IMPLEMENTACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS APLICADO EN EL ÁREA DE PLANIFICACIÓN DE RECURSOS

EDSON ALEJANDRO FLÓREZ SUÁREZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2013

IMPLEMENTACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS APLICADO EN EL ÁREA DE PLANIFICACIÓN DE RECURSOS

EDSON ALEJANDRO FLÓREZ SUÁREZ

Trabajo de Grado para optar al título de
Ingeniero de Sistemas

Directora
Msc. LOLA XIOMARA BAUTISTA ROZO

Codirector

Ing. WILFREDO ARIEL GÓMEZ BUENO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2013

DEDICATORIA

A Dios

A mis padres Jaime y Luz Mary

A mis hermanos Jaimito y Mary

A toda mi familia

A mis compañeros y amigos

A todos los que me guiaron para alcanzar esta meta

Edson Flórez

AGRADECIMIENTOS

A la Universidad Industrial de Santander y a la Escuela de Ingeniería de Sistemas e Informática, por la formación académica brindada por sus profesores.

Al Grupo de Investigación en Informática Biomédica, específicamente a su directora, Msc. Lola Xiomara Bautista Rozo, y al Ing. Wilfredo Ariel Gómez Bueno, por sus trascendentales aportes y orientaciones durante el desarrollo del proyecto.

A mis compañeros y amigos de la universidad, por todas las enseñanzas para la vida que me quedaron del tiempo compartido, y por siempre estar dispuestos a trabajar juntos para cumplir los compromisos académicos.

CONTENIDO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABLAS

Pág.

LISTA DE ANEXOS

RESUMEN

TITULO: IMPLEMENTACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS APLICADO EN EL ÁREA DE PLANIFICACIÓN DE RECURSOS[*]

AUTOR: Edson Alejandro Flórez Suárez[**]

PALABRAS CLAVE: Optimización combinatoria, Job Shop Scheduling Problem (JSP), Metaheurísticas, Optimización basada en colonia de hormigas (OCH), GRASP.

DESCRIPCIÓN: Las metaheuristicas son las mejores técnicas para solucionar los problemas de optimización, entre las cuales se destaca la *Optimización basada en Colonia de Hormigas* (OCH), que ha probado ser muy eficiente y efectiva en problemas de gran complejidad (NP-hard) en optimización combinatoria. En este trabajo de grado se describe la implementación de un algoritmo del modelo OCH conocido como *Elitist Ant System* (EAS), aplicado al problema de planificación de recursos denominado *Job Shop Scheduling Problem* (JSP). Se propone un método que busca reducir los retardos designando la operación disponible inmediatamente, pero teniendo en cuenta a las operaciones que les falta poco para estar disponibles y tienen una gran cantidad de feromona, porque son buenas candidatas. El desempeño del algoritmo fue evaluado para problemas de referencia en JSP, comparando la calidad de las soluciones obtenidas respecto a GRASP y a la mejor solución conocida para estos problemas. Las soluciones con OCH fueron de buena calidad (con 96% de aproximación a la mejor solución conocida), obtenidas con una destacable eficiencia al tener que realizar un número muy bajo de evaluaciones de la función objetivo. Por último, se planteó un escenario de aplicación del JSP en los sistemas distribuidos, donde se realizó la planificación de trabajos en los nodos de un clúster.

[*] Trabajo de Grado.

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Directora: Lola Xiomara Bautista Rozo. Codirector: Wilfredo Ariel Gómez Bueno.

ABSTRACT

TITLE: IMPLEMENTATION OF AN ALGORITHM OF ANT COLONY FOR JOB SHOP SCHEDULING PROBLEMS[*]

AUTHOR: Edson Alejandro Flórez Suárez[**]

KEYWORDS: Combinatorial optimization, Job Shop Scheduling Problem (JSP), Metaheuristics, Ant Colony Optimization (ACO), GRASP.

DESCRIPTION: The metaheuristics are the best techniques to solve optimization problems, outstanding among these is *Ant Colony Optimization* (ACO), which have proved to be very effective and efficient in problems of high complexity (NP-hard) in combinatorial optimization. This degree work describes the implementation of an ACO model algorithm known as *Elitist Ant System* (EAS), applied to a combinatorial optimization problem called *Job Shop Scheduling Problem* (JSP). We propose a method that seeks to reduce delays designating the operation immediately available, but considering the operations that lack little to be available and have a greater amount of pheromone, because they are good candidates. The performance of the algorithm was evaluated for problems of JSP reference, comparing the quality of the solutions obtained regarding GRASP and the best known solution for these problems. The OCH solutions were of good quality (96% of approximation to the *Best Knowed Solution*), obtained with remarkable efficiency by having to make a very low number of objective function evaluations. Finally, we proposed a JSP application scenario in distributed systems, where we performed the job scheduling at the nodes of a cluster.

[*] Degree work.

[**] Faculty of Physical-Mechanical Engineering. School of Engineering and Computing Systems. Director: Lola Xiomara Bautista Rozo. Codirector: Wilfredo Ariel Gómez Bueno.

GLOSARIO

Instancia de Lawrence (LA): problemas JSP especificados en [11], utilizados para comparar el desempeño de los algoritmos implementados respecto a los resultados obtenidos en otros trabajos.

JSP (*Job Shop Scheduling Problem*): el problema de planificación de tareas (calendarización o programación de tareas) es un problema de optimización combinatoria que es una generalización del problema del vendedor viajero o *Travelling Salesman Problem* (TSP). Es un ejemplo destacado de una clase de problemas en la teoría de la complejidad computacional que son difíciles de resolver.

Metaheurística: son procedimientos para la búsqueda de soluciones en problemas de optimización que introducen reglas sistemáticas que les permiten, en la mayoría de los casos, continuar con la búsqueda del valor óptimo dejando de lado los óptimos locales [49].

Optimización combinatoria: también conocida como optimización discreta, porque las variables son de naturaleza discreta.

Grafo: representación gráfica del problema, consiste en un conjunto finito de vértices (nodos) y un conjunto de aristas que unen vértices adyacentes. Pueden ser grafos dirigidos (dígrafo), donde a cada arista se le indica un sentido mediante una flecha, y grafos disyuntivos o no dirigidos, donde las aristas no tienen un sentido definido.

NP: tiempo polinomial no determinista (en inglés, *Nondeterministic Polynomial time*), son los problemas que pueden solucionarse en tiempo polinómico solo por una *Máquina de Turing no determinista*.

INTRODUCCIÓN

En este trabajo se aplica la inteligencia colectiva de las colonias de hormigas al problema *Job Shop Sheduling* [22]. Este problema consiste en encontrar un plan óptimo para terminar un número finito de tareas, en un número finito de máquinas en el menor tiempo posible [13], por tanto, la función objetivo será minimizar el flujo máximo de tareas. Cada tarea es una secuencia de operaciones, cada operación tiene una máquina y tiempo de procesamiento determinados. Las soluciones factibles deben cumplir con las restricciones a las que está sujeto el problema de *Jop Shop Sheduling*, como respetar la precedencia entre operaciones establecida por la secuencia tecnológica, sin interrumpir ninguna operación hasta su finalización [21].

La solución adecuada de este problema de optimización combinatoria solo se puede hallar utilizando metaheurísticas [54], que logran sortear la complejidad computacional del problema, requiriendo poco tiempo y memoria para la obtención de una solución aproximada. Estas técnicas optimizan (maximizar o minimizar según la función objetivo) costos, tiempos, desplazamientos, etc., en áreas como la industria y la informática.

Una de esas técnicas es la *Optimización basada en Colonia de Hormigas* (OCH), metaheurística que reúne conceptos de campos como la Inteligencia Artificial y la Biología, inspirado en el comportamiento colectivo de las hormigas. Estos insectos sociales forman complejos sistemas auto-organizados y descentralizados de donde surge la Inteligencia de Enjambre de las colonias de hormigas [12]. Gracias a esa inteligencia emergente de relaciones simples entre cada individuo, que solo poseen información local del sistema compartida a través de señales químicas, una colonia puede resolver problemas complejos presentes en su entorno, como el problema de encontrar el camino más corto entre la colonia y la comida, lo que puede ser adaptado para encontrar la mejor solución al problema de optimización

combinatoria tratado en este trabajo de investigación. Esta técnica es implementada junto a otra metaheurística llamada GRASP, para diseñar algoritmos de gran eficiencia y eficacia en JSP.

El presente documento está organizado de la siguiente manera. En el capítulo I se presenta el proyecto, con los objetivos de este y el planteamiento y justificación del problema. En el capítulo II se proporciona el marco teórico de los ejes centrales del proyecto, con una introducción a la Optimización combinatoria, la descripción del problema JSP y la descripción general de OCH y GRASP. En el capítulo III se hace un recuento del estado del arte, con los diversos algoritmos de la familia OCH y los métodos que han abordado el problema de *Job Shop Scheduling Problem*. En el capítulo IV se presenta el algoritmo de OCH implementado para solucionar el JSP, el *Elitist Ant System* con la descripción de la variante propuesta. En el capítulo V se realiza la caracterización de un posible escenario de aplicación del problema JSP en el campo de la computación. En el capítulo VI se exponen los resultados experimentales con los problemas de referencia y los comparamos con respecto a GRASP y otros algoritmos del estado del arte de JSP. Por último, en el capítulo VII y VIII se establecen las conclusiones del proyecto y recomendaciones para el trabajo futuro respectivamente.

1. PRESENTACIÓN DEL PROYECTO

1.1 PLANTEAMIENTO DEL PROBLEMA

Recientemente se han desarrollado técnicas inspiradas en la naturaleza que tienen como objetivo los problemas de optimización combinatoria, para los cuales es muy difícil encontrar soluciones exactas por su complejidad algorítmica (NP-hard). Las técnicas tradicionales requieren demasiados recursos computacionales y tiempo para obtener soluciones aproximadas, mientras que las técnicas metaheurísticas de optimización inspiradas en el comportamiento de las hormigas, han demostrado una gran eficiencia para resolver los problemas de optimización combinatoria.

Estos problemas son importantes por sus diferentes aplicaciones en la industria, que requiere gestionar sus cuantiosos procesos entre sus recursos limitados. En el presente estudio se considera un problema de calendarización conocido como *Job Shop Scheduling Problem* (JSP), en el que se tienen trabajos (jobs) que deben ser realizados en determinadas máquinas, y la relación entre ambos conforma una operación [13]. El objetivo es encontrar un calendario que utilice de manera óptima los recursos disponibles en el menor tiempo posible.

Los algoritmos de *Optimización basada en Colonias de Hormigas* (OCH) son un área de la inteligencia artificial muy reciente y activa. Para evaluar su desempeño en instancias comunes de problemas de planificación de trabajos, se realizará un contraste con el otro algoritmo implementado (GRASP) y con otras técnicas eficientes encontradas en la literatura.

1.2 JUSTIFICACIÓN

En la optimización combinatoria se encuentran problemas en los que el tiempo de computación aumenta de manera exponencial en la medida que crecen las restricciones y el número de variables. Algunos de estos problemas, como los de planificación de recursos se clasifican como NP-duro ya que su solución no se alcanza en un tiempo de cómputo polinomial. Debido a su alta incidencia en la industria en problemas de producción [5], aterrizajes y despegues de aviones, etapas de un proyecto de ingeniería, ejecución de un programa de computación, se hace necesario abordar este tipo de problemas con técnicas que produzcan cuasi-soluciones, es decir, aproximaciones a la solución exacta, permitiendo con esto la reducción en los tiempos de cómputo.

Dado que existe un vasto panorama de técnicas convencionales para la solución de estos problemas [9], se pretende explorar nuevas alternativas que han tenido un uso creciente en el ámbito investigativo internacional, como lo son los algoritmos de *Optimización basada en Colonias de Hormigas* (OCH), con el fin de encontrar aproximaciones eficientes a las soluciones de dichos problemas. En el país es escasa la cantidad de estudios y aplicaciones de esta técnica debido a lo relativamente nueva, y a los pocos grupos de investigación que trabajan en dicha área o han abordado dicha temática.

La aplicación funcional permitirá conocer el rendimiento de OCH y GRASP en instancias JSP, y además, con el planteamiento de posibles instancias de problemas de planificación de recursos relacionados al área computacional, se podrá determinar los mejores escenarios de aplicación.

1.3 OBJETIVOS

1.3.1 Objetivo general

Implementar un algoritmo de colonia de hormigas para resolver instancias de un problema de planificación de recursos y compararlo frente a otras técnicas encontradas en la literatura.

1.3.2 Objetivos específicos

- Implementar un algoritmo de colonia de hormigas sobre la familia de instancias LA del problema de *Job Shop Scheduling* (JSP).
- Evaluar el desempeño del algoritmo implementado frente a un algoritmo enumerativo GRASP sobre la familia de instancias LA del problema de *Job Shop Scheduling*.
- Caracterizar posibles escenarios de aplicación del problema JSP en el campo de la computación basados en la familia de instancias utilizadas.

1.4 IMPACTO

Uno de los objetivos del Grupo de Investigación en Informática Biomédica (GIIB) de la de la Escuela de ingeniería de Sistemas e Informática de la Universidad Industrial de Santander, es diseñar e implementar algoritmos basados en bioingeniería para dar soporte a proyectos desarrollados en sus áreas de investigación. Por lo tanto, este proyecto será una herramienta muy útil que apoyará futuros trabajos de investigación, para desarrollar aplicaciones que solucionen problemas de optimización combinatoria presentes a nivel informático, científico, industrial, entre otros ámbitos en que se requiera optimizar el uso de recursos, para minimizar costos, tiempos, etc.

Este proyecto es uno de los dos trabajos de grado que se desarrollaron de manera paralela, sobre diferentes técnicas bioinspiradas para apoyar el trabajo de maestría, “IMPLEMENTACIÓN DE ALGORITMOS BIOINSPIRADOS PARA LA SOLUCIÓN DEL PROBLEMA DE PLANIFICACIÓN DE TRABAJOS” que está siendo desarrollado por el Ingeniero Wilfredo Ariel Gómez Bueno.

2. MARCO TEÓRICO

2.1 OPTIMIZACIÓN COMBINATORIA

La optimización combinatoria (no lineal) es una rama de la optimización matemática, la cual es un área muy activa de las matemáticas aplicadas relacionadas con la Investigación de Operaciones (I.O.). El propósito de este estudio es obtener la mejor solución factible de un problema, tomando las decisiones más adecuadas para maximizar o minimizar la función objetivo según el problema, por ejemplo, aumentar la utilidad producida por una empresa o reducir los costos de producción.

Este propósito está presente en muchas actividades humanas, especialmente en actividades económicas, administrativas y militares, que buscan incrementar ganancias, desempeño o rendimiento, y disminuir perdidas, riesgo o costo. “La optimización combinatoria trata una clase de problemas en los que el número de soluciones candidatas es de tamaño combinatorio. Cada posible solución tiene un costo asociado y el objetivo consiste en encontrar la solución con el menor costo” [30], donde las variables son de naturaleza discreta y solo pueden tomar valores dentro de un conjunto de soluciones factibles que también es discreto. Si fueran variables continuas como en Optimización Numérica, las soluciones factibles podrían ser infinitas y estarían dentro de un intervalo fijado por las restricciones.

Encontrar la mejor solución posible del problema se vuelve muy difícil cuando la función objetivo depende de muchas variables de decisión, las cuales pueden tomar cualquier valor perteneciente a un espacio de búsqueda limitado por un amplio conjunto de restricciones a las que está sujeto el problema a resolver, por lo que en muchos casos solo se pueden encontrar soluciones casi óptimas a los complejos problemas de optimización combinatoria. Por ejemplo, si se tienen 5 personas para realizar 100 trabajos con diferente duración, analizar cada una de las combinaciones posibles sería muy dispendioso.

La Investigación de Operaciones genera modelos de donde se puede obtener la configuración optima de un sistema, mejorando el funcionamiento de redes de transporte, la gestión de redes de computadores, cadenas de suministro, energía, ingeniería financiera, manufactura, y toda área en que la toma de decisiones no puede depender exclusivamente de la experiencia de las personas.

En un modelo de optimización combinatoria se establece [3]:

La función objetivo que se pretende optimizar: minimizar[1] $f: S \rightarrow \mathbb{R}_0^+$

- Ω Un conjunto de Restricciones entre las variables definido por el problema a solucionar.
- *S* El espacio de búsqueda definida sobre un conjunto finito de variables de decisión discretas.

El espacio de búsqueda (o dominio) *S* se define como un conjunto de variables discretas $X_i,\ i = 1, \dots, n$, en donde $X_i \leftarrow v_i^j$ y los valores $v_i^j \in D_i = \{v_i^1, \dots, v_i^{|D_i|}\}$. Los elementos de *S* son asignaciones completas, es decir, asignaciones en las que cada variable X_i tiene un valor asignado v_i^j de su dominio D_i. El conjunto de soluciones factibles S_Ω está dado por los elementos de *S* que satisfacen todas las restricciones en el conjunto Ω.

Una solución $s^* \in S_\Omega$ se denomina óptimo global si y sólo si $f(s^*) \leq f(s)\ \forall_s \in S_\Omega$. El conjunto de todas las soluciones óptimas a nivel global se denota por $S_\Omega^* \in S_\Omega$. Resolver un problema de optimización combinatoria exige encontrar al menos una $s^* \in S_\Omega^*$.

[1] Maximizar sobre *f* es lo mismo que minimizar sobre *–f*, por consiguiente, cualquier problema de optimización combinatoria puede ser descrito como un problema de minimización.

Los problemas de optimización combinatoria se clasifican en varias grupos [31], entre los cuales están los problemas de asignación, como el Problema de Asignación Cuadrática (QAP) y el problema de organización de cursos en la universidad. También se tienen problemas de enrutamiento como el Problema de Ruteo de Vehículos (VRP). *Job Shop Scheduling Problem* está catalogado dentro de los problemas de programación (Scheduling), junto a otros problemas como *Flow Shop Problem* (FSP), *Open Shop Problem* (OSP) y *Single Machine Total Tardiness Problem* (SMTTP).

Para resolver estos problemas, la Investigación de Operaciones los separa en dos categorías:

- Determinísticos: son problemas en los que se conoce toda la información necesaria para obtener la solución exacta, ya que no existe ningún tipo de incertidumbre.
- Estocásticos (no determinístico): son problemas en los que se desconoce parte de la información, por lo tanto el resultado exacto es incierto y los métodos de solución se comportan de una manera probabilística. En esta categoría se encuentra OCH y GRASP, que tienen componentes aleatorios para buscar soluciones. Estos métodos heurísticos no pueden garantizar que la solución exacta sea encontrada, solo encuentran soluciones aproximadas al óptimo.

2.2 JOB SHOP SCHEDULING PROBLEM (JSP)

El Problema de *Job Shop Scheduling* (JSP) o problema de planificación de trabajos consiste en “acomodar los recursos en el tiempo para realizar un conjunto de trabajos” [6], construyendo un plan o secuencia de ejecución de *j* trabajos en un conjunto *m* de máquinas [13], donde una operación es cada trabajo que se

procesa en cada máquina (*Operación(j,m)*), y tiene asignado un tiempo de procesamiento determinado.

Este problema se presenta en múltiples actividades humanas, desde la sencilla organización que hace una persona (de su tiempo) en su agenda semanal, hasta problemas complejos como la designación de horarios a los estudiantes de una universidad. Tiene aplicaciones en labores que requieren de una gestión precisa del tiempo (con estrictas fechas de entrega), como la programación de horarios para la entrega de paquetes (vía aérea p.ej.), en la planificación de los despegues y aterrizajes de aviones a través de las pistas de un aeropuerto, envío de datos en una red de computadores, computación (multitarea y multiprocesamiento), gestión de proyectos (plan o cronograma), procesos de producción y administración (en líneas de ensamblaje p.ej.), etc. [20].

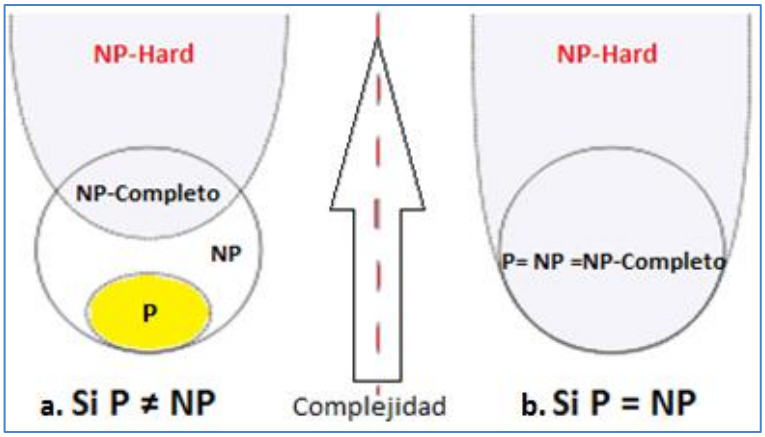
En JSP se debe cumplir con ciertas restricciones en la ejecución de los trabajos y el objetivo es concluir todos los trabajos (en conjunto) en el menor tiempo posible. Ese tiempo a optimizar conocido como makespan (C_{MAX}) o Flujo de Trabajo Máximo, conforma la función objetivo (a minimizar): $C_{MAX} = max_{i=1\ldots n}(C_i)$, donde C_i es el tiempo de finalización del trabajo J_i. Es un problema de optimización combinatoria porque el número de soluciones candidatas es de tamaño combinatorio con variables de naturaleza discreta, por tanto las soluciones son permutaciones (con repetición) de las operaciones de cada máquina, siendo imposible determinar todas las posibles soluciones en un tiempo razonable.

Se pueden incluir otros objetivos como la minimización del retardo de todos los trabajos (total tardiness), tiempo de flujo total (total flow time), por lo que puede ser un problema de optimización multi-objetivo [46]. Aunque no sea un objetivo oficial del presente trabajo, de forma indirecta se reducen los retardos al reducir el makespan.

2.2.1 Complejidad Computacional del JSP

En 1976, Michael Garey proporcionó una prueba de que este problema es NP-hard para m>2 [17], es decir, puede no encontrarse rápidamente (en tiempo polinómico) una solución óptima para JSP con más de dos máquinas. Junto con David Johnson en 1979, terminaron de demostrar que JSP es un problema NP-hard [18], a menos que en la Teoría de la Complejidad Computacional se demuestre que P=NP [19], si es así (Ver Figura 1b), cualquier problema que pueda ser verificado rápidamente por una computadora, también podría ser rápidamente resuelto por dicha computadora.

Figura 1: Diagrama de Euler de las familias de problemas P, NP, NP-completo, y NP-hard



Esta cuestión se conoce como el problema P versus NP [29], y es uno de los siete Problemas del Milenio que no se han resuelto. La postura generalizada entre los matemáticos es que P≠NP (Ver Figura 1b), por lo tanto, existen problemas complejos que no tendrán solución exacta en tiempo polinómico, así que los mejores algoritmos ejecutados en las computadoras actuales (más poderosas) solo pueden obtener soluciones aproximadas, que podrían ser la mejor solución, pero no se puede tener certeza de esto sin realizar todas las innumerables soluciones posibles. En ese aspecto radica la complejidad NP-hard de JSP, porque es imposible realizar todas las combinaciones posibles en un tiempo

razonable. Cada secuencia de operaciones en una máquina puede ser permutada independientemente de la secuencia de operaciones de otra máquina, por lo que con unos cuantos trabajos y máquinas puede tener $(j!)^m$ soluciones posibles, que corresponde al espacio de búsqueda (S) del problema.

Para un problema de solo una máquina (o recurso) y un total de 30 operaciones por realizar:

$Comb(30)(1) = (30!) = 2.65 * 10^{32}$ posibles soluciones

Que es un número enorme de combinaciones si se compara con la edad del universo que está estimada en:

$(14 * 10^9) * 365.25 * 24 * 60 * 60 = 4.42 * 10^{17}\ seg.$

JSP es un problema semejante al famoso problema del vendedor viajero (TSP), porque el TSP puede compararse con JSP al tener que el vendedor es el trabajo y las ciudades son las máquinas, y el vendedor tiene que completar todas las operaciones, visitando cada ciudad en el menor tiempo posible.

2.2.2 Definición Formal del JSP

El modelo cualitativo del problema se define así [5]:

Sea:
$J = \{j_1, j_2, j_3, \dots, j_n\}$: Conjunto de *n* trabajos a ser procesados
$M = \{m_1, m_2, m_3, \dots, m_m\}$: Conjunto de *m* recursos o máquinas
$O_{ik} = \{o_{i1}, o_{i2}, o_{i3}, \dots, o_{ik_i}\}$: Operación del trabajo J_i que debe ser procesado en la máquina M_k por un τ_{ik}
$\tau_{ik} = \{\tau_{i1}, \tau_{i2}, \tau_{i3}, \dots, \tau_{ik_i}\}$: Periodo ininterrumpido de tiempo de procesamiento de cada operación.

Función objetivo: Minimizar $C_{MAX} = max(\ t_{ik} + \ \tau_{ik}) : \forall \ \ (\ J_i \ \in J \quad y \quad M_k \in \ M\ )$

Sujeto a:

Restricción de tiempos de inicio de cada operación $\ t_{ik} \geq 0$

Restricción de precedencia $\ t_{ik} \ - \ \ t_{ih} \ \geq \ \tau_{ih}$ si O_{ih} precede a O_{ik} .

Restricción disyuntiva $\ t_{pk} - \ t_{ik} \ + K\big(1 - \ y_{ipk}\ \big) \geq \tau_{ik} \quad y_{ipk} = 1$, si O_{ik} precede a O_{pk}, $\ t_{ik} \ - \ \ t_{pk} \ + K\big(\ y_{ipk}\ \big) \geq \tau_{pk} \quad y_{ipk} = 0$ en otro caso.

