IMPLEMENTACIÓN DE ALGORITMOS BIOINSPIRADOS PARA LA SOLUCIÓN DEL PROBLEMA DE PLANIFICACIÓN DE TRABAJOS

ELABORADO POR

WILFREDO ARIEL GÓMEZ BUENO
Ingeniero de Sistemas

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

IMPLEMENTACIÓN DE ALGORITMOS BIOINSPIRADOS PARA LA SOLUCIÓN DEL PROBLEMA DE PLANIFICACIÓN DE TRABAJOS

WILFREDO ARIEL GÓMEZ BUENO

Trabajo de grado presentado para optar el título de Magister en Ingeniería de Sistemas e Informática

DIRIGIDO POR

Lola Xiomara Bautista, M.Sc.

CO-DIRIGIDO POR

Darío José Delgado, M.Sc.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMÉCANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
MAESTRÍA EN INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA, 2013

Reconocimientos

El presente trabajo de investigación ha sido posible bajo el apoyo del Grupo de Investigación en Ingeniería Biomédica de la Universidad Industrial de Santander, liderado por la profesora Lola Xiomara Bautista, quien como directora aporto inmensamente con su confianza y experiencia.

Adicional se desea rescatar los aportes invaluables del profesor Darío José Delgado como codirector del presente trabajo, sus conocimientos y consejos permitieron la finalización del mismo.

Resaltar la labor desarrollada por los ingenieros Nelson Eduardo Diaz, Leydy Luna Martínez y Edson Flórez que con su disciplina y constancia a la hora de desarrollar sus respectivos trabajos de grado, fueron soporte importantísimo para alcanzar los objetivos propuestos en el presente trabajo, además su completa disposición y colaboración, facilitaron el trabajo de divulgación de los resultados de este proyecto de investigación.

A todos los compañeros y familiares que sin su respaldo y apoyo incondicional no sería posible ninguna de las metas alcanzadas.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE ANEXOS

RESUMEN

TÍTULO: IMPLEMENTACIÓN DE ALGORITMOS BIOINSPIRADOS PARA LA SOLUCIÓN DEL PROBLEMA DE PLANIFICACIÓN DE TRABAJOS[*].

AUTOR: WILFREDO ARIEL GÓMEZ BUENO[**]

PALABRAS CLAVE: algoritmos bioinspirados; job shop scheduling; metaheurísticas; optimización combinatoria, planificación de tareas.

CONTENIDO:

El problema de Job shop Scheduling (JSP) o planificación de tareas, es un problema de gran interés para la industria que se encuentra distribuido en múltiples escenarios, como la gestión de proyectos, gestión de producción de bienes, gestión de sistemas computacionales distribuidos, telecomunicaciones entre otras. Una de sus principales características es su naturaleza combinatoria y su complejidad, NP-Hard, la cual implica altos costos computacionales. Por tanto, se hace necesario abordar este tipo de problemáticas por medio de métodos que a pesar de que no encuentren soluciones óptimas en la mayoría de los casos, produzcan aproximaciones competitivas, a este tipo de métodos se les denomina metaheurísticas. Éstas garantizan soluciones aproximadas, y reducen el consumo de recursos de cómputo. Para esta investigación se utilizaron metaheurísticas de inspiración biológica, como sistemas inmunes artificiales y los algoritmos de colonias de hormigas que parten de una serie de características y comportamientos de los seres vivos que son interesantes en el área computacional. El desempeño de los algoritmos fue caracterizado y evaluado para instancias de referencia del problema de job shop scheduling, comparando la calidad de las soluciones obtenidas respecto a la mejor solución conocida de los métodos más eficaces. Adicional se desarrolló una aproximación a un escenario real en el campo de planificación y gestión de los recursos de cómputo en clústeres de computadores Las soluciones fueron de buena calidad competitivas frente a la literatura y obtenidas con una destacable eficiencia al tener que realizar un número muy bajo de evaluaciones de la función objetivo.

[*] Trabajo de investigación

[**] Facultad de Ingenierías Fisicomecánicas, Escuela de Ingeniería de sistemas. Director: Lola Xiomara Bautista Rozo. Codirector: Darío José Delgado Quintero.

ABSTRACT

TITLE: IMPLEMENTATION OF THE BIO-INSPIRED ALGORITHMS FOR THE SOLUTION THE JOB SHOP SCHEDULING PROBLEM [*].

