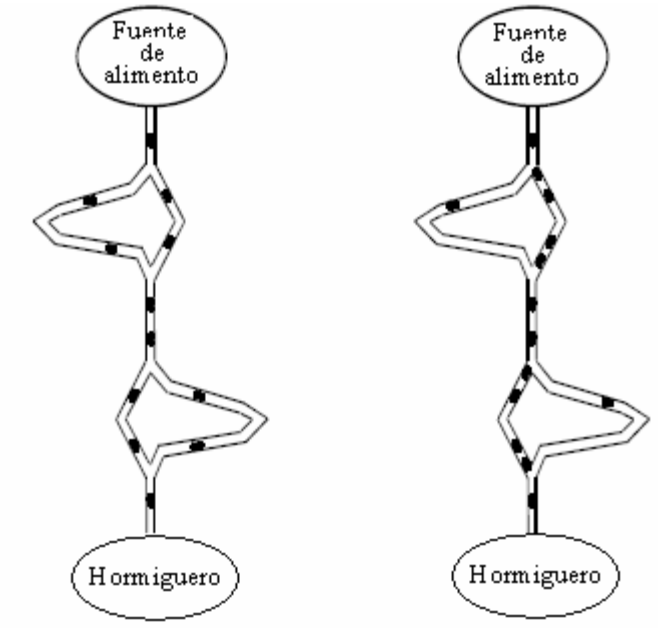
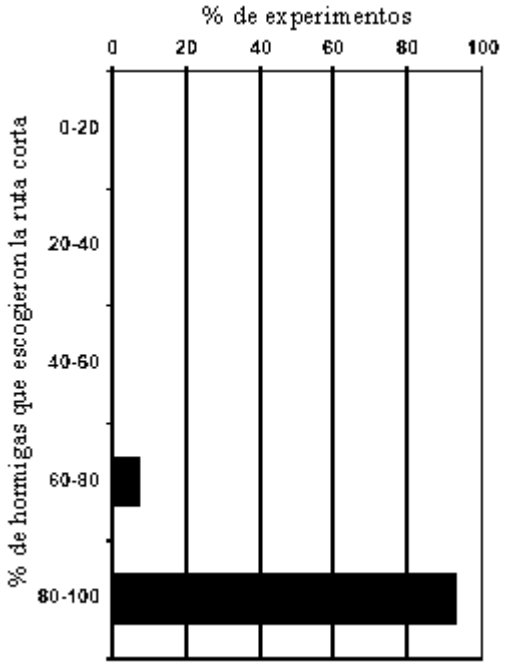


Figura 6. Resultados del experimento de Goss [37]



Resulta atractivo el hecho que aunque cada hormiga por si sola es capaz de encontrar una ruta entre el hormiguero y la fuente, solo con la cooperación de toda la colonia se puede encontrar la ruta más corta al integrar la pequeña contribución de cada uno de sus miembros.

Cada una de las hormigas puede mejorar su desempeño mediante la comunicación indirecta con las demás por medio de la segregación de la feromona, este mecanismo es conocido como comunicación por estimulo (stigmergy en ingles) [38].

Como esta descrito en el trabajo de [38] stigmergy es “la estimulación a los individuos por el desempeño que ellos alcancen”. Los insectos son capaces de responder a esos estímulos, activando reacciones codificadas genéticamente. En insectos sociales, como las hormigas y las termitas, los efectos de esas reacciones pueden dar lugar a un nuevo estimulo tanto para el insecto que lo emitió como para el que lo recibió. La producción de nuevos estímulos

debidos a la reacción a estímulos significantes determina la forma de coordinación de las actividades y por ello se puede considerar como una forma indirecta de comunicación.
Las características de la comunicación por estimulo son:

- La naturaleza física de la información liberada por los insectos que se comunican, modificando el ambiente de los sitios que visitan.

- La naturaleza local de la información que proporcionan, la cual solo puede ser percibida por aquellos insectos que visitan el estado en que fue liberada.

De acuerdo con [17] se puede hablar de comunicación por estimulo cuando exista una "comunicación indirecta mediante la modificación física del entorno de los estados que solo son localmente accesibles por los agentes comunicantes".

Debido a las características mencionadas en el modelo de comunicación por estimulo, el de las hormigas en particular, este es interesante para extenderlo a sistemas de multi-agentes artificiales aplicados a la solución de difíciles problemas de optimización, tal como lo hizo [19] dando origen a la heurística de colonia de hormigas ACO (Ants Colony Optimization).

En las hormigas la comunicación por estimulo se da por medio de las feromonas que depositan mientras caminan, mientras que en las hormigas artificiales de [19] se da mediante la modificación de "variables de feromonas" definidas apropiadamente y asociadas con los estados del problema que ellas visitan, mientras construyen la solución al problema de optimización al que fueron asignadas. Acorde con la comunicación por estimulo, las hormigas artificiales solo tienen acceso local a las variables de feromonas.

Otro aspecto en el que las hormigas artificiales imitan a las reales es en la retroalimentación, en el caso de estas últimas cuando una hormiga completa su ruta se devuelve reforzando el rastro de feromonas, en el caso de las artificiales se realizan ajustes

del nivel de feromonas cuando se encuentran buenas soluciones. Sin embargo se debe tener cuidado en este ajuste ya que puede llevar a la convergencia prematura (caer en óptimos locales).

Para solucionar este inconveniente, el sistema artificial imita al natural en otro de sus aspectos, la evaporación de la feromona. Al ser una sustancia química la feromona se evapora en la vida real haciendo que el rastro de feromonas disminuya, llegando a desaparecer si las hormigas abandonan o hacen poso uso de una ruta. A pesar de la evaporación, si una ruta es muy transitada mantendrá un nivel de feromonas que la mantiene atractiva para las hormigas que vienen atrás.

Las hormigas artificiales tienen una gran ventaja sobre las naturales, pueden imitar de ellas las mejores características observadas de su comportamiento en la búsqueda de rutas, y a la vez ser dotadas con atributos que estas últimas no poseen. Las semejanzas entre hormigas naturales y artificiales son:

- En ambos casos se tiene una colonia de individuos cooperantes que buscan encontrar una buena solución a un problema. Tanto en las colonias naturales como en las artificiales, una sola hormiga esta en capacidad de encontrar una solución [ó de construir una ruta], pero las soluciones realmente buenas nacen de la cooperación entre los individuos.

- Rastro de feromonas y comunicación por estimulo: Las hormigas reales depositan feromonas sobre el suelo que pisan para que las hormigas que lleguen a ese sitio puedan percibir el rastro dejado, las artificiales realizan modificaciones a valores numéricos de “variables de feromonas” que dependen del estado local del problema que solucionan y a las que pueden acceder los individuos que lleguen a ese mismo estado.
- Las hormigas reales no saltan, tienen que encontrar una paso a paso, las hormigas artificiales construyen soluciones mediante movimientos locales entre estados adyacentes.

- Ambas clases de hormigas usan información local proveniente de las modificaciones realizadas en el entorno por sus antecesoras, para tomar decisiones ó predecir sus movimientos futuros.

Entre las principales diferencias se pueden contar las siguientes:

- Las hormigas artificiales pertenecen a un mundo discreto, en el que deben trasladarse de un estado discreto a otro.

- Las hormigas artificiales pueden almacenar información acerca de sus movimientos pasados.

- Las hormigas artificiales pueden variar el nivel de secreción de feromonas, mientras que las naturales siempre depositan una determinada cantidad (la mayoría de las veces, ya que existen algunas especies que pueden variar la cantidad de feromona depositada). Es más, las hormigas artificiales pueden decidir si actualizan o no el nivel de feromonas de acuerdo con la calidad de las solución encontrada y lo pueden hacer al final de la solución.

- Las hormigas artificiales se pueden dotar con factores extra de visibilidad que les permitan tener visibilidad para la toma de decisiones y no depender únicamente del rastro dejado por sus compañeras de colonia.

3.2 ALGORITMO DE COLONIA DE HORMIGAS (ACO)

En los algoritmos de colonia de hormigas se tiene un conjunto finito de agentes con las características antes mencionadas y que cooperativamente buscan una solución de calidad al problema en consideración. Cada hormiga construye una solución partiendo de un estado al que ha sido asignada, va recolectando información con la cual mide su desempeño, modifica los valores de las variables de representación del problema y los transmite a las

demás hormigas. Las hormigas son individuos relativamente simples que pueden construir por si solos soluciones pobres al problema, las soluciones de alta calidad se obtienen por la interacción y cooperación entre las hormigas [17][19]. Los movimientos de cada hormiga son locales y se realizan de acuerdo a la definición de vecinos dada para el problema, la información privada de cada hormiga, a reglas de decisión estocásticas y al rastro de feromonas que es accesible para todos loe miembros de la colonia.

La información privada ó interna de las hormigas se puede considerar como su memoria y les permite a cada una, teniendo en cuenta los movimientos que realizo en el pasado, no caer en soluciones no factibles. Gracias a esta característica, cada hormiga construirá solamente soluciones factibles independientemente de la calidad de estas.

Los movimientos locales de las hormigas se realizan apoyándose en la información pública disponible ya sean las características del problema o el rastro de feromonas actual. La secreción de feromonas es realizada por una hormiga cuando esta construye una solución, antes de construirla o ambas, esto depende del tipo de implementación. La cantidad de feromona depositada depende de la calidad de la solución hallada. Cuando un movimiento es bueno y contribuye a la construcción de soluciones de alta calidad, varias hormigas lo realizaran con lo que el rastro de feromonas se incrementará haciéndolo más atractivo para las demás.

Un aspecto importante es que los componentes estocásticos de las reglas de decisión sobre el movimiento siguiente que realizara una hormiga, así como la evaporación de las feromona, permiten que no se presente convergencia prematura hacia óptimos locales.

Una vez que las hormigas han cumplido con su misión, construir una solución y proporcionar información, mueren es decir son borradas del sistema. De forma equivalente se pude hacer que la hormiga continué en el proceso borrándole toda su información privada una vez ha terminado una solución [21].

En el siguiente esquema se presenta el algoritmo propuesto por [17] [19] en los trabajos pioneros de colonia de hormigas, aunque algunas de sus etapas son opcionales este sigue siendo la base de los trabajos sobre el tema.

Procedimiento meta heurística de colonia de hormigas

Mientras [el criterio de terminación no este satisfecho]

Programación de actividades

Generación de hormigas y actividades

Evaporación de feromonas

Acciones de ajuste secundario [opcional]

Fin de la programación de actividades

Fin mientras

Fin procedimiento

Procedimiento de generación de hormigas y actividades

Mientras [recursos disponibles]

Programar la creación de una nueva hormiga

Nueva actividad de la hormiga

Fin Mientras

Fin procedimiento

Procedimiento nueva hormiga activa [ciclo de vida de la hormiga]

Inicializar la hormiga

μ = Ajustar la memoria de la hormiga

Mientras [estado actual $\neq$ estado objetivo]

A = lectura actual y secuencia

P = calculo de las probabilidades de transición

Estado siguiente = aplicar la regla de decisión

Movimiento al nuevo estado

Si [actualización en línea de las feromonas]

Segregación de feromona en el arco visitado

Actualización de la secuencia

Fin Si

$\mu =$ Actualización del estado interno

Fin Mientras

Si [actualización tardía del nivel de feromona]

Evaluar la solución

Depositar feromonas en todos los arcos visitados

Actualizar la secuencia

Fin Si

Muerte de la hormiga

Fin del procedimiento.

El estado objetivo se refiere a la terminación de las soluciones. Los procedimientos de actualización tardía del nivel de feromonas y actualización paso a paso son mutuamente excluyentes y muy rara vez se realizan en forma simultánea [17].

3.3 APLICACIONES DEL ALGORITMO DE COLONIA DE HORMIGAS

Aunque el algoritmo es relativamente nuevo, en la actualidad se pueden encontrar varias aplicaciones, tanto en situaciones estáticas como en situaciones dinámicas. En el siguiente cuadro se da un resumen de ellas.

Cuadro 1. Aplicaciones de Colonia de Hormigas.

PROBLEMA	AÑO	REFERENCIAS
Agente viajero	1991	[20][21] [32][56]
Asignación cuadrática	1994, 1997 y 1998	[9] [29][46]
Programación en talleres	1994	[8]
Ruteo de vehículos	1999	[30]
Ordenamiento secuencial	1997	[31]
Coloreado de grafos	1997	[12]
Redes	1997 y 1998	[18]

3.4 SEMEJANZAS CON OTRAS METAHEURÍSTICAS

Los algoritmos de colonia de hormigas muestran semejanzas con algunas técnicas de optimización, algoritmos de aprendizaje y de simulación tales como búsqueda heurística en grafos, simulación de Monte Carlo, redes neuronales y computación evolutiva [17].

3.4.1 Redes Neuronales. Las colonias de hormigas están conformadas por numerosas unidades locales que interactúan concurrentemente por lo que pueden ser vistas como sistemas de conexiones [25], cuyo más famoso ejemplo son las redes neuronales [4][39]. Desde un punto de vista estructural, el paralelo entre la metaheurística de colonia de hormigas y una red neuronal es claro al considerar un estado j visitado por las hormigas como una neurona j y la estructura del vecindario correspondiente del estado j con el conjunto de lazos sinápticos que salen de la neurona j.