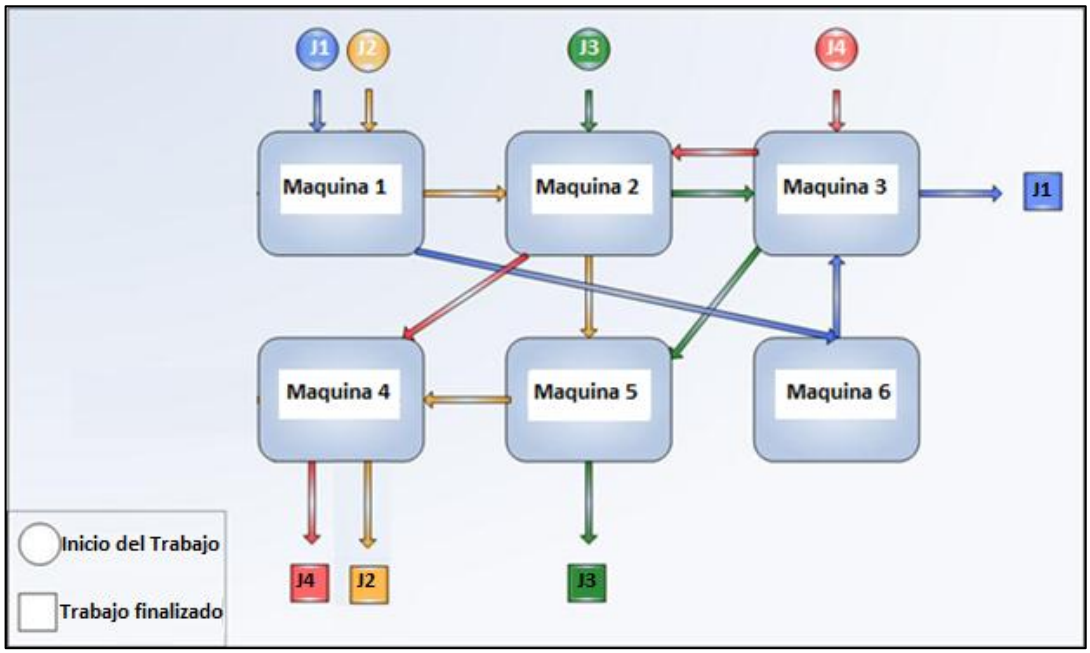
Donde: $\{i, p\} \ \in J \ \ y \ \ \{k, h\} \in M$. Con K > $\sum_{i=1}^{n}(\ \sum_{k=1}^{m} \tau_{ik} - min(\tau_{ik})\ )$

El conjunto anterior de restricciones (Ω) del JSP se explica así [21]:

- Restricción de inicio: El tiempo en que empieza cada operación no se especifica, por lo tanto un trabajo puede empezar en cualquier instante de tiempo siempre y cuando esté disponible la máquina requerida.
- Restricción de precedencia: Cada trabajo debe atravesar una secuencia particular de operaciones que esta predefinida, por lo que las operaciones no podrán iniciar hasta finalizar su antecesora, impidiendo procesar dos operaciones del mismo trabajo simultáneamente, es decir, no se permite el procesamiento en paralelo de un trabajo. En la Figura 2 se establece la secuencia que debe llevarse a cabo para procesar correctamente cuatro trabajos, pasando en estricto orden por tres de las seis máquinas del sistema.
- Restricciones Disyuntivas: Una máquina puede procesar sólo un trabajo a la vez. Cada operación deberá ser procesada en su totalidad en una única máquina, no puede ser interrumpida aunque haya trabajos esperando que esa máquina esté disponible. Es decir, ningún trabajo podrá ser procesado más de una vez en la misma máquina.

Además de las anteriores restricciones, se tiene determinado que todas las operaciones tienen la misma prioridad de procesamiento, y todas las máquinas son iguales y pueden estar ociosas en cualquier momento.

Figura 2: Secuencia del procesamiento de 4 trabajos en 6 máquinas



En este trabajo se resolverán (en principio) instancias JSP muy utilizadas conocidas como instancias LA, que fueron planteadas por Lawrence [11]. En estas instancias definidas mediante una matriz [15], cada fila corresponde a un trabajo (J1, J2 y J3), donde se establece su orden o secuencia de ejecución en las máquinas, y al lado de cada máquina se muestra el tiempo de procesamiento de la respectiva operación. En la Tabla 1 se tiene una instancia del problema de planificación de trabajos de tamaño 3x3 con el flujo de máquinas en cada trabajo.

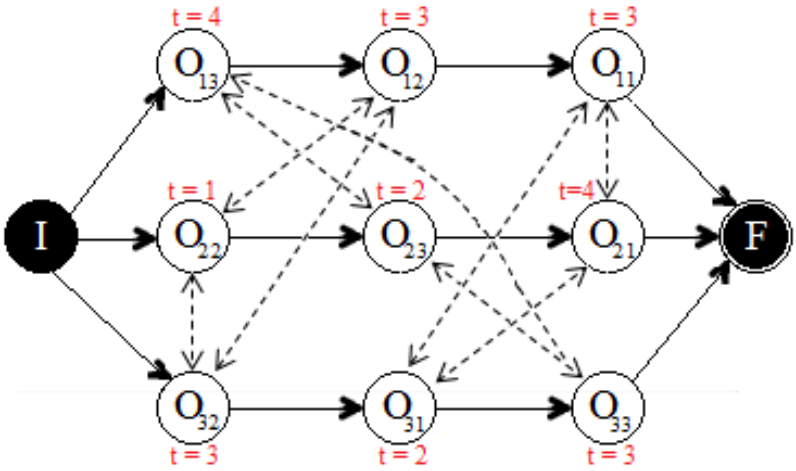
Tabla 1: Instancia del JSP de 3×3

Trabajo (J)	**Máquina (*tiempo*)**		
Secuencia:	*S1*	*S2*	*S3*
J1	3 (4)	2 (3)	1 (3)
J2	2 (1)	3 (2)	1 (4)
J3	2 (3)	1 (2)	3 (3)

Representación del problema

El modelo cualitativo de JSP se representa normalmente como un grafo disyuntivo $G = (V, C \cup D)$ [14], donde V es el conjunto de nodos que en este problema son las *Operaciones*(*trabajo*, *máquina*), con excepción de los nodos de inicio (I) y final (T) del grafo. C es un conjunto grafos dirigidos (→) que unen operaciones correspondientes al mismo trabajo (secuencia tecnológica) y D es un conjunto de grafos no dirigidos (⇠⇢) que conectan operaciones que se ejecutan en una misma máquina (Ver Figura 3). Además se coloca el tiempo de procesamiento de cada operación en la parte superior del nodo. En la Figura 3 se representa el problema definido en la instancia anterior (Ver Tabla 1).

Figura 3: Grafo de la instancia 3×3 de la Tabla 1.

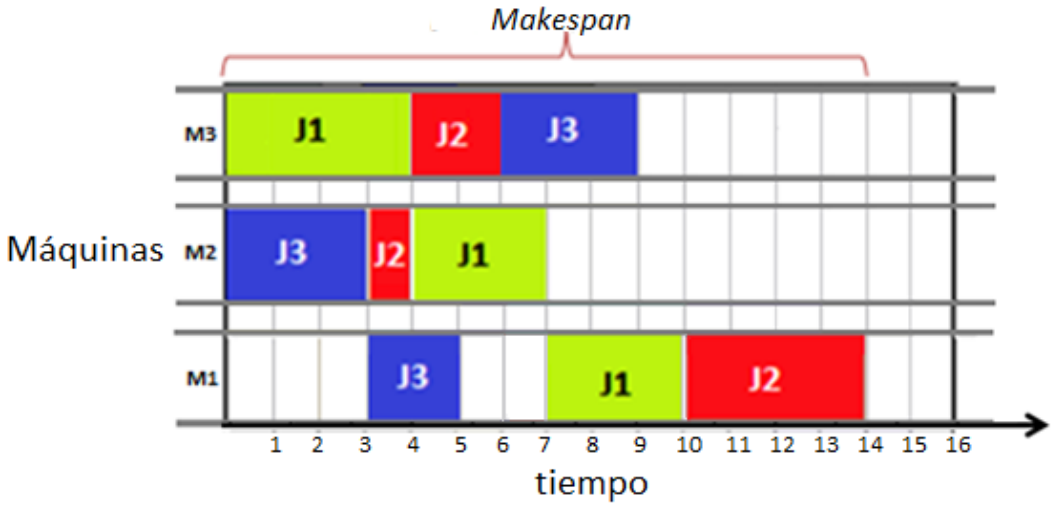


Representación de la solución

El cumplimiento de las restricciones del problema se puede ver claramente mediante un diagrama de Gantt (Ver Figura 4), que representa una de las soluciones factibles para la instancia JSP de la Tabla 1. En el diagrama se muestra el tiempo de ejecución (eje horizontal) de las máquinas (eje vertical), cada una con una fila (renglón) correspondiente, entre las que se ubican las celdas (cuadros) que muestran el trabajo que se realizó en cada intervalo de tiempo, de

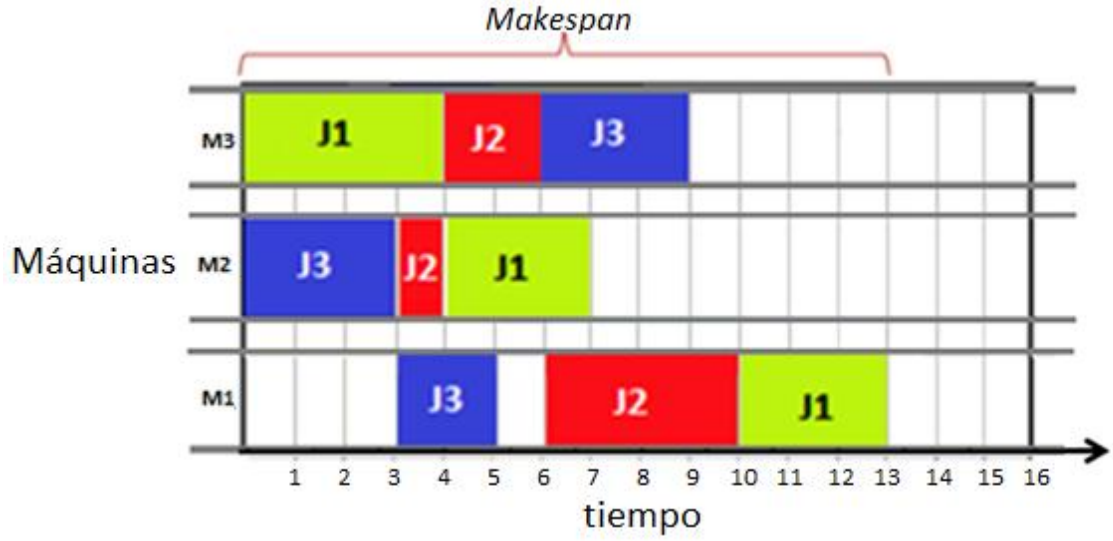
donde se pueden ver las operaciones realizadas en cada máquina, por ejemplo, se puede ver que en el intervalo [0,4] de la máquina 3 se realizó el trabajo J1, es decir la operación O_{13}.

Figura 4: Diagrama de Gantt que muestra un plan de trabajo



Se puede obtener una mejor solución para esa instancia si en la máquina 1, el trabajo J2 se ejecuta antes que J1 (Ver Figura 5), con lo cual se reduce el makespan en una unidad de tiempo. Para asegurarnos de que esta es la solución óptima, se deben realizar todas las combinaciones posibles, algo que es muy difícil para instancias de gran tamaño. En este caso, la Figura 5 es la solución óptima para la instancia de JSP de la Tabla 1, porque se logró el makespan mínimo que es de 13 unidades de tiempo, con retardos solo en la máquina 1 (de 4 unidades de tiempo) porque al estar ejecutándose los trabajos J3 y J2 en las otras máquinas, se debe esperar para no violar la restricción que impide operaciones simultaneas del mismo trabajo.

Figura 5: Diagrama de Gantt con plan de trabajo óptimo.



El problema de *Job Shop Scheduling* ha sido abordado con métodos que solo pueden resolver instancias de un número limitado de operaciones, porque realizan búsquedas exhaustivas para encontrar la solución exacta, como Branch and Bound (B&B) propuesto en 1960 [23], solo puede resolver problemas de hasta 15 x 14, es decir, máximo 220 operaciones [24]. Por lo que se deben utilizar métodos aproximados (Ver Tabla 2) como Recocido Simulado (SA), Búsqueda Tabú (TS) [10], Búsqueda Local Iterativa (ILS), GRASP, OCH, Algoritmos Evolutivos (EA) como el algoritmo Cultural (CULT), los algoritmos Genéticos (GA) o el Sistema Inmune Artificial (AIS), entre otros [9].

Tabla 2: Características principales de las metaheurísticas [9]

Metaheurística	Características
SA	Criterio de aceptación Horario de enfriamiento
TS	Elección de vecino (lista tabú) Criterio de aspiración
EA	Recombinación Mutación Selección
ILS	Búsqueda local Movimiento inicial Criterio de aceptación
OCH	Construcción probabilística Actualización de feromona
GRASP	Búsqueda local Lista de Candidatos Restringida (RCL)

2.3 OPTIMIZACIÓN BASADA EN COLONIAS DE HORMIGAS (OCH)

La *Optimización basada en Colonias de Hormigas* (OCH) o *Ant Colony Optimization* (ACO), es una metaheurística que aplica conceptos de campos como la Inteligencia Artificial y la Biología, aprendiendo de la forma en que seres vivos resuelven problemas específicos presentes en su entorno. Esta técnica pertenece a una de las ramas de la Inteligencia Artificial conocida como Inteligencia Computacional, que se basa en sistemas complejos de la naturaleza. De esta rama se deriva la Inteligencia de Enjambre, inspirada en el comportamiento colectivo de sistemas descentralizados y auto-organizados como las colonias de hormigas. OCH emula la inteligencia emergente de las relaciones entre las hormigas, cuando buscan el camino más corto entre el hormiguero y la comida, que es equivalente al problema de JSP.

Inspiración en la biología de las hormigas

Los sistemas biológicos auto-organizados como las complejas estructuras sociales de las hormigas, abejas y muchas manadas de mamíferos (ñus, cebras), o el comportamiento en masa de las bandadas de pájaros y cardúmenes de peces, han sido la inspiración de muchas heurísticas, entre las que sobresalen OCH y *Particle Swarm Optimization* (PSO) [34].

"La auto-organización en sistemas biológicos es un proceso en el que patrón a nivel global de un sistema surge únicamente de numerosas interacciones entre los componentes de bajo nivel del sistema. Además, las reglas que especifican las interacciones entre los componentes del sistema se ejecutan utilizando sólo la información local, sin referencia al patrón global" [32]. Las organizaciones jerárquicas tienen un orden central o liderazgo único, en cambio, en las organizaciones horizontales como las colonias de hormigas no se tiene este tipo de dirección y control, los planes surgen del sistema desordenado (sin planificación) a través de las fluctuaciones aleatorias que son amplificadas por un bucle de realimentación positiva, es decir, si una hormiga realiza pequeños cambios en una estructura compartida, induce a otras hormigas a seguir ese proceso, y estas influyen más al resto de hormigas.

Se afirma que las colonias de hormigas son modelos ideales, con una organización descentralizada o sistema distribuido sobre todos sus componentes, que les permite tomar decisiones más adecuadas y adaptarse rápidamente a cambios sustanciales en el ambiente, logrando sobrevivir durante más de 110 millones de años, hoy conforman el 20% (aprox.) de la biomasa de todos los animales terrestres, que sería similar a la biomasa total de todos los seres humanos (7.000 millones).

Figura 6: Hormigas recolectoras

Fuente: Kim Taylor [44]

Las hormigas perciben olores con sus antenas, con las cuales determinan la dirección y la intensidad de los olores. En una población de hormigas que recolecta comida de forma cooperativa, las hormigas se comunican por medio de una hormona llamada feromona [33], que dejan cuando vuelven al hormiguero con alimentos (Ver Figura 6). Cuando se agota la fuente de alimento ya no van dejando el rastro, y las feromonas se disipan lentamente.

Figura 7: Hormigas siguiendo el mejor camino

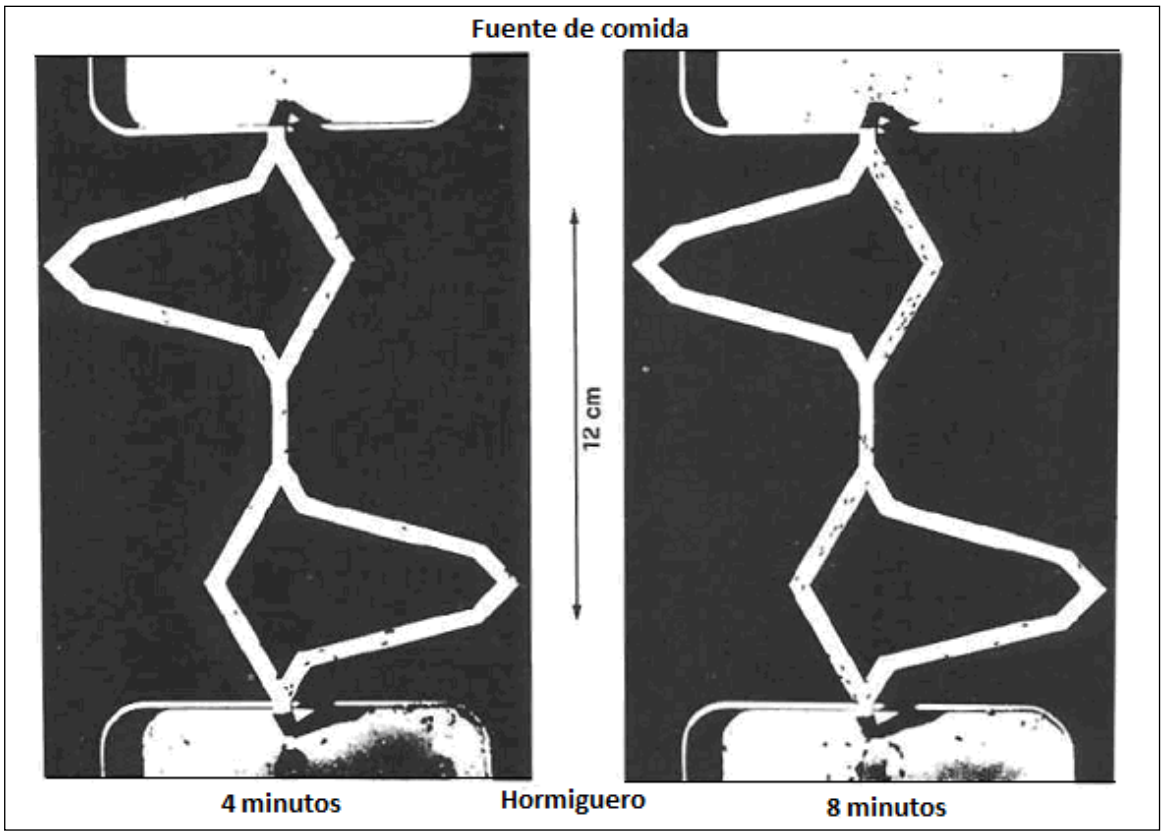
Fuente: Daily Mail [43]

Cada hormiga por si misma puede perderse al no encontrar rastros de feromona, porque tienen una memoria e inteligencia muy limitada por su cerebro de tan solo 250.000 neuronas aproximadamente [41], mientras que un ser humano tiene miles de millones. Sin embargo, una numerosa colonia de hormigas tendría colectivamente un cerebro tan grande como el de cualquier mamífero, que les permite encontrar el mejor camino (Ver Figura 7).

Inteligencia de enjambre

En un experimento sobre la auto-organización de las hormigas Argentinas realizado en 1989 [12], se observó el comportamiento de alimentación de la colonia de hormigas, que lograron encontrar las ramas más cortas de un puente entre el hormiguero y la comida (Ver Figura 8), por medio del rastro de feromona que van dejando al moverse. Las hormigas salen a recolectar comida, moviéndose inicialmente en forma aleatoria al no existir rastros de feromona, y a lo largo de su camino de regreso a la colonia depositan la feromona. Por los caminos cortos aumenta rápidamente la cantidad de feromona porque las hormigas tardan poco tiempo para realizar un recorrido (ida y vuelta). Cuando una hormiga encuentra rastros de feromona, sigue el camino con más feromona, aumentando aún más la cantidad acumulada, lo que estimula a otras hormigas a seguir esa trayectoria. Con el paso del tiempo el rastro de la feromona se va evaporando, reduciendo la fuerza atractiva de los caminos poco frecuentados. Al final, las hormigas solo siguen las trayectorias más rápidas, lo que provoca la convergencia a una solución localmente óptima que será la única trayectoria que finalmente siguen la mayoría de hormigas.

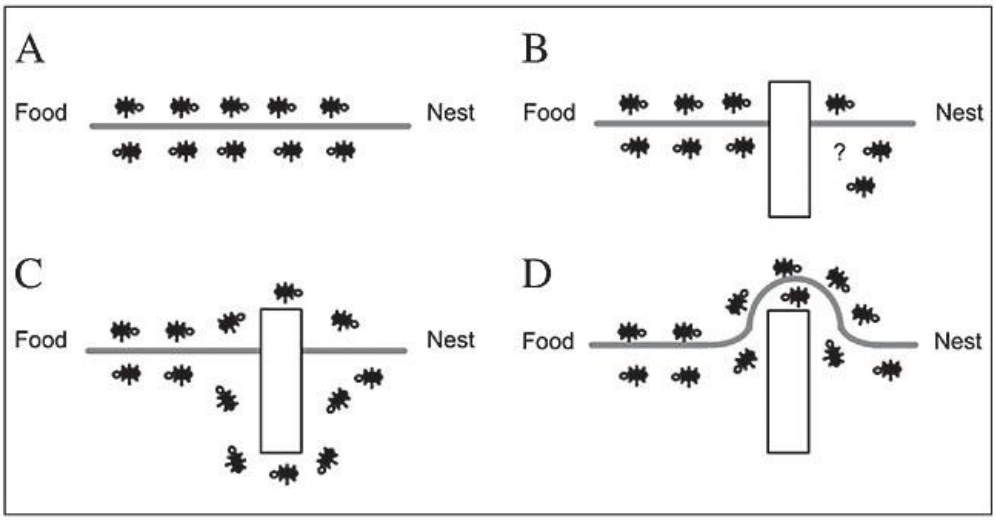
Figura 8: Fotografía de hormigas que demuestra su *inteligencia de enjambre*



Fuente: Goss et al. [12]

El comportamiento adaptativo de la hormigas se observa cuando el camino más corto entre la comida y el hormiguero queda bloqueado por un obstáculo (Ver Figura 9B), las hormigas empiezan a explorar nuevas rutas, dejando rastros nuevos de feromona. Las rutas más cortas son seguidas por más hormigas, reforzando el rastro de feromona y reencontrando de manera gradual el mejor camino (Ver Figura 9D) [12].

Figura 9: A. Hormigas en un rastro de feromona entre el hormiguero y la comida; B. un obstáculo interrumpe el rastro; C. las hormigas encuentran dos caminos para pasar alrededor del obstáculo; D. un nuevo rastro de feromona se forma a lo largo del camino corto



Fuente: Perretto y Lopes [60]

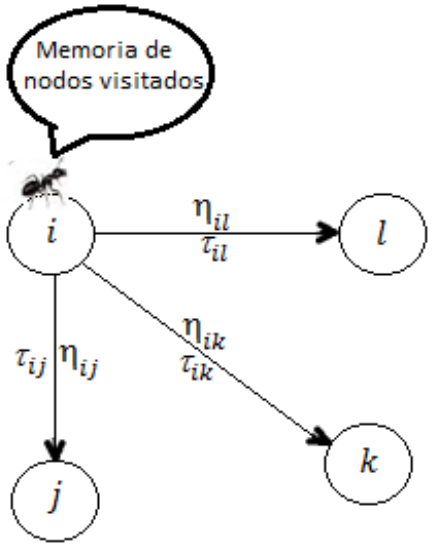
El comportamiento de las hormigas se simula con un agente virtual que tiene la capacidad de explorar un espacio de búsqueda limitado y obtener información acerca del entorno que lo rodea. Las hormigas artificiales (k) caminan sobre un grafo que representa el problema a solucionar, donde cada arista les permite moverse del nodo origen i al nodo destino j determinado según la información disponible localmente en cada arista (ecuación probabilística) y las restricciones (reglas de transición), realizando una búsqueda estocástica para construir paso a paso una solución que se va guardando en la memoria privada de cada hormiga llamada lista $Tabu_k$ (que almacena información sobre la secuencia de nodos o ruta seguida hasta el momento t), que termina cuando alcanza alguno de los estados de aceptación determinados por las restricciones y el objetivo del problema.

Así, en cada iteración, las hormigas pueden construir soluciones aproximadas a problemas complejos (como los de asignación, secuenciación, planificación o programación) hasta que se cumpla un criterio de terminación. Cada arista del

grafo (Ver Figura 10) tiene asociada dos tipos de información que guían el movimiento de la hormiga [4] y cuyos valores son modificados por las hormigas en cada iteración:

- η_{ij} Información heurística del espacio de exploración, mide la preferencia heurística de moverse desde el nodo i hasta el nodo j, al recorrer la arista a_{ij}. Las hormigas no modifican esta información durante la ejecución del algoritmo.
- τ_{ij} Información de los rastros de feromona artificiales, que mide la "deseabilidad aprendida" del movimiento de i a j. Esta información se modifica durante la ejecución del algoritmo dependiendo de calidad de las soluciones encontradas por las hormigas para reflejar la experiencia adquirida por estos agentes.

Figura 10: Información disponible para seleccionar el siguiente nodo [16]



Pseudocódigo de la metaheurística OCH [3]**:**

$Procedimiento\ OCH$

$Establecer\ parámetros, Inicializar\ los\ rastros\ de\ feromona$

$Actividades\ programadas$

$Construcción\ de\ soluciones\ por\ hormigas$

$Servidor\ de\ acciones\ (Opcional)$

$Actualización\ de\ feromona$

$Fin - Actividades\ programadas$

$Fin - Procedimiento$

La metaheurística se compone de un paso de inicialización de parámetros y de tres procedimientos algorítmicos cuya activación está regulada por el constructor *Actividades programadas*, que se repite hasta que un condición de terminación se cumple, como alcanzar un número de iteraciones máximo o un tiempo máximo de la CPU. Los tres procedimientos algorítmicos sometidos al bucle de *Actividades programadas* consisten en [25]:

1) Construcción de soluciones por hormigas: Es la construcción probabilística de soluciones por todas las hormigas de una colonia, que visitan estados adyacentes de un problema considerado. Las hormigas pueden moverse aplicando una política de decisión estocástica usando la información de los rastros de feromona y la información heurística, con lo cual las hormigas construyen incrementalmente una solución al problema.