AUTHOR: WILFREDO ARIEL GÓMEZ BUENO[**]

KEYWORD: bio-inspired algorithms; job shop scheduling; metaheuristics; combinatorial optimization; scheduling;

CONTENT:

The Job Shop Scheduling is a problem of great interest to the industry that is distributed in many scenarios, such as project management, production management of assets, management of distributed computer systems, telecommunications and others. One of its main features is its combinatorial nature and complexity, NP-Hard, in real instance which implies high computational costs. Therefore, it becomes necessary to deal with such problems by methods which although are not optimal solutions in most instances, produce competitive approximations to such methods are called metaheuristics. These ensure approximate solutions, and reduce the consumption of computing resources. For this research we used biologically inspired metaheuristics, as artificial immune systems and ant colony algorithms that are based on a number of characteristics and behaviors of living things that are interesting in the computer area. The performance of the algorithms was characterized and evaluated for reference instances of job shop scheduling problem, comparing the quality of the solutions obtained with respect to the best known solution of the most effective methods. Additional developed an approximation to a real scenario in the field of planning and management of computing resources in computer clusters. The solutions were of good quality, competitive with the literature and obtained with remarkable efficiency by having to make a very low number of objective function evaluations.

[*] Research project

[**] Physical-Mechanical Engineering Faculty, Systems and Informatics Engineering School. Director: Lola Xiomara Bautista Rozo. Codirector: Darío José Delgado Quintero

GLOSARIO

Antígeno: cualquier sustancia (casi siempre ajena) que se une de manera específica con un anticuerpo o un receptor en la célula T; a menudo se emplea como sinónimo de inmunógeno[1].

Anticuerpo: proteína (inmunoglobulina) que consiste en dos cadenas pesadas idénticas y dos cadenas ligeras idénticas; reconoce un epítopo particular en un antígeno y facilita la eliminación de ese antígeno[1].

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[2] .

Optimización combinatoria: La optimización combinatoria trata una clase de problemas en los que el número de soluciones candidatas es de tamaño combinatorio, lo que hace que la complejidad de dichos problemas sea alta[3].

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[4].

Makespan**:** tiempo total en el que todos los trabajos de una máquina completan su ejecución[5]

ACRÓNIMOS

AIS: Artificial Immune System (Sistema Inmune Artificial)

JSP: Job Shop Scheduling Problem (Problema de planificación de trabajos)

GRASP: Greedy Randomized Algorithm Search Procedure (Algoritmo de Busqueda Aleatoria Greedy)

BKS: Best Know Solution (Mejor Solución Conocida)

INTRODUCCIÓN

El estudio de los problemas de optimización combinatoria se centra en hallar métodos o algoritmos que procuren la resolución eficiente de los mismos, la dificultad radica en que los problemas de optimización combinatoria se encuentran contenidos en la clase NP-Duros, o problemas en los que el tiempo de cómputo necesario para resolverlos crece de manera no polinomial conforme aumenta el tamaño del problema. Dentro del área de optimización combinatoria se encuentra una familia de problemas relacionados con la planificación de trabajos entre ellos el que es objeto de estudio en el presente trabajo denominado Job Shop Scheduling que consiste en encontrar un conjunto de permutaciones entre recursos y trabajos que sean desarrollados en el menor tiempo posible. El presente trabajo de investigación surge del interrogante ¿Qué tan eficientes son los algoritmos bioinspirados aplicados al problema de planificación de trabajos?

Para dar solución a esa pregunta de investigación, primero se estableció un escenario común para verificar el desempeño de los algoritmos, la familia de instancias seleccionada fue la de Lawrence compuesta de 40 problemas de diferentes complejidades y de gran difusión en la literatura. Posteriormente se implementaron un algoritmo inmune artificial, un algoritmo de colonia de hormigas y un algoritmo de búsqueda local miope conocido como GRASP, con los cuales se realizaron pruebas de calidad de la solución (Minimizar Makespan) y pruebas de desempeño (Numero de Evaluaciones requeridas para encontrar la solución) sobre el escenario de instancias mencionado anteriormente, contrastando los algoritmos desarrollados entre si y comparando frente a resultados de la literatura. Para finalizar se realizó una aproximación a un escenario real en el área de la computación sobre sistemas distribuidos validando los resultados con una herramienta de la literatura denominada Lekin. Se observa que los algoritmos implementados especialmente el algoritmo inmune artificial alcanzan soluciones competitivas tanto en la calidad como en desempeño, frente a las soluciones presentadas en la literatura.