Las hormigas por si mismas pueden ser consideradas como señales de entrada concurrentes que se propagan a través de la red neuronal [17] y modifican la fuerza de las conexiones sinápticas entre neuronas. Las señales (hormigas) se propagan localmente por medio de una función de la transferencia estocástica y cuanto mas se utiliza una sinapsis más se refuerza la conexión entre dos neuronas terminales. La regla de aprendizaje sináptico en los sistemas de optimización con colonia de hormigas puede considerarse como una regla a posteriori: las señales relacionadas con buenas soluciones encontradas por las hormigas refuerzan las conexiones sinápticas más que las relacionadas con soluciones malas.

3.4.2 Búsqueda Heurística En Grafos. En los procedimientos de optimización con colonia de hormigas, cada hormiga realiza una búsqueda heurística en un grafo [17]: las hormigas toman decisiones probabilísticas para incorporar el siguiente componente de una solución y dicho movimiento es realizado después de una evaluación heurística que favorece los componentes que parecen más prometedores. Dorigo [17] hace notar que esta evaluación es diferente a la que se realiza por ejemplo en recocido simulado, ya que en este

se define un criterio de aceptación y solamente se ejecutan movimientos generados aleatoriamente que satisfacen dicho criterio y las búsquedas son realizadas generalmente en el espacio de las soluciones.

3.4.3 Simulación De Monte Carlo. Los algoritmos de colonia de hormigas se pueden interpretar como replicas paralelas de los sistemas de Monte Carlo [17]. Los sistemas de Monte Carlo [55] son simulaciones estocásticas, es decir, técnicas que realizan repetidamente experimentos bajo un modelo que describe el sistema en consideración. Los resultados son usados para obtener conocimiento estadístico del comportamiento del sistema y realizar estimaciones de las variables de interés para el investigador. Los resultados también se pueden utilizar para realizar nuevas simulaciones en un nuevo espacio de estudio.

Análogamente, en algoritmos de colonia de hormigas, estas exploran el espacio de la soluciones del problema en varias ocasiones, aplicando una política estocástica de la decisión hasta que construyen una solución factible. Cada hormiga se puede considerar como "un experimento" que tiene en consideración los resultados obtenidos en replicas anteriores y que permite modificar el conocimiento estadístico del problema para la realización de nuevas replicas.

3.4.5 Computación Evolutiva. Existen algunas semejanzas generales entre la metaheurística de colonia de hormigas y la computación evolutiva [27]. Ambos enfoques utilizan una población de individuos que representan soluciones del problema, y en los dos procedimientos el conocimiento recogido por la población sobre el problema se utiliza generar estocásticamente una nueva población de individuos. La principal diferencia radica en que en EC todo el conocimiento sobre el problema se almacena en la población actual, mientras que en colonia de hormigas la memoria del desempeño pasado es mantenida bajo la forma del rastro de feromonas. Existe un algoritmo del tipo EC, llamado población basada en el aprendizaje incremental (PBIL), que es muy similar a ACO, para mayor detalle ver [1].

4. IMPLEMENTACION DEL ALGORITMO ACO AJUSTADO CON LAS MEJORAS PROPUESTAS

El problema que se pretende solucionar con el algoritmo diseñado es el de secuenciar n trabajos en una máquina, como se menciono anteriormente este es un problema de tipo NP - Hard que ha sido atacado por medio de varios procedimientos tanto heurísticos como de ramificación y acotación. Para un problema de tamaño n, existen n! alternativas de ordenar los trabajos, es decir que para una situación en la que se deban programar tan solo 25 trabajos se tienen $1.55 * 10^{25}$ alternativas de ordenamiento, esta cantidad es mayor al numero de Avogadro (el numero de átomos que hay en una mol de un elemento) ó que la edad estimada del universo medida en segundos.

Como se menciona en el título de este trabajo, para solucionar el problema descrito se implemento la metaheurística de colonia de hormigas, la cuál se adaptó al problema considerado y se le propusieron mejoras con el fin de obtener mejores soluciones. Se presenta una completa y clara descripción del algoritmo que se implementó junto con los cambios que se realizaron con el fin de mejorar su desempeño.

4.1 ALGORITMO DE COLONIA DE HORMIGAS APLICADO AL PROBLEMA DE PROGRAMACIÓN DE UNA MÁQUINA

A grandes rasgos se puede decir que el algoritmo ACO consiste en una colonia de ***m*** hormigas creando secuencias de n trabajos obedeciendo ciertas reglas. Se determina un número máximo de iteraciones (***tmàx***) y en cada iteración cada una de las m hormigas construye una secuencia de n trabajos. Al final se escoge como solución la mejor secuencia de las ***m* tmàx*** construidas durante todo el proceso.

En cuanto a nomenclatura, (i,j) representa el trabajo i seguido del trabajo j (j e i pertenecen al conjunto de los n trabajos a secuenciar). S^+ simboliza la mejor secuencia encontrada

hasta el momento, es decir la secuencia de las halladas que tiene el menor valor de la función objetivo. L_+ es el valor de la función objetivo correspondiente a la secuencia S_+. Debido a que en las diferentes iteraciones pueden obtenerse soluciones que no mejoran la mejor solución actual acumulada, de las m soluciones construidas en una iteración, la mejor de ellas se representa como S_* y su correspondiente valor L_*.

Como el algoritmo trata de imitar el comportamiento de una colonia de hormigas, usa términos que pueden estar relacionados con estas, tales como la intensidad del rastro de feromonas el cual para el contexto de este problema se simboliza mediante $\tau_{(i,j)}$ (nivel del rastro de feromonas para los trabajos adyacentes i,j) **,** el cual varía en cada iteración. Con la intensidad del rastro se escoge entre los candidatos el siguiente trabajo que se adicionará a la secuencia que se esta formando para al final tener una secuencia de programación completa. Existen otros parámetros que se usan y se explicaran cuando sea conveniente.

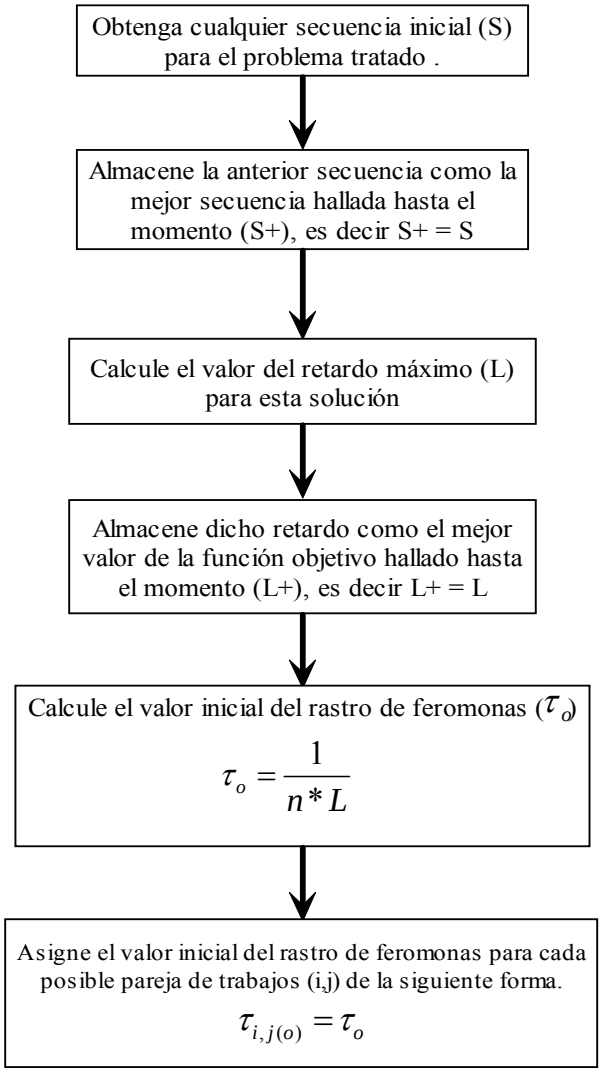
En la naturaleza las hormigas construyen caminos basándose en el rastro de feromonas dejado por sus antecesoras, por esto la extensión de ACO al problema del agente viajero no es difícil de aceptar. En cuanto a la programación de la producción, algoritmo ACO se puede visualizar mejor considerando la representación gráfica del problema descrita en el capítulo 2, en este caso también se trata de construir una buena ruta que vaya del nodo inicial al nodo terminal. Por ello es que dos trabajos consecutivos pueden considerarse como parte de un camino creado por una hormiga y al igual que en la naturaleza, si muchas hormigas siguen el mismo camino ó escogen el mismo par de trabajos consecutivos incentivarán a las que vienen atrás a escoger el mismo camino.

El algoritmo se puede dividir en tres grandes bloques; uno de inicialización, otro de construcción de soluciones y uno de evaluación de soluciones. Se describen a continuación.

- **Inicialización**: En esta fase se realizan los ajustes para que las hormigas puedan iniciar la construcción de soluciones. Se da un valor inicial al rastro de feromonas y se asignan

los trabajos iniciales a las secuencias que cada una de las m hormigas empiezan a elaborar.

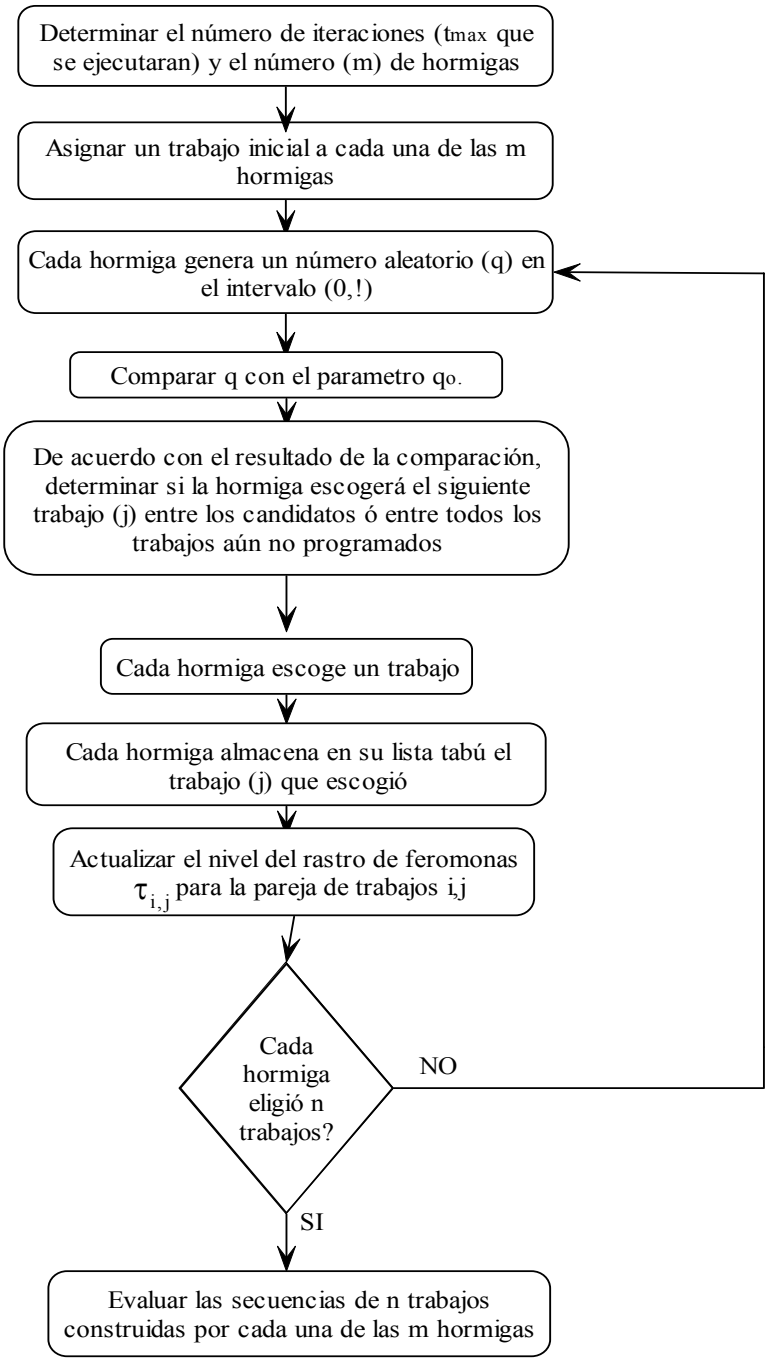
Figura 7. Diagrama del bloque de inicialización del algoritmo ACO



- **Construcción de soluciones por cada una de las hormigas**: En esta etapa las hormigas realizan la búsqueda de soluciones al problema. En cada una de las iteraciones, las hormigas van construyendo simultáneamente secuencias de trabajos basándose en reglas de decisión para la adición del trabajo siguiente. Estas reglas son función del rastro de feromonas y de los factores de visibilidad. También se realizan los