2) Servidor de acciones: Son acciones centralizadas que modifican el comportamiento del algoritmo y que no pueden ser desarrolladas por las hormigas en forma individual. La más común es la optimización local o mejora de las soluciones con la aplicación de un algoritmo de búsqueda local. Las soluciones optimizadas a nivel local luego se utilizan para establecer los valores de feromonas para actualizar.

3) Actualización de feromona: Es el proceso que actualiza los rastros de feromona en cada arista a_{ij}, denominado *actualización en línea a posteriori* o *fuera de línea* porque se realiza al finalizar un camino o ruta completa. La cantidad de feromona que deposita cada hormiga en las aristas, depende de la longitud total del camino recorrido (ecuación 3). También se puede realizar una *actualización en línea de los rastros de feromona paso a paso*, que es una actualización local o en "tiempo real" de la feromona [1], realizada cuando una hormiga se mueve desde el nodo *i* hasta el nodo *j.* El valor del rastro de feromona se reduce por medio de una constante de evaporación de feromona, lo que evita una convergencia prematura del algoritmo al ir descartando las aristas menos frecuentadas.

En la literatura se han propuesto diversos algoritmos que siguen la técnica probabilística de OCH, para encontrar soluciones aproximadas a problemas complejos de optimización. El primer algoritmo OCH fue el *Ant Sytem* (AS), propuesto por Marco Dorigo en 1991 [26], terminado con los aportes de Maniezzo y Colorni [1]. Nuevos desarrollos dieron mejores resultados, como el *Ant Colony System* (ACS) [2], el *Max-Min Ant Sytem* (MMAS) [7], el *Rank-based Ant System* (AS_{rank}) [8], entre otros. En este artículo se presenta una variante del *Elitist Ant System* (EAS), propuesto también por Dorigo como una mejora al SH [1].

Los algoritmos de OCH han producido soluciones subóptimas con gran eficiencia al Problema del viajante de comercio (TSP), a problemas de enrutamiento de redes y sistemas urbanos del transporte, entre otros, porque se adaptan continuamente a los cambios en tiempo real.

2.4 GRASP (PROCEDIMIENTO DE BÚSQUEDA MIOPE ALEATORIA Y ADAPTATIVA)

Esta metaheurística es un método aproximado para problemas de optimización combinatoria [36]. GRASP implementa una búsqueda local multi-arranque, iniciando desde una solución básica diversa y de buena calidad, para asegurar el éxito en los métodos de búsqueda local de manera iterativa.

GRASP construye soluciones probabilísticas al igual que OCH, por medio de dos fases que se repiten numerosas veces se encuentra la mejor solución (aproximada) [35]**.**

Empieza con una fase de construcción que busca equilibrar (combina) un componente aleatorio con uno miope, para encontrar una solución inicial de la cual partirá la segunda fase del algoritmo. En cada paso se tiene un conjunto de soluciones candidatas (de máximo |J| elementos) definido por las restricciones, ordenando los elementos de según alguna información heurística que tiene en cuenta la solución parcial que ya está disponible (componente adaptativa), que determina los componentes más prometedores (componente miope o voraz) con una función Greedy para cada nodo candidato, la cual encuentra el incremento del valor del makespan “parcial” al incluir el nodo candidato en la solución.

Mediante un parámetro alfa (que controla el equilibrio entre el componente aleatorio y el miope) se crea un intervalo que limitará los nodos candidatos que son seleccionables para entrar en la Lista Restringida de Candidatas (RCL). Todos los elementos que quedaron en la lista tienen la misma probabilidad de ser seleccionados sin importar su valor en la función Greedy (componente aleatoria), de donde se escoge aleatoriamente el nodo a insertar en la nueva solución.

En la segunda fase, la búsqueda local prueba aleatoriamente con uno de los otros nodos de RCL para tratar de mejorar la solución inicial, utilizando la misma regla

de decisión miope aleatoria anterior se mantiene a RCL con los mismos nodos de la fase de construcción. Se busca en la vecindad de la solución inicial, y al encontrar una mejor solución, esta se guarda y se explora su subespacio de vecindad en un nuevo ciclo de mejora.

Alfa es un valor entre el intervalo (0,1), impidiendo que alfa sea 1 porque sería un algoritmo totalmente aleatorio al entrar todos los nodos candidatos a la lista RCL (no excluye a ningún nodo por lo tanto la función greedy no guía el proceso, daría soluciones muy diversas de poca calidad), o cero porque sería miope al entrar solo el mejor nodo de los candidatos a RCL (que es el de menor valor en la función Greedy).

Pseudocódigo de la metaheristica GRASP [42]:

$Procedimiento\ GRASP\ (Max\_Iteraciones, Semilla)$
$Establecer\ parámetros\ (\alpha)$
$Constructor\_Soluciones$
$Solución \leftarrow Construcción\_Miope\_Aleatoria(Semilla)$
$Solución \leftarrow Búsqueda\_Local(Solución)$
$Actualizar\_Solución\ (Solucion, Mejor\_Solución)$
$Fin - Constructor\_Soluciones$
$Fin - Procedimiento\ GRASP$

Con la descripción anterior se puede ver que los algoritmos GRASP y la OCH difieren en que la OCH usa una forma explícita de memoria que son los rastros de feromona, los algoritmos GRASP no consideran ese tipo de información para guiar el proceso de construcción. Además, la OCH no usa la información heurística adaptativa (dinámica) que es dependiente de la solución parcial y que siempre utiliza GRASP.

Este algoritmo difícilmente encuentra la solución óptima porque depende mucho de la solución inicial obtenida en la fase de construcción. Es esencial que esta sea de buena calidad y muy diversa para que la búsqueda local encuentre el óptimo global.

3. ESTADO DEL ARTE

3.1 MÉTODOS DE SOLUCIÓN DEL JSP

Existen métodos que dan soluciones con resultados exactos con escasa eficiencia al realizar búsquedas exhaustivas, y métodos aproximados que solucionan los problemas en menos tiempo y con menor utilización de recursos computacionales (Ver Figura 11).

3.1.1 Métodos exactos

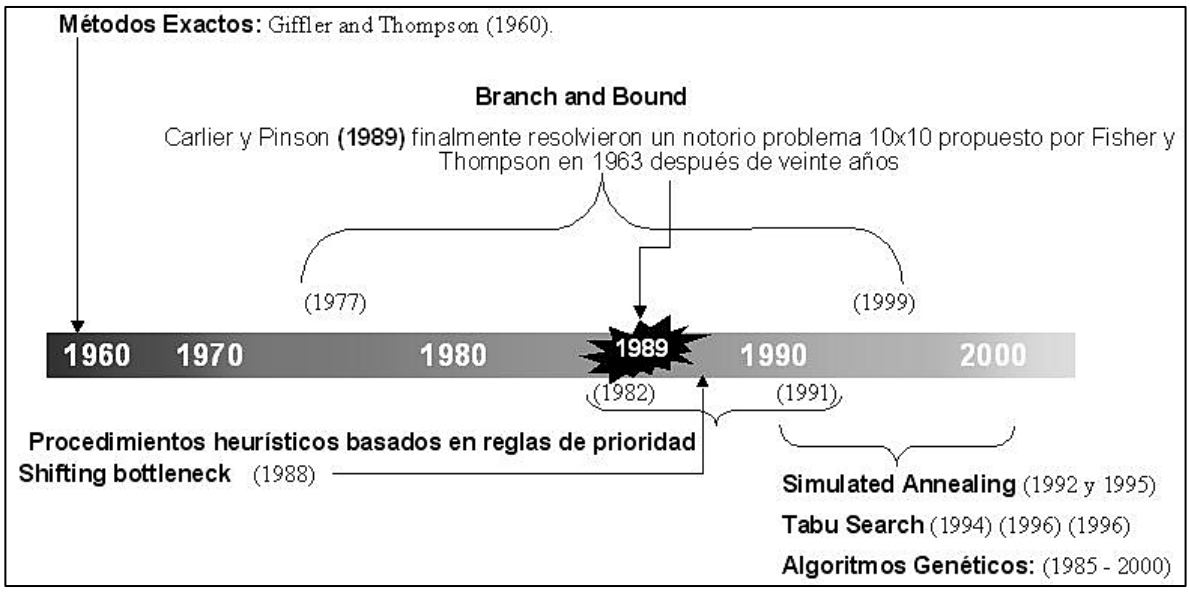
Vuelta atrás (Backtracking): realiza una búsqueda en profundidad dentro del grafo dirigido para encontrar soluciones al problema, por lo que va generando un árbol de búsqueda en el que construye soluciones parciales que limitan las regiones en las que se puede encontrar un estado de aceptación. Si no puede encontrar una solución, retrocede eliminando los elementos que se hubieran añadido en cada paso. Cuando llega a un nodo que tiene algún nodo vecino sin explorar, reanuda la búsqueda de una solución.

Ramificación y poda (B&B, Branch and Bound): propuesto por Land y Doig en 1960 [39], es una variante del Backtracking, en el que cada solución tiene asociado un costo y la solución que se busca es una de mínimo costo llamada óptima. Además de ramificar (branch) una solución padre en hijos, se trata de eliminar de consideración los hijos cuyos descendientes tienen un costo que supera al óptimo, acotando (bound) el costo de los descendientes del hijo. Esta "poda" reduce el tiempo de búsqueda, cortando las ramas del árbol que tienen descendientes costosos.

3.1.2 Métodos aproximados

Encuentran una buena aproximación al valor óptimo de problemas con un espacio de búsqueda muy amplio. En la Figura 11 se puede ver como los métodos exactos como B&B han sido reemplazados por metaheurísticas que encuentran soluciones con más eficiencia.

Figura 11: Diagrama de tiempo de métodos que han abordado el JSP



Fuente: Emanuel Téllez [40]

Algoritmos Evolutivos (EA) [51]: Están inspirados en la evolución de los seres vivos, por lo que utiliza mecanismos de selección y combinación. Los algoritmos más reconocidos de esta técnica son los *algoritmos Genéticos (GA)* [52], en que a partir de una población inicial de cromosomas (solución) se busca encontrar la más apta (solución con el mejor costo en la función objetivo) en el transcurso de varias generaciones, por medio de recombinaciones (crossover) de los cromosomas, mutaciones aleatorias de sus genes y selección de los cromosomas que sobrevivirán para producir la siguiente generación. Al igual que otras metaheurísticas, su éxito en JSP depende de soluciones iniciales adecuadas.

Otro algoritmo de computación evolutiva es el *Sistema Inmune Artificial SIA* (Artificial Immune System, AIS), inspiradas en el sistema inmune de los seres vivos, en el que las células aprenden y memorizan la forma de superar dificultades. "Los Sistemas Inmune Artificial son sistemas adaptativos, inspirados en la inmunología teórica y las funciones inmunes observadas, principios y modelos, que se aplican a la resolución de problemas" [59].

Otras metaheurísticas realizan un mejoramiento iterativo de la solución inicial: *Búsqueda Local* (Local Search, LS): este algoritmo realiza una búsqueda en la vecindad de una solución inicial (que puede ser obtenida por diferentes métodos o aleatoriamente). Realiza solo modificaciones que mejoran la solución actual, hasta encontrar la solución óptima o se agote el tiempo máximo de ejecución. Sólo requiere de unas cuantas especificaciones como una función de evaluación y un método eficiente para explorar entornos. Este método determinístico y sin memoria, puede encontrar soluciones rápidamente pero su solución final depende fuertemente de la solución inicial para no estancarse en óptimos locales.

Para evitar los óptimos locales se pueden utilizar otras metaheurísticas iterativas como *Búsqueda Local Iterativa (ILS)* [53]. ILS realiza pequeños cambios sucesivamente para explorar el vecindario de la solución actual y comprobar si hay alguna mejor que haya quedado oculta por la actual, y vuelve a empezar desde diferentes configuraciones iniciales aleatorias para diversificar la exploración, por lo que es un algoritmo de un alto consumo de recursos computacionales.

Busqueda Tabú (Tabu Search, TS) [54]: es un algoritmo determinístico, que en cada iteración puede aceptar soluciones de mayor costo para explorar otras áreas del espacio de búsqueda, teniendo en cuenta una lista tabú que prohíbe algunos movimientos repetidos. Actualmente es la mejor metaheurística para resolver JSP, pero requiere partir de una solución inicial adecuada para encontrar el óptimo eficientemente.

Recocido Simulado (SA, Simulated Annealing) o enfriamiento simulado es un algoritmo de búsqueda estocástico sin ningún tipo de memoria [55], inspirado en el proceso de recocido en Metalurgia. En este proceso se calienta un material (como el acero) hasta una temperatura específica y luego se enfría progresivamente hasta la temperatura ambiente, para modificar sus propiedades físicas (como ductilidad, tenacidad, etc.) y obtener el óptimo global (mínimo de energía). El calor causa que los átomos aumenten su energía y así pueden desplazarse fácilmente de sus posiciones iníciales, con lo que logran explorar una gran porción del espacio de búsqueda. El enfriamiento lento les da mayores probabilidades de pasar a nodos vecinos que tienen menos energía, hasta aproximarse al mínimo global de energía del sistema. Es una búsqueda local prácticamente aleatoria, por lo que es poco eficiente en problemas de optimización combinatoria.

3.2 ALGORITMOS DE OPTIMIZACION BASADOS EN COLONIAS DE HORMIGAS

La Optimización basada en Colonias de Hormigas se fundamenta en trabajos previos sobre el comportamiento colectivo de las hormigas de biólogos como Deneubourg y Pasteels en 1983 [22], quienes junto a Goss y Aron estudiaron la auto-organización entre las hormigas que les permite hallar el camino más corto entre el nido y la comida [12]. Con esto, Ebling, et al. implementaron un modelo de hormigas en el TWOS (Time Warp Operating System) [37].

La primera propuesta de OCH fue realizada en 1992 por Marco Dorigo, denominado Ant System (AS) [26], que fue implementado con ayuda de Maniezzo y Colorni [1]. Este se caracteriza principalmente por actualizar la feromona para todas las hormigas que terminan un plan, la cantidad depositada en las aristas del plan es inversamente proporcional a la longitud del plan.

Dorigo propuso una variante para el Ant System conocida como *Elitist Ant System* (EAS) [1], en la que se aumenta la cantidad de feromona depositada de la mejor solución encontrada en cada iteración (se deposita más feromona utilizando todas las otras hormigas).

En 1997, Dorigo y Gambardella desarrollaron el Ant Colony System (ACS) [2], que fue una gran mejora del AS. Empezando por la regla de decisión utilizada por las hormigas durante la etapa de construcción, que en ACS es pseudoaleatoria. La probabilidad de que una hormiga para moverse de un nodo a otro depende de una variable aleatoria uniforme q (entre [0,1]) y un parámetro q_0. Si $q \leq q_0$ entonces, entre los componentes viables, se escoge el componente que maximiza el producto $\left(\tau_{ij}\right)^{\alpha} * \left(\eta_{ij}\right)^{\beta}$, de lo contrario, se utiliza la misma ecuación del AS. Esta regla es más voraz o miope (greedy), favoreciendo la explotación de la información de feromonas, por lo que debe ser contrarrestada por la actualización de feromona local para diversificar la búsqueda realizada por las hormigas posteriores durante una iteración. Esta se realiza después de cada paso de la hormiga:

$$\tau_{ij} = (1 - \rho) * \tau_{ij} + \rho * \tau_0, \quad donde\ \rho\ \epsilon\ (0,1]\ es\ la\ evaporización$$

También se realiza la actualización fuera de línea (o global) de la feromona como se hace en AS, pero solo para la mejor (best) hormiga de cada ciclo:

$$\tau_{ij} \leftarrow (1 - \rho) * \tau_{ij} + \rho * \Delta\tau_{ij}^{best}$$

En el Rank-based Ant System (AS_{rank}) todas las soluciones se ordenan de acuerdo a su longitud [8]. La cantidad de feromona depositada se pondera para cada solución, de tal manera que en las soluciones con trayectorias más cortas se deposite más feromona que en las soluciones con caminos más largos.

Stützle y Hoos presentaron una de las últimas versiones destacables, el MAX-MIN Ant System (MMAS) [7], donde la feromona varía entre un máximo y mínimo específicos, empezando el algoritmo con la cantidad máxima de feromona (τ_{MAX}), y se reinicia cuando se estanca el algoritmo, es decir, no se mejora en un determinado número de iteraciones. τ_{ij} se ajusta a τ_{MAX} si $\tau_{ij} > \tau_{MAX}$ y a τ_{MIN} si $\tau_{ij} < \tau_{MIN}$.Solo se deposita feromona en las aristas de la mejor solución de cada iteración.

En recientes desarrollos, se ha planteado un método que introduce Lógica Difusa en las hormigas de OCH para acelerar la capacidad de búsqueda [38].

3.3 TRABAJOS DE OCH PARA JSP

Se han aplicado los diferentes algoritmos de OCH con múltiples variantes para la solución del problema de planificación de tareas (JSP).

En el trabajo de Dorigo et al. se realizó la implementación del primer algoritmo desarrollado de OCH [56], el Ant System (AS), para resolver el problema de planificación de trabajos, y se estableció mediante un análisis de sensibilidad de los parámetros que tienen mayor influencia en la convergencia del algoritmo, que los mejores resultados se obtienen cuando $\alpha = \beta$. Esto fue confirmado en el trabajo de Van der Swaan y Marques en el cual resolvieron instancias JSP planteadas por Muth-Thompson con una variante de OCH [47], en la que introdujeron un parámetro de variación (similar al operador de mutación en algoritmos Genéticos) para guiar la búsqueda a las regiones de los óptimos. En Colombia, Buitrago et al. [48]. en el artículo “Minimización de la tardanza ponderada total en talleres de manufactura aplicando colonia de hormigas”, presentaron un diseño simplex para configurar adecuadamente todos los parámetros del algoritmo.

En el trabajo de Téllez [40], también se realizó la implementación del Ant System (AS), con algunas variantes que dieron resultados de gran calidad para las instancias JSP de Lawrence (LA), como el contador de pasos de las hormigas que penaliza las operaciones que generan retardos, lo que lo hace muy competitivo respecto a los mejores resultados de Búsqueda Tabú.

Montgomery et al. aplicaron un enfoque alternativo del MAX-MIN Ant System (MMAS) en un problema JSP real de una empresa de impresión (o imprenta) [50], en la que a cada máquina (de impresión, corte, plegado, relieve, recopilación, grapado o empaque) se le asigna una regla única de secuenciación (que determina el orden de los despachos) y los tiempos de procesamiento y las fechas de entrega del trabajo se modelan con conjuntos difusos (fuzzy). Los resultados indicaron que este enfoque produce mejores soluciones rápidamente porque la asignación de reglas de secuenciación restringe el espacio de búsqueda a un área de soluciones de buena calidad.

En un trabajo reciente [46], Udomsakdigool y Khachitvichyanukul desarrollaron un algoritmo de OCH para el problema JSP multiobjetivo, donde se minimizo además del makespan, el flujo de tiempo promedio y la tardanza media, demostrando que esta técnica puede alcanzar simultáneamente los múltiples objetivos de JSP.

4. DESARROLLO DEL ALGORITMO

El *Elitist Ant Sistem* fue el algoritmo de la familia OCH implementado.

4.1 DISEÑO DEL *ELITIST ANT SYSTEM* (EAS)

El EAS (Sistema de Hormigas Elitista) implementa un sencillo cambio al *Ant System* que mejora los resultados, simplemente reforzando el rastro de feromona de la mejor ruta encontrada en cada iteración. A las aristas de la mejor solución generada por una de las hormigas, se le deposita más feromona por medio de todas las otras hormigas.

En este algoritmo las hormigas artificiales realizan una construcción probabilística de soluciones en cada ciclo, para lo cual se requiere representar el problema por medio de un grafo, en el que las hormigas se mueven a lo largo de cada arista de un nodo a otro para construir caminos que representan soluciones, desde un nodo inicial seleccionado aleatoriamente, las siguientes elecciones del próximo nodo en este camino se hacen de acuerdo a la regla de transición de estado:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}\left(\eta_{ij}\right)^{\beta}}{\sum_{l \in Tabu_k}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}} \qquad (1)$$

$$si\ j,l\ \notin Tabu_k$$

i es el nodo origen ($i \in Tabu_k$), j es el nodo destino al que se evalúa la probabilidad y h son los otros nodos destino que son candidatos a ser seleccionados. Donde los parámetros α y β determinan la influencia de los valores de la información de la feromona (τ) y de la información heurística (η) respectivamente, sobre la decisión de cada hormiga (k).

El propósito de esta ecuación es que las aristas con gran cantidad de feromona sean las más visibles porque las hormigas deben tener preferencia probabilística por las rutas cortas que formarán esas aristas. Para obtener una probabilidad de

transición mayor en esas aristas, sería aparentemente adecuado colocar un $\alpha > \beta$. Sin embargo, para que el algoritmo este equilibrado, se debe tener un apropiado ajuste de los parámetros α y β, evitando valores cercanos a cero, porque si $\alpha = 0$, únicamente la información heurística determinaría que posibles elementos de la solución tendrán mayor probabilidad de ser seleccionados, lo que corresponde a un algoritmo greedy (miope) estocástico; y si $\beta = 0$, solo se tendría en cuenta la cantidad de feromona de las aristas para seleccionar el próximo nodo. En ambos casos las hormigas podrían estancarse en un óptimo local, generando la misma solución en las siguientes iteraciones, sin oportunidades de encontrar una mejor solución que sea la solución óptima global. Estos parámetros están configurados normalmente en valores enteros entre 1 y 5, pero en este caso se relacionarán de la siguiente forma $\beta = (1 - \alpha)$ con $\alpha \in (0,1]$.

La cantidad de feromona $\tau_{ij}(t)$ presente en cada arista del camino en la generación t está dada por la siguiente ecuación:

$$\tau_{ij}(t) = \sum_{k=1}^{n} \Delta\tau_{ij}^{k} + \rho * \tau_{ij}(t-1) \quad (2)$$

Donde $\tau_{ij}^{k}(t)$ es la contribución de la hormiga k a la feromona total en la generación t y ρ es la tasa de evaporación de la feromona. La razón para incluir la tasa de evaporación es que la feromona antigua no debería tener mucha influencia en las decisiones futuras de las hormigas.

La cantidad de feromona con la que contribuye cada hormiga depende de calidad de la solución obtenida, siendo inversamente proporcional al costo de la función objetivo de la solución (Ver Ecuación 3), por lo que las rutas más cortas tienden a tener una razón más alta de crecimiento del valor de la feromona.

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \quad (3)$$

Donde Q es una constante de feromona y L_k es la longitud del makespan de la solución obtenida por la hormiga k.

Hasta el punto anterior las ecuaciones son idénticas a las del Ant System, la modificación planteada por Dorigo permite acelerar la convergencia del algoritmo, aumentando la visibilidad del rastro de feromona en todas las aristas del camino más corto, pasando con todas las hormigas elitistas (e) del sistema. Por lo tanto, la ecuación 3 para el mejor camino construido en cada ciclo es reemplazada por:

$$\Delta \tau_{ij}^{k} = \frac{Q}{L_k} * e \tag{4}$$

Con mayor velocidad de convergencia, el algoritmo tendrá menor capacidad de exploración del espacio de soluciones factibles, y no lograría encontrar la mejor solución. Para evitar esto, se pueden permitir incrementos de costos al tener máquinas que estarán ociosas (o en pausa) en vez de la máquina que está disponible inmediatamente.

4.2 ADAPTACIONES DEL *ELITIST ANT SYSTEM* AL JSP

La convergencia rápida de este algoritmo hace que las hormigas pierdan capacidad de exploración y terminaran muy pronto en un solo camino, que puede ser un óptimo local. Para compensar esto, se permite incluir en el conjunto de operaciones realizables (punto 3.3 del pseudocódigo), operaciones que generan retardos en las máquinas por tener que esperar unas unidades de tiempo para empezar su ejecución, porque el trabajo correspondiente aún está activo en otra máquina. Pero esta operación que demora el inicio de las máquinas, solo será seleccionada si la arista que llega a ese nodo tiene suficiente feromona para hacer que su probabilidad sea mayor a la de las operaciones que tiene trabajos disponibles inmediatamente. Lo que se dará solo con grandes cantidades de feromona, porque tener máquinas ociosas no es muy adecuado y es penalizado reduciendo la visibilidad de la operación.