El presente documento aborda en primera instancia una conceptualización del problema y realiza una caracterización de las familias de instancias disponibles en la literatura, posteriormente aborda cada una de las técnicas implementadas desde el punto de vista del modelo biológico y caracterizando su implementación, finalmente se presentan los resultados y la aproximación sobre el escenario real en el área de sistemas distribuidos.

PROBLEMA DE PLANIFICACIÓN DE TRABAJOS O JOB SHOP SCHEDULING

El Job Shop Scheduling Problem (JSPP), está compuesto por un conjunto finito de trabajos es procesado sobre un conjunto finito de máquinas. Cada trabajo se caracteriza por un orden fijo de las operaciones, cada una de las cuales será procesada en una máquina específica y con un tiempo específico. Cada máquina puede procesar a lo más un trabajo al mismo tiempo y una vez que un trabajo ha iniciado sobre una máquina se debe completar su procesamiento sobre esa máquina por un tiempo ininterrumpido. Un Calendario es una asignación de operaciones en intervalos de tiempo sobre las máquinas. El makespan es el máximo tiempo en completar los trabajos. El objetivo de JSP es encontrar un calendario que minimice el makespan.

Formalmente, el JSP puede ser definido como se muestra a continuación[6]. Dado un conjunto M de máquinas ($|M|$ denota el tamaño de M) y un conjunto J de trabajos ($|J|$ denota el tamaño de J), sean $\sigma_1^j \prec \sigma_2^j \prec \cdots \prec \sigma_{|M|}^j$ sea el orden de un conjunto $|M|$operaciones del trabajo j, donde $\sigma_k^j \prec \sigma_{k+1}^j$ indica que la operación σ_{k+1}^j solo puede empezar el procesamiento después de completar la operación σ_k^j. Sea O el conjunto de operaciones. Cada operación es definida por dos parámetros: M_k^j es la máquina sobre la cual σ_k^j es procesada y $p_k^j = p(\sigma_k^j)$ es el tiempo de procesamiento de la operación σ_k^j. Definiendo $t(\sigma_k^j)$ como el tiempo de inicio de la $k-esima$ operación $\sigma_k^j \in O$, una formulación de programación disyuntiva para el JSP se muestra a continuación:

$$\min C_{max} \qquad (1)$$

$sujeto\ a$:

$C_{max} \geq t(\sigma_k^j) + p(\sigma_k^j),\ \ para\ toda\ \sigma_k^j \in O,$

$(1a).\ \ t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \qquad para\ toda\ \ \sigma_l^j \prec \sigma_k^j,$

$(1b).\ \ t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i)\ \vee\ t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j)\ \ para\ todo\ i,j\ \in J$

$\ni M_{\sigma_l^i} = M_{\sigma_k^j},$

$t(\sigma_k^j)\ \geq\ 0, para\ toda\ \ \sigma_k^j\ \in O$

C_{max} Es el makespan a ser minimizado.

Una solución factible puede ser construida de una permutación de J sobre cada una de las máquinas M, observando las restricciones de precedencia, la restricción de que cada máquina puede procesar solo una operación a la vez y requiriendo una vez iniciada, el procesamiento de una operación debe ser ininterrumpido hasta ser completada. Cada conjunto de permutaciones tiene un correspondiente Calendario.

Por tanto, el objetivo del JSP es encontrar un conjunto de permutaciones con el makespan más pequeño.

Algunas medidas asociadas al problema de planificación de tareas por flujos de trabajo son[7]:

- $Ci.\ (completion\ time)$ El tiempo de terminación corresponde al tiempo de la última operación del trabajo *Ji*.
- $Fi.\ \ (flow\ time)$ El tiempo de flujo del trabajo *Ji.*
- $Li.\ \ (lateness)$ El retraso de *Ji*. Corresponde al tiempo que se excede un trabajo en ser concluido.

Las medidas relacionadas con medir el desempeño para poder obtener una reducción de costos en el problema de planificación de tareas por flujos de trabajo son:

- *Makespan*: Tiempo mínimo para completar todos los trabajos.
- *Total flow time*: Es el tiempo consumido por todos los trabajos.
- *Total lateness :* La suma de todos los retrasos de los trabajos.
- *Maximum lateness*: El retraso del más atrasado de los trabajos.

El Job Shop Scheduling[1] es un problema de optimización combinatoria, según Gary William Flake del MIT en su libro "*Computer Explorations of Fractals, Chaos,Complex Systems, and Adaptation"* [3] 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. Debido a la gran cantidad de soluciones posibles, se trata de expresar en un espacio de búsqueda completo, aunque esto no siempre es posible".