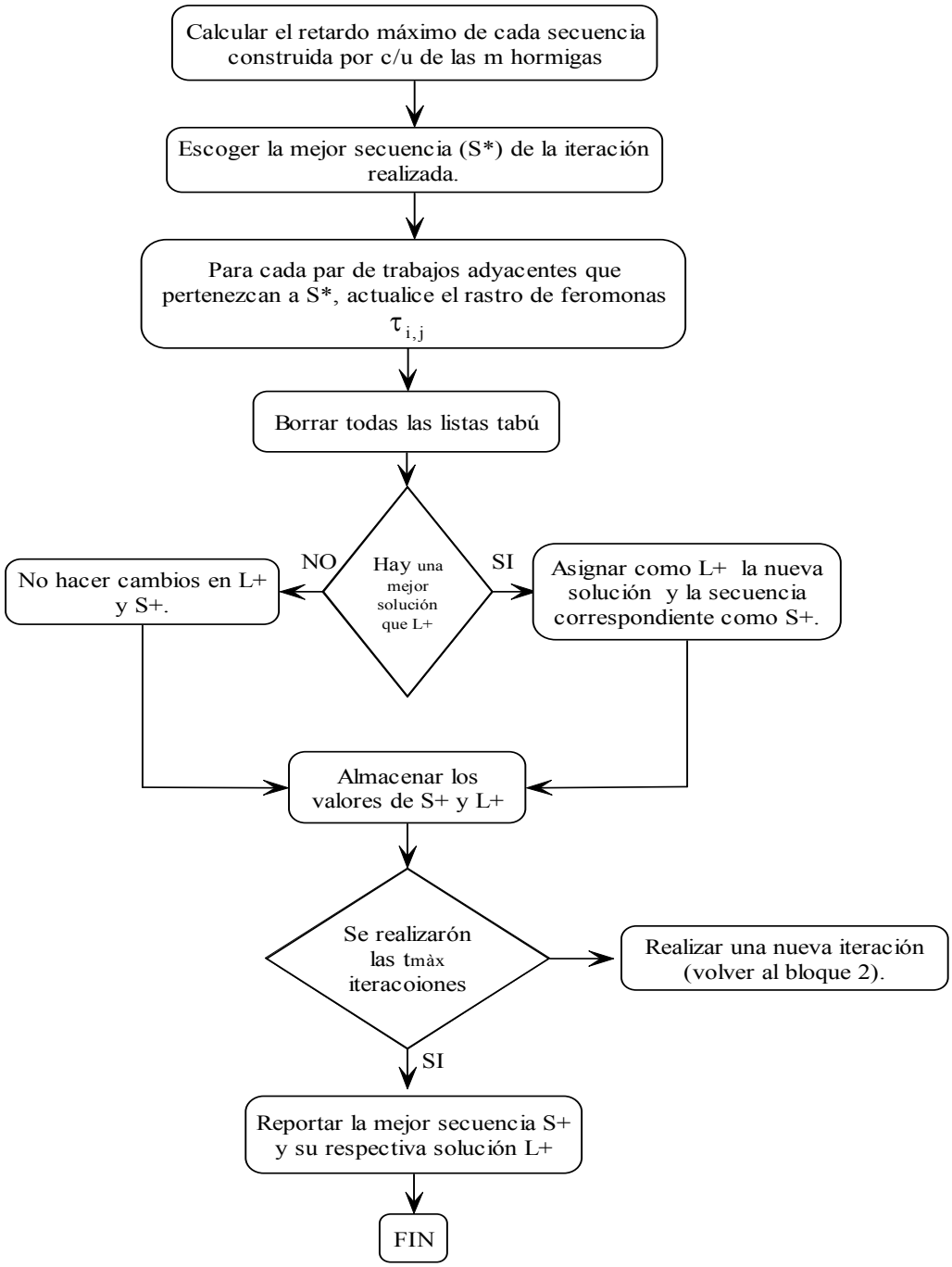
ajustes correspondientes al nivel del rastro de feromonas ($\tau_{i,j}$) con el fin de no caer en óptimos locales.

Figura 8. Diagrama del bloque de construcción de soluciones



- **Evaluación de las soluciones construidas**: Se evalúa cada una de las soluciones encontradas con el fin de almacenar la mejor de todo el procedimiento. A la mejor solución de cada iteración, se le hace un incremento en el valor del rastro de feromona.

Figura 9. Diagrama del bloque de evaluación de soluciones.



A continuación se presenta una descripción más detallada del algoritmo implementado, de las acciones realizadas en cada paso y de la nomenclatura utilizada. El código en C para este procedimiento se encuentra en el Anexo 1.

En el paso de inicialización del rastro de feromonas (bloque 1), cada par de trabajos (i,j) es inicializado con una pequeña cantidad de feromonas, es decir se hace $\tau_{i,j(0)} = \tau_o$. Cada uno de estos pares de trabajos son candidatos a ser los dos primeros trabajos en la secuencia. El rastro de feromonas es inicializado con un valor $\tau_o = (1/(n*L_{cs})$ donde n es el número de trabajos y L_{cs} es el valor de una solución encontrada de cualquier forma. Dado que no es importante como se construye la secuencia con la que se obtiene el valor L_{cs}, en el algoritmo implementado ésta corresponde al ordenamiento de los trabajos de acuerdo a su número.

En cuanto al número de veces que se ejecuta el ciclo principal (en el que cada una de las m hormigas construye una secuencia de n trabajos durante el número de iteraciones fijado), se determinó detener el algoritmo cuando pasan 100 iteraciones sin que haya mejoramiento en el valor de la función objetivo. Alternativamente se puede fijar un número máximo de iteraciones $t_{máx}$.

Según se reporta en [2][23] la asignación del trabajo inicial se realiza de la siguiente forma: cuando se empieza la construcción de una secuencia de trabajos, el último agregado en la iteración anterior se toma como el trabajo inicial de la iteración actual, esto se hace con el fin de mantener la continuidad en el rastro de feromonas.

Con el fin de disminuir el número de búsquedas que realiza una hormiga en cada iteración, se elabora una lista de candidatos. Para ello se busca una forma de ordenar los trabajos (en este estudio se ordenaron por tiempo de liberación) y se toman como candidatos un determinado número de los primeros trabajos que no estén en la lista Tabú. El tamaño de la lista de candidatos se fijo en 0.3*n según los reportes de [2][23]. Por ejemplo si se fija el tamaño de la lista de candidatos en 0.3*n y se tienen 100 trabajos, estos se ordenan por

algún criterio y se toman como candidatos los primeros 30 que no pertenecientes a la lista tabú.

En el capitulo dedicado a la metaheurística de colonia de hormigas se menciono que las hormigas artificiales se pueden dotar de atributos que no tienen las naturales. Mientras que las hormigas naturales realizan su búsqueda solo guiándose por el rastro de feromonas, las artificiales pueden ser provistas de factores de visibilidad (representados mediante el simbolo *fv*), con los que la búsqueda no se realiza a ciegas.

Cada hormiga debe construir una secuencia de trabajos de la siguiente forma (bloque 2): para una secuencia parcial de trabajos, una hormiga escoge el siguiente trabajo a añadir teniendo en cuenta la intensidad del rastro de feromonas $\tau_{i,j(t)}$ y los factores de visibilidad. Para ello se fija el valor del parámetro q_o, en el intervalo (0, 1), y en cada iteración se genera un número aleatorio q en el mismo intervalo. Si $q_o \geq q$, el siguiente trabajo de la secuencia es escogido considerando solo el grupo de candidatos y mediante la siguiente regla:

$$\arg\max_{l \notin Tabu_k} \left\{ \left[\tau_{il}(t)\right]^{\alpha} \left[1/\ fv^{1}_{il}\right]^{\beta} \left[1/\ fv^{2}_{il}\right]^{\varphi} \right\} \quad (8)$$

El subíndice 1 indica que el trabajo pertenece a la lista de candidatos mientras que α, β y φ representan los exponentes del rastro de feromonas y de los factores de visibilidad. En [2] [23] estos exponentes toman valores entre 1 y 4.

Si $q_o < q$, entonces el siguiente trabajo en la secuencia se escoge considerando la totalidad de los trabajos que no han sido programados. El escogido es aquel que tenga la mayor probabilidad de acuerdo a:

$$p^{k}_{ij}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \left[1/\ fv^{1}_{ij}\right]^{\beta} \left[1/\ fv^{2}_{ij}\right]^{\varphi}}{\sum_{j \notin Tabuu_k} \left[\tau_{ij}(t)\right]^{\alpha} \left[1/\ fv^{1}_{ij}\right]^{\beta} \left[1/\ fv^{2}_{ij}\right]^{\varphi}} \quad (9)$$

Imitando otro aspecto de la naturaleza, la evaporación de la feromona, se realiza una actualización local del rastro de feromonas. Con esta actualización se pretende evitar la convergencia prematura. Esta actualización realiza una disminución del nivel de feromonas

para un par de trabajos adyacentes $\tau_{i,j}$, de esta manera se desanima a la siguiente hormiga elegir el mismo par de trabajos consecutivos durante la misma iteración. La disminución es pequeña pero suficiente para mantener baja la atracción hacia el par de trabajos i,j. La actualización se realiza de acuerdo con la siguiente expresión.

$$\tau_{ij(t)} = \rho_l \, \tau_{ij(t)} + (1 - \rho_l) \, \Delta \tau_{ij} \quad \text{con } \Delta \tau_{ij} = \tau_o \qquad (10)$$

El parámetro ρ_l es una medida del nivel de disminución ó "evaporación" de $\tau_{ij[t)}$ y su valor esta entre 0 y 1.

Cuando todas las hormigas han terminado de construir su solución se debe realizar una evaluación de soluciones (bloque 3). Se evalúan las m secuencias halladas durante la iteración y si el valor de la función objetivo en alguna de ellas es mejor que el que se tiene hasta el momento, entonces se realiza la actualización del mejor valor encontrado para la función objetivo L_+ y para la secuencia correspondiente S_+. Después de evaluadas todas las soluciones, se hace una actualización global de la intensidad del rastro de feromonas. Al contrario de la actualización local, en este paso se incrementa el rastro de feromonas para cada par de trabajos pertenecientes a la mejor secuencia de las m construidas durante la iteración. La actualización se realiza de acuerdo a la calidad de dicha secuencia, para cada par de trabajos adyacentes i,j que pertenezcan a esta se incrementa el valor de $\tau_{i,j}$.

$$\tau_{i,j(t+1)} = \rho_g \, \tau_{i,j(t)} + (1 - \rho_g) \, \Delta \tau_{i,j(t)} \text{ donde } \Delta \tau_{i,j(t)} = 1 / L_* \qquad (11)$$

La magnitud del incremento del rastro de feromonas esta dado por ρ_g, este parámetro toma valores en el intervalo [0,1]. L_* es el valor de la función objetivo correspondiente a la mejor secuencia de la iteración, dicha secuencia se representa por S*. En los trabajos de [2][23] q_0, ρ_l y ρ_g toman los siguientes valores; $q_0 = \rho_l = \rho_g = 0.9$ de acuerdo a las recomendaciones de [17], sin explicar el porque de estos ni reportar algún intento de probar otros valores.

En el siguiente esquema se resume en términos generales el algoritmo implementado, las mejoras propuestas presentan en la siguiente sección.

Inicialización del rastro de feromonas.
Para cada par de trabajos (i,j)
Iniciar $\tau_{i,j(0)} = \tau_o$
Fin
Sea S_+ la mejor secuencia hallada hasta el momento y L_+ el valor de la función objetivo.
Ciclo principal.
Para t = 1 hasta t = tmàx
Asignación del trabajo inicial.
Para k = 1 hasta k = m, haga
Asignar el trabajo inicial para la hormiga k y almacenarlo en la lista $Tabu_k$
Fin
Construcción de secuencias por cada hormiga.
Sea S_* la mejor secuencia encontrada en la iteración actual y L_* su valor.
Para i = 1 a n
Para k =1 a m
Escoger el siguiente trabajo j (j no pertenece a Tabú k)
si $q \leq q_o$, se escoge entre los trabajos de la lista de candidatos.

$$j = \arg\max_{l \notin Tabu_k} \left\{ \left[\tau_{il}(t)\right]^{\alpha} \left[1/\, fv^{1}_{il}\right]^{\beta} \left[1/\, fv^{2}_{il}\right]^{\varphi} \right\}$$

si $q > q_o$ entonces se escoge entre todos los trabajos aún no programados (se escoge el que obtenga el mayor valor de la siguiente expresión)

$$p^{k}_{ij}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha} \left[1/\, fv^{1}_{ij}\right]^{\beta} \left[1/\, fv^{2}_{ij}\right]^{\varphi}}{\sum_{j \notin Tabuu_k} \left[\tau_{ij}(t)\right]^{\alpha} \left[1/\, fv^{1}_{ij}\right]^{\beta} \left[1/\, fv^{2}_{ij}\right]^{\varphi}}$$

Almacenar esta información en $Tabu_k$
Realizar la actualización local del nivel del rastro de feromonas para el par (i,j) de trabajos escogidos:
$\tau_{ij[t)} = \rho_t\, \tau_{ij(t)} + (1 - \rho_t)\, \Delta\, \tau_{ij}$ donde $\Delta\, \tau_{ij} = \tau_o$
Fin
Fin
Evaluación de soluciones.
Para k =1 a m
Calcular $L_k(t)$ para la secuencia $S_k(t)$ construida por la hormiga k.
Si se encuentra una solución mejor que la actual entonces actualizar S_+ y L_+
Fin.
Actualización global del rastro de feromonas.
Para cada par de trabajos adyacentes (i,j) que pertenecen a S_*
Actualizar el rastro de feromonas de acuerdo a:
$\tau_{i,j(t+1)} = \rho_g\, \tau_{i,j(\,t)} + (1 - \rho_s)\, \Delta\, \tau_{i,j\,(t)}$ donde $\Delta\, \tau_{i,j(\,t)} = 1/\, L_*$
Fin.
Limpiar todas las listas $Tabu_k$

Fin

Al final de este procedimiento se obtiene como solución el valor L^+ correspondiente a la mejor secuencia S^+.