Con este método se explora más el espacio de búsqueda para obtener soluciones muy diversas, a partir de las cuales se pueden obtener soluciones que sobrepasen

los óptimos locales hallados en las primeras iteraciones. Estos óptimos son los que limitan la búsqueda, deteniéndola en soluciones alejadas hasta en un 5% del óptimo global. La diversidad inicial del algoritmo es la que asegura que las hormigas se encaminen hacia el espacio de búsqueda donde se halla el camino correspondiente a la solución óptima global. El siguiente es el Pseudocódigo implementado para solucionar el JSP:

Procedimiento OCH: EAS

1. $Inicializacion\ de\ los\ parametros$:
   $\propto, \beta, \rho, \tau_0, Q, K\ (Hormigas), C\ (Ciclos)$

2. $Para\ cada\ arista\ a_{ij}\ hacer$:
   $\tau_{0ij} = c\ ;\ donde\ c\ es\ una\ constante$
   $\Delta\tau_{ij} = 0\ ;\ \ Acumulador\ de\ feromona\ actual$

3. $Para\ cada\ Ciclo\ C\ hacer$:
   3.1 $Asignación\ aleatoria\ de\ la\ primera\ operación$
   3.2 $Definir\ la\ regla\ de\ decibilidad\ para\ cada\ hormiga\ k$
   3.3 $Mientras\ tabu_k\ no\ este\ llena, hacer$:
   $Para\ cada\ hormiga\ k\ hacer$:
   $Determinar\ el\ conjunto\ de\ operaciones\ realizables\ desde\ el\ Nodo\ actual$
   $Seleccionar\ la\ próxima\ operación\ a\ ser\ visitada\ según\ ecuación\ 1$
   $Mover\ la\ hormiga\ a\ la\ operación\ seleccionada$
   $Guardar\ la\ operación\ seleccionada\ en\ \ tabu_k$
   3.4 $Para\ cada\ hormiga\ k\ hacer$:
   $Obtener\ el\ makespan\ L_k\ del\ plan\ construido$
   $Guardar\ el\ plan\ con\ el\ menor\ makespan\ del\ Ciclo\ C$
   $Para\ cada\ arista\ a_{ij}\ hacer$:
   $Calcular\ \Delta\tau_{ij}\ según\ ecuación\ 3\ o\ 4\ que\ corresponda$
   3.5 $Para\ cada\ arista\ a_{ij}\ hacer$:
   $Actualizar\ feromona\ \tau_{ij}\ según\ ecuación\ 2$
   $\Delta\tau_{ij} = 0\ ;$
   3.6 $Mostrar\ plan\ mas\ corto\ del\ ciclo\ C$
   3.7 $tabu_k = \phi\ ;\ \ vaciar\ lista\ de\ visitados$
   $Fin - Ciclos\ C$
4. $Mostrar\ plan\ con\ el\ makespan\ mas\ corto$
$Fin - Procedimiento$

Descripción del pseudocódigo:

1. Los mejores resultados se obtuvieron con los parámetros inicializados en $\alpha = 0.2,\ \beta = 0.8\ y\ \rho = 0.7$, se fijó el número de ciclos (o iteraciones) en 1000, la feromona inicial $\tau_0 = 0.002$, la feromona ganada $Q = 0.001$, y la cantidad de hormigas (K) se calcula según el número de trabajos, que es la cantidad de elementos que tiene el conjunto *J*, así:

$$K = \frac{|J|}{2} \tag{5}$$

2. Se inicia el rastro de feromona de todas las aristas en una constante positiva y pequeña.

3. Empieza la fase de Construcción Probabilística de soluciones por las K hormigas.

3.1 La primera operación se selecciona aleatoriamente entre los nodos visitables inicialmente según las restricciones del problema.

3.2 La selección de la regla de "decibilidad" se realiza aleatoriamente, con igual probabilidad entre la regla con el tiempo de procesamiento más corto SPT (*Shortest Processing Time*) o la regla con el tiempo de procesamiento más largo LPT (*Longest Processing Time*) de las operaciones [56].

3.3 Mientras no termine de llenarse la memoria $tabú_k$, significa que la hormiga no ha finalizado la generación del plan por lo que continua recorriendo el grafo hasta completar el total de operaciones ($|O| = |J| * |M|$). La lista $tabu_k$ restringe la elección de operaciones para prevenir el regreso a nodos recientemente visitados, en el algoritmo esta lista es una sarta, que contiene las operaciones en cada máquina ordenadas por tiempo de inicio (Ver Figura 12). En la Figura 12 se muestra la sarta para el diagrama de Gantt de la Figura 5, en donde los primeros trabajos en iniciarse son J3 y J1 en t0, y el ultimó trabajo en empezarse es J1 en t10.

Figura 12: Lista $tabu_k$ con operaciones en orden de inicio

Tiempo de inicio	t0	t0	t3	t3	t4	t4	t6	t6	t10
Operaciones	O_{32}	O_{13}	O_{31}	O_{22}	O_{12}	O_{23}	O_{21}	O_{33}	O_{11}

En el conjunto de operaciones visitables se incluyen operaciones que generan un retardo en las máquinas menor o igual a cinco unidades de tiempo, para no descartar nodos a los que se accede con aristas de un alto nivel de feromona. Para mantener el equilibrio afectado por el retardo generado, se reduce la visibilidad del nodo en un punto porcentual por cada unidad de tiempo perdido, es decir, máxima 5% de reducción.

3.4 Una vez que cada hormiga ha construido su solución, se empieza la fase de actualización de la feromona, repasando el camino recorrido para agregar la cantidad de feromona correspondiente (según la ecuación 3 o 4) al acumulador de feromona del ciclo actual. Si el makespan de la solución es muy costoso, se da menor relevancia al camino depositando poca feromona en sus aristas.

3.5 Actualiza los rastros de feromona de las aristas recorridas utilizando un proceso conocido como *Actualización en línea a posteriori*, que es una actualización global realizada fuera de línea (offline). Después de la ejecución de cada ciclo del algoritmo, se deposita en los rastros de feromona de cada una de las aristas del grafo, lo que han añadido las hormigas en el acumulador de feromona respectivo. Luego se reinicia en cero el acumulador de feromona actual para no volver a depositar esta feromona en el próximo ciclo.

3.6 El plan de mejor calidad del ciclo actual se guarda con su respectivo makespan.

3.7 Se borra la memoria de cada hormiga ($tabu_k$) para empezar a construir nuevos planes en el próximo ciclo.

4. Muestra el mejor plan de todos los ciclos realizados por el algoritmo.

CARACTERIZACIÓN ESCENARIO COMPUTACIONAL

Se debe establecer posibles instancias de aplicación de los algoritmos desarrollados sobre problemas de planificación de recursos relacionados con el área computacional.

5.1 PROBLEMA JSP EN EL CAMPO DE LA COMPUTACIÓN

Uno de los campos de la computación en que se presenta el problema de calendarización, son los sistemas distribuidos como los clústeres y grids, que pueden llegar a tener miles de nodos para procesar las tareas. Estos sistemas pueden realizar una gran cantidad de cálculos en corto tiempo, por lo que son sistemas de computación de alto rendimiento HPC (high-performance computing), y se pueden gestionar con plataformas como *Microsoft Windows Compute Cluster Server*. Estas arquitecturas son muy escalables y brindan grandes prestaciones y tolerancia a fallos.

Los sistemas distribuidos requieren de métodos de optimización combinatoria adecuados para realizar la calendarización de trabajos (job scheduling), que logren optimizar el tiempo y uso de recursos en forma paralela. Con OCH se ha abordado el problema de calendarización del flujo de trabajos en entornos grid para lograr un alto rendimiento [57], que debe realizar una actualización local de la feromona (en cada paso de las hormigas) para presentar mejores resultados.

OCH se ha utilizado para encontrar una asignación adecuada de recursos para cada trabajo, que minimice la tardanza en el tiempo de cada trabajo. En el trabajo de Lorpunmanee y sus colegas [58], aprovechan la información dinámica y una colonia de hormigas para realizar la planificación de trabajos, encontrando que este sistema de planificación utilizando es bastante eficiente y eficaz para

asignar los trabajos a los recursos adecuados, y logra reducir significativamente la longitud del calendario (makespan).

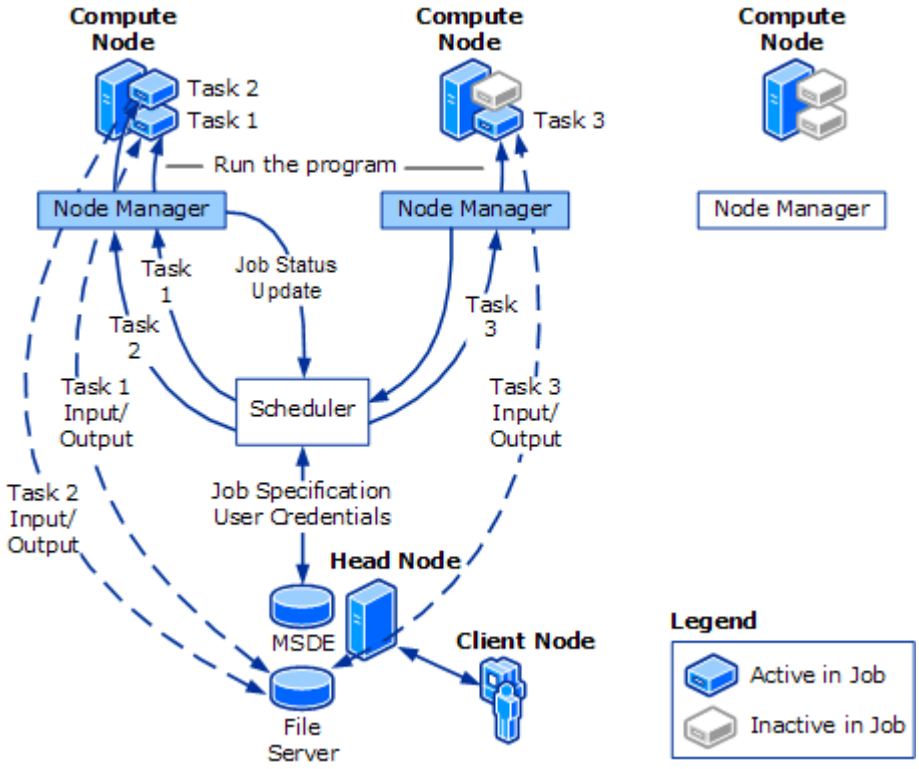
En este contexto, un trabajo es un conjunto de tareas (u operaciones) para realizar en diferentes recursos (procesadores). Una tarea representa la ejecución de un programa en el nodo de procesamiento asignado, puede ser un programa secuencial con un solo proceso (como en JSP) o un programa paralelo con múltiples procesos simultáneos.

La administración de los recursos de un clúster es realizada por un Job Scheduler, que distribuye las tareas en los nodos de cómputo para su ejecución, en el orden en que aparecen en la lista de tareas (candidatos factibles). Para despachar la tarea, el Job Scheduler pasa la tarea al *nodo designado* (máquina especifica de la operación), que puede ser cualquiera de los nodos asignados al trabajo (en paralelo). A menos que se hayan especificado las dependencias (orden por restricciones), las tareas son despachadas por (prioridad basada en) *primero en llegar, primero en ser atendido* (First Come, First Serve, FCFS). También se pueden aplicar otras reglas, como la de *atender de primero al trabajo más corto* (Shortest Job First, SJF) que genera una cola de tareas con tiempo de procesamiento creciente, o al contrario, *atender de primero al trabajo más largo* (Longest Job First, LJF). Estas dos últimas reglas son similares a la regla de SPT (Shortest Process Time) y LPT (Longest Process Time) respectivamente, aplicadas en OCH para definir la visibilidad de las operaciones candidatas según la extensión de su tiempo de procesamiento.

En la figura 13 se muestra un ejemplo de despacho de tareas secuenciales (como se debe realizar en JSP), donde se asigna la tarea 1 al primer procesador del primer nodo, después la tarea 2 se asigna a el segundo procesador del mismo nodo, la tarea 3 se asigna al primer procesador del segundo nodo, y así sucesivamente hasta que no falten más tareas del trabajo por ejecutar o hasta que

estén ocupados todos los procesadores del clúster. Las tareas pendientes tienen que esperar al siguiente procesador disponible para ser ejecutadas.

Figura 13: Ejecución de tareas secuenciales



Fuente: technet.microsoft.com[2]

Un tipo de calendarizador para clústeres es batch scheduler, calendarizador a nivel de batch (por lotes) que despliega las tareas en un intervalo de tiempo en el que los recursos están disponibles. Algunos job scheduler tipo batch son PBS (Portable Batch System) y LSF (Load Sharing Facility). En la GridUIS se utiliza el batch scheduler OAR (Resource Management System for HPC), open source que soporta toda clase de aplicaciones paralelas de usuario. OAR opera como un daemon (proceso en segundo plano) únicamente sobre el servidor OAR, utiliza un gestor de base de datos como MySQL, un lenguaje de script como Perl (con la DBI, DB independent interfaz para Perl). Ejecuta trabajos interactivos (de forma

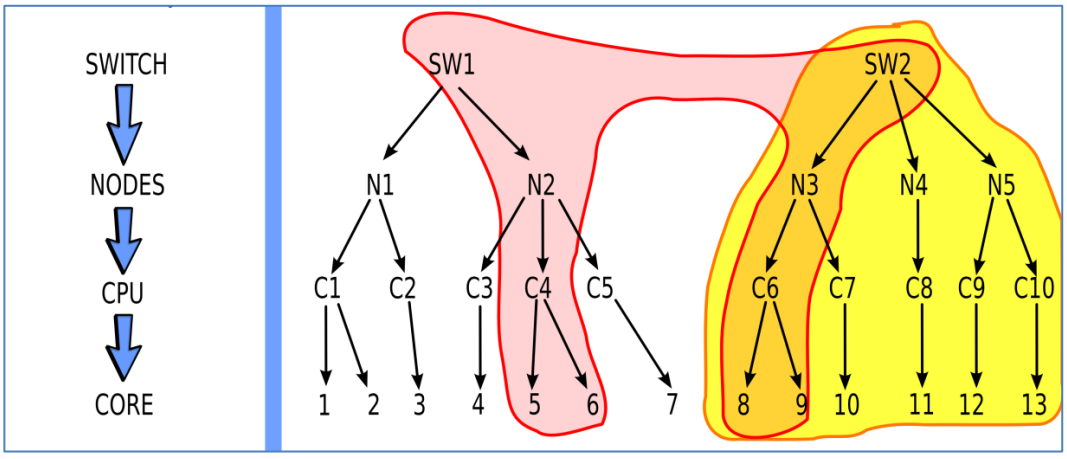
[2] http://technet.microsoft.com/en-us/library/cc720125(v=ws.10).aspx

inmediata) o por lotes con reserva para un tiempo determinado, realizada por usuarios autenticados (logging) que especifican el Walltime (horas de uso, fecha) y los recursos que necesitan. Organiza los trabajos en múltiples colas (Multi-queues) con prioridades, buscando aprovechar los nodos ociosos, y puede suspender y reanudar trabajos.

En un clúster heterogéneo, la jerarquía tradicional empieza desde los nodos, los cuales tienen varias CPU, que a su vez contienen varios núcleos. Un usuario que ha iniciado sesión puede hacer una reserva interactiva de recursos en OAR con el comando oarsub (Submit job), que para el ejemplo de la Figura 14 podría ser:

Oarsub -I -l switch=2/nodes=1/cpu=1/core=2, que reserva 4 cores en total, 2 cores en cada CPU de un nodo de 2 switches diferentes. Con este comando se asignaría el tiempo de reserva por defecto. En otra sentencia, *oarsub -I -l switch=1 walltime=10*, reserva todos los recursos unidos a un switch completo durante 10 horas, SW2 en la Figura 14.

Figura 14: Ejemplo de jerarquía de un clúster heterogéneo

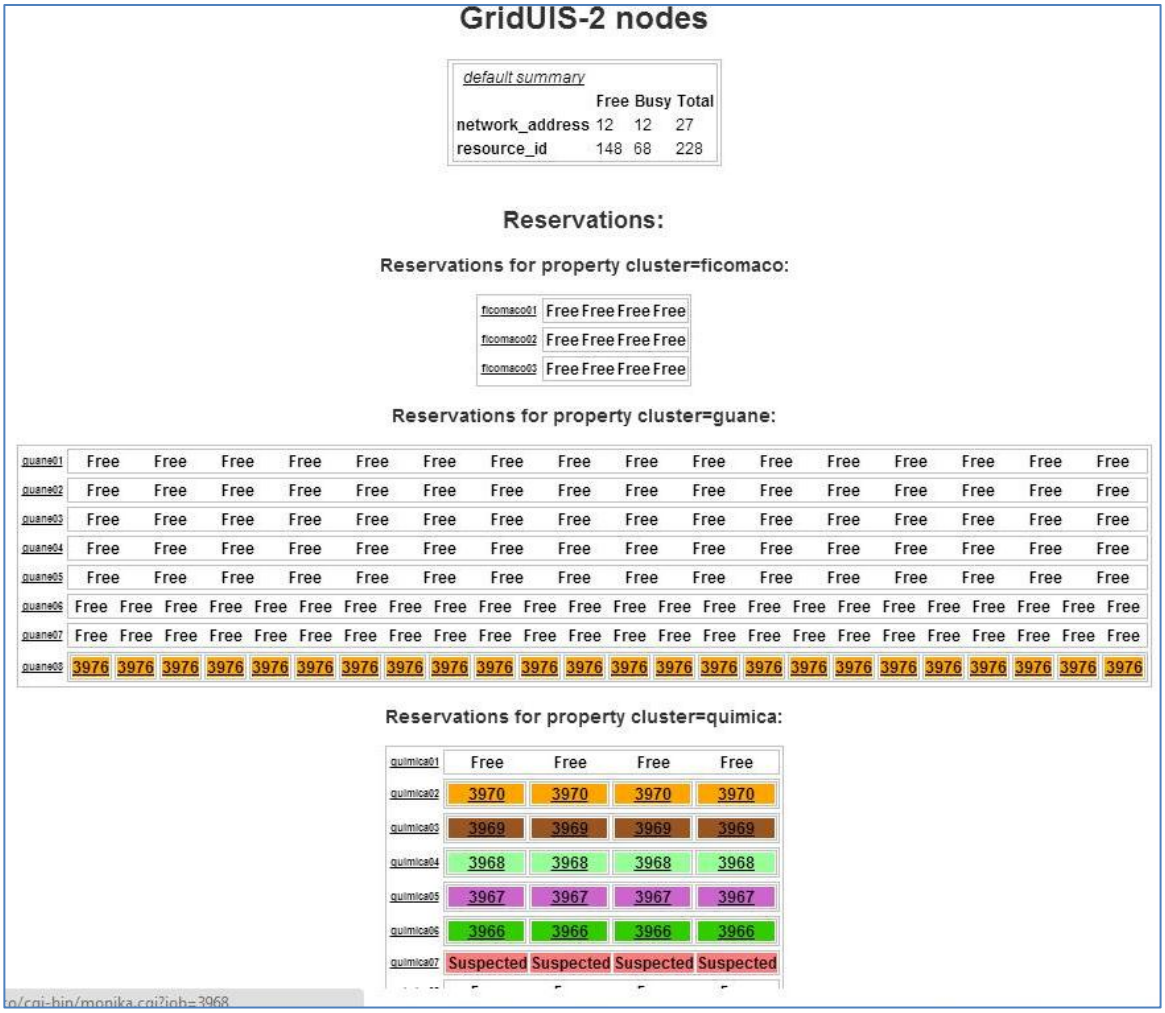


Fuente: OAR project[3]

[3] http://oar.imag.fr/sources/2.5/docs/documentation/OAR-DOCUMENTATION-USER/

Para cada reserva se crea un Job_ID, con fecha de inicio (Start) y final (End), que se puede visualizar en la herramienta de monitoreo de recursos on-line de OAR llamado Monika. Los recursos de la GridUIS-2 se pueden consultar en http://grid.uis.edu.co/monika (Ver Figura 15).

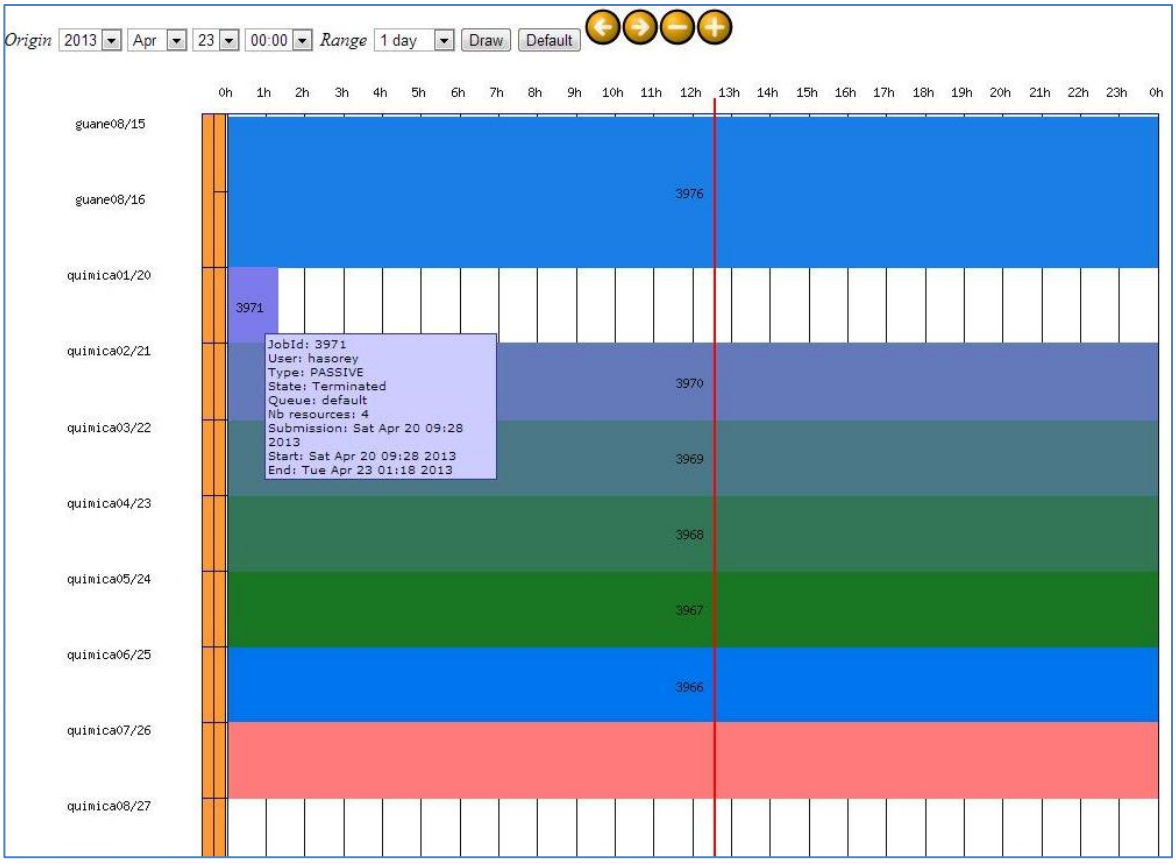
Figura 15: Sistema de monitoreo (Monika) de GridUIS-2



Fuente: http://grid.uis.edu.co/monika

La GridUIS-2 tiene 34 nodos distribuidos en 3 clústeres (Guane, Ficomaco y Química), cada nodo con varias CPUs y hasta 70 núcleos. En la Figura 16 se puede ver algunos de esos nodos con los trabajos que tienen planificados para un día.

Figura 16: Diagrama de Gantt de la GridUIS-2

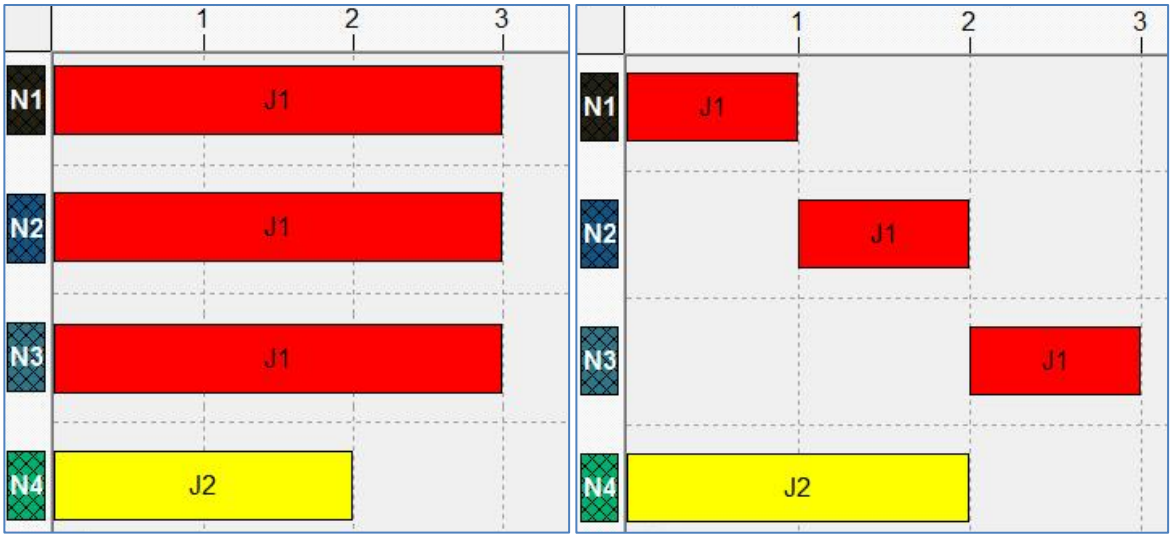


Fuente: http://grid.uis.edu.co/drawgantt

5.2 ADAPTACIÓN DEL JOB SCHEDULING A UN ESCENARIO JSP

En Cómputo de Alto Rendimiento, calendarizar implica multitareas (entre nodos) y multiprocesos (entre procesadores de un nodo algoritmo, que dividen en procesos una tarea), en este escenario solo se realizará la calendarización de las múltiples tareas. Tampoco se ejecutarán las tareas en forma paralela porque no sería exactamente un problema JSP, en consecuencia, se cambiará la restricción de paralelismo de tareas de un clúster por la restricción de precedencia (secuencialidad) de JSP (Ver Figura 17), que es una restricción importante para mantener la complejidad del problema aunque el procesamiento en paralelo también sería igual de complejo.

Figura 17: Cambio de tareas paralelas a tareas secuenciales



Además, los algoritmos EAS y GRASP solo actuarán como planificadores del flujo de trabajos (usuarios) en un momento determinado (tiempo inicial cuando todos los nodos están libres), y no realizarán la selección de los nodos para cada tarea, que es una de las complicadas funciones del administrador de recursos (OAR). Será como una replanificación de las asignaciones de trabajos a los nodos que hace el batch scheduler (que busca iniciar inmediatamente el job sin ningún retardo) teniendo en cuenta la capacidad de procesamiento de los nodos, los algoritmos los vuelven a organizar tratando de reducir el flujo máximo de trabajos (makespan).