Para un problema de 20 operaciones y 1 recurso se tienen $(20!) = 2.4 * 10^{18}$ posibles soluciones, para un problema de 24 operaciones y 1 recurso se tienen $(24!) = 6.2 * 10^{23}$ posibles soluciones por ultimo para un problema de 30 operaciones y 1 recurso se tiene $(24!) = 2.6 * 10^{32}$ posibles soluciones. Teniendo en cuenta esto, una estimación de la edad del universo se encuentra alrededor de 20 mil millones de años, pasando a segundos seria $20 * 10^9 * 365.25 * 24 * 60 * 60 = 6.31 * 10^{17}\ segundos$, llevando a μseg seria 6.31*10^{23} μseg. Una computadora capaz de procesar una solución cada μseg, seria tan solo capaz de resolver por enumeración el problema de 20 operaciones y 24 operaciones dentro de la edad del universo, para el problema de 30 operaciones no bastarían los 20000 años del universo para dar una solución completa.

Formalmente se considera un problema de optimización combinatoria:

Definición 1: [6]

Un problema de optimización combinatoria esta denotado por un conjunto de instancias del problema y una prescripción de maximizar o minimizar. Consideraremos sin pérdida de generalidad únicamente problemas de minimización. De lo contrario, si f es la función de costo y S el conjunto de soluciones usamos que

[1] Se puede traducir como *el problema de asignación de trabajos, aquí se usan las siglas en inglés* **JSP**

$$\boldsymbol{max}\,\{\boldsymbol{f}(\boldsymbol{x}):\ \boldsymbol{x}\in\boldsymbol{S}\}-\boldsymbol{min}\,\{-\boldsymbol{f}(\boldsymbol{x}):\ \boldsymbol{x}\in\boldsymbol{S}\} \quad \textbf{(2)}$$

Normalmente el factor determinante en el área de eficiencia algorítmica corresponde al tiempo, y está relacionado de manera directa con el procesamiento de información pero depende fuertemente del sistema, del lenguaje en que están escritos y la arquitectura del equipo de cómputo.

Para evaluar la eficiencia de un algoritmo, no se utilizan unidades en tiempo, en cambio se usan unidades lógicas que expresan una relación lineal entre el tamaño *n* de un archivo o un arreglo y la cantidad de tiempo que se requiere para el procesamiento de los datos. La expresión en la que se encapsula la relación entre tiempo y magnitud de información es denominada Complejidad Asintótica.

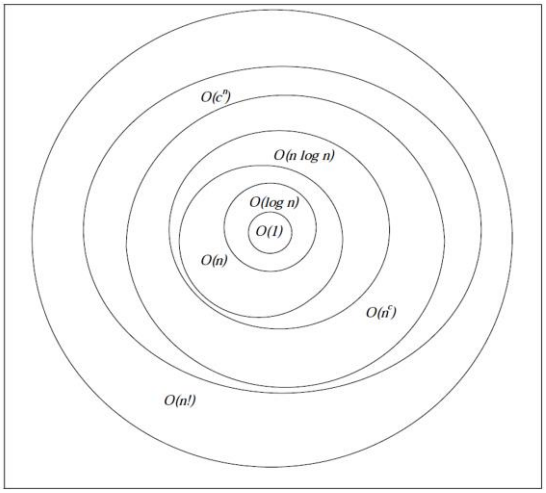
Las principales notaciones de la complejidad asintótica son: O (Big O o big Omicron), Θ (Big Theta) y Ω (Big Omega) [8] .En 1864 Paul Bachmannn introduce la notación de orden O u notación asintótica superior que más tarde sería popularizada por Edmund Landau en 1899.

La alta complejidad de los problemas combinatorios es uno de los aspectos por los cuales adquieren relevancia, debido a que los recursos computacionales actuales no son suficientes para encontrar soluciones exactas en la mayoría de los casos.

Los algoritmos pueden clasificarse según las complejidades de tiempo y de espacio como se ilustra en la Figura 1:

Figura 1 Clases de complejidad y su representación en diagramas de Venn

CLASE	COMPLEJIDAD
CONSTANTE	$O(1)$
LOGARÍTMICA	$O(\log n)$
LINEAL	$O(n)$
LINEAL LOGARÍTMICA	$O(n \log n)$
CUADRÁTICA	$O(n^2)$
CÚBICA	$O(n^3)$
POLINOMIAL	$O(n^c)$
EXPONENCIAL	$O(c^n)$
FACTORIAL	$O(n!)$



Para poder determinar la eficiencia de un algoritmo es necesario precisar por lo menos tres casos de evaluación, el primero que corresponde al máximo número de pasos (El peor Caso), el segundo que correspondería al mínimo número de pasos (El mejor Caso) y uno que corresponda entre los dos extremos (El Caso Promedio)[9].