4.2 MEJORAS PROPUESTAS

En el presente trabajo se proponen algunas modificaciones a la forma convencional de realizar algunas de las etapas de la metaheurística. Las modificaciones se dividen en cuatro frentes; la asignación del trabajo inicial de cada hormiga en cada una de las iteraciones, el esquema con el que se le permite a cada hormiga visitar nuevas soluciones, los factores de visibilidad y la forma en que se asignan los valores de los parámetros utilizados en el algoritmo.

4.2.1 Asignación del trabajo inicial. Con el fin de mantener la continuidad en el rastro de feromonas [23] propone asignar en cada iteración como primer trabajo de las secuencias al último trabajo de la secuencia construida por la misma hormiga en la iteración anterior.

En la presente propuesta se modifica esta regla y proponen dos esquemas de asignación. En el primero se realiza la asignación de forma totalmente aleatoria, es decir que cualquier trabajo puede ser colocado en la primera posición de la secuencia. Esto con el fin de permitir una amplia exploración de nuevas soluciones por parte de las hormigas, a este esquema se le denomino ***asignación aleatoria***.

En el segunda forma de asignación se toma como trabajo inicial cualquiera de los trabajos pertenecientes al primer 30 por ciento de las posiciones de la secuencia construida en la iteración anterior, a este esquema se le denomino ***asignación selectiva***. La justificación es que se considera que los trabajos que están en las últimas posiciones en una secuencia determinada no tienen, de acuerdo con la filosofía del algoritmo, los atributos necesarios para ser programados prioritariamente.

4.2.2 Elección de los trabajos. En cada iteración las hormigas deben ir escogiendo cual de los trabajos aún no programados añade a la secuencia que esta construyendo. Ya se menciono que para ello se fija el valor de un parámetro q_o, se genera un número aleatorio q y de acuerdo a la comparación entre estos dos números se realiza la elección entre los candidatos ó entre todos los candidatos.

En el presente trabajo se propone que cuando se deba escoger entre todos los trabajos no se tenga en cuenta a los que se encuentran en la lista de candidatos. El fundamento de esta propuesta es que se considera que si al escoger entre todos los trabajos los candidatos están incluidos, muy probablemente la hormiga escogerá a uno de estos. Con la propuesta hecha se pretende permitirle a las hormigas explorar nuevas soluciones.

4.2.3 Factores de visibilidad. En este trabajo se proponen los siguientes factores de visibilidad:

$$fv^1_{ij} = d_j \qquad (12)$$

$$fv_{ij} = \max(r_j, ts) + p_j \qquad (13)$$

$$fv_{ij} = \max(r_j, ts) \qquad (14)$$

$$fv_{ij} = \max(r_j, ts) + p_j - d_j \qquad (15)$$

ts es el tiempo en el que se terminara de procesar el último trabajo de la secuencia parcial (el tiempo en que la hormiga debe tomar la decisión) y el subíndice j corresponde al trabajo que se esta considerado para añadir a la secuencia dado que el último en ésta es el trabajo i.

Con el primer factor se busca dar prioridad a los trabajos que deben ser entregados antes. Se escogió este factor ya que el tiempo de entrega es uno de los parámetros con que se calcula el retardo de un trabajo, este factor se ha empleado en otros trabajos [2].

Con el segundo se pretende considerar los trabajos que ya están disponibles para ser programados y que liberarán la máquina en el menor tiempo, con ello se ayuda a que la máquina no este desocupada innecesariamente. No se encontraron registros bibliográficos del uso de este factor.

El tercer factor es una variante del anterior con el que se busca dar prioridad a los trabajos que ya están disponibles para ser procesados sin tener en cuenta cuando liberaran la máquina.

El último factor es el cálculo del retardo para el trabajo que se esta considerando añadir a la secuencia, se debe tener en cuenta que el retardo puede ser negativo por lo tanto se hacen los arreglos para no castigar los trabajos que presenten retardos negativos.

En el algoritmo implementado siempre se trabajo con dos factores de visibilidad, como se considera que el tiempo de entrega es un factor importante a la hora de dar prioridad a los trabajos, se probo el uso de cada uno de los factores correspondientes a las ecuaciones (13) (14) y (15) como el acompañante del factor de visibilidad dado por el tiempo de entrega.

4.2.4 Diseño De Experimentos. Como se ha mencionado, el diseño de experimentos se realizó para determinar los valores de los parámetros $q_{0,}$ ρ_l , ρ_g, α, β y φ que se usaron en el algoritmo debido a que en [2][23] se utilizan valores recomendados por [20] pero no se prueban otros valores.

Se realizo un diseño factorial fraccionado 3^{6-2} , debido a que es de resolución III (es decir que no confunde los efectos principales entre si), permite estudiar los factores en el rango deseado y que reduce de manera considerable el número de corridas que se deben realizar.

Los niveles a los que se estudio cada uno de los factores se presentan en la siguiente tabla. Como variable de respuesta se tomo la solución obtenida al correr un problema de tamaño 50, Prob 509.

Tabla 1. Valores de los niveles de los factores estudiados

Factor	Nivel bajo	Nivel medio	Nivel alto
q_0	0.1	0.5	0.9
ρ_l	0.1	0.5	0.9
ρ_g	0.1	0.5	0.9
α	1	3	5
β	1	3	5
φ	1	3	5

4.2.4.1 Diseño Generado. Los factores $q_0, \rho_l, \rho_g, \alpha, \beta$ y φ se representan con las letras A, B, C, D, E y F. Los componentes de interacción escogidos para generar el diseño son ABCDEF y BC^2D, con lo que se tiene que sus interacciones generalizadas son:

$$(ABCDEF)(BC^2D) = AB^2D^2EF$$

$$(ABCDEF)(BC^2D)^2 = (ABCDEF)(B^2CD^2) = AC^2EF$$

Con lo que se tiene:

$$I = ABCDEF = BC^2D = AB^2D^2EF = AC^2EF$$

$$I^2 = A^2B^2C^2D^2E^2F^2 = B^2CD^2 = A^2BDE^2F^2 = A^2CE^2F$$

De acuerdo con lo anterior se tienen los contrastes definición:

$$L_1 = X_A + X_B + X_C + X_D + X_E + X_F = u \pmod 3 \quad u = 0, 1, 2$$

$$L_2 = X_B + 2X_C + X_D = h \pmod 3 \quad h = 0, 1, 2$$

Las X representan el valor que toma cada factor en las corridas del diseño factorial completo, estos valores son 0, 1 y 2 para los niveles bajo, medio y alto respectivamente. Se tomo el bloque que contiene las corridas para las cuales $L_1 = L_2 = 0$.

La estructura de los alias para este diseño es la siguiente:

A $AB^2C^2D^2E^2F^2$ ABC^2D $ABDE^2F^2$ ACE^2F^2 $AB^2C^2D^2E^2F^2$ AB^2CD^2 BDE^2F^2 CE^2F
B AB^2CDEF BCD^2 AD^2EF ABC^2EF $ACDEF$ CD^2 ABD^2EF^2 AB^2C^2EF
C ABC^2DEF BD $ABCD^2EF$ AEF $ABDEF$ BCD AB^2CD^2EF $ACEF^2$
D $ABCD^2EF$ BC^2D^2 AB^2EF AC^2DEF $ABCEF$ BC^2 AB^2DEF AC^2DEF^2
E $ABCDE^2F$ BC^2DE $AB^2D^2E^2F$ AC^2E^2F $ABCDF$ BC^2E^2D AB^2D^2F AC^2F^2
F $ABCDEF^2$ BC^2DF $AB^2D^2EF^2$ AC^2EF^2 $ABCDE$ BC^2DF^2 AB^2D^2E AC^2EF^2

Es importante anotar que debido a que efectos principales están confundidos con interacciones de dos factores, este diseño es de resolución III.

4.3 GENERACIÓN DE LOS PROBLEMAS

Con el algoritmo presentado se solucionaron problemas de programación de 50, 100 y 200 trabajos. Con los problemas generados se puede analizar la influencia de varios aspectos relativos a los problemas en el desempeño del algoritmo. Dentro de estos aspectos se cuentan el tamaño del problema y el valor de los parámetros con que se generan los tiempos de entrega y de liberación de los trabajos

4.3.1 Tiempos de proceso. Para cada trabajo se genero un tiempo de proceso p_j a partir de una distribución uniforme (1, 100).

4.3.2 Tiempos de entrega. Los tiempos d_j se obtienen a partir de una distribución uniforme en el siguiente intervalo, de acuerdo con [2]:

$$\sum_{i=1}^{n} p_j * (1- TF - \frac{RDD}{2}),\ \sum_{i=1}^{n} p_j * (1- TF + \frac{RDD}{2}) \quad (16)$$

Donde RDD y TF son conocidos como los factores de tardanza y rango relativo de tiempos de entrega respectivamente, estos factores permiten variar la complejidad (dureza) de los problemas generados [2]. Ambos toman valores del intervalo [0, 1].

4.3.3 Tiempos de liberación. No se encontraron reportes para generar los tiempos de liberación r_j, por lo tanto se buscó generarlos de una manera que incluyera una amplia variedad de situaciones. Por ello se considero conveniente generar los tiempos de liberación a partir de una distribución uniforme

$$0,\ 0.9*\sum_{i=1}^{n} p_j * (1- TF + \frac{RDD}{2}) \quad (17)$$

4.3.4 Nomenclatura de los problemas. En la nomenclatura de los problemas los primeros dígitos indican el tamaño del problema y los restantes el número de este. Por ejemplo Prob 501 se refiere al primer problema de tamaño 50 mientras que Prob 10010 se refiere al décimo problema de tamaño 100.

Se generaron 20 problemas de 50 trabajos, 20 de tamaño 100 y 20 de tamaño 200, distribuidos de la siguiente forma en cuanto al valor de los parámetros TF y RDD.

En los problemas 501 a 505, 1001 a 1005 y 2001 a 2005, TF = RDD = 0.2
En los problemas 506 a 5010, 1006 a 10010 y 2006 a 20010, TF = RDD = 0.4
En los problemas 5011 a 505, 1001 a 10015 y 20011 a 20015, TF = RDD = 0.6
En los problemas 5016 a 5020, 10016 a 10020 y 20016 a 20020, TF = RDD = 0.8

Con los problemas generados se puede analizar si aspectos tales como el tamaño del problema y la forma en que se generaron influyen en el desempeño del algoritmo.

5. ANÁLISIS DE RESULTADOS

Para la obtención de resultados la ejecución del programa de colonia de hormigas se detiene cuando pasan 100 iteraciones sin que haya una mejora en la función objetivo. Se uso un computador personal con procesador Athlon de 1.7 GHz y 256 MB de RAM.

5.1 DISEÑO DE EXPERIMENTOS.

A los resultados obtenidos del diseño de experimentos se les realizo un análisis de varianza, el valor P (P value) pára cada factor estudiado se presenta en la siguiente tabla.

Tabla 2. Valor P para cada factor estudiado

Factor	Valor P
q_0	0.000
ρ_l	0.227
ρ_g	0.270
α	0.259
β	0.000
φ	0.000

De acuerdo con los resultados (en el anexo 1 se muestran las salidas del análisis de varianza realizado con Minitab), se observa que los factores más significativos son los exponentes de los factores de visibilidad (β, φ) y el parámetro q_0. El factor menos significativo es el nivel de actualización global del rastro de feromona ρ_g.

Con los datos obtenidos también se realizó un análisis de regresión con el fin de obtener una superficie de respuesta que incluyera los componentes lineal y cuadrático de los

factores estudiados. Con la superficie obtenida se planteo el siguiente problema de programación no lineal que se soluciono mediante el uso de Gams.

Minimizar $848 - 423\,q_0 - 138\,\rho_l - 42.4\,\rho_g + 12{,}9\,\alpha + 35.63\,\beta - 9.4\,\varphi$

$$+ 252\,q_0^2 + 147\,\rho_l^2 + 9.5\,\rho_g^2 - 1.05\,\alpha^2 - 2.54\,\beta^2 - 2.02\,\varphi^2$$

sujeto a:

$0 \le q_0 \le 1$

$0 \le \rho_l \le 1$

$0 \le \rho_g \le 1$

$\alpha, \beta, \varphi \ge 1$

Los resultados obtenidos son:

$q_0 = 0.839,\ \rho_l = 0.469,\ \rho_g = 0.617,\ \alpha = 1.0,\ \beta = 4.86,\ \varphi = 5.$

Estos valores se utilizaron en la ejecución del programa codificado en lenguaje C.

5.2 COMPARACIÓN ENTRE LAS DIFERENTES MODIFICACIONES REALIZADAS EN ACO.

Con el objetivo de realizar un análisis del desempeño del algoritmo propuesto con sus respectivas modificaciones, se corrieron con cada uno de ellos una serie de problemas de diferentes tamaños. La calidad de la respuesta siempre se midió de la misma forma; el porcentaje de mejora con respecto a la solución obtenida mediante la regla de despacho EDD.

5.2.1 Elección de los trabajos. Se realizaron corridas para comparar la propuesta de no incluir los trabajos de la lista de candidatos cuando se escoge entre todos los trabajos contra sí incluirlos. El algoritmo en el que no se excluyen los candidatos se denomina ACO 1 y en el que si se excluyen ACO 2.