Un buen administrador de recursos debería enviar cada trabajo al nodo que tenga un número igual de procesadores solicitados (para no malgastar el tiempo de los nodos de mayor capacidad) pero sin crear una cola extensa en una de las máquinas (que aumenta el makespan). Con un adecuado uso de máquinas (balanceo de carga) se puede atender eficientemente a los usuarios, y se mantiene disponible a los nodos más potentes (de mayor número de núcleos) a la espera de trabajos exigentes que no puedan ser procesados en los demás nodos.

El administrador de recursos (en este escenario JSP) define para cada trabajo una secuencia de tareas (con un tiempo de procesamiento especifico), que se deben ejecutar en el orden exacto y en la máquina determinada, para atender a todos los usuarios en el menor tiempo posible. Como en JSP, cada tarea no se puede interrumpir, son operaciones que se deben ejecutar hasta que acabe su trabajo, para evitar errores en la ejecución de aplicaciones.

Escenario *Job Shop Scheduling*

Calendarizar las tareas de cada usuario (trabajo) que solicita el uso de nodos de un clúster, minimizando el flujo de trabajo de todos los usuarios. Con todos los nodos disponibles inicialmente y una instancia de trabajos definida por el administrador de recursos, los algoritmos (EAS y GRASP) buscarán el plan con el menor makespan.

Trabajo: tiene múltiples tareas secuenciales y es definido por un usuario. Se le asigna un Job_ID para su identificación.

Operación: Tarea que puede ser independiente o depender de otras tareas, y no se permite operaciones simultáneas del mismo trabajo para JSP.

Máquina: Recurso computacional que es un nodo del clúster (que tiene múltiples procesadores).

Tiempo de procesamiento: es el tiempo solicitado en la reserva por el usuario, que es igual para todas las tareas del trabajo correspondiente.

Tamaño de la instancia: 10 trabajos x 20 máquinas

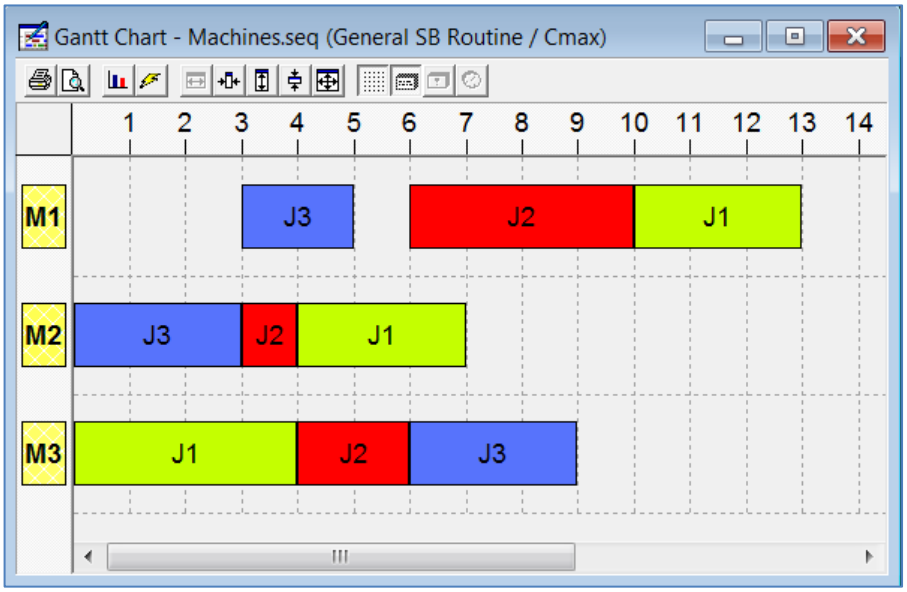
La instancia planteada está en la Tabla 3, los trabajos tienen un número de operaciones variables porque dependen del número de nodos que solicitó el usuario. Los resultados obtenidos por EAS y GRASP sobre esta instancia se pueden ver en la Tabla 4.

Tabla 3: Instancia JSP de un clúster

Sec:	S1		S2		S3		S4		S5		S6		S7		S8		S9		S10	
Job	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**	**N**	**t**
J1	1	2	2	2	3	2	4	2	5	2	6	2	7	2	8	2	9	2	10	2
J2	2	3	3	3	5	3	7	3	11	3	12	3								
J3	3	4	1	4	2	4	13	4	14	4	15	4	16	4						
J4	17	2	18	2	19	2	20	2	5	2	6	2	7	2						
J5	5	9	6	9	7	9	11	9	13	9	8	9	10	9	9	9				
J6	1	12	0	12																
J7	15	24	16	24																
J8	5	36																		
J9	8	3	9	3	11	3	10	3	5	3	4	3	6	3	3	3	0	3		
J10	12	5	13	5	10	5	9	5	5	5	14	5	15	5	16	5	17	5		

Para comprobar que los resultados obtenidos son correctos, se utilizó el software Lekin[4], que lee la matriz de flujo de trabajos producida por los algoritmos y calcula los tiempos de inicio y final de cada operación y el makespan del plan generado. Lekin también puede generar planes de trabajos, para lo cual usa la heurística Generic Shifting Bottleneck (desarrollado por Nutthapol Asadathorn). Solucionando la instancia de la Tabla 1, Lekin genera el plan óptimo (con makespan de 13 unidades de tiempo) igual al expuesto en la Figura 5 (Ver Figura 18).

Figura 18: Obtención mediante Lekin del óptimo de la instancia de la Tabla 1



[4] Flexible Job-Shop Scheduling System. http://community.stern.nyu.edu/om/software/lekin/

Para la instancia planteada en la tabla 3, el algoritmo usado por Lekin encuentra soluciones con makespan de 76 unidades de tiempo, que es mayor al mejor makespan obtenido por OCH y GRASP (Ver Tabla 4).

Tabla 4: Resultados con la instancia planteada en Tabla 3

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	73	1280	1	72	1001
2	72	2930	2	72	1001
3	76	25	3	72	1001
4	72	3005	4	72	1001
5	72	3025	5	72	1001
6	72	3270	6	72	1001
7	72	4375	7	72	1001
8	72	760	8	72	1001
9	72	1895	9	72	1001
10	73	3680	10	72	1501
11	75	1440	11	72	1001
12	72	2135	12	72	1001
13	72	1050	13	72	1001
14	72	3035	14	72	1001
15	72	2120	15	72	1001
16	72	3725	16	72	1001
17	76	20	17	72	1001
18	72	3160	18	72	1001
19	72	4400	19	72	1501
20	75	4930	20	72	1001
21	72	1560	21	108	1001
22	72	2215	22	72	1001
23	76	10	23	72	1001
24	72	725	24	72	1001
25	72	3630	25	108	501
26	73	1160	26	108	1001
27	72	2275	27	72	1001
28	76	40	28	108	1001
29	72	1500	29	72	1001
30	72	2170	30	72	1001
Mejor	72	10	Mejor	72	501
Promedio	72,83	2184,83	Promedio	76,80	1017,67
Desviación	1,49	1392,53	Desviación	12,45	159,92

El makespan de la mejor solución de los dos algoritmos es el mismo, generando un buen plan para realizar todos los trabajos en de 3 días (72 horas), que es la mejor solución conocida (BKS) por ahora. EAS tiene mejor desempeño porque logra un promedio de makespan menor que GRASP, aunque requiere de más evaluaciones de la función objetivo y su desviación estándar es muy alta. Ambos algoritmos encontraron el makespan mínimo de cada ejecución en las primeras iteraciones, y no lograron mejorarla en el transcurso de la búsqueda. Esto indica que si no se encuentra el óptimo global de esta instancia al principio, los algoritmos caen en óptimos locales que no logran superar, porque existen muy pocas combinaciones con el mínimo makespan, y están ubicadas en la misma región del espacio de búsqueda, es decir, se tienen muchos operaciones iniciales (de mayor tiempo de procesamiento) que solo ubicándolas en momentos muy específicos, se asegura encontrar el óptimo global.

6. PRUEBAS Y ANÁLISIS DE RESULTADOS

Las instancias JSP (LA) de prueba se obtuvieron de la OR-Library (biblioteca online de Investigación de Operaciones) con el siguiente formato [45]:

Tabla 5: Instancia de Lawrence LA01

Secuencia	1		2		3		4		5	
Trabajos	**Máquina**	**Tiempo**	**Máquina**	**Tiempo**	**Máquina**	**Tiempo**	**Máquina**	**Tiempo**	**Máquina**	**Tiempo**
J1	1	21	0	53	4	95	3	55	2	34
J2	0	21	3	52	4	16	2	26	1	71
J3	3	39	4	98	1	42	2	31	0	12
J4	1	77	0	55	4	79	2	66	3	77
J5	0	83	3	34	2	64	1	19	4	37
J6	1	54	2	43	4	79	0	92	3	62
J7	3	69	4	77	1	87	2	87	0	93
J8	2	38	0	60	1	41	3	24	4	83
J9	3	17	1	49	4	25	0	44	2	98
J10	4	77	3	79	2	43	1	75	0	96

Cada fila representa un trabajo, y en orden secuencial se presenta la máquina y el tiempo de procesamiento de la operación conformada (Ver Tabla 5), empezando desde cero la enumeración de las máquinas, y con tiempos de procesamiento en unidades de tiempo entre el intervalo [5,100).

Presentación de resultados

El algoritmo (desarrollado en lenguaje JAVA) tiene una salida en forma de matriz, donde se coloca la secuencia de trabajos de cada máquina que conforman las operaciones. La Tabla 6 tiene la matriz de salida de la instancia de la Tabla 1.

Tabla 6: Matriz del flujo de trabajos en las máquinas

Máquinas	O1	O2	O3
1	J3	J2	J1
2	J3	J2	J1
3	J1	J2	J3

Los problemas de optimización combinatoria son poco predecibles, por lo que los algoritmos requieren de un análisis de sensibilidad para comparar diversas condiciones y determinar los posibles valores de cada parámetro con los cuales se obtienen las mejores soluciones.

Con un análisis de sensibilidad se estableció el valor de los parámetros de entrada para EAS en $\alpha = 0.2$, $\beta = 0.8$ y $\rho = 0.7$, la feromona inicial (τ_0) en 0.002 y la feromona ganada (Q) en 0.001, y el número de ciclos se fijó en 1000.

Para GRASP, el parámetro alfa que determina el número de elementos que entran en la lista RCL, se fijó en 0.4 que es más cercano a un algoritmo voraz o codicioso (Greedy) que a uno totalmente aleatorio (alfa=1). Con valores menores o mayores las soluciones pierden calidad. El número de iteraciones en la búsqueda local se fijó en 500 para cada operación de la solución inicial, porque con un número más alto la mejora de las soluciones no es significativa y el tiempo de ejecución del algoritmo aumenta proporcionalmente.

6.1 ANÁLISIS DE RESULTADOS

Los resultados se obtuvieron al realizar 30 ejecuciones de cada uno de los algoritmos (EAS y GRASP) por cada una de las 40 instancias JSP planteadas por Lawrence (Ver Tabla 15) [11], que son de diferentes tamaños y dificultad, y al ser muy utilizadas se puede comparar los algoritmos con otras técnicas que generan la *mejor solución conocida* BKS (Best Knowed Solution) tomada de [13] y [27].

Para el análisis de resultados de EAS y GRASP, se determinó el mejor makespan (C_{max}) y el menor número de evaluaciones realizadas a la función objetivo, y se calculó el promedio y desviación estándar del makespan y del número de evaluaciones. Esto se muestra para una instancia representativa de cada uno de los diferente grados de dificultad del conjunto de instancias LA, es decir, 8

instancias en total de tamaños 10 x 5, 15 x 5, 20 x 5, 10 x 10, 15 x 10, 20 x 10, 30 x 10 y 15 x 15 (Ver Tablas 7, 8, 9, 10, 11, 12,13 y 14).

Tabla 7: Resultados de los algoritmos en LA01

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	666	3060	1	726	16525
2	667	705	2	715	13025
3	671	3480	3	707	8025
4	666	4400	4	692	15025
5	668	1420	5	755	15525
6	666	1620	6	752	13525
7	669	2405	7	681	16025
8	666	4120	8	730	15525
9	668	3530	9	728	16025
10	668	1525	10	701	15525
11	669	4915	11	712	14025
12	666	4555	12	666	14525
13	666	2080	13	774	15525
14	668	4445	14	722	12525
15	669	910	15	724	13525
16	668	1045	16	727	15525
17	666	945	17	676	11025
18	666	285	18	739	13025
19	672	2765	19	740	11525
20	668	580	20	726	11525
21	666	515	21	725	14001
22	675	4925	22	753	9001
23	668	1835	23	669	16501
24	666	3840	24	699	15501
25	666	640	25	745	15001
26	666	4310	26	742	17501
27	666	1760	27	707	17001
28	673	3795	28	705	14001
29	666	585	29	739	16501
30	668	255	30	704	15501
Mejor	666	255	Mejor	666	8025
Promedio	667,77	2375,00	Promedio	719,37	14267,00
Desviación	2,33	1594,06	Desviación	26,44	2281,82

Tabla 8: Resultados de los algoritmos en LA06

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	926	721	1	961	28501
2	926	1330	2	945	22001
3	926	427	3	976	16001
4	926	518	4	926	18001
5	926	1610	5	926	30501
6	926	427	6	927	29501
7	926	371	7	966	27501
8	926	350	8	959	21001
9	926	392	9	961	25501
10	926	721	10	998	22001
11	926	385	11	985	25001
12	926	357	12	935	29001
13	926	287	13	926	27001
14	926	581	14	974	28001
15	926	399	15	995	28501
16	926	287	16	947	27001
17	926	287	17	1000	23001
18	926	679	18	956	26001
19	926	252	19	945	28501
20	926	301	20	930	26001
21	926	336	21	956	20001
22	926	581	22	990	28001
23	926	308	23	1036	27001
24	926	840	24	1061	15501
25	926	777	25	926	27501
26	926	238	26	969	25501
27	926	196	27	926	25501
28	926	539	28	938	27501
29	926	1190	29	972	20001
30	926	252	30	956	23001
Mejor	926	196	Mejor	926	15501
Promedio	926,00	531,30	Promedio	962,27	24934,33
Desviación	0,00	339,05	Desviación	33,15	4012,34

Para comparar la estabilidad de EAS y GRASP se graficó la variación del makespan en las 30 ejecuciones junto con la desviación estándar (Ver Figura 19).

Figura 19: Gráfica del makespan de EAS y GRASP en LA01

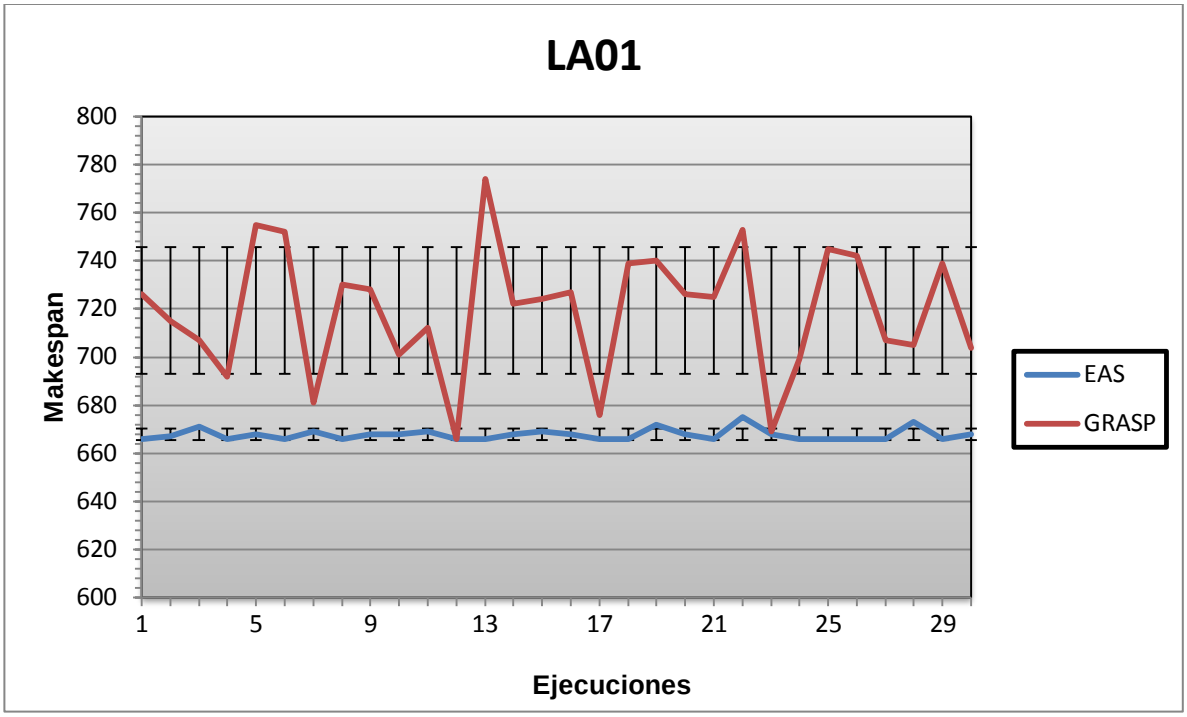


Figura 20: Gráfica del makespan de EAS y GRASP en LA06

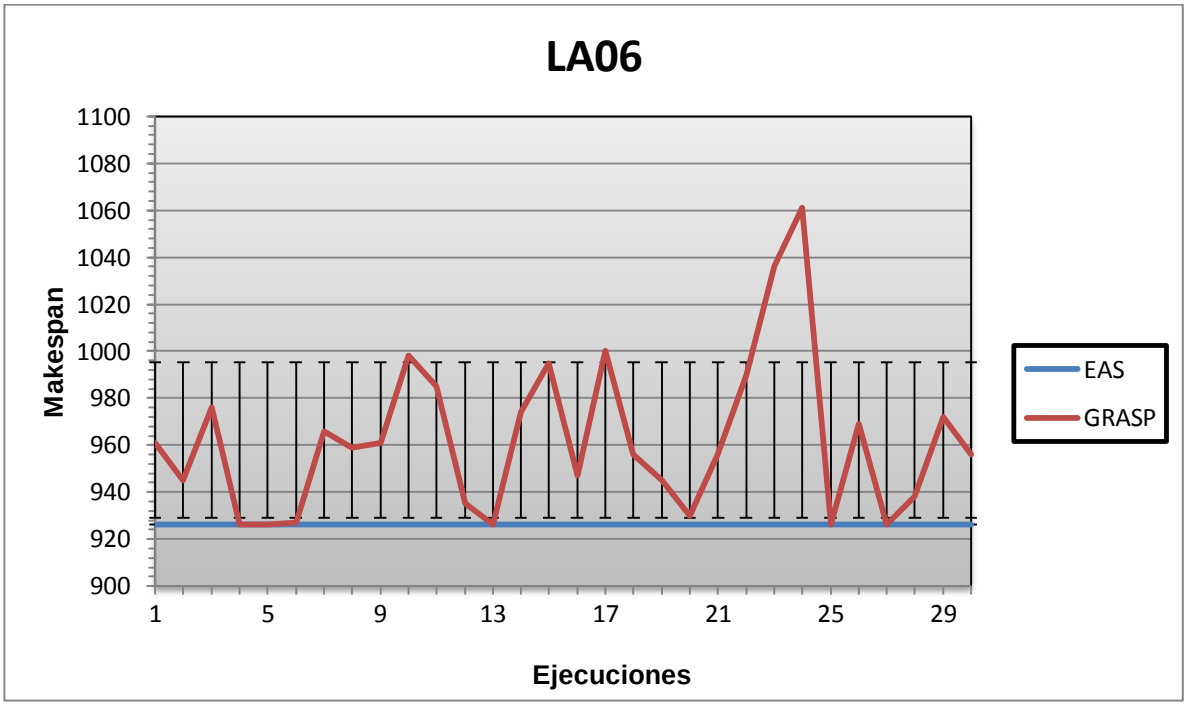


Tabla 9: Resultados de los algoritmos en LA11

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	1222	700	1	1282	32001
2	1222	2770	2	1309	26001
3	1222	2100	3	1293	36501
4	1222	1520	4	1317	25501
5	1222	520	5	1222	38501
6	1222	4040	6	1328	38001
7	1222	800	7	1233	43001
8	1222	410	8	1316	39001
9	1222	1810	9	1298	34001
10	1222	2330	10	1286	29001
11	1222	1070	11	1285	40001
12	1222	2610	12	1310	27501
13	1222	2350	13	1335	33501
14	1222	4360	14	1307	37001
15	1222	560	15	1255	36501
16	1222	1920	16	1323	33501
17	1222	660	17	1323	38501
18	1222	1400	18	1290	28501
19	1222	1210	19	1242	39001
20	1222	1350	20	1283	26001
21	1222	710	21	1254	35001
22	1222	670	22	1253	32001
23	1222	720	23	1286	37001
24	1222	2870	24	1255	40001
25	1222	500	25	1328	11001
26	1222	930	26	1279	35501
27	1222	910	27	1309	31501
28	1222	1290	28	1345	20001
29	1222	1070	29	1290	40501
30	1222	960	30	1305	36001
Mejor	1222	410	Mejor	1222	11001
Promedio	1222,00	1504,00	Promedio	1291,37	33334,33
Desviación	0,00	1028,58	Desviación	31,49	6857,33

Tabla 10: Resultados de los algoritmos en LA16

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	1017	220	1	1108	39025
2	1019	4750	2	1107	33025
3	1028	1695	3	1080	35525
4	1018	3555	4	1156	36025
5	1013	1630	5	1097	35025
6	1031	2280	6	1136	34525
7	1024	3060	7	1052	25525
8	1023	3095	8	1044	33025
9	1008	350	9	1063	31025
10	1028	4555	10	1097	39025
11	1041	580	11	1020	35525
12	1008	2995	12	1068	39025
13	1011	3960	13	1041	31525
14	1037	4600	14	1116	39525
15	1034	2215	15	1065	37525
16	1021	3400	16	1154	30025
17	1005	3010	17	1086	33525
18	1024	4725	18	1165	29025
19	1030	3850	19	1133	39025
20	1022	555	20	1126	30525
21	1011	1785	21	1125	33001
22	1016	3610	22	1082	32001
23	1030	1220	23	1160	34501
24	1008	1035	24	1175	32501
25	1006	4210	25	1092	35001
26	1035	1135	26	1169	35001
27	1011	4245	27	1112	27001
28	1014	2450	28	1128	33501
29	1010	2170	29	1100	17501
30	1021	4070	30	1045	29001
Mejor	1005	220	Mejor	1020	17501
Promedio	1020,13	2700,33	Promedio	1103,40	33200,33
Desviación	10,10	1429,41	Desviación	42,29	4669,57