Un algoritmo determinista es una secuencia de pasos definida de manera exclusiva para una entrada en particular. Están relacionados mayormente a los problemas de función, a diferencia de un algoritmo no determinista que es una secuencia de pasos que para una entrada particular puede presentar diversidad de resultados y están mayormente relacionados a los problemas de decisión. Teniendo en cuenta esto y las notaciones de complejidad se puede clasificar a los problemas de la siguiente manera:

- *Problemas Tipo P:*

Son problemas para los cuales existen algoritmos computacionales deterministas de tipo polinomial $\mathrm{O(n^m)}$, que permiten encontrar la solución óptima global del problema en tiempos racionales.

- *Problemas tipo NP:*

Estos son problemas a los cuales no se les conoce algoritmo computacional determinista polinomial $O(n^m)$ capaz de encontrar la respuesta óptima en un tiempo razonable. Estos problemas utilizan algoritmos no deterministas para hallar soluciones en tiempo polinomial.

Dentro del grupo de problemas NP existen algunos problemas especialmente difíciles de resolver por su naturaleza propia, estos problemas reciben el nombre de Problemas No Polinomiales Completos (NP-Completo). Actualmente para pequeñas instancias de estos problemas, los algoritmos deterministas conocidos para problemas NP-completos utilizan tiempo exponencial con respecto al tamaño de la entrada.

Garey y Johnson en su libro Computers and Intractability: A Guide to the Theory of NP-Completeness en 1979[10], demostraron que el problema de planificación es un problema NP-duro, y Tapan P. Bagchi en MultiObjective Scheduling by Genetic Algorithms de 1999[11], describe que de entre esa clase es uno de los menos tratables.

Una instancia de un problema de optimización combinatoria se define como:

Definición 2: [6]

Una instancia de un problema de optimización combinatoria es un par (S, f) donde S es el conjunto de soluciones factibles y $f: S \to \tilde{N}$ es la función de costo. El problema reside en encontrar una solución óptima global, esto es, un $i* \in S$ tal que $f(i*) \leq f(i)$ para todo $i \in S$. Además, si un $f* = f(i*)$

Entonces $S* = \{i \in S: f(i) = f*\}$ denota el conjunto de soluciones óptimas.

Una instancia generalmente no está dada explícitamente, esto es, expuesta como una lista de soluciones con sus costos respectivos. Usualmente tenemos una representación compacta de los datos formada por un algoritmo de tiempo polinomial para verificar si una solución pertenece a S y computar su costo [10].

Definición 3: [6]

Sea (S, f) una instancia de un problema de optimización combinatoria. Una función entorno es una aplicación $N: S \rightarrow P(S)$ que define para cada solución $i \in S$ un conjunto $N(i) \subseteq S$ de soluciones que están, en algún sentido, cerca de i. El conjunto $N(i)$ es el entorno de la solución i, y cada $j \in N(i)$ es un vecino de i. Asumiremos que $i \in N(i)$ para todo $i \in S$.

Un problema computacional podría ser considerado como una relación entre una serie de información que compone unas instancias y un conjunto de valores considerado solución a esas instancias. Esas instancias deben tener por lo menos una solución que es el producto de la transformación de una serie de entradas.

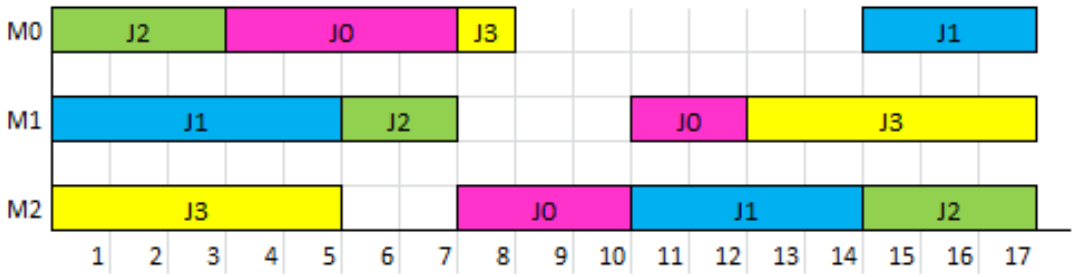
Una instancia del problema de planificación de tareas o job shop scheduling problema se define mediante una matriz, la cual contiene cada uno de los trabajos, el orden dentro de las máquinas y el tiempo de procesamiento correspondiente a cada trabajo en cada máquina. (Véase Tabla 1)