5.2.1.1 Resultados. En la tabla 2 se muestran los resultados obtenidos en las corridas de 10 problemas de tamaño 100 y el promedio del porcentaje de mejora con respecto a la solución obtenida mediante la regla de despacho de menor tiempo de entrega primero, la cual se simboliza como EDD.

Tabla 3. Mejora porcentual con respecto a EDD para las dos formas propuestas de elección de los trabajos

Problema	**ACO 1**	**ACO 2**
Prob1001	53.18 %	53.40 %
Prob1002	55.24 %	61.04 %
Prob1003	54.97 %	57.41 %
Prob1004	56.82 %	56.33 %
Prol1005	44.60 %	56.10 %
Prob1006	47.92 %	48.76 %
Prob1007	44.18 %	49.98 %
Prob1008	43.10 %	44.61 %
Prob1009	35.48 %	36.50 %
Prob10010	44.09 %	42.90 %
Promedio	47.95 %	50.70 %

ACO 1 no se excluyen los candidatos, en ACO 2 si.

5.2.1.2 Discusión. Se observa que en promedio y para cada uno de los problemas, salvo el 10010, el algoritmo ACO 2 arroja los mejores resultados. De acuerdo con lo anterior se puede decir que se obtienen mejores resultados cuando al hacer la elección entre todos los trabajos no programados, no se considera a los que están en la lista de candidatos. Se puede interpretar esto de otra forma; el algoritmo que reporta los mejores resultados es aquel en el que en ocasiones se escoge un trabajo de acuerdo a una lista de candidatos y en otras ocasiones se escoge entre los que no son candidatos.

Los resultados favorables a ACO 2 pueden ser debidos a que con esta forma de elegir entre los trabajos se permite una mayor exploración de nuevas soluciones. Si se permite que los trabajos de la lista de candidatos sean "elegibles" siempre, puede pasar que se caiga en óptimos locales con facilidad.

5.2.2 Asignación del trabajo inicial. Se probaron las dos formas de asignación del trabajo inicial propuestas, asignación aleatoria y asignación selectiva) y la asignación hecha de forma convencional.

5.2.2.1 Resultados. En ACO 2 se usa la ***asignación selectiva***, en ACO 3 se hace la asignación de acuerdo con [23] y en ACO 4 se realiza la ***selección aleatoria***.

Tabla 4. Mejora porcentual con respecto a EDD para las diferentes formas de asignación del trabajo inicial.

Problema	**ACO 2**	**ACO 3**	**ACO 4**
Prob1001	53.40 %	52.70 %	72.18 %
Prob1002	61.04 %	53.03 %	57.38 %
Prob1003	57.41 %	55.35 %	74.98 %
Prob1004	56.33 %	55.49 %	60.30 %
Prol1005	56.10 %	49.25 %	57.24 %
Prob1006	48.26 %	46.30 %	53.36 %
Prob1007	49.98 %	42.90 %	49.98 %
Prob1008	44.61 %	42.73 %	44.61 %
Prob1009	36.50 %	29.91 %	36.61 %
Prob10010	42.90 %	45.55 %	36.51 %
Promedio	50.65 %	46.82 %	54.31 %

5.2.2.2 Discusión. Los mejores resultados son los obtenidos con ACO 4, es decir cuando

la asignación del trabajo inicial se realiza aleatoriamente. Los resultados de más baja calidad se obtienen con el algoritmo ACO 2 en el que el último trabajo de la secuencia anterior pasa a ser el primero de la secuencia que se empezara a construir.

Los resultados de ACO 2 se deben a que bajo este esquema de asignación se permite que trabajos que no tienen atributos para estar en las primeras posiciones ocupen estas. Cuando una hormiga tiene en cuenta el rastro para un trabajo dejado por sus antecesoras y por otra parte calcula los factores de visibilidad, trata de darle prioridad a los trabajos que parecen ser los mejores y en consecuencia los que parecen no ser los mejores estarán ubicados en las ultimas posiciones. Por las razones discutidas la asignación de acuerdo a [23] aunque trata de ser muy fiel al comportamiento de las hormigas reales, no favorece el desarrollo del Algoritmo.

5.2.3 Factores de visibilidad. Los factores de visibilidad propuestos están dados por las ecuaciones (12) (13) (14) (15). A los algoritmos utilizados en esta sección se les implementaron ajustes de acuerdo a los resultados encontrados en las pruebas anteriores. La asignación del trabajo inicial se realiza de manera aleatoria y cuando se escoge entre todos los trabajos no se considera a los que están en la lista de candidatos.

5.2.3.1 Resultados. En el algoritmo ACO 4 se tienen como factores de visibilidad los correspondientes a las ecuaciones (12) y (13):

$fv^1_{ij} = d_j$ y $fv_{ij} = \max(r_j, ts) + p_j$

En ACO 5 los factores de visibilidad son:

$fv^1_{ij} = d_j$ y $fv_{ij} = \max(r_j, ts)$

Mientras que para ACO 6 se tiene:

$$fv^1_{ij} = d_j \text{ y } fv_{ij} = \max(r_j, ts) + p_j - d_j$$

Primero se realizo una fase de exploración con 10 problemas de tamaño 100 con el objeto de verificar cual de las tres propuestas es la que reporta peores resultados para

posteriormente con las dos mejores algoritmos hacer los análisis con problemas de 200 trabajos.

Tabla 5. Mejora porcentual con respecto a EDD para los algoritmos con diferentes factores de visibilidad

Problema	**ACO 4**	**ACO 5**	**ACO 6**
Prob1001	72.18 %	78.45 %	51.47 %
Prob1002	57.38 %	83.50 %	39.91 %
Prob1003	74.98 %	82.48 %	61.54 %
Prob1004	60.36 %	74.7 %	51.81 %
Prol1005	57.24 %	74.51 %	46.16 %
Prob1006	53.36 %	61.22 %	35.74 %
Prob1007	49.98 %	56.73 %	39.79 %
Prob1008	44.61 %	53.78 %	47.06 %
Prob1009	36.61 %	47.65 %	22.70 %
Prob10010	36.50 %	46.23 %	38.17 %
Promedio	54.31 %	63.82 %	43.45 %

ACO 4, ACO 5 y ACO 6 se diferencian en los factores de visibilidad

Como se puede observar, los mejores resultados problema a problema y por ende en promedio se logran con ACO 5 mientras que los peores resultados se obtienen con ACO 6 posiblemente porque los factores de visibilidad utilizados hacen que el tiempo de entrega sea muy determinante en la elección del siguiente trabajo a añadir.

Los algoritmos ACO 4 y ACO 5 por tener un mejor resultado promedio se estudiaron con problemas de tamaño 50 y 200. Los resultados se muestran a continuación.

Tabla 6. Mejora porcentual con respecto a EDD de algoritmos con diferentes factores de visibilidad aplicados a problemas de tamaño 50

Problema	**ACO 4**	**ACO 5**	**Problema**	**ACO 4**	**ACO 5**
Prob501	66.30 %	66.82 %	Prob5011	7.93 %	3.51 %
Prob502	66.00 %	66.00 %	Prob5012	24.07 %	21.88 %
Prob503	70.65 %	70.65 %	Prob5013	9.54 %	6.65 %
Prob504	84.25 %	84.04 %	Prob5014	12.25 %	6.47 %
Prob505	66.82 %	77.30 %	Prob5015	2.44 %	1.45 %
Prob506	57.00 %	48.92 %	Prob5016	0.00 %	0.00 %
Prob507	74.96 %	75.52 %	Prob5017	1.84 %	0.87 %
Prob508	78.60 %	77.03 %	Prob5018	0.57 %	0.31 %
Prob509	82.00 %	79.85 %	Prob5019	0.53 %	0.52 %
Prob5010	49.24 %	44.35 %	Prob5020	1.60 %	1.20 %

Los resultados obtenidos cuando se realizaron corridas con problemas de tamaño 200 se presentan en la tabla 7.

Tabla 7. Mejora porcentual con respecto a EDD de algoritmos con diferentes factores de visibilidad aplicados a problemas de tamaño 200

Problema	**ACO 4**	**ACO 5**	**Problema**	**ACO 4**	**ACO 5**
Prob2001	63,21 %	67.60 %	Prob20011	12,20 %	14.98 %
Prob2002	61,13 %	63.03 %	Prob20012	13,66 %	19.64 %
Prob2003	54,49 %	66.43 %	Prob20013	12,01 %	18.77 %
Prob2004	60,27 %	61.37 %	Prob20014	16,42 %	22.26 %
Prob2005	57,90 %	59.15 %	Prob20015	12,31 %	16.21 %
Prob2006	44,86 %	45.38 %	Prob20016	0 %	0 %
Prob2007	38,99 %	38.90 %	Prob20017	3,77 %	3.77 %
Prob2008	48,12 %	45.87 %	Prob20018	1,17 %	1.17 %
Prob2009	47,76 %	46.70 %	Prob20019	0,48 %	6.48 %
Prob20010	31,32 %	38.50 %	Prob20020	1,63 %	1.43 %

5.2.3.2 Discusión. En los problemas de tamaño 50 se logran mejores resultados con ACO 4 (37.82 % en promedio de mejora con respecto a EDD contra 36.67% de ACO 5). Para los problemas de tamaño 200 el algoritmo ACO 5 arroja resultados superiores frente a ACO 4 (31.88% vs 29.08%). Aunque con ACO 5 en los problemas 2003, 20012, 20014 y 20015 se obtienen resultados claramente superiores, en promedio la diferencia entre los dos algoritmos no es muy grande (en ninguno de los dos tamaños de problema excede el 2.79%). La razón puede ser que en ambos casos los factores de visibilidad tratan de escoger los trabajos que eviten tener la máquina desocupada innecesariamente.

5.3 COMPARACION DE ACO CON OTRAS METAHEURÍSTICAS

Para realizar las comparaciones del desempeño de colonia de hormigas frente a búsqueda tabú y recocido simulado se escogió la mejor respuesta entre las arrojadas por los algoritmos ACO 4 y ACO 5. Los resultados reportados son nuevamente el porcentaje de mejora con respecto a la solución hallada con EDD, pero en esta sección se dividen los problemas por tamaño, por valores de los parámetros TF y RDD y se reporta el tiempo de ejecución del programa con el fin de realizar los análisis correspondientes.

5.3.1 Problemas de tamaño 50.

Tabla 8. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 50 con TF = RDD = 0.2

Problema	SA	Tiempo	TS	Tiempo	ACO	Tiempo
Probl501	79.76 %	1s	79.76 %	1 s	66.82 %	13 s
Probl502	81.70 %	1 s	82.38 %	1 s	66.00 %	11 s
Probl503	76.73 %	1 s	79.24 %	1 s	70.65 %	14 s
Probl504	89.09 %	1 s	89.14 %	1 s	84.25 %	13 s
Probl505	85.09 %	1 s	85.20 %	1 s	77.30 %	14 s
Promedio	82.47 %	1 s	83.14 %	1 s	73.04 %	13 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 9. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 50 con TF = RDD = 0.4

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl506	64.04 %	1s	64.04 %	1 s	57.00 %	12 s
Probl507	80.09 %	1 s	80.09 %	1 s	75.52 %	11 s
Probl508	83.90 %	1 s	83.90 %	1 s	78.60 %	12 s
Probl509	84.90 %	1 s	84.90 %	1 s	82.00 %	20 s
Probl5010	60.99 %	1 s	60.99 %	1 s	49.24 %	12 s
Promedio	74.78 %	1 s	74.78 %	1 s	68.47 %	13.4 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 10. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 50 con TF = RDD = 0.6

Problema	**S A**	**Tiempo**	**T S**	**Tiempo**	**ACO**	**Tiempo**
Probl5011	12,69 %	1 s	12,69 %	1 s	7.96 %	14 s
Probl5012	28,99 %	1 s	28,99 %	1 s	24.07 %	16 s
Probl5013	17,38 %	1 s	17,4 %8	1 s	9.54 %	15 s
Probl5014	15,78 %	1 s	15,78 %	1 s	12.25 %	13 s
Probl5015	0,64 %	1 s	5,78 %	1 s	2.44 %	14 s
Promedio	15.1 %	1 s	16.14 %	1 s	11.25 %	14.4 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 11. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 50 con TF = RDD = 0.8

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl5016	0.0 %	1 s	0.0 %	1s	0.0 %	14 s
Probl5017	1,84 %	1 s	1,84 %	1 s	1.84 %	11 s
Probl5018	0,57 %	1 s	0,57 %	1 s	0.57 %	17 s
Probl5019	0,53 %	1 s	0,53 %	1 s	0.53 %	14 s
Probl5020	2,17 %	1 s	2,17 %	1 s	1.60 %	8 s
Promedio	1.02 %	1 s	1.02 %	1 s	0.91 %	12.8 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

5.3.1.2 Discusión Se observa que ACO es la metaheurística que presenta las soluciones más costosas en tiempo y de menor calidad en la solución. Esta tendencia se mantiene en todos los problemas analizados menos en los reportados en la tabla 5. En este caso la colonia de hormigas logra igualar en calidad de soluciones a las otras dos metaheurísticas. Sin embargo a pesar que el tiempo consumido por la colonia de hormigas es muy grande comparado con el consumido por las otras dos heurísticas es un tiempo aceptable en términos prácticos.

A medida que aumenta el valor de los parámetros TF y RDD es más difícil mejorar la solución de referencia. Esto se puede ser debido a que la regla de despacho para valores altos de los mencionados parámetros proporciona una buena solución ò porque el valor de la función objetivo para este caso no es sensible a los cambios de orden en la secuencia.