Figura 21: Gráfica del makespan de EAS y GRASP en LA11

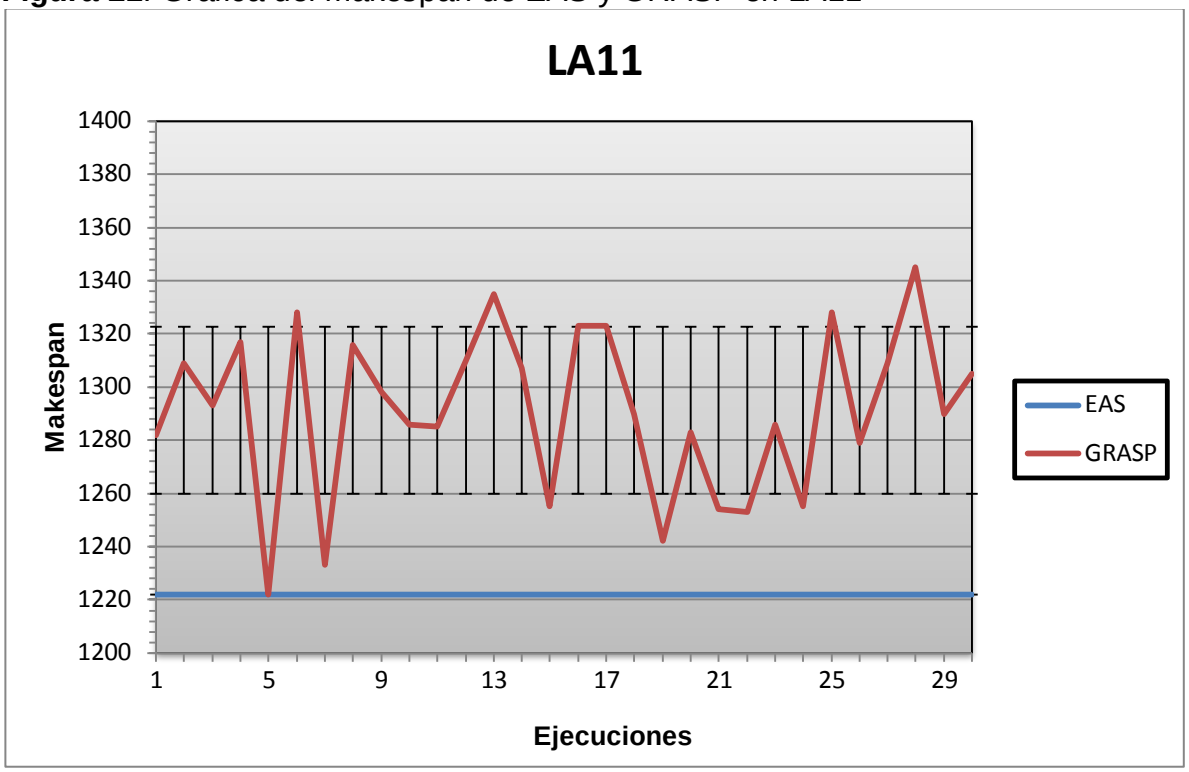


Figura 22: Gráfica del makespan de EAS y GRASP en LA16

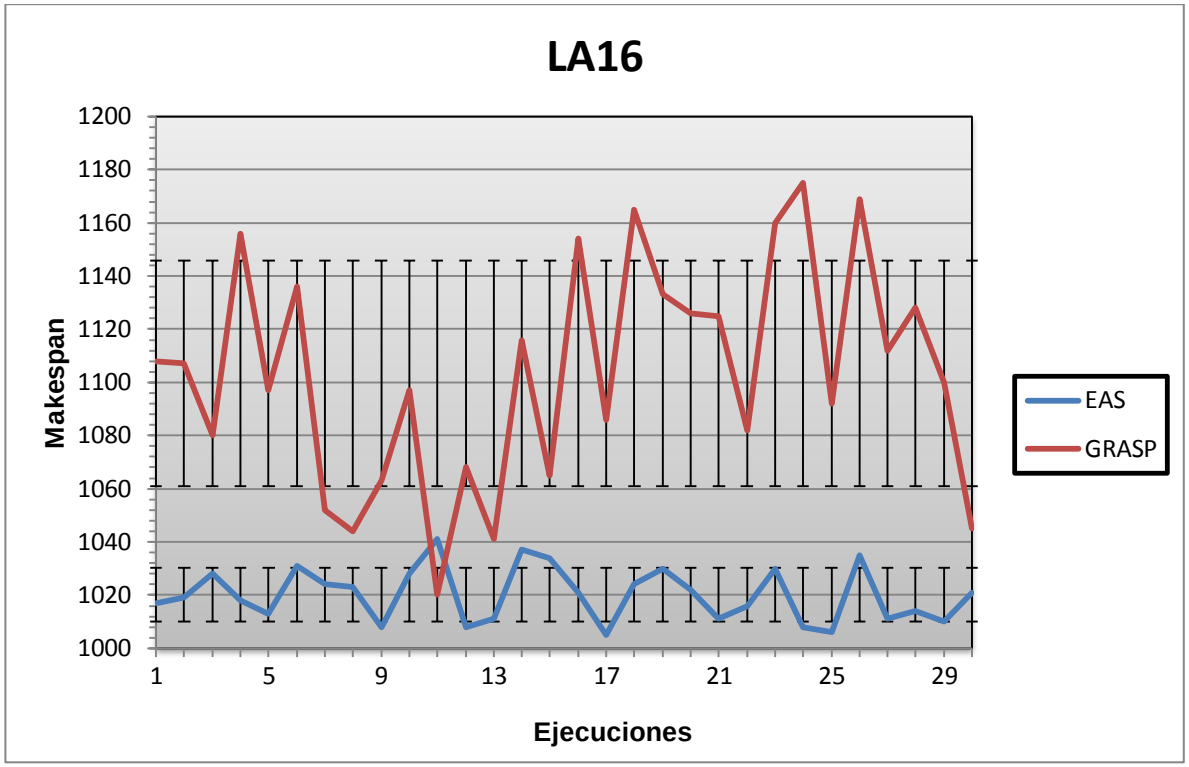


Tabla 11: Resultados de los algoritmos en LA21

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	1175	4473	1	1461	60001
2	1152	112	2	1384	63501
3	1160	721	3	1350	63501
4	1143	6538	4	1393	61001
5	1172	1918	5	1325	60501
6	1161	3017	6	1446	60001
7	1161	4375	7	1424	55501
8	1122	4382	8	1338	59001
9	1171	119	9	1371	60501
10	1159	5985	10	1457	60001
11	1176	203	11	1405	61001
12	1150	1414	12	1376	51501
13	1157	1568	13	1306	62501
14	1180	4697	14	1363	60501
15	1160	5208	15	1304	57501
16	1165	3941	16	1388	58501
17	1174	6650	17	1516	57001
18	1178	1862	18	1383	31001
19	1163	4753	19	1341	56001
20	1170	6314	20	1322	58001
21	1149	5978	21	1447	55001
22	1162	4452	22	1390	56001
23	1107	4669	23	1328	56501
24	1175	2842	24	1318	50501
25	1179	1330	25	1355	60001
26	1174	5929	26	1474	54501
27	1159	5558	27	1387	59501
28	1179	3927	28	1445	59501
29	1169	3297	29	1362	57001
30	1166	3521	30	1456	56001
Mejor	1175	4473	Mejor	1304	31001
Promedio	1152	112	Promedio	1387,17	57384,33
Desviación	1160	721	Desviación	55,65	5870,27

Tabla 12: Resultados de los algoritmos en LA26

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	1353	860	1	1556	84001
2	1349	5860	2	1606	83001
3	1358	9160	3	1762	87501
4	1352	6600	4	1606	85501
5	1343	1940	5	1673	88001
6	1349	8080	6	1560	82501
7	1334	8020	7	1624	80501
8	1337	6780	8	1574	86501
9	1343	4580	9	1728	91001
10	1331	4650	10	1566	82001
11	1363	1570	11	1691	90001
12	1343	1690	12	1572	86501
13	1368	5000	13	1734	82001
14	1326	9430	14	1618	68501
15	1341	9550	15	1649	87001
16	1359	7000	16	1718	66501
17	1343	6290	17	1621	84501
18	1354	3660	18	1720	83001
19	1332	6940	19	1665	57001
20	1342	1810	20	1672	86001
21	1335	6340	21	1687	78501
22	1320	8170	22	1680	84001
23	1320	9620	23	1682	84501
24	1339	2230	24	1679	75501
25	1296	6580	25	1624	77501
26	1339	6980	26	1693	81501
27	1310	2590	27	1665	88001
28	1318	9350	28	1639	73001
29	1333	9530	29	1635	82001
30	1357	7800	30	1610	82501
Mejor	1296	860	Mejor	1556	57001
Promedio	1339,57	5955,33	Promedio	1650,30	81617,67
Desviación	16,16	2795,44	Desviación	55,47	7346,92

Figura 23: Gráfica del makespan de EAS y GRASP en LA21

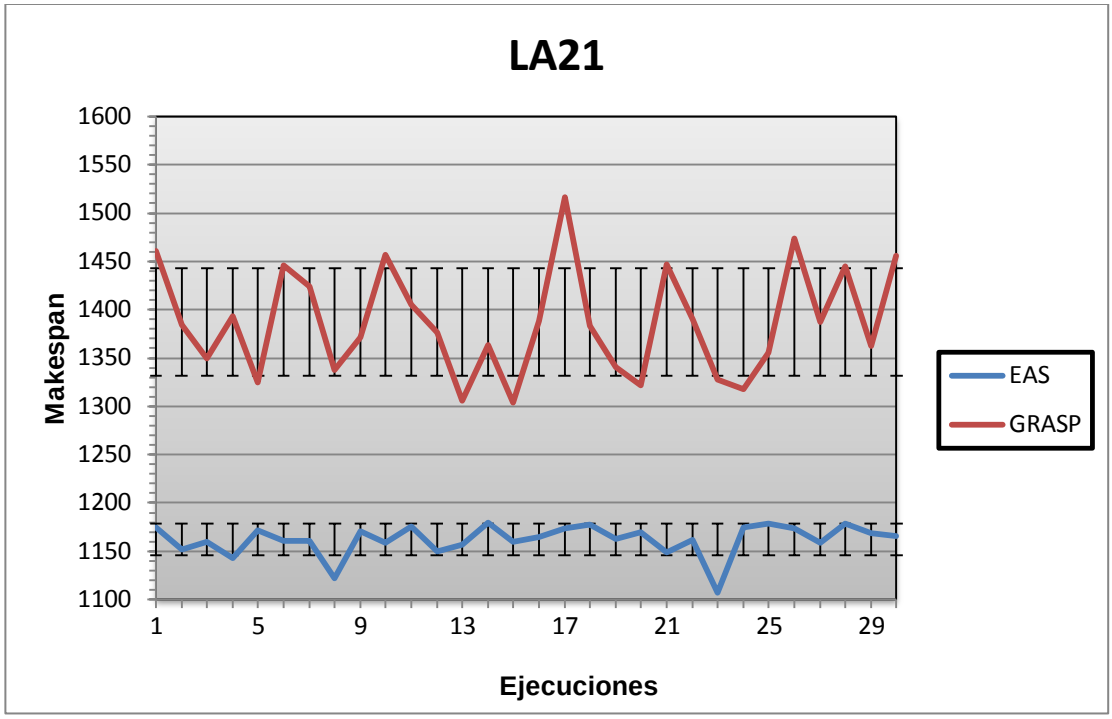


Figura 24: Gráfica del makespan de EAS y GRASP en LA26

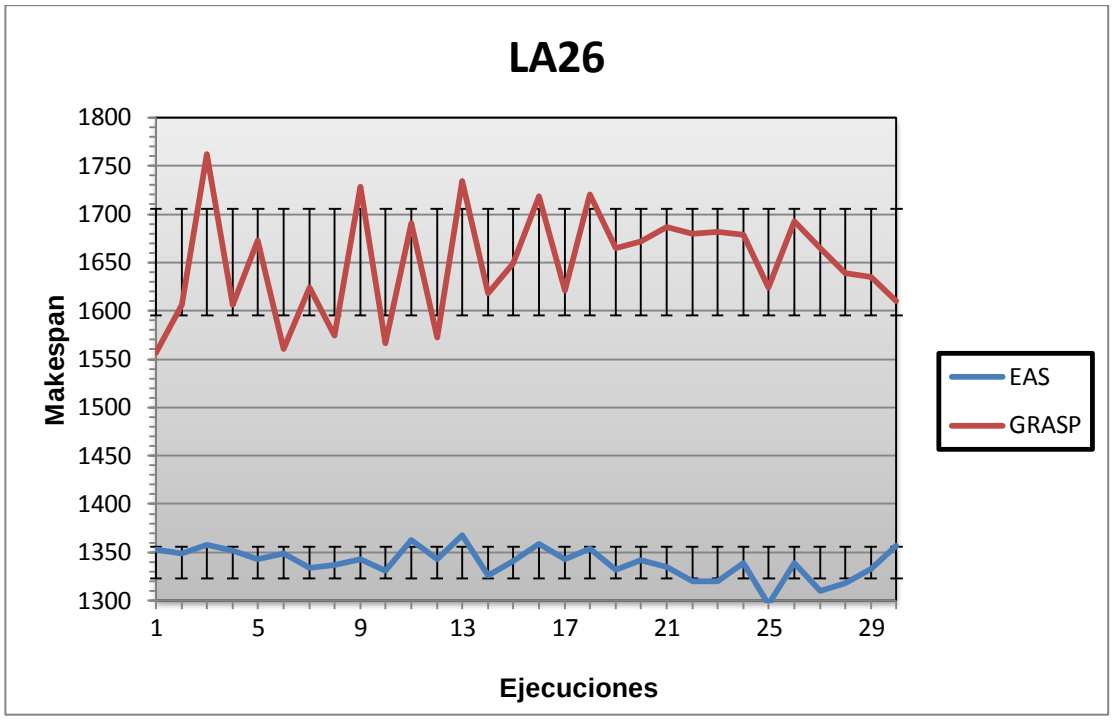


Tabla 13: Resultados de los algoritmos en LA31

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	1839	3885	1	2310	133501
2	1798	13920	2	2240	127001
3	1819	7470	3	2266	109501
4	1803	1410	4	2325	111501
5	1836	540	5	2233	131501
6	1815	5730	6	2230	129001
7	1822	2385	7	2275	122001
8	1818	12660	8	2317	129501
9	1826	13005	9	2220	122001
10	1820	11640	10	2367	129501
11	1841	11895	11	2277	132001
12	1843	10725	12	2231	124001
13	1809	345	13	2319	135001
14	1818	5850	14	2195	129501
15	1818	3195	15	2237	129501
16	1837	9915	16	2343	130001
17	1845	12000	17	2401	131501
18	1826	5790	18	2295	114501
19	1835	9765	19	2293	139501
20	1835	11085	20	2278	135501
21	1845	14145	21	2243	130501
22	1830	12390	22	2298	128501
23	1808	525	23	2363	136501
24	1835	6285	24	2285	96501
25	1827	4620	25	2267	133501
26	1821	7590	26	2308	124501
27	1817	7605	27	2259	136001
28	1823	3510	28	2177	139001
29	1825	375	29	2295	136001
30	1839	2670	30	2229	136001
Mejor	1798	345	Mejor	2177	96501
Promedio	1825,77	7097,50	Promedio	2279,20	128101,00
Desviación	12,47	4563,72	Desviación	51,67	9530,71

Tabla 14: Resultados de los algoritmos en LA36

	EAS			GRASP	
Ejecuciones	**C_{max}**	**Evaluaciones**	**Ejecuciones**	**C_{max}**	**Evaluaciones**
1	1410	2135	1	1655	93001
2	1444	1820	2	1766	94001
3	1396	3598	3	1714	90001
4	1447	3171	4	1948	89001
5	1446	5726	5	1781	86001
6	1444	2289	6	1817	91001
7	1397	3738	7	1556	83001
8	1440	1869	8	1774	88501
9	1417	4634	9	1805	91501
10	1408	4942	10	1589	95001
11	1456	483	11	1790	84501
12	1443	6111	12	1763	97001
13	1442	2303	13	1826	75501
14	1424	252	14	1778	85001
15	1417	2457	15	1642	83501
16	1428	5775	16	1670	93001
17	1425	5929	17	1873	92501
18	1408	1246	18	1696	82001
19	1447	4368	19	1704	90501
20	1416	882	20	1753	94001
21	1437	2429	21	1740	93501
22	1410	3108	22	1793	92001
23	1459	5852	23	1705	90001
24	1439	1827	24	1569	97001
25	1409	5278	25	1744	89501
26	1451	2751	26	1800	95001
27	1434	4137	27	1637	96001
28	1463	6944	28	1793	93001
29	1422	3031	29	1767	82501
30	1434	3066	30	1800	95501
Mejor	1396	252	Mejor	1556	75501
Promedio	1430,43	3405,03	Promedio	1741,60	90084,33
Desviación	18,53	1817,72	Desviación	87,79	5237,60

Figura 25: Gráfica del makespan de EAS y GRASP en LA31

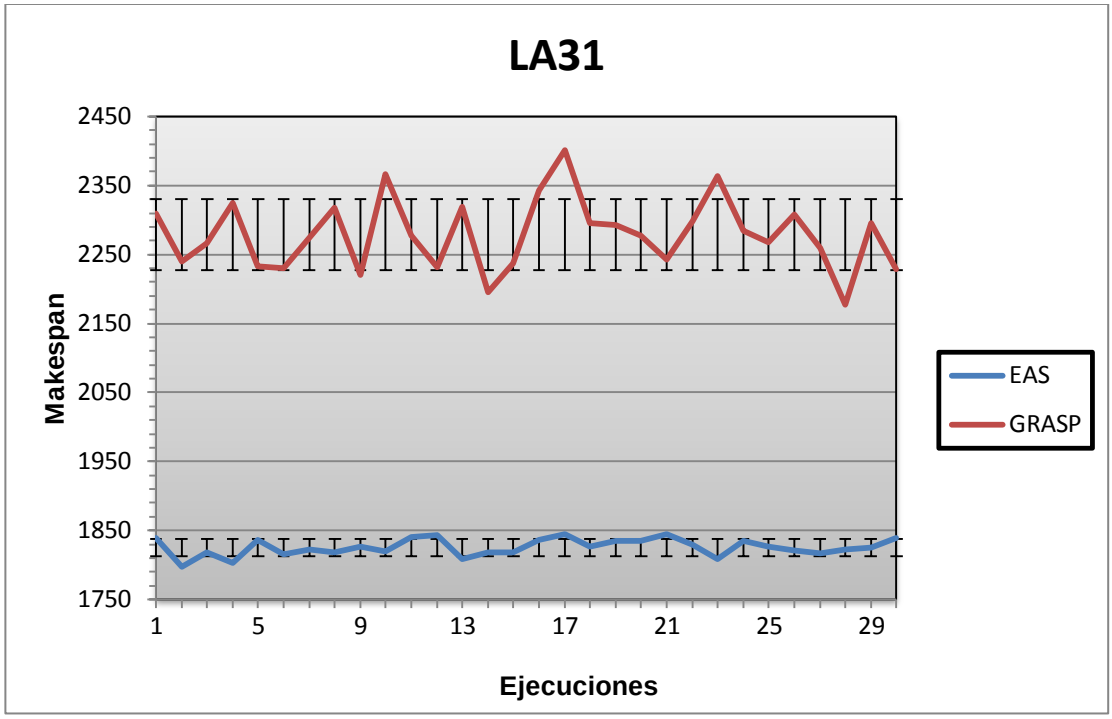


Figura 26: Gráfica del makespan de EAS y GRASP en LA36

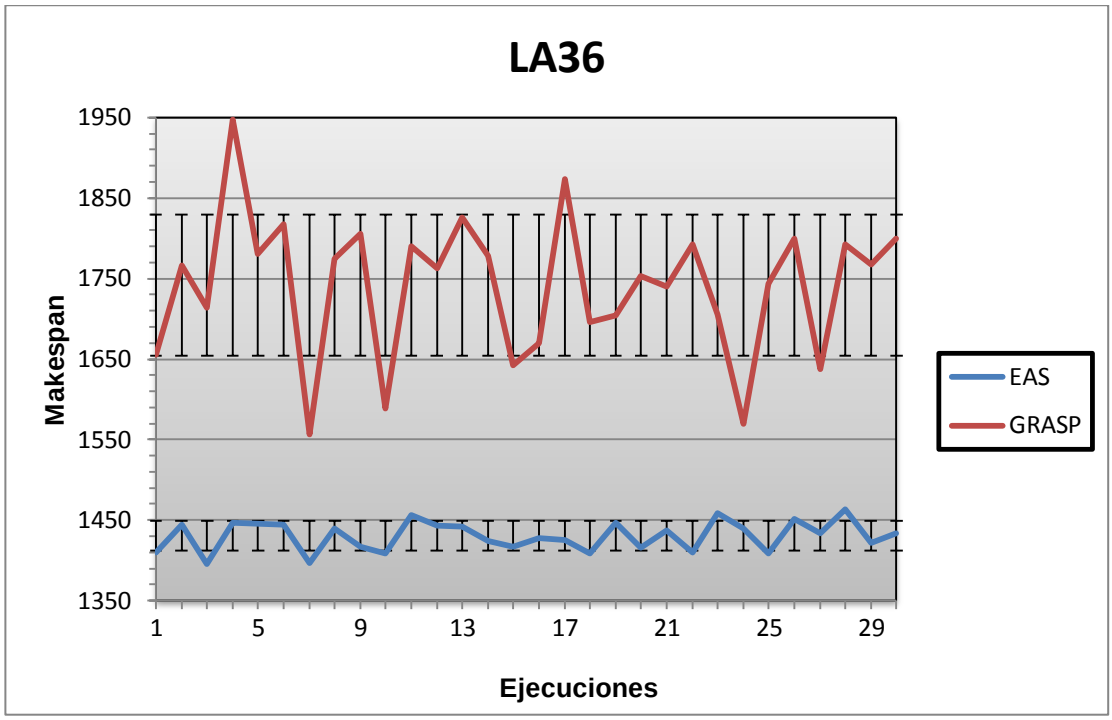


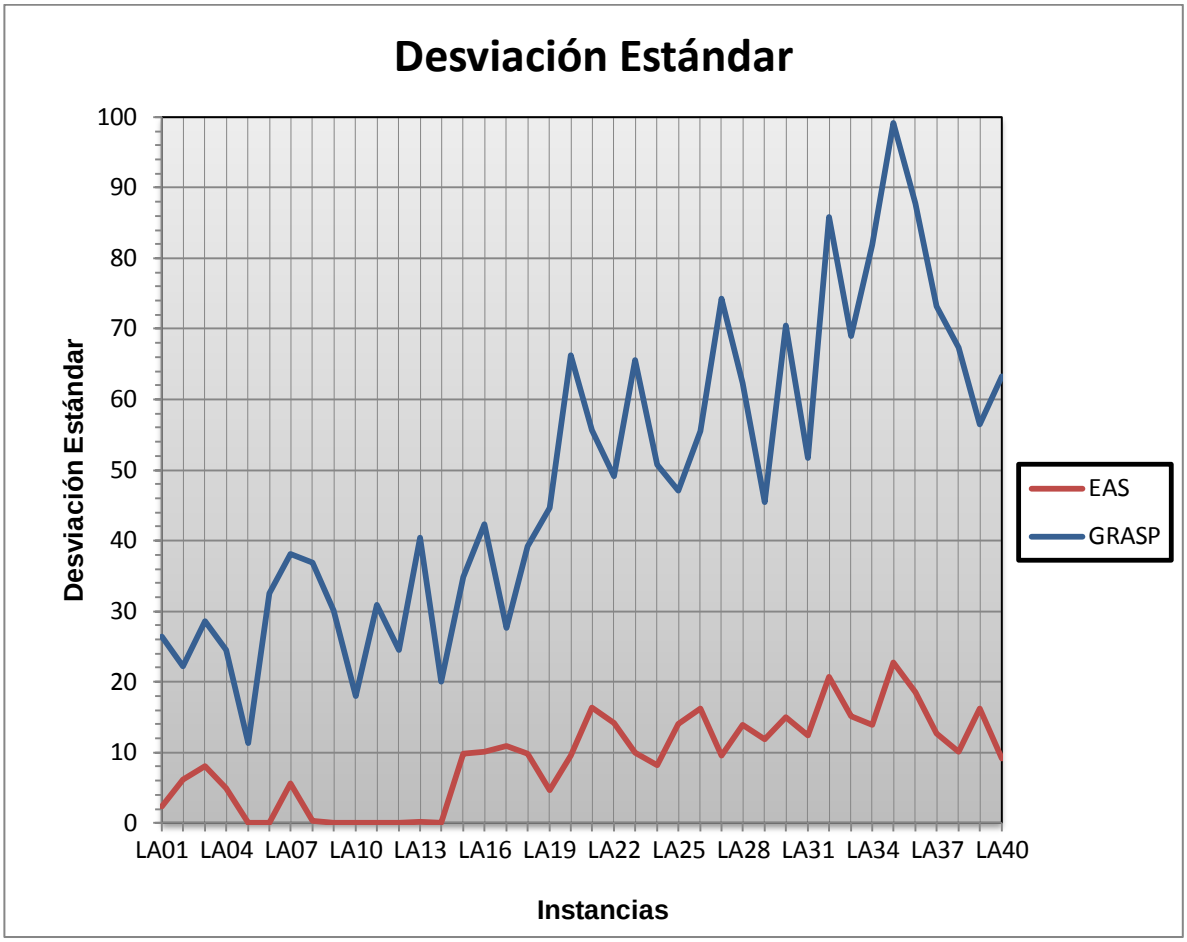
Tabla 15: Comparación de la estabilidad de EAS y GRASP

Instancia	Tamaño	EAS		GRASP	
		C_{max} promedio	Desviación Estándar	C_{max} promedio	Desviación Estándar
LA01	10 x 5	667,8	2,3	719,4	26,4
LA02	10 x 5	689,7	6,2	750,0	22,1
LA03	10 x 5	644,8	8,0	686,6	28,6
LA04	10 x 5	617,7	5,0	676,7	24,5
LA05	10 x 5	593,0	0,0	600,7	11,3
LA06	15 x 5	926,0	0,0	962,3	32,6
LA07	15 x 5	898,2	5,6	972,0	38,2
LA08	15 x 5	863,1	0,4	941,4	36,8
LA09	15 x 5	951,0	0,0	1010,0	30,1
LA10	15 x 5	958,0	0,0	975,6	18,0
LA11	20 x 5	1222,0	0,0	1291,4	31,0
LA12	20 x 5	1039,0	0,0	1110,6	24,5
LA13	20 x 5	1150,0	0,2	1223,6	40,5
LA14	20 x 5	1292,0	0,0	1306,7	20,0
LA15	20 x 5	1245,6	9,9	1407,2	34,8
LA16	10 x 10	1020,1	10,1	1103,4	42,3
LA17	10 x 10	836,1	10,9	910,8	27,6
LA18	10 x 10	904,8	9,8	1028,9	39,1
LA19	10 x 10	881,7	4,7	1015,7	44,7
LA20	10 x 10	936,8	9,6	1070,6	66,3
LA21	15 x 10	1162,3	16,4	1387,2	55,7
LA22	15 x 10	1050,2	14,2	1270,2	49,1
LA23	15 x 10	1069,2	10,0	1327,9	65,6
LA24	15 x 10	1033,5	8,3	1271,8	50,7
LA25	15 x 10	1093,3	14,1	1283,2	47,1
LA26	20 x 10	1339,6	16,2	1650,3	55,5
LA27	20 x 10	1379,8	9,6	1685,4	74,2
LA28	20 x 10	1363,8	13,9	1683,9	62,3
LA29	20 x 10	1374,4	11,9	1669,6	45,5
LA30	20 x 10	1443,2	15,0	1780,7	70,5
LA31	30 x 10	1825,8	12,5	2279,2	51,7
LA32	30 x 10	1906,0	20,7	2398,1	85,9
LA33	30 x 10	1771,0	15,1	2197,0	68,9
LA34	30 x 10	1823,9	13,9	2273,8	81,9
LA35	30 x 10	1974,1	22,8	2403,2	99,2
LA36	15 x 15	1430,4	18,5	1741,6	87,8
LA37	15 x 15	1544,2	12,7	1895,5	73,1
LA38	15 x 15	1343,8	10,1	1704,0	67,4
LA39	15 x 15	1359,5	16,2	1741,6	56,4
LA40	15 x 15	1323,7	9,2	1722,3	63,3
		Promedio:	9,09	Promedio:	48,78

En la tabla anterior, se muestra el promedio y desviación estándar (de las 30 ejecuciones) de las 40 instancias para establecer la estabilidad de los algoritmos (Ver Tabla 15).

Las diferencias observadas entre ambos algoritmos son significativas, siendo EAS mejor en el promedio de makespan y la desviación estándar de todas las instancias (ver Tabla 15). EAS es mucho más estable, presentando resultados menos dispersos con una desviación estándar promedio muy inferior (de solo 9,09 respecto al 48,78 de GRASP), por lo que el mejor makespan se obtiene con más frecuencia que en GRASP (Ver Figura 27).

Figura 27: Gráfica de desviación estándar de EAS y GRASP por instancia.