Tabla 1 Flujo y tiempos de los trabajos en el Job shop Scheduling Problem

Trabajo	Máquina	Tiempo	Máquina	Tiempo	Máquina	Tiempo
J0	0	4	2	3	1	2
J1	1	5	2	4	0	3
J2	0	3	1	2	2	3
J3	2	5	0	1	1	5

Para facilitar la visualización de la programación de los trabajos, se hace uso de planes de trabajos donde se acomodan los trabajos en las máquinas de tal forma que cumplan con las restricciones del problema, dicha visualización se conoce como *diagrama de Gantt* (ver figura 2, 3, 4, 5).

Figura 2 Diagrama de Gantt para la representación de trabajos y máquinas



Los diagramas de Gantt representan cada máquina en un renglón diferente y cada cuadro representa una operación. Los cuadros están marcados con el número de trabajo al que corresponde, es decir se identifica por el color y el número especificado, por último en el eje x se encuentran las unidades de tiempo que los trabajos gastan en completar cada una de las operaciones.

En los problemas de Job Shop Scheduling se encuentran dos tipos de planes de trabajos: activo o semi-activo. Los primeros son planes de trabajo donde ninguna operación se inicia sin que se retarde cualquier otra operación o sin que se viole alguna de las restricciones. Por el contrario los planes de trabajos semi-activos son planes en los cuales ninguna operación puede iniciarse sin cambiar el orden de procesamiento o sin que se viole alguna de las restricciones[12].

Figura 3 Paso 1: Optimización de un plan de trabajo

Figura 4 Paso 2: Optimización de un plan de trabajo

Figura 5 Paso 3: Optimización de un plan de trabajo

En la Figura 3, se muestra un calendario semi-activo, en el cual se puede optimizar al cambiar el orden del trabajo $J0$ por el trabajo $J3$ en la máquina $M1$, dado que si se inicia antes no afecta el tiempo de la operación del trabajo.

En la Figura 4 se puede observar que al cambiar el orden del trabajo $J1$ en la máquina $M2$ permitirá reducir el tiempo de retardo y a su vez este mismo trabajo en la máquina $M0$ podrá llevarse a cabo antes.

Finalmente, en la Figura 5 se observa un calendario no retardado, es decir, que no existe ningún movimiento que haga que no se retrase ninguna operación.

Un calendario donde ninguna de las máquinas está ociosa es decir, que está lista para procesar los trabajos, se conoce como calendario o plan de trabajo *sin retardo.*

1.1 CARACTERIZACIÓN FAMILIA DE INSTANCIAS PARA EL PROBLEMA DE JSP

En la literatura se puede encontrar un serie de colecciones de familias de instancias de diferentes configuraciones del problema de Job shop scheduling, estas familias de instancias se utilizan para la realización de pruebas y mediciones de rendimiento de algoritmos. A continuación se realiza una caracterización de las de mayor uso[13]:

Tabla 2 Familias de Instancias del Problema de Job Shop Scheduling

Siglas	Nombre	Autor	Numero de problemas	Tamaños	Características
FT	Applegate y Cook [14].	Fisher y Thompson	3 problemas de 3 tamaños	6x6, 10x10 y 20x5	Las máquinas con los números más pequeños le son asignadas las primeras operaciones y a las máquinas con los números más grandes se le asignan las últimas operaciones.
LA	Applegate y Cook	Lawrence	40 problemas de 8 tamaños	10x5, 15x5, 20x5, 10x10, 15x10, 20x10, 30x10, 15x15.	Es uno de los más comúnmente utilizados. Los tiempos de procesamiento fueron generados en el intervalo [5; 99].
ABZ	Applegate y Cook	Adams, Balas y Zawack	5 problemas de 2 diferentes	10x10 20x15.	Los tiempos de procesamiento de ABZ5, ABZ6 y ABZ(7-9) fueron generados en los intervalos [50*;* 100], [25*;* 100] y [11*;* 40] respectivamente.
ORB	Applegate y Cook	Bonn	10 problemas	10*x*10	Son caracterizados como problemas difíciles
SWV	Vaessens	Storer,Wu y Vaccari	20 problemas de 4 tamaños	20*x*10, 20*x*15, 50*x*10 50*x*10	Los tiempos de procesamiento son generados en el intervalo [1*;* 100]. Estas instancias son consideradas difíciles. En estos problemas el conjunto de máquinas es divido en k ($1 \cdot k \cdot m$) subconjuntos del mismo tamaño. Una secuencia de precedencia es generada pasando el trabajo a través de toda la máquina de un sistema usando asignaciones uniformemente distribuidas antes de que llegue a una máquina del siguiente conjunto. Para el conjunto fácil $k = 1$, mientras que para las instancias difíciles $k = 2$.
YN	Vaessens	Yamada y Nakano	4 problemas	20x20	Los tiempos de procesamiento fueron generados en el intervalo [10*;* 50].
TA	Taillard.	Taillard	80 problemas de 8 Tamaños	15x15, 20x15, 20x20, 30x15, 30x20, 50x15, 50x20 100x20.	Los tiempos de procesamiento fueron generados en el intervalo [1; 99].