5.3.2 Problemas de tamaño 100. El mismo análisis realizado con los problemas de tamaño 50 se realizo con problemas de 100 trabajos. Los resultados se encuentran reportados en las tablas 12 a 15.

Tabla 12. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 100 con TF = RDD = 0.2

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl1001	76,94 %	1 s	76,94 %	2 s	78.45 %	140 s
Probl1002	39.05 %	1 s	72,44 %	2 s	83.50 %	20 s
Probl1003	80,20 %	1 s	80,20 %	2 s	82.48 %	17 s
Probl1004	74,70 %	1 s	79,41 %	2 s	74.7 %	20 s
Probl1005	72,84 %	1 s	74,27 %	2 s	74.51 %	18 s
Promedio	68.75 %	1 s	76.65 %	2 s	78.73 %	67.6 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 13. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 100 con TF = RDD = 0.4

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl1006	63,28 %	1 s	64,05 %	2 s	61.22 %	1100s
Probl1007	50,72 %	1 s	61,33 %	2 s	56.73 %	960 s
Probl1008	61,74 %	1 s	61,74 %	2 s	53.78 %	120 s
Probl1009	57,80 %	1 s	58,12 %	2 s	47.65 %	115 s
Probl10010	51,99 %	1 s	29,00 %	2 s	46.23 %	120 s
Promedio	57.11 %	1 s	54.85 %	2 s	53.12 %	483

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 14. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 100 con TF = RDD = 0.6

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl10011	24,46 %	1 s	24,47 %	2 s	22.00 %	60 s
Probl10012	17,24 %	1 s	18,09 %	2 s	16.34 %	62 s
Probl10013	21,97 %	1 s	21,17 %	2 s	21.60 %	60 s
Probl10014	21,80 %	1 s	21,80 %	2 s	21.57 %	62 s
Probl10015	19,68 %	1 s	19,70 %	2 s	18.28 %	80 s
Promedio	21.03 %	1 s	21.38 %	2 s	19.96 %	64.8 s

Tabla 15. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 100 con TF = RDD = 0.8

Problema	SA	Tiempo	TS	Tiempo	ACO	Tiempo
Probl10016	0,36 %	1 s	0 %	2 s	0,36 %	27 s
Probl10017	0 %	1 s	0,09 %	2 s	0.09 %	30 s
Probl10018	3,66 %	1 s	3,66 %	2 s	3,66 %	27 s
Probl10019	0,32 %	1 s	0 %	2 s	1,30 %	27 s
Probl10020	1,04 %	1 s	0 %	2 s	0 %	28 s
Promedio	1.08 %	1 s	1.75 %	2 s	1.08 %	27.8 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

5.3.2.1 Discusión. Los resultados obtenidos en los problemas de tamaño 100 muestran que ACO es competitiva frente a las otras dos metaheurísticas, esto es especialmente notorio en los resultados de la tabla 12 en los que ACO proporciona las mejores soluciones.
Al igual que en los problemas de tamaño 50, cuando aumentan los valores de TF y RDD es más difícil realizar mejoras en la función objetivo.

5.3.3 Problemas de tamaño 200.

Tabla 16. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 200 con TF = RDD = 0.2

Problema	SA	Tiempo	TS	Tiempo	ACO	Tiempo
Probl2001	65,76 %	4 s	84,56 %	6 s	67.60 %	1140 s
Probl2002	84,39 %	4 s	75,10 %	6 s	63.03 %	1020 s
Probl2003	66,08 %	4 s	72,13 %	6 s	66.43 %	1020 s
Probl2004	68,16 %	4 s	78,08 %	6 s	61.37 %	950 s
Probl2005	69,99 %	4 s	73,79 %	6 s	59.15 %	980 s
Promedio	70.88 %	4 s	76.63 %	6 s	63.52 %	1022 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 17. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 200 con TF = RDD = 0.4

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl2006	59,11 %	4 s	63,96 %	6 s	45.38 %	1010 s
Probl2007	54,95 %	4 s	62,76 %	6 s	38,99 %	960 s
Probl2008	59,64 %	4 s	64,81 %	6 s	48,12 %	1140 s
Probl2009	59,17 %	4 s	62,26 %	6 s	47,76 %	920 s
Probl20010	49,42 %	4 s	60,51 %	6 s	38.50 %	1740s
Promedio	56.46 %	4 s	62.86 %	6 s	43.75 %	1154 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 18. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 200 con TF = RDD = 0.6

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl20011	17,88 %	4 s	21,69 %	6 s	14.98 %	600 s
Probl20012	21,62 %	4 s	23,25 %	6 s	19.64 %	1620 s
Probl20013	18,98 %	4 s	18,93 %	6 s	18.77 %	1500 s
Probl20014	17,06 %	4 s	24,22 %	6 s	22.26 %	1600 s
Probl20015	22,58 %	4 s	24,44 %	6 s	16.21 %	780 s
Promedio	19.62 %	4 s	22.51 %	6 s	18.37 %	1220 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

Tabla 19. Mejora porcentual con respecto a EDD para las metaheurísticas aplicadas a problemas tamaño 200 con TF = RDD = 0.8

Problema	**SA**	**Tiempo**	**TS**	**Tiempo**	**ACO**	**Tiempo**
Probl20016	0,02 %	4 s	0,023 %	6 s	0 %	700 s
Probl20017	2,81 %	4 s	3,77 %	6 s	3.77 %	510 s
Probl20018	1,19 %	4 s	1,59 %	6 s	1.17 %	225 s
Probl20019	0 %	4 s	0,73 %	6 s	6.48 %	492 s
Probl20020	0,77 %	4 s	2,03 %	6 s	1.63 %	480 s
Promedio	0.96 %	4 s	1.63 %	6 s	2.61 %	481.4 s

Recocido simulado SA, búsqueda tabú TS y colonia de hormigas ACO.

5.3.3.1 Discusión. En este caso colonia de hormigas reporta soluciones que son competitivas al compararlas con recocido simulado pero es superada en la mayoría de los casos por la búsqueda Tabú. A favor de colonia de hormigas se puede decir que arrojo mejores resultados en la situación en la que parece ser más difícil mejorar la solución arrojada por la regla EDD. El aspecto más critico es que el tiempo consumido por la colonia para construir las soluciones es considerablemente mayor que el consumido por sus enfrentados, sin reportar soluciones significativamente mejores.

Los resultados muestran que independientemente al tamaño del problema a medida que el valor de los parámetros TF y RDD se incrementa, el porcentaje de mejora de las soluciones obtenidas con cualquiera de las metaheurísticas con respecto a la solución obtenida al aplicar EDD disminuye.

6. CONCLUSIONES Y RECOMENDACIONES

El diseño de experimentos muestra que para el algoritmo implementado los factores más significativos son los factores de visibilidad y el número de veces que se debe escoger entre los trabajos de la lista de candidatos.

En el algoritmo de colonia de hormigas implementado, la propuesta realizada en este estudio para la asignación del trabajo inicial arroja mejores resultados que la reportada en [28].

La asignación aleatoria del trabajo inicial conduce a mejores resultados que cuando se escoge entre uno de los trabajos ubicados en el primer 30% de las posiciones de la secuencia anterior.

Un aspecto importante en el desempeño del algoritmo es permitirle explorar nuevas soluciones con el fin de evitar caer en óptimos locales. Este hecho se manifiesta en los resultados obtenidos cuando al hacer la elección entre todos los trabajos no se consideran los que están en la lista de candidatos, así como en la asignación aleatoria del trabajo inicial En este sentido la propuesta realizada proporciona mejores resultados que los obtenidos cuando se aplica el esquema reportado en [28].

El mejor comportamiento de ACO se presentó en los problemas de 100 trabajos generados con el valor 0.2 para los parámetros TF y RDD. En este caso, de las metaheurísticas analizadas ACO presenta el mejor desempeño.

El comportamiento de ACO también presenta el mejor desempeño en los problemas de tamaño 200 generados con un valor de 0.8 para TF y RDD. Esto es importante ya que en este tipo de problemas es más difícil mejorar la solución obtenida con la regla EDD.

Sin los pasos de mejoramiento local el algoritmo de colonia de hormigas es competitivo frente al procedimiento de búsqueda local recocido simulado, ya que en muchos de los casos analizados proporciona soluciones de igual o mejor calidad que éste. A pesar de que la cantidad de tiempo es significativamente mayor, en muchos casos reales es puede llegar a ser aceptable.

Frente a la búsqueda tabú el algoritmo de colonia de hormigas no resulta competitivo ya que no supera la calidad de las soluciones que el primero obtiene y por otro lado consume mucho más tiempo de computador.

A medida que el valor de los parámetros TF y RDD crece, es más difícil mejorar la solución obtenida con la regla de despacho EDD.

Cuando en la generación de los problemas TF y RDD toman valores de 0.2 el mejor desempeño de ACO se presenta en los problemas de tamaño 100.

Para valores de los parámetros TF y RDD igual a 0.4 en la generación de los problemas, ACO reporta mejores resultados en los problemas de tamaño 50.

En el caso en que TF y RDD valen 0.6, el mejor comportamiento de ACO se presenta en los problemas de tamaño 100.

En los problemas generados con un valor de TF y RDD igual a 0.8, ACO obtiene mejores respuestas en los problemas de tamaño 200.

Los resultados obtenidos pueden representar un avance en el tiempo de ejecución de algoritmos de colonia de hormigas en problemas similares ya que están obteniendo mejores soluciones que las asignaciones convencionales y por lo tanto se puede requerir menos esfuerzo en los pasos de mejoramiento local.

La líneas de investigación futuras se pueden dirigir a la utilización de colonia de hormigas en otro tipo de problemas tales como programación de producción en ambientes más complejos ó problemas de ruteo.

BIBLIOGRAFÍA

1. Baluja S. and Caruana R. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russell, editors, Proceedings of the Twelfth International Conference on Machine Learning, ML-95, p 38 - 46. Palo Alto, CA: Morgan Kaufmann, 1995.

2. Bauer A., Bullnheimer B., Hartl F. and Strauss C. Minimizing total tardiness on a single machine using Ant Colony Optimization. Cejor 8: 125-141. 2000.

3. Bersini H., Dorigo M., Langerman S., Seront G. and Gambardella L. M. Results of the First international contest on evolutionary optimisation [1st ICEO). In Proceedings of IEEE International Conference on Evolutionary Computation, IEEE-EC 96, pp 611- 615. IEEE Press, 1996.

4. Bishop C. M. Neural Networks for Pattern Recognition. Oxford University Press,1995.

5. Bruckstein M., Mallows C. L. and Wagner I. A. Probabilistic pursuits on the grid. AMM: The American Mathematical Monthly, 104, 1997.

6. Campanini R., Di Caro G., Villani M., D'Antone I. and Giusti G. Parallel architectures and intrinsically parallel algorithms: Genetic algorithms. International Journal of Modern Physics C, 5(1): 95-112, 1994.

7. Chen K., A simple learning algorithm for the traveling salesman problem. Physical Review E, 55, 1997.

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

9. Colorni A, Dorigo M and Maniezzo V. Ants System applied to the Quadratic Assignment Problem. Technical Report No. 94- 28, IRIDIA, Brusels, belgium. 1994

10. Colorni A, Dorigo M and Maniezzo V. Distributed optimization by ant-colonies. Ind: Verela F and Bourgine P [eds). Procedings of the First European Conference on Artificial Life (ECAL'91). MIT Press: Cambridge MA, pp 134-142. 1991

11. Corne D., Dorigo M. and F. Glover, editors. New Ideas in Optimization. McGraw-Hill, 1999.

12. Costa D. and Hertz A. Ants can colour graphs. Journal of the Operational Research Society, 48: 295 - 305, 1997.

13. Cox G M and Cocharan W. Diseños experimentales. Jhon Wiley & Sons. 1965

14. Della Crose F. Tadei R, Baracco P, Grosso A. A New Descomposition Approach for the Single machina Total Tardiness Scheduling problem. Journal of the operational research society 49, 1101 – 1106. 1998.

15. Deneubourg Jl, PasteelsJm and Verhaeghe JC. Probabilistic behavior in ants: a strategy of errors? J Theor Biol **105**: 259-271, 1983

16. Di Caro G. and M. Dorigo. An adaptive multi-agent routing algorithm inspired by ants behavior. In Proceedings of PART98 - 5th Annual Australasian Conference on Parallel and Real-Time Systems, pp 261-272. Springer-Verlag, 1998.

17. Di Caro Gianni, Gambardella L. M. and Dorigo M. Ants Algorithms for Discrete Optimization, IRIDIA, Universite Libre de Bruxelles, Belgium.

18. Di Caro G. and M. Dorigo. AntNet: Distributed stigmergetic control for communications networks. Journal of Artificial Intelligence Research [JAIR), 9:317-365, December 1998.

19. Dorigo M. Optimization, Learning and Natural Algorithms [in Italian). PhD thesis, Dipartimento di Elettronica e Informazione, Politecnico di Milano, IT, 1992.

20. Dorigo M. and Gambardella L.M.. Ant colonies for the traveling salesman problem. BioSystems, 43: 73-81, 1997.

21. Dorigo M. and Gambardella L. M. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1): 53 - 66, 1997.

22. Dorigo M. and Maniezzo V. Parallel genetic algorithms: Introduction and overview of the current research. In J. Stender, editor, Parallel Genetic Algorithms: Theory & Applications, pages 5- 42. IOS Press, 1993.