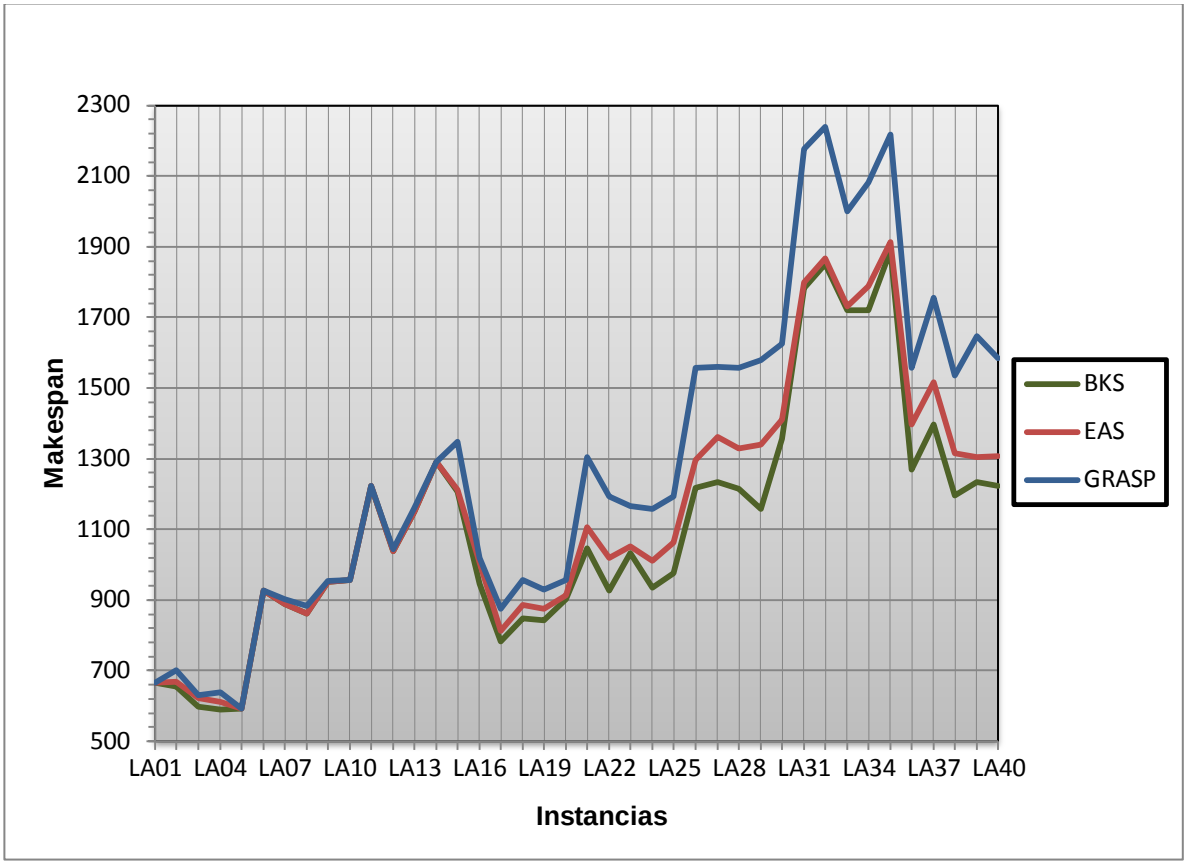
En la Tabla 16 se muestra en orden, el nombre de la instancia de Lawrence, su tamaño y BKS, el mejor makespan hallado y su error relativo porcentual respecto al BKS, y por último el número de evaluaciones promedio de la función objetivo.

Tabla 16: Comparación de resultados respecto al BKS

Instancia	Tamaño	BKS	EAS			GRASP		
			C_{max}	Error Relativo (%)	#Eval. Promedio	C_{max}	Error Relativo (%)	#Eval. Promedio
LA01	10 x 5	666	666	0	2375	666	0	13634
LA02	10 x 5	655	669	2.13	2809	702	7,18	10301
LA03	10 x 5	597	623	4.36	2230	630	5,53	11568
LA04	10 x 5	590	611	3.56	2257	638	8,14	12734
LA05	10 x 5	593	593	0	101	593	0	9001
LA06	15 x 5	926	926	0	531	926	0	24934
LA07	15 x 5	890	890	0	3443	902	1,35	22001
LA08	15 x 5	863	863	0	2251	883	2,32	20351
LA09	15 x 5	951	951	0	391	954	0,32	21751
LA10	15 x 5	958	958	0	637	958	0	21618
LA11	20 x 5	1222	1222	0	1504	1222	0	33334
LA12	20 x 5	1039	1039	0	1752	1045	0,58	30468
LA13	20 x 5	1150	1150	0	2952	1160	0,87	32418
LA14	20 x 5	1292	1292	0	471	1292	0	24034
LA15	20 x 5	1207	1212	0.41	3836	1348	11,68	23068
LA16	10 x 10	945	1005	6.35	2700	1020	7,94	31384
LA17	10 x 10	784	812	3.57	2401	874	11,48	33318
LA18	10 x 10	848	885	4.36	2946	956	12,74	33001
LA19	10 x 10	842	875	3.92	2394	930	10,45	33551
LA20	10 x 10	902	912	1.11	2496	958	6,21	31751
LA21	15 x 10	1046	1107	5.38	3658	1304	24,67	57384
LA22	15 x 10	927	1018	9.82	2938	1192	28,59	56484
LA23	15 x 10	1032	1051	1.84	3826	1167	13,08	57084
LA24	15 x 10	935	1011	8.13	3097	1159	23,96	56701
LA25	15 x 10	977	1062	8.7	3632	1192	22,01	58884
LA26	20 x 10	1218	1296	6.4	5955	1556	27,75	81618
LA27	20 x 10	1235	1362	10.28	4450	1560	26,32	81984
LA28	20 x 10	1216	1330	9.38	3938	1558	28,12	83118
LA29	20 x 10	1157	1339	15.73	4532	1580	36,56	80734
LA30	20 x 10	1355	1410	4.06	5186	1626	20	78751
LA31	30 x 10	1784	1798	0.78	7098	2177	22,03	128101
LA32	30 x 10	1850	1868	0.97	8016	2238	20,97	130151
LA33	30 x 10	1719	1731	0.7	5796	2000	16,35	131301
LA34	30 x 10	1721	1788	3.89	6811	2082	20,98	127084
LA35	30 x 10	1888	1913	1.32	7357	2218	17,48	122734
LA36	15 x 15	1268	1396	10.09	3405	1556	22,71	90084
LA37	15 x 15	1397	1517	8.59	2142	1756	25,7	90601
LA38	15 x 15	1196	1315	9.95	4051	1536	28,43	90668
LA39	15 x 15	1233	1304	5.76	3266	1647	33,58	90334
LA40	15 x 15	1222	1307	6.96	2655	1585	29,71	90851
Promedio:				3.96	3307		14.39	56472

EAS obtiene un makespan menor que el de GRASP en 34 instancias (Ver Tabla 16), GRASP nunca logra superar a EAS, solo alcanza a igualar sus resultados en 6 instancias en las que ambos obtienen el BKS (Ver Figura 28).

Figura 28: Gráfica del mejor makespan de EAS y GRASP respecto al BKS



Aunque la eficacia de EAS para encontrar el óptimo no es alta, alcanzando el BKS solo en el 27.5% de las instancias (Ver Figura 29), el promedio del error relativo en las 40 instancias es de solo 3.96% (Ver Tabla 16), lo que es una buena aproximación a la óptima de JSP. Y lo más destacable es el bajo número de evaluaciones promedio de la función objetivo, que es muy inferior comparado con GRASP y otras metaheurísticas que tienen los mejores resultados en las instancias de Lawrence (Ver Tabla 17). Respecto a GRASP, el EAS realiza 17 veces menos evaluaciones. La metaheurística AIS en promedio realiza 52 veces más evaluaciones que EAS, y CULT realiza 137 veces más evaluaciones.

Comparado con Tabu Search (TS) el número de evaluaciones es solo es 3 veces menor porque no se incluyen el número de evaluaciones realizadas por el algoritmo INSA, que produce la solución inicial de la que parte la búsqueda de Tabu Search [28].

Así se puede afirmar que EAS algoritmo tiene una gran eficiencia computacional, reduciendo los costos en tiempo y memoria, algo muy importante en este tipo de problemas en los que conseguir una solución "económica" es tan importante como la calidad de la misma.

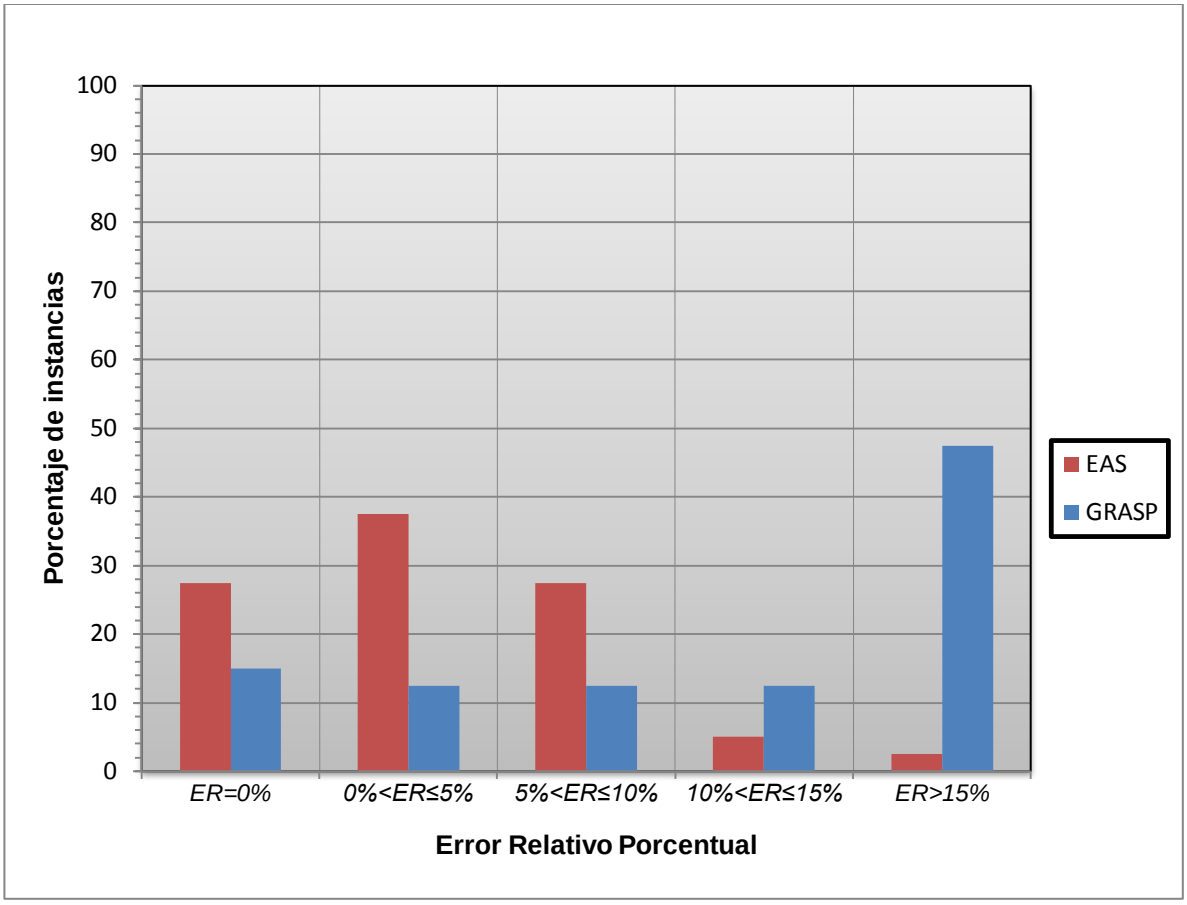
En la siguiente tabla se compara el número promedio de evaluaciones de la función objetivo realizado mediante EAS, con los realizados por GRASP, TS, AIS y CULT:

Tabla 17: Número de evaluaciones comparado con otras metaheurísticas [40]

Algoritmo	**N° de Evaluaciones promedio**
EAS	3307
GRASP	56472
AIS	175058
CULT	454525
TS	11108

Aunque en los estudios no se acostumbra a comparar el tiempo de ejecución de los algoritmos, se puedo ver que el tiempo de ejecución de EAS fue mucho menor que GRASP, sobretodo porque realiza menos evaluaciones de la función objetivo.

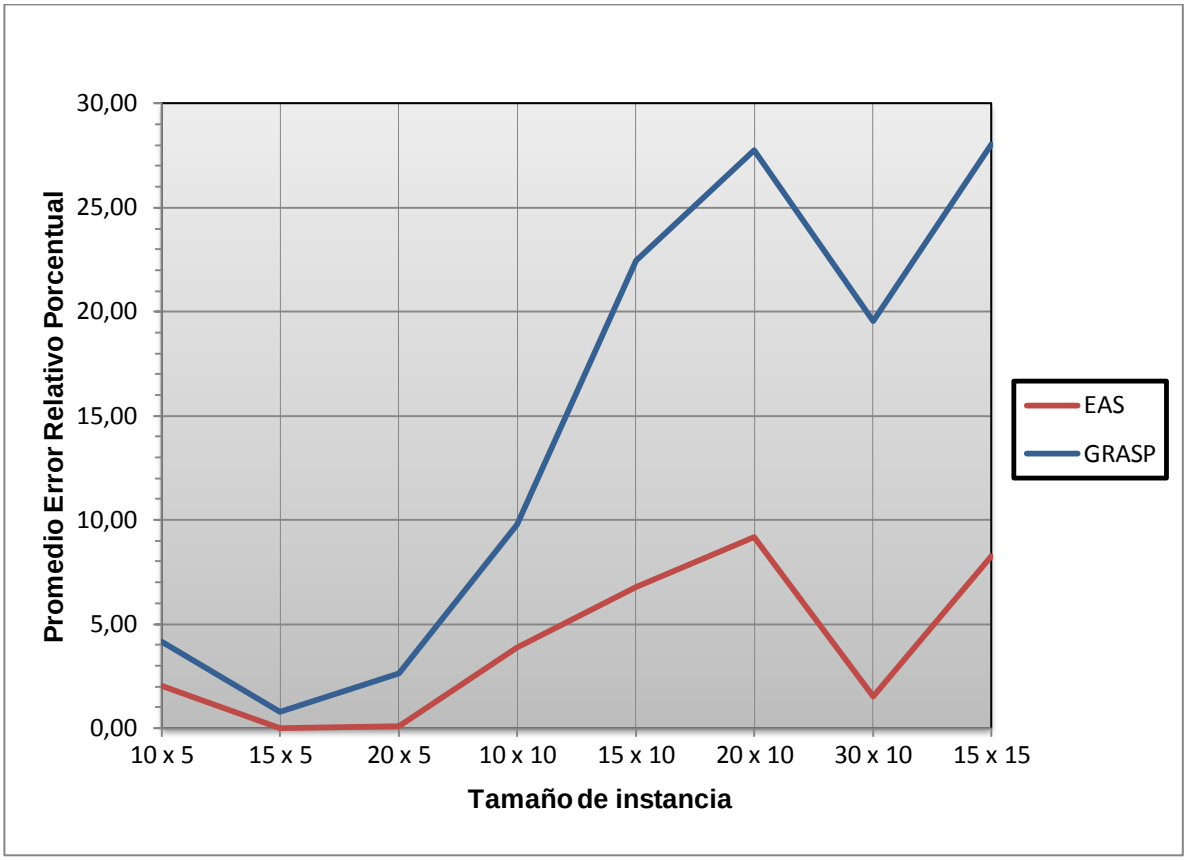
Figura 29: Distribución de las instancias en cada algoritmo según el error relativo



En la Tabla 16 se observa que en los problemas de tamaño 10 x 5 no se tienen problemas para encontrar el BKS, con la excepción de la instancia LA02 hasta la LA04, donde el mejor resultado obtenido está cerca al BKS (a menos de 5%). Igualmente para las instancias de tamaño 15 x 5 y 20 x 5, con la excepción de la LA15 que solo se aleja del BKS en 5 unidades de tiempo. En las demás instancias (de tamaño 10 x 10, 15 x 10, 20 x 10, 30 x 10 y 15 x 15), que tienen 5 o 10 máquinas más que las anteriores, la complejidad es muy alta debido al considerable número de operaciones que se deben realizar, por lo que se obtienen soluciones de una menor calidad. Por ejemplo, para las instancias de tamaño 30 x 10, se deben realizar 300 operaciones, y el número total de combinaciones posibles es de $(30!)^{10}$, que es aproximadamente 2.65×10^{42}. A pesar de ello, el algoritmo logra presentar soluciones de gran calidad en las instancias de 30 x 10,

con un error relativo promedio en esas 5 instancias de menos de la mitad (solo 1,5%) del error promedio de todas las 40 instancias (Ver Figura 30).

Figura 30: Error relativo promedio por tamaño de instancia



En general, 65% de las instancias ejecutadas se aproximan a menos de 5% del BKS (Ver Figura 29), y 47.5% se desvían en menos del 3% del BKS. Por consiguiente, EAS es un algoritmo que tiene el potencial de obtener la mejor solución conocida de estas instancias JSP, a un costo mínimo de tiempo y recursos computacionales.

7. CONCLUSIONES

- El algoritmo implementado basado en colonia de hormigas, el *Elistist Ant System*, ha presentado mejores resultados respecto a GRASP sobre la familia de instancias LA del problema de *Job Shop Scheduling* (JSP), y con un número de evaluaciones (de la función objetivo) menor a GRASP y a las metaheurísticas que obtienen el BKS en estas instancias. El algoritmo EAS presenta un comportamiento más estable que GRASP, porque en las instancias evaluadas la desviación estándar es menor, y obtiene su mejor solución con más frecuencia porque el makespan promedio se encuentra más cerca del mejor makespan obtenido.
- Comparado con las metaheurísticas que logran las mejores soluciones conocidas en la mayoría de instancias LA, el *Elistist Ant System* tiene un buen desempeño con soluciones muy aproximadas al BKS, mostrando un grado de proximidad de 96.04% en promedio de todas las instancias LA. Alcanza el BKS en 11 de las 40 instancias, pero la calidad de sus soluciones fue inferior en las instancias de mayor tamaño.
- Los sistemas distribuidos son un buen escenario de aplicación de JSP, aunque se deba adaptar al procesamiento en paralelo realizado en clústeres y grids. En el escenario planteado EAS y GRASP consiguieron resultados iguales o muy similares en el promedio, desviación estándar y mejor makespan.

En definitiva, la *Optimización basada en colonia de hormigas* (OCH) es un método aproximado muy eficiente, que puede encontrar soluciones de buena calidad a problemas de planificación de recursos.

8. RECOMENDACIONES

Como trabajo futuro se propone aplicar OCH al problema de *Job Scheduling* que se presenta en las Grids, enfocándose en uno de los principales objetivos de los sistemas distribuidos, prestar un servicio eficiente a cada usuario evitando retardos en los trabajos.

Para mejorar el desempeño de OCH se recomienda trabajar sobre un método que permita limitar el conjunto de operaciones factibles a solo las más adecuadas (puede ser un método parecido al que genera la lista RCL de GRASP), para que las hormigas puedan tomar la mejor decisión posible en cada uno de sus pasos, que reduzca el tiempo en el que las máquinas están ociosas.

También se puede aplicar Búsqueda Local para mejorar OCH (como se hace en GRASP), que se propone en el pseudocódigo de OCH para el *Servidor de Acciones*. Se debe aprovechar el bajo número de evaluaciones que realiza OCH, realizando una búsqueda local extensa para alcanzar los resultados de las mejores técnicas que han abordado el JSP.

En el caso de GRASP, la forma de mejorar su desempeño es utilizando una solución inicial de más calidad, obtenida por otra metaheurística como OCH, porque es un método que depende en gran medida de esto para poder encontrar la solución óptima.

9. BIBLIOGRAFÍA

[1] M. Dorigo, V. Maniezzo, and V. M. Colorni, "*The Ant System: Optimization by a colony of cooperating agents*," in IEEE: Transactions on Systems, Man, and Cybernetics, Part B, vol. 26, no. 1, pp. 29-41, 1996.

[2] M. Dorigo and L. M. Gambardella, "*Ant Colony System: A cooperative learning approach to the Traveling Salesman Problem*," IEEE: Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 53-66, 1997.

[3] M. Dorigo and K. Socha, "*An introduction to Ant Colony Optimization*," Technical report N° 10 of the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université Libre de Bruxelles, Belgium, 2006.

[4] S. Alonso, O. Cordón, I. Fernández, and F. Herrera, "*La metaheurística de Optimización basada en Colonias de Hormigas: Modelos y nuevos enfoques*," Trabajo realizado en el marco del proyecto Mejora de Metaheurísticas mediante Hibridación y sus Aplicaciones, de la Universidad de Granada, España, 2004.

[5] V. Peña and L. Zumelzu, Estado del arte del Job Shop Scheduling Problem, Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile, 2006.

[6] J. Blazewicz, K. H. Ecker, G. Schmidt and J. Weglarz. Scheduling in Computer and Manufacturing System. Springer, 1994.

[7] T. Stützle and H.H. Hoos, "*MAX-MIN Ant System".* Future Generation Computer Systems, vol. 16, no. 8, pp. 889-914, 2000.

[8] B. Bullnheimer, R.F. Hartl and C. Strauss, "*A New Rank-Based version of the Ant System: A computational study,"* Central European Journal for Operations Research and Economics, vol. 7, no. 1, pp. 25-38, 1999.

[9] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” ACM Computing Surveys, vol. 35, no. 3, pp. 268-308, 2003.

[10] M. Ben-Daya and M. Al-Fawzan, A tabu search approach for the flow shop scheduling problem. In European Journal of Operational Research, vol. 109, pp. 88-95, 1998.

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

[12] S. Goss, S. Aron, J. Deneubourg, and J. Pasteels, “*Self-organized shortcuts in the Argentine ant,”* Naturwissenschaften, vol. 76, no.12, pp. 579-581, 1989.

[13] D. Applegate and W. Cook, “*A computational study of the job-shop scheduling problem*,” In ORSA Journal on Computing, vol. 3, No. 2, pp. 149–156, 1991.

[14] R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra, “*Job Shop Scheduling by Local Search*,” INFORMS J. Comput., vol. 8, no. 3, pp. 302–317, 1996.

[15] M. Ventresca and B. M. Ombuki, “*Ant Colony Optimization for Job Shop Scheduling Problem*,” Tech. Rep. N° CS-04-04, Department of Computer Science, Brock University, 2004.

[16] O. Cordon, Sistemas Complejos, Algoritmos Evolutivos y Bioinspirados. Universidad de Granada, España, 2005.

[17] M. R. Garey, “*The complexity of Flowshop and Job Shop Scheduling*,” Mathematics of Operations Research, vol. 1, no. 2, pp. 117–129, 1976.

[18] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, 1979.

[19] S. Cook, “The complexity of theorem proving procedures”. Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158, 1971.

[20] V. A. Peña, Cadena de Suministros: sus niveles e importancia. Modelado de Procesos de Negocios, 2006.

[21] A. Manne, "*On the Job-Shop Scheduling Problem*", Operations Research, vol. 8, no. 2, pp. 219-223, 1960.

[22] J. L. Denebourg, J. M. Pasteels, and J. C. Verhaeghe, "*Probabilistic behaviour in ants: a strategy of errors*?", Journal of Theoretical Biology, no. 105, 1983.

[23] A. H. Land and A. G. Doig, "*An automatic method of solving discrete programming problems*". Econometrica, vol. 28, no 3, pp. 497–520, 1960.

[24] T. Yamada and R. Nakano. Job-shop scheduling. In P. J. Fleming A. M. S. Zalzala, editor, Genetic Algorithms in Engineering Systems, chapter 7, pp. 134–160. IET, 1999.

[25] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, Massachusetts, USA: The MIT Press, 2004.

[26] M. Dorigo, Optimization, Learning and Natural Algorithms. Ph.D.Thesis, Politecnico di Milano, Italy, 1992.

[27] C. Coello Coello, D. Cortez Rivera, and N. Cruz Cortez, "*Use of an artificial immune system for job shop scheduling,"* in Artificial Immune Systems," Proceedings of ICARIS, Lecture Notes in Computer Science vol. 2787, pp. 1–10, 2003.

[28] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," Management Science, vol. 42, no. 6, pp. 797–813, 1996.

[29] S. Cook, (April 2000). The P versus NP Problem. Clay Mathematics Institute. Retrieved 18 October 2006.

[30] G. Flake, The computational beauty of nature: computer explorations of fractals, chaos, complex systems, and adaptation. Cambridge, Mass: MIT Press, 1998.

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

[32] J. Deneubourg, S. Camazine, N. Franks, J. Sneyd, G. Theraulaz and E. Bonabeau, *Self-Organization in Biological Systems*, Princeton University Press, 2003.

[33] D. E. Jackson and F. L. Ratnieks. *Communication in ants*. Current Biology 16 (15), pp. 570–574, 2006.

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

[35] S. Alonso, O. Cordón, I. Fernández y F. Herrera. *La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques*. Trabajo realizado en el marco del proyecto Mejora de Metaheurísticas mediante Hibridación y sus Aplicaciones, de la Universidad de Granada, España, 2004.

[36] M. Resende y J. L. González Velarde. GRASP Procedimientos de búsquedas miopes aleatorizados y adaptativos. *Revista Iberoamericana de Inteligencia Artificial, 7*(19), 61-76, 2003.

[37] M. Ebling, M. Di Loreto, M. Presley, F. Wieland, et D. Jefferson,*An Ant Foraging Model Implemented on the Time Warp Operating System*, Proceedings of the SCS Multiconference on Distributed Simulation, 1989.

[38] G. Aloysius and B. R. Rajakumar, Fuzzy Aided Ant Colony Optimization Algorithm to Solve Optimization Problem, Intelligent Informatics, Advances in Intelligent Systems and Computing, vol. 182, pp. 207-215, 2013.

[39] A. H. Land and A. G. Doig. "An automatic method of solving discrete programming problems". *Econometrica* 28 (3): pp. 497–520, 1960.