A continuación se presentan todas las instancias de cada una de las familias, para cada instancia se detalla el tamaño(n,m), el límite inferior (LI), y el límite superior(LS) y las fechas correspondientes a la consecución de esos límites.

Tabla 3 Características de la Familia de Instancias La

Instancia	Trabajos	Máquinas	LI	LS	AÑO	Ref. LI	Ref. LS
la01	10	5	666	666	1988	[15]	
la02	10	5	655	655	1988/1988	[15]	[16]
la03	10	5	597	597	1991/1988	[17]	[16]
la04	10	5	590	590	1991/1988	[17]	[16]
la05	10	5	593	593	1988	[15]	
la06	15	5	926	926	1988	[15]	
la07	15	5	890	890	1988	[15]	
la08	15	5	863	863	1988	[15]	
la09	15	5	951	951	1988	[15]	
la10	15	5	958	958	1988/1992	[15]	[18]
la11	20	5	1222	1222	1988	[15]	
la12	20	5	1039	1039	1988	[15]	
la13	20	5	1150	1150	1988	[15]	
la14	20	5	1292	1292	1988	[15]	
la15	20	5	1207	1207	1988	[15]	
la16	10	10	945	945	1990	[19]	
la17	10	10	784	784	1990/1988	[19]	[16]
la18	10	10	848	848	1991/1988	[17]	[16]
la19	10	10	842	842	1991/1988	[17]	[16]
la20	10	10	902	902	1991/1992	[17]	[18]
la21	15	10	1046	1046	1996	[20]	
la22	15	10	927	927	1991/1988	[17]	[16]
la23	15	10	1032	1032	1988	[15]	
la24	15	10	935	935	1991	[17]	
la25	15	10	977	977	1991	[17]	
la26	20	10	1218	1218	1988/1988	[15]	[16]
la27	20	10	1235	1235	1988/1994	[15]	[21]
la28	20	10	1216	1216	1988/1988	[15]	[16]
la29	20	10	1152	1152	1996	[22]	
la30	20	10	1355	1355	1988	[15]	
la31	30	10	1784	1784	1988	[15]	
la32	30	10	1850	1850	1988	[15]	
la33	30	10	1719	1719	1988	[15]	
la34	30	10	1721	1721	1988	[15]	
la35	30	10	1888	1888	1988	[15]	
la36	15	15	1268	1268	1990	[19]	
la37	15	15	1397	1397	1990/1991	[19]	[17]
la38	15	15	1196	1196	1995/1996	[20]	[23]
la39	15	15	1233	1233	1991	[17]	

la40	15	15	1222	1222	1991	[17]	

Fuente: Adaptación Cortes[24]

Tabla 4 Características de la Familia de Instancias abz

Instancia	Trabajos	Máquinas	LI	LS	AÑO	Ref. LI	Ref. LS
abz5	10	10	1234	1234	1991	[17]	
abz6	10	10	943	943	1991/1988	[17]	[15]
abz7	20	15	656	656	1996/1996	[25]	[25]
abz8	20	15	646	665	1999/1996	[26]	[22]
abz9	20	15	662	679	1999/1995	[26]	[27]

Fuente: Adaptación Cortes[24]