23. Du J, Leung JY- T [1990) Minimizing Total tardiness on One Machina is NP- Hard. Mathematics of Operations Research 15, 483-495.

24. Emmons H. One Machine Sequencing to minimize certain Funtions of Tardiness. Operations research 17, 701- 715. 1969.

25. Feldman J.A and Ballard D. H.. Connectionist models and their properties. Cognitive Science, 6: 205- 254, 1982.

26. Fisher ML. A Dual Algorithm for the One Machine Scheduling Problem. Mathematical Programming 11, 229 – 251 1976.

27. Fogel D. B. Evolutionary Computation. IEEE Press, 1995.

28. Gagne C., Gravel M & Pric W I. [2002). Comparing an ACO algorithm whit other heuristics for the single machine scheduling problem with sequence dependent setup times. Journal of the Operational Research Society. **53**: 895 – 906

29. Gambardella L. M., Taillard E. D. and M. Dorigo. Ant colonies for the QAP. Technical Report 4-97, IDSIA, Lugano, Switzerland, 1997. Accepted for publication in the Journal of the Operational Research Society [JORS).

30. Gambardella L. M., Taillard E. and G. Agazzi. Ant colonies for vehicle routing problems. In D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization. McGraw- Hill, 1999.

31. Gambardella L.M. and Dorigo M. HAS-SOP: An hybrid ant system for the sequential ordering problem. Technical Report 11-97, IDSIA, Lugano, CH, 1997.

32. Gambardella L. M. and Dorigo M. Solving symmetric and asymmetric TSPs by ant colonies. In Proceedings of the IEEE Conference on Evolutionary Computation, ICEC96, pp 622- 627. IEEE Press, 1996.

33. Garey M. R., Johnson D. S., and Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research, 2[2):117-129, 1976.

34. Gavett J. Three heuristics rules for sequencing jobs to a single production facility. Management Science, 11:166 -176, 1965.

35. Glover F. Tabu search, part I. ORSA Journal on Computing, 1:190 - 206, 1989.

36. Goldberg D. E. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989.

37. Goss S., Aron S., Deneubourg J.L., and Pasteels J. M.. Self-organized shortcuts in the Argentine ant. Naturwissenschaften, 76: 579 – 581, 1989.

38. Grasse P.P. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. La theorie de la stigmergie : essai dinterpretation du comportement des termites constructeurs. Insectes Sociaux, 6 : 41 81, 1959.

39. Hertz, J. Krogh A., and Palmer R. G.. Introduction to the Theory of Neural Computation. Addison Wesley, 1991.

40. Holland J. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.

41. Hoop W. and Spearman M. Factory Physics. New York. Mc Graw Hill. 2001.

42. Kirkpatrick S., Gelatt C. D. and M. P. Vecchi. Optimization by simulated annealing. Science, 220, pp:671- 680, 1983.

43. Koulamas C [1994). The total tardiness problem: review and extensions. Opns Res **42:**1025 – 1041.

44. Lawler E. L. A Pseudopolynomial Algorithm for Sequencing Jobs to Minimize total Tardiness. Annals of discrete Mathematics 1, 331- 342, 1977.

45. Lenstra Jk, Rinnoy Kan AHG, Bruckner P [1977) Complexity of machine scheduling problems. In: Hammer PL, Johnson EL, Korte BH, nemhauser GL [eds) Studies in Integer Programming, Annals of Discrete Mathematics 1. North-Holland, Amsterdam, pp 343- 362, 1977.

46. Maniezzo V. and Colorni A. The ant system applied to the quadratic assignment problem. IEEE Trans. Knowledge and Data Engineering, 1999, in press.

47. Merkie D and Middendorf M. An ant algorthm with new pheromone evaluation rule for the total tardiness problem. In: cagnioni et al [eds). Real – Word applications of Evolutionary Computing. Springer: Berlin, pp 287 – 296, 2000

48. Michel R and Middendof M. An ACO algorithm for the shortest common super sequence problem. New Ideas in Optimization. Mc Graw- Hill. 1999

49. Montgomery D. C., Design and Analisis of Experiments, United States. Jhon Wiley & Sons. 1997.

50. Narendra K. and Thathachar M. Learning Automata: An Introduction. Prentice - Hall, 1989.

51. Pasteels J. M., Deneubourg J.-L. and Goss S. Self-organization mechanisms in ant societies (i): Trail recruitment to newly discovered food sources. Experientia Supplementum, 54:155 -175, 1987.

52. Pinedo M & Xiuli C. [1999).Operations Scheduling. United States. Mc Graw Hill..

53. Potts CN, Van Wassenhove LN. A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. Operations Research 33, 363 – 377. 1985

54. Rubin PA and Ragatz GL. Scheduling in a sequence dependent setup environment with genetic search. Computers Opns Res **22**: 85-99. 1995

55. Rubistein R. Y. Simulation and the Monte Carlo Method. John Wiley & Sons, 1981.

56. St*f*utzle T. and Hoos H. The MAX - MIN Ant system and local search for the traveling salesman problem. In T. Baeck, Z. Michalewicz, and X. Yao, editors, Proceedings of IEEE-ICEC-EPS'97, IEEE International Conference on Evolutionary Computation and Evolutionary Programming Conference, pages 309 - 314. IEEE Press, 1997.

57. St*f*utzle T. and Hoos H. Improvements on the ant system: Introducing MAX - MIN ant system. In Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, pages 245 - 249. Springer Verlag, Wien, 1997.

58. Whitley D., Starkweather T. and Fuquay D. Scheduling problems and travelling salesman: The genetic edge recombination operator. In Proceedings of the Third International Conference on Genetic Algorithms, pp 133-140. Palo Alto, CA: Morgan Kaufmann, 1989. 78

ANEXO 1. CÓDIGO DEL ALGORITMO IMPLEMENTADO EN LENGUAJE C.

A continuación se presenta el código del algoritmo que se implemento en este estudio, el compilador utilizado fue el de la casa Borland. Se recuerda que no se pueden colocar tildes.

Se agregan comentarios de manera conveniente, sin embargo para una mejor comprensión se recomienda tener presente el esquema presentado en la pagina 41 Las explicaciones escritas en letra tamaño 12 no hacen parte del código y su objetivo es orientar sobre el propósito de cada segmento del programa.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#define N  100   /*numero de trabajos (cambiar de acuerdo al tamaño del problema)*/
#define MH 30  /*numero de hormigas (cambiar de acuerdo al problema)*/
#define TF  3   /*tamaño lista de candidatos (cambiar de acuerdo al problema)*/
/* se define una estructura para poder manejar de mejor forma los datos correspondientes a cada trabajo*/
 struct job{
          int njob;       /*numero del trabajo*/
          double proc ;/*tiempo de proceso*/
          double rele;  /*release time*/
          double due;  /*tiempo de entrega*/
          double term; /*tiempo de terminacion*/
 };

 /*variables globales*/
   double tau0,rol,rog,alf,bet,gam,qo,Lm;
   int     m,n,It,tf,cand,Lteus,iter;
   struct job *datos;
```

Los nombres de las variables respetan la nomenclatura que se utilizo en la descripción de la metaheurística, sin embargo ya que no se pueden usar símbolos en el código se presenta esta correspondencia.

τ_o = tau 0 , ρ_l = rol, ρ_g = rog, α = alf, β = bet, φ = gam, L+ = Lm.

```
   /*prototipo de funciones*/
   void leer (void)  ;/*lee los datos de los trabajos de un aschivo*/
   double iniciacion (void);  /*da una secuencia inicial y calcula tau0*/
   void iteracion (void)  ;/*calcula S* y L* */
   double aleat (double, double);  /*genera un numero aleatorio*/
   double maxX(double *,int);  /*calcula maximo de un vector*/
   int maxXP(double *,int);   /*calcula posicion del maximo en un vector*/
   int minXP(double *,int);   /*calcula el minimo de un vector*/
   double minX(double *,int);   /*posicion del minimo de un vector*/
```

```
/*programa principal*/

void main (void){
        clrscr();
        printf("digite el numero de trabajos n");
        scanf("%d",&n);
        printf("digite el numero de hormigas m");
        scanf("%d",&m);
        printf("digite el numero maximo de iteraciones It");
        scanf("%d",&It);
        printf("digite el numero de iteraciones maximas permitidas sin mejorar el objetivo Lteus");
        scanf("%d",&Lteus);
        printf("digite el tamaño de las lista de candidatos tf");
        scanf("%d",&tf);
        printf("digite el nivel de evaporacion local y global rol, rog");
        scanf("%lf%lf",&rol,&rog);
        printf("digite el valor de qo");
        scanf("%lf",&qo);
        printf("digite los exponentes de las ecuaciones alf, bet, gam");
        scanf("%lf%lf%lf",&alf,&bet,&gam);
        leer();
        iniciacion();
        iteracion();
        free(datos);
}
```

El siguiente segmento de código lee de un archivo el número del trabajo (njob), su tiempo de proceso (proc), tiempo de inicio (rele) y tiempo de entrega (due).

```
void leer (void){
        FILE *archDat;
        int i;
        archDat= fopen("nombre.txt","r");
        if(archDat==NULL){
                printf ("\n el archivo no existe\n");
                exit(1);
        }
        else{
        datos=(struct job *) malloc(n*sizeof(struct job));
          for (i=0;i<n;i++){
             fscanf(archDat,"%d%lf%lf%lf",&datos(i).njob,&datos(i).proc,&datos(i).rele,&datos(i).due);
          }

        }

}
```

Esta parte del programa realiza los cálculos necesarios para poder empezar las iteraciones de la metaheurística ACO. Ordena los trabajos de acuerdo a su número y calcula el valor del retardo máximo (guarda este valor como L+), almacena esta secuencia como S+ y por último calcula el valor de $\tau o = 1 / n*Lm$.

```
double iniciacion (void){
  int i;
  datos(0).term=datos(0).rele+datos(0).proc;
  Lm=datos[0].term-datos[0].due;
  for (i=1;i<n;i++){
        datos(i).term=max(datos[i].rele,datos[i-1].term)+ datos[i].proc;
        Lm=max(max(datos[i-1].term, datos[i]..rele)+datos[i]..proc-datos[i].due,Lm); /*Tau inicial y mejor
                                                                                      solucion acumulada*/
  }
  tau0=max((1/(n*Lm)),-1*(1/(n*Lm)));
  return tau0;
}
```

Se explica la nomenclatura del siguiente segmento del código.

M es una matriz de m*n (m es el número de hormigas y n el número de trabajos) en la cual en cada fila se almacena la secuencia que esta construyendo la hormiga correspondiente

Mtau es una matriz donde se almacena el valor del rastro de feromonas $\tau_{i,j}$ para cada par de trabajos adyacentes.

Tmej es un vector en el que se almacenan los valores del retardo máximo de las soluciones construidas.

Tiem es un vector en el que se almacena el tiempo en que la máquina acaba de procesar el último trabajo que entro a ella.

Fvisi corresponde al valor del factor de visibilidad en el momento de hacer la elección del siguiente trabajo.

P es un vector en el que se almacenan los trabajos que aún no se han programado.

C es un vector en el que se almacenan los trabajos que pertenecen a la lista de candidatos.

Mtabu es una matriz que en cada fila contiene los trabajos que ya han sido programados por cada hormiga, en otras palabras es la lista tabu.

Jsec es un vector en el que se encuentra el valor de la regla de decisión (ecuaciónes) para cada uno de los trabajos que aún no se han programado. El trabajo que presente el mayor valor de dicha regla, es el que se programa a continuación.

Com es un vector donde se almacena el tiempo de terminación de cada uno de los trabajos.

Tobj es el vector en el que se almacena la mejor secuencia de la iteración.

Lest es una variable que toma el valor de la función objetivo correspondiente a la mejor solución hallada.

El siguiente fragmento del programa corresponde a lo que en el algoritmo se designo como ciclo principal, es decir la sección de la metaheurística donde cada hormiga construye una secuencia de trabajos.

```
void iteracion (void){   /*INICIO DEL CICLO PRINCIPAL */
 int a,b,f,g,i,j,k,l,r,s,u,v;
 double
M[MH][N],Mtau[N][N],Tmej[N],Tiem[N],Fvisi,Tobj[N],P[N/TF],Mtabu[MH][N],Jsec[N],Com[N],Lamx[N]
,La[N];
 int C[N];
 double q,Lest;
 int PosLest,CTA,Lte;
 cand=(n/tf);
 Lte=0;
 CTA=0;
 for (i=0;i<n;i++){    /*llenado de la matriz Tau, a cada posicion se la asigna el valor tau0*/
             for(j=0;j<n;j++){
       Mtau[i][j]=tau0;
   }
 }/*fin de llenado de la matriz Tau*/
 for(r=0;r<n;r++){    /*mejor secuencia inicial*/
      Tobj[r]= datos[r].njob;
 }
```

Se hace una primera lista de candidatos de acuerdo al número del trabajo.

```
 for(r=0;r<n;r++){
     C[r]=datos[r].njob; /*hace una primera lista de candidatos por n job*/
 }
 for(u=0;u<(n-1);u++){
   for(v=0;v<(n-u-1);v++){
     if(datos(C(v)-1).rele>datos(C((v+1))-1).rele){ /*ordena los candidatos por release time*/
        r= C[[v+1]];
        C[[v+1]]= C[v];
        C[[v]]=r;
     }
   }
}

 for(k=0;k<m;k++){   /*Pone en la ultima posicion de la matriz de secuencias los candidatos*/
    M[k][n-1]=C[k];
}
```

En el siguiente segmento del algoritmo se inicia la realización de las iteraciones en las que cada hormiga construye una solución al problema, se puede decir que es el núcleo del programa.