[40] Emanuel Téllez Enríquez. Uso de una Colonia de Hormigas para resolver Problemas de programación de Horarios, 2007.

[41] John and Sarah. "Interesting Facts About Ants". Retrieved December 23, 2010. http://www.lingolex.com/ants.htm

[42] M. Resende and C. Ribeiro (2002). GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES. AT&T Labs Research Technical Report TD-

53RSJY, version 2. To appear in State of the Art Handbook in Metaheuristics, F. Glover and G. Kochenberger, eds., Kluwer, 2002.

[43] Daily Mail Reporter. *In a jam? Intelligent ants could put an end to traffic snarl-ups*. Published by Associated Newspapers Ltd, Part of the Daily Mail, 6 November 2008. http://www.dailymail.co.uk/sciencetech/article-1083475/In-jam-Intelligent-ants-end-traffic-snarl-ups.html

[44] K. Taylor. *Leaf-cutting ants (Atta cephalotes) carrying leaf sections back to nest. Trinidad, West Indies also ocurrs in South America*. Nature Picture Library (2002-2013). http://www.naturepl.com/cache/pcache/01154201.jpg

[45] J. E. Beasley, *Distributing test Problems by Electronic Mail*, in OR-Library, Journal of Operations Research Society, pp. 1069-1072, 1990.

[46] A. Udomsakdigool and V. Khachitvichyanukul. Ant colony algorithm for multi-criteria job shop scheduling. International Journal of Management Science and Engineering Management. 6(2): 117-123, 2011.

[47] S. Van der Zwaan and C. Marques, Ant colony optimisation for job shop scheduling. In *Proceedings of the third workshop on genetic algorithms and artificial life (GAAL 99)*, 1999.

[48] Ó. Y. Buitrago Suescun, R. A. Britto Agudelo y G. Mejía Delgadillo. Minimización de la tardanza ponderada total en talleres de manufactura aplicando colonia de hormigas. Revista *Ingeniería*, *14*(1), 25-30, 2008.

[49] A. B. Badiru. Handbook of industrial and systems engineering. CRC Press, 2005.

[50] J. Montgomery, C. Fayad and S. Petrovic. Solution representation for job shop scheduling problems in ant colony optimisation. In *Ant Colony Optimization and Swarm Intelligence,* Springer Berlin, pp. 484-491, 2006.

[51] T. Bäck (1996). Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press on Demand.

[52] D. E. Goldberg (1989). Genetic algorithms in search, optimization, and machine learning.

[53] H. R. Lourenço, O. C. Martin and T. Stützle, (2001). Iterated local search. arXiv preprint math/0102188.

[54] F. Glover and M. Laguna. Tabu search. Vol. 22. Boston: Kluwer academic publishers, 1997.

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

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

[57] W. N. Chen and J. ZHANG "Ant Colony Optimization Approach to Grid Workflow Scheduling Problem with Various QoS Requirements", IEEE Transactions on Systems, Man, and Cybernetics--Part C: Applications and Reviews, Vol. 31, No. 1, pp.29-43, 2009.

[58] S. Lorpunmanee, M. N. Sap, A. H. Abdullah and C. Chompoo-inwai. An ant colony optimization for dynamic job scheduling in grid environment. *International Journal of Computer and Information Science and Engineering*, *1*(4), pp. 207-214, 2007.

[59] L. N. De Castro and J. Timmis. Artificial Immune Systems: A New Computational Intelligence Approach. Springer. pp. 57–58, 2002.

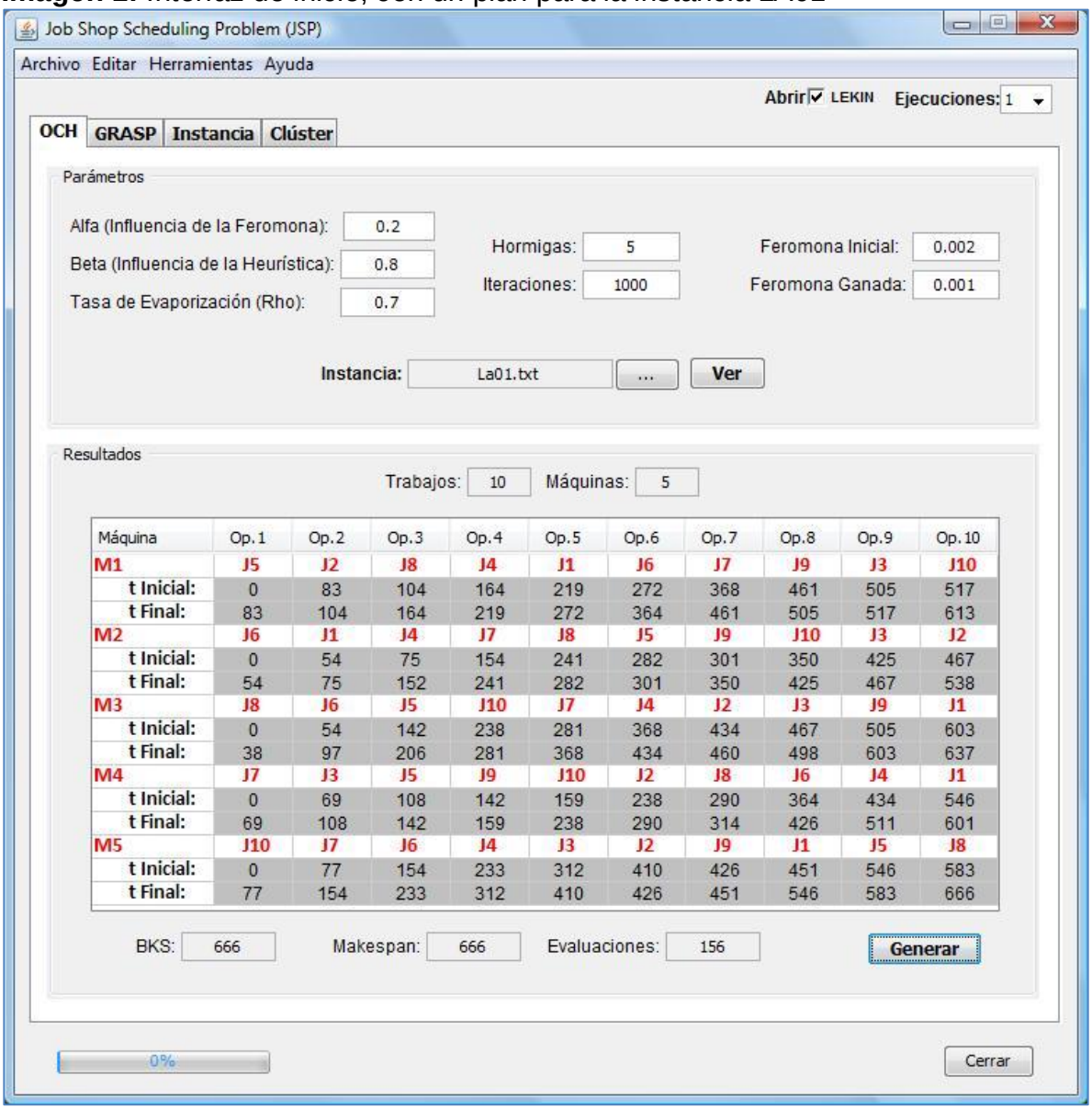
[60] M. Perretto and S. Lopes, Reconstruction of phylogenetic trees using the ant colony optimization paradigm. Genetics and Molecular Research, 4(3), pp. 581-589, 2005.

10. ANEXOS

Anexo A. FUNCIONAMIENTO DEL PROTOTIPO

La interfaz de inicio del prototipo tiene 4 pestañas (OCH, GRASP, Instancia y Clúster) donde se manejan las principales funciones, y una barra de menú para funciones adicionales.

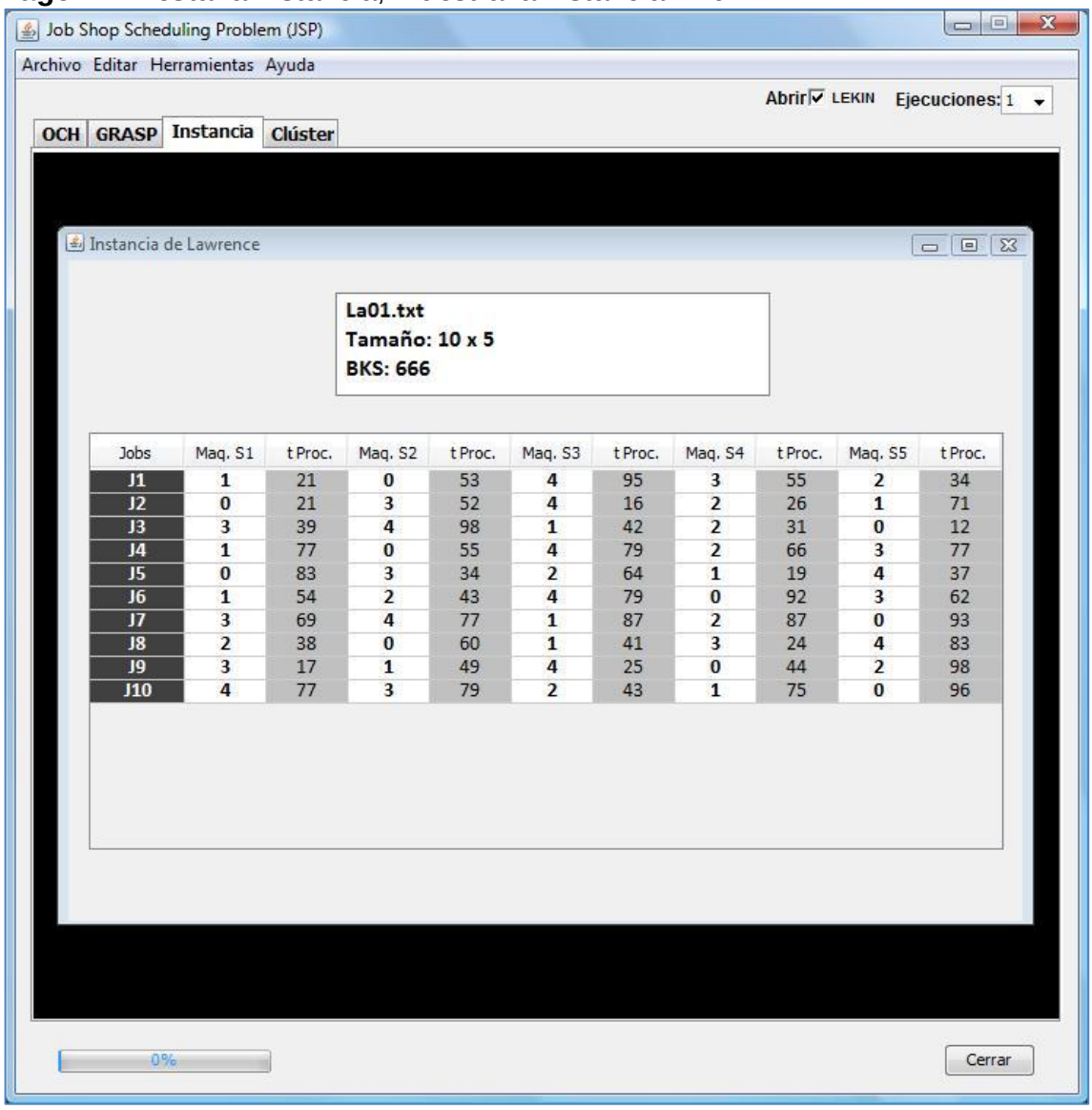
Imagen 1: Interfaz de inicio, con un plan para la instancia LA01



Máquina	Op.1	Op.2	Op.3	Op.4	Op.5	Op.6	Op.7	Op.8	Op.9	Op.10
M1	J5	J2	J8	J4	J1	J6	J7	J9	J3	J10
t Inicial:	0	83	104	164	219	272	368	461	505	517
t Final:	83	104	164	219	272	364	461	505	517	613
M2	J6	J1	J4	J7	J8	J5	J9	J10	J3	J2
t Inicial:	0	54	75	154	241	282	301	350	425	467
t Final:	54	75	152	241	282	301	350	425	467	538
M3	J8	J6	J5	J10	J7	J4	J2	J3	J9	J1
t Inicial:	0	54	142	238	281	368	434	467	505	603
t Final:	38	97	206	281	368	434	460	498	603	637
M4	J7	J3	J5	J9	J10	J2	J8	J6	J4	J1
t Inicial:	0	69	108	142	159	238	290	364	434	546
t Final:	69	108	142	159	238	290	314	426	511	601
M5	J10	J7	J6	J4	J3	J2	J9	J1	J5	J8
t Inicial:	0	77	154	233	312	410	426	451	546	583
t Final:	77	154	233	312	410	426	451	546	583	666

Las pestañas OCH y GRASP son similares, contienen un panel para definir todos los parámetros de los algoritmos, y otro donde se muestran los resultados (makespan, número de evaluaciones de la función objetivo y la matriz del flujo de trabajos en cada máquina con el correspondiente tiempo de inicio y finalización de cada operación) de la ejecución. Además contienen botones para seleccionar y ver la instancia que se va a ejecutar. Para ver la instancia se pasa a la pestaña Instancia, donde se muestra en una tabla, junto a los datos de su tamaño y BKS (si es conocido).

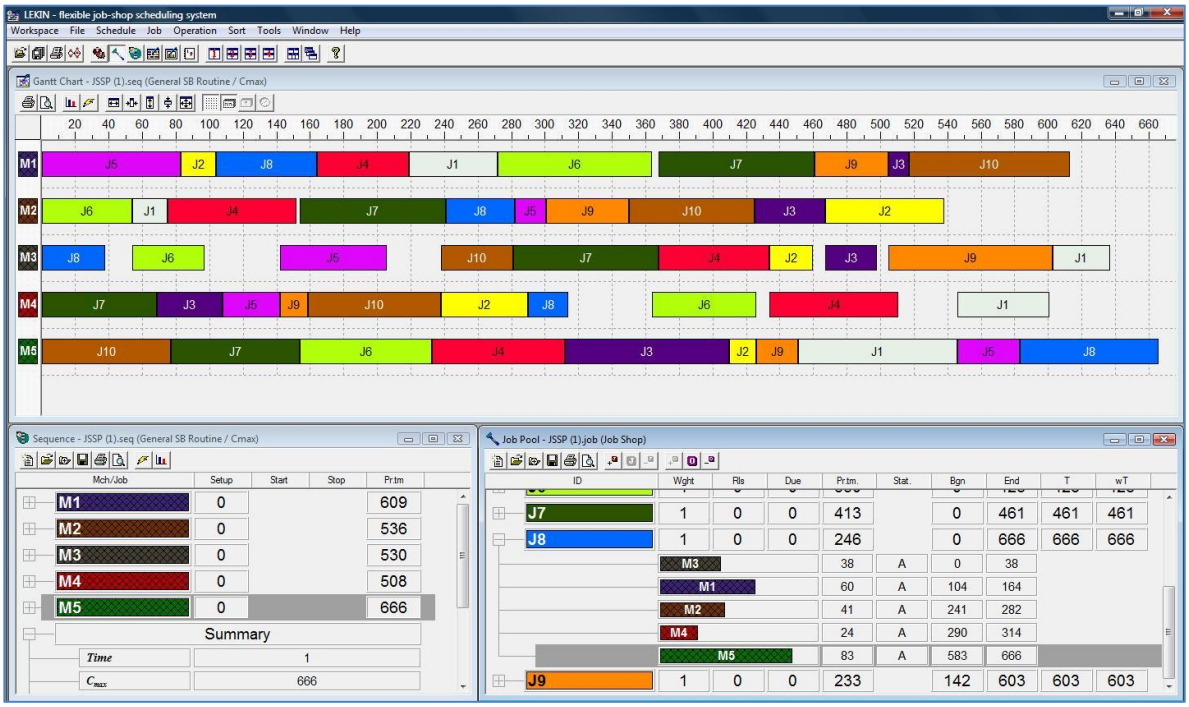
Imagen 2: Pestaña Instancia, muestra la instancia LA01



Jobs	Maq. S1	t Proc.	Maq. S2	t Proc.	Maq. S3	t Proc.	Maq. S4	t Proc.	Maq. S5	t Proc.
J1	1	21	0	53	4	95	3	55	2	34
J2	0	21	3	52	4	16	2	26	1	71
J3	3	39	4	98	1	42	2	31	0	12
J4	1	77	0	55	4	79	2	66	3	77
J5	0	83	3	34	2	64	1	19	4	37
J6	1	54	2	43	4	79	0	92	3	62
J7	3	69	4	77	1	87	2	87	0	93
J8	2	38	0	60	1	41	3	24	4	83
J9	3	17	1	49	4	25	0	44	2	98
J10	4	77	3	79	2	43	1	75	0	96

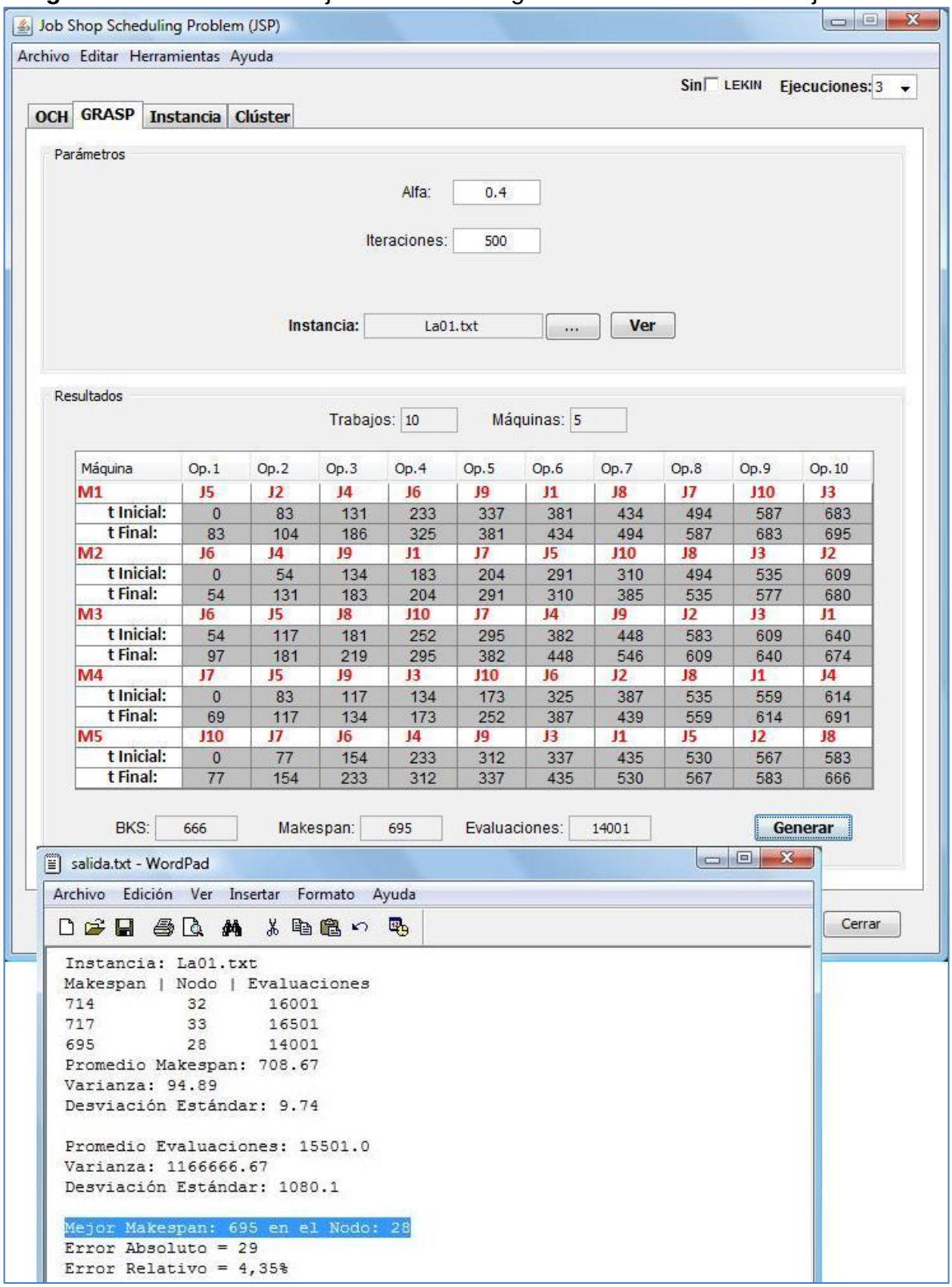
Tambien se tiene un casilla de verificación (checkbox) para determinar si se abre Lekin, el cual es utilizado para verificar que el plan generado es correcto.

Imagen 3: Interfaz de LEKIN con el diagrama de Gantt para el plan de la Imagen 1



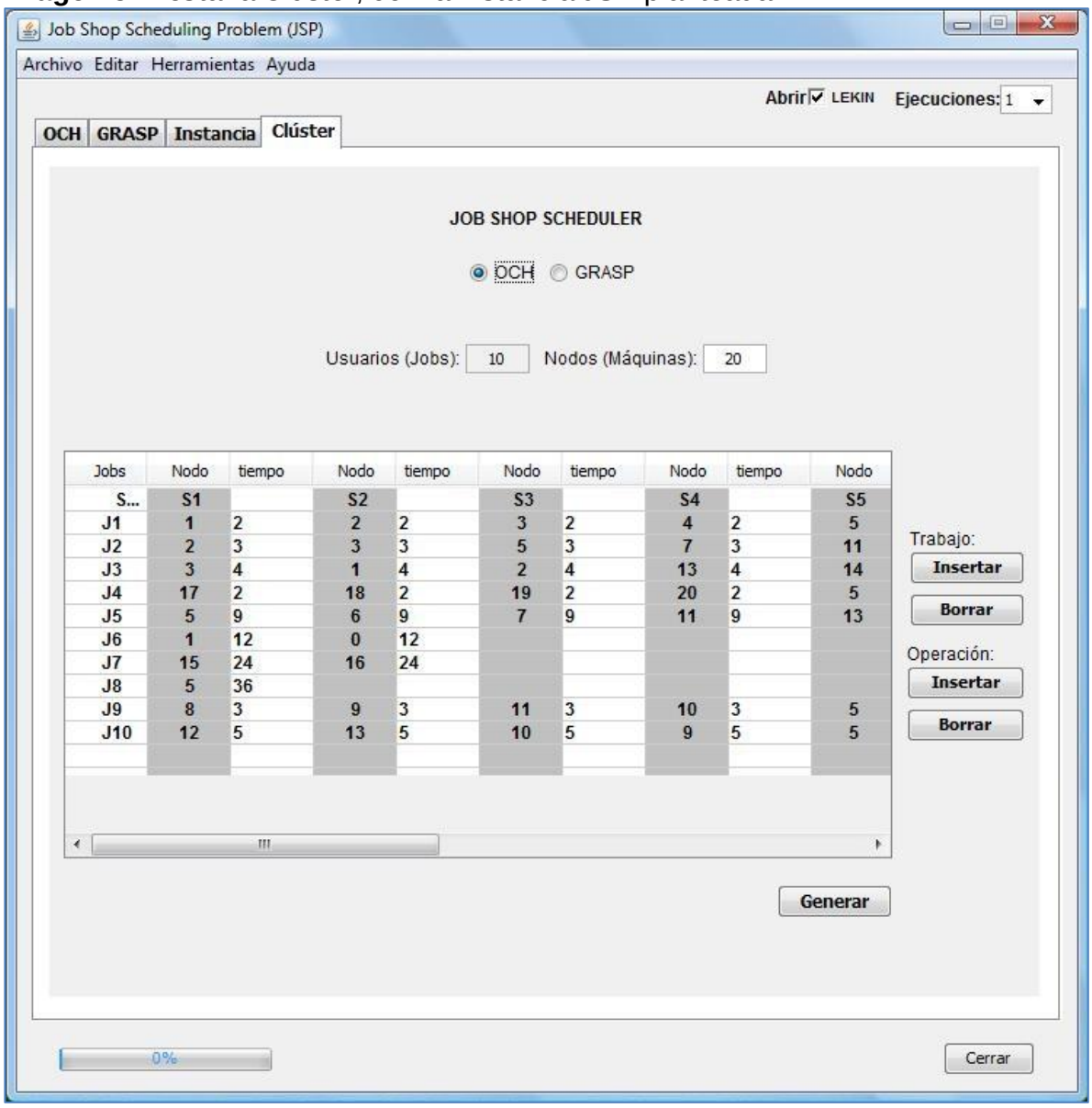
Si se selecciona mas de una ejecucion (o prueba) para la instancia abierta, despues de mostrarse los respectivos resultados, se abrirá un archivo de texto que contiene la información de las ejecuciones con datos estadísticos sobre el makespan y número de evaluaciones (promedio, varianza, desviación estandar, error absoluto y error relativo porcentual del makespan respecto al BKS). En la siguiente imagen se muestra la pestaña GRASP, donde alfa es el parámetro entre 0 y 1, que determina la influencia de la información heurística y del componente aleatorio del algoritmo. Las iteraciones son el número de búsquedas locales que se realizan en cada nodo (operación), por lo que el número de evaluaciones de la función objetivo se halla multiplicado las iteraciones por el nodo donde se encontró la mejor solución, más la solución inicial.

Imagen 4: Pestaña GRASP junto a la salida generada de LA01 con 3 ejecuciones



En la ultima pestaña titulada Clúster, se presenta la instancia JSP planteada para un sistema distribuido, sus datos pueden modificarse directamente en la tabla, se puede agregar mas operaciones y trabajos a la instancia, y definir el número de nodos (máquinas) del clúster. Los resultados se muestran en la pestaña del algoritmo seleccionado para generar el plan.

Imagen 5: Pestaña Clúster, con la instancia JSP planteada



Una de las funciones adicionales es Gráfica, se localiza en el ítem Herramientas de la barra de menú, permite ver el makespan obtenido en cada iteración del algoritmo ejecutado, y tiene una línea recta que es el BKS de la instancia.

Imagen 6: Gráfica de las iteraciones de OCH para LA01