```
   for (iter=0;iter<It&&Lte<=Lteus;iter++){    /*INICIO DE ITERACIONES */
```

```
 for (a=0;a<m;a++){     /*pone en la primera posicion el último trabajo de la secuencia anterior*/
     M[a][0]= M[a][n-1];
}
for (b=0;b<m;b++){
    for (a=0;a<n;a++){
       Mtabu[b][a]= C[a];  /*primera lista tabu para cada hormiga*/
    }
}
```

En la matriz de la lista tabu para cada hormiga se marcan con –1 los trabajos ya programados.

```
for (b=1;b<n;b++){       /* se construye una columna de la matriz M*/
    for (f=0;f<m;f++){/*for f*/
        for (g=0;g<n;g++){
           if(Mtabu(f)(g) == M(f)(b-1)){ /*matriz tabu actualizada con la columna anterior de la matriz
                                          M*/
              Mtabu[f][g]= -1;
           }
        }
    }
    for (f=0;f<m;f++){       /*cada hormiga escoge un trabajo*/
        for (r=0;r<(cand);r++){
            P[r]= -1;
        }
        for(j=0, k=0;j<(cand)&& k<n ;j++,k++){ /*posibles candidatos para cada hormiga */
           if(Mtabu(f)(k)!=-1){
             P[j]= Mtabu[f][k];
           }
           else {
              j=j-1;
           }
        }/*fin posibles candidatos */
        q = aleat(0,1);  /*se elige con que regla se asignara el trabajo a la hormiga*/
        for(g=0;g<n;g++){
          Jsec[g]=-1;
        }
```

Después de generar el número aleatorio q, se compara con el parámetro q_o, en caso de cumplirse que q sea menor o igual al parámetro, se escoge de acuerdo con la regla indicada en la ecuación (8) la cual permite escoger solo entre los trabajos que se encuentren en la lista de candidatos.

```
        if(q<=qo){ /*eleccion greedy*/
          for(l=0;l<(cand) ;l++){
            if(P(l)!=-1){ /*escoge entre los candidatos*/
              Tiem(0)= datos((int)M(f)(0)-1).rele+datos((int)M(f)(0)-1).proc;
              for(g=1;g<b;g++){
                Tiem(g)=max(datos((int)M(f)(g)-1).rele,Tiem(g-1))+datos((int)M(f)(g)-1).proc;
                                                          /*calcula tiempos de terminacion*/
              }
              Fvisi = max(datos((int)P[l]-1).rele,Tiem[b-1])+datos((int)P[l]-1).proc;
              Jsec[l] = pow((double)Mtau((int)M[f][b-1]-1)((int)P[l]-1),alf);
              Jsec[l]=Jsec[l]*(pow(((double)1/datos((int)P[l]- 1)
```

```
                .due),bet))*(pow(((double)1/Fvisi),gam));/*vector con resultado de la funcion greddy*/
              }
            }
            l= maxXP(Jsec,(cand));
            M[f][b]= P[l];
          }/*fin de la elección geedy*/
```

De no haberse cumplido la condición para escoger entre los candidatos, la elección del siguiente trabajo se debe realizar considerando todos los trabajos aún no programados tal como lo indica la ecuación (9) y como se realiza en el siguiente fragmento de código.

.

```
else{ /*escoge entre todos los trabajos*/
            for(l=0;l<n;l++){
              if(b<(n-cand)){
                if(Mtabu[f][l]!=-1&&Mtabu[f][l]&&P[l]){
                  Tiem[0]= datos((int)M[f][0]-1).rele+datos((int)M[f][0]-1).proc;
                  for(g=1;g<b;g++){
                     Tiem(g)=max(datos((int)M[f][g]-1).rele,Tiem[g-1])+datos((int)M[f][g]-1).proc;
                  }
                  Fvisi = max(datos((int)Mtabu[f][l]-1).rele,Tiem[b-1])+datos((int)Mtabu[f][l]-1).proc;
                  Jsec[l) = pow((double)Mtau((int)M[f][b-1]-1)((int)Mtabu[f][l]-1),alf);
                  Jsec[l)=Jsec[l]*(pow(((double)1/datos((int)Mtabu[f][l]-
                  1).due),bet))*(pow(((double)1/Fvisi),gam));
                }
                M[f][b]= Mtabu[f]((int)maxXP(Jsec,n));
              }
              else{
                if(Mtabu[f][l]!=-1){
                  Tiem[0]= datos((int)M[f][0]-1).rele+datos((int)M[f][0]-1).proc;
                  for(g=1;g<b;g++){
                     Tiem(g)=max(datos((int)M[f][g]-1).rele,Tiem[g-1])+datos((int)M[f][g]-1).proc;
                  }
                  Fvisi = max(datos((int)Mtabu[f][l]-1).rele,Tiem[b-1])+datos((int)Mtabu[f][l]-1).proc;
                  Jsec[l] = pow((double)Mtau((int)M[f][b-1]-1)((int)Mtabu[f][l]-1),alf);
                  Jsec[l]=Jsec[l]*(pow(((double)1/datos((int)Mtabu[f][l]- ).
                  due),bet))*(pow(((double)1/Fvisi),gam));
                }
                M[f][b]= Mtabu[f]((int)maxXP(Jsec,n));
              }
            }
          } /*fin elleccion entre todos los trabajos*/
          Mtau((int)M[f][b-1]-1)((int)M[f][b]-1)=(rol*Mtau((int)M[f][b-1]-1)((int)M[f][b]-1))+(double)(1-
                                                  rol)*tau0;       /*actualizacion local de tau*/
      }    /*fin cada hormiga escoge un trabajo*/
  }   /*fin se construye una columna de la matriz M*/
```

En el siguiente segmento se realiza la evaluación de las soluciones construidas por cada una de las m hormigas, con el objetivo de seleccionar la mejor y realizarle un incremento en el rastro de feromonas.

```
for (b=0;b<m;b++){      /*evaluacion de soluciones*/
```

```
        Com(0)= datos((int)M(b)(0)-1).rele+datos((int)M(b)(0)-1).proc; /*tiempo de terminacion del primer
                                                                        trabajo de la secuencia*/
        La(0)= Com(0)-datos((int)M(b)(0)-1).due;
          for(f=1;f<n;f++){
            Com(f)=max(datos((int)M(b)(f)-1).rele,Com(f-1))+datos((int)M(b)(f)-1).proc;
/*calcula tiempos de terminacion*/
              La(f)=max(Com(f-1),datos((int)M(b)(f)-1).rele)+datos((int)M(b)(f)-1).proc-datos((int)M(b)(f)-
                                    1).due;    /*calcula el retardo maximo de las m soluciones*/
          }
          Lamx(b) = maxX(La,n);
    }     /*fin evaluación de soluciones*/
```

Se escoge la mejor solución de la iteración y se compara con la mejor solución que se tiene hasta el momento en todo el proceso, de ser mejor se realiza la actualización correspondiente.

```
    Lest= minX(Lamx,MH);    /*escoge la mejor solucion de la iteracion*/
    PosLest=minXP(Lamx,MH); /*indica la posicion de la mejor hormiga*/
    if(Lest< Lm){
      Lm=Lest;              /*Lm = mejor solucion de la meta-heuristica*/
        for(r=0;r<n;r++){  /*mejor secuencia de la meta-heuristica*/
          Tobj(r)= M(PosLest)(r);
        }
      CTA=CTA+1;
      Lte=0;
    }/*fin if*/
    else{
    Lte=Lte+1;
    }
    for(r=0;r<n;r++){             /*mejor secuencia de la iteracion*/
      Tmej(r)= M(PosLest)(r);
    }
```

Una vez identificada la mejor secuencia de la iteración, a todos los pares de trabajos adyacentes pertenecientes a esta se les realiza el incremento correspondiente en el nivel de feromonas.

```
    for(s=0;s<(n-1);s++){         /*actualizacion global de tau*/
      Mtau((int)Tmej(s)-1)((int)Tmej(s+1)-1)=(rog*Mtau((int)Tmej(s)-1)((int)Tmej(s+1)-1))+(1-
                                              rog)*((double)1/(Lest));
    }
  }     /*FIN DE ITERACIONES */
```

Una vez terminadas las iteraciones del programa se reporta cual fue la mejor solución hallada y su respectivo valor de la función objetivo.

```
  printf("La mejor secuencia encontrada es\n");
  for(g=0;g<n;g++){
    printf("%lf\n",Tobj(g));
  }
  printf("Su valor de retardo maximo es %lf\n",Lm);
  getch();
```

```
}     /*FINAL DEL CICLO PRINCIPAL */
```

Las siguientes son subrutinas que se necesitaban para la ejecución del programa, pero que por si mismas no forman parte del programa principal.

```
 /* PROGRAMAS INVOCADOS*/
   double maxX(double *x,int T){          /*halla el valor máximo de un vector*/
     double *pxi,*pxn,maxi=-1.0E100,xi;
     pxi=x;
     pxn=x+T;
     while(pxi<pxn){
           xi=*pxi++;
           if(xi>maxi){
             maxi=xi;
           }
    }
    return maxi;
  }
 int maxXP(double *x,int T){          /*posicion del maximo en un vector*/
    int i,posi;
    double *pxi,*pxn,maxi=-1.0E100,xi;
     posi=-1;
     pxi=x;
     pxn=x+T;
     i=0;
      while(pxi<pxn){
        xi=*pxi++;
        if(xi>maxi){
          maxi=xi;
          posi=i;
        }
        i++;
     }
     return posi;
 }

 double minX(double *x,int T){       /*valor del valor minimo de un vector*/
    double *pxi,*pxn,mini=1.0E100,xi;
    pxi=x;
    pxn=x+T;
    while(pxi<pxn){
      xi=*pxi++;
      if(xi<mini){
        mini=xi;
      }
   }
   return mini;
}

 int minXP(double *x,int T){        /*posicion del minimo en un vector*/
   int i,posi;
   double *pxi,*pxn,mini=1.0E100,xi;
```

```
  posi=-1;
  pxi=x;
  pxn=x+T;
  i=0;
    while(pxi<pxn){
      xi=*pxi++;
      if(xi<mini){
        mini=xi;
        posi=i;
      }
      i++;
    }
    return posi;
}

double aleat (double w, double v){ /*calcula un aleatotio entre w y v*/
   double r;
   r=w + (v-w)*double(rand())/double(RAND_MAX);
   return r;
}
```

ANEXO 2. SALIDAS EN MINITAB DEL ANALISÍS DE VARIANZA Y REGRESIÓN PARA LA BUSQUEDA DE LOS MEJORES VALORES DE LOS PARAMETROS DE ACO.

```
Factor     Type Levels Values
Qo        fixed      3 0,1 0,5 0,9
Rol       fixed      3 0,1 0,5 0,9
Rog       fixed      3 0,1 0,5 0,9
Alf       fixed      3 1 3 5
Bet       fixed      3 1 3 5
Gam       fixed      3 1 3 5

Analysis of Variance for Rta, using Adjusted SS for Tests

Source     DF     Seq SS     Adj SS     Adj MS       F      P
Qo          2     280776     280776     140388   39,99  0,000
Rol         2      10639      10639       5319    1,52  0,227
Rog         2       9377       9377       4688    1,34  0,270
Alf         2       9680       9680       4840    1,38  0,259
Bet         2      91646      91646      45823   13,05  0,000
Gam         2     101800     101800      50900   14,50  0,000
Error      68     238745     238745       3511
Total      80     742662

Unusual Observations for Rta
```

Regression Analysis: Rta versus Qo. Rol. ...

```
The regression equation is
Rta = 898 - 423 Qo - 138 Rol - 42,4 Rog + 12,9 Alf + 35,6 Bet - 9,4 Gam
          + 252 Qo^2 + 147 Rol^2 + 9,5 Rog^2 - 1,05 Alf^2 - 2,54 Bet^2
          - 2,02 Gam^2

Predictor        Coef     SE Coef          T        P
Constant       897,88       52,19      17,20    0,000
Qo            -422,80       89,59      -4,72    0,000
Rol           -138,10       89,59      -1,54    0,128
Rog            -42,36       89,59      -0,47    0,638
Alf             12,89       21,33       0,60    0,548
Bet             35,61       21,33       1,67    0,100
Gam             -9,44       21,33      -0,44    0,659
Qo^2           252,20       87,29       2,89    0,005
Rol^2          146,99       87,29       1,68    0,097
Rog^2            9,49       87,29       0,11    0,914
Alf^2           -1,051       3,492      -0,30    0,764
Bet^2           -2,537       3,492      -0,73    0,470
Gam^2           -2,023       3,492      -0,58    0,564

S = 59,25       R-Sq = 67,9%     R-Sq[adj) = 62,2%
Analysis of Variance
```

Source	DF	SS	MS	F	P
Regression	12	503918	41993	11,96	0,000
Residual Error	68	238745	3511		
Total	80	742662			

Source	DF	Seq SS
Qo	1	251467
Rol	1	683
Rog	1	9335
Alf	1	9361
Bet	1	89793
Gam	1	100621
Qo^2	1	29309
Rol^2	1	9956
Rog^2	1	42
Alf^2	1	318
Bet^2	1	1854
Gam^2	1	1179