Tabla 5 Características de la Familia de Instancias orb

Instancia	Trabajos	Máquinas	LI	LS	AÑO	Ref. LI	Ref. LS
orb01	10	10	1059	1059	1991	[17]	
orb02	10	10	888	888	1991	[17]	
orb03	10	10	1005	1005	1991	[17]	
orb04	10	10	1005	1005	1991	[17]	
orb05	10	10	887	887	1991	[17]	
orb06	10	10	1010	1010	1991	[17]	
orb07	10	10	397	397	1991	[17]	
orb08	10	10	899	899	1991	[17]	
orb09	10	10	934	934	1991	[17]	
orb10	10	10	944	944	1991	[17]	

Fuente: Adaptación Cortes[24]

Tabla 6 Características de la Familia de Instancias swv

Instancia	Trabajos	Máquinas	LI	LS	AÑO	Ref. LI	Ref. LS
swv01	20	10	1407	1407	1996	[22]	
swv02	20	10	1475	1475	1995/1996	[28]	[28]
swv03	20	10	1398	1398	1999/1996	[26]	[28]
swv04	20	10	1450	1474	1996/2002	[28]	[23]
swv05	20	10	1424	1424	1996	[22]	
swv06	20	15	1591	1673	1996/2003	[28]	[29]
swv07	20	15	1447	1600	1999/2002	[26]	[23]
swv08	20	15	1641	1759	1999/2002	[26]	[29]
swv09	20	15	1605	1661	1999/2002	[26]	[23]
swv10	20	15	1632	1761	1999/2002	[26]	[29]

swv11	50	10	2983	2983	1996/2002	[28]	[23]
swv12	50	10	2972	3003	1996/1997	[28]	[30]
swv13	50	10	3104	3104	1996/1997	[28]	[30]
swv14	50	10	2968	2968	1996/1995	[28]	[27]
swv15	50	10	2885	2903	1996/2002	[28]	[29]
swv16	50	10	2924	2924	1996/1992	[28]	[31]
swv17	50	10	2794	2794	1996/1992	[28]	[31]
swv18	50	10	2852	2852	1996/1992	[28]	[31]
swv19	50	10	2843	2843	1996/1992	[28]	[31]
swv20	50	10	2823	2823	1996/1992	[28]	[31]

Fuente: Adaptación Cortes[24]

Tabla 7 Características de la Familia de Instancias yn

Instancia	Trabajos	Máquinas	LI	LS	AÑO	Ref. LI	Ref. LS
yn1	20	20	846	885	1999/2002	[26]	[23]
yn2	20	20	870	909	1999/1996	[26]	[28]
yn3	20	20	840	892	1999/2002	[26]	[23]
yn4	20	20	920	968	1999/1997	[26]	[30]

Fuente: Adaptación Cortes[24]

Tabla 8 Características de la Familia de Instancias ta

Instancia	Trabajos	Máquinas	LI	LS	AÑO	Ref. LI	Ref. LS
ta01	15	15	1231	1231	1994	[32]	
ta02	15	15	1244	1244	1995/1993	[28]	[23]
ta03	15	15	1218	1218	1999/1995	[26]	[27]
ta04	15	15	1175	1175	1999/1995	[26]	[33]
ta05	15	15	1224	1224	1999/1999	[26]	[26]
ta06	15	15	1238	1238	1999/1999	[26]	[26]
ta07	15	15	1227	1227	1999/1999	[26]	[26]
ta08	15	15	1217	1217	1999/1995	[26]	[27]
ta09	15	15	1274	1274	1999/1995	[26]	[27]
ta10	15	15	1241	1241	1995/1995	[28]	[27]
ta11	20	15	1323	1358	2000/2003	[34]	[29]
ta12	20	15	1351	1367	2000/1995	[34]	[27]

ta13	20	15	1282	1342	2000/2002	[34]	[29]
ta14	20	15	1345	1345	1995/1996	[28]	[23]
ta15	20	15	1304	1340	2000/2002	[34]	[29]
ta16	20	15	1302	1360	2000/2000	[34]	[29]
ta17	20	15	1462	1462	2000/2002	[34]	[23]
ta18	20	15	1369	1396	1995/1996	[28]	[27]
ta19	20	15	1297	1335	2000/2001	[34]	[23]
ta20	20	15	1318	1351	2000/2001	[34]	[23]
ta21	20	20	1539	1644	1995/2002	[28]	[23]
ta22	20	20	1511	1600	1995/2001	[28]	[23]
ta23	20	20	1472	1557	1995/2001	[28]	[23]
ta24	20	20	1602	1647	1995/2002	[28]	[23]
ta25	20	20	1504	1595	1995/2002	[28]	[23]
ta26	20	20	1539	1645	1995/2002	[28]	[23]
ta27	20	20	1616	1680	1995/2001	[28]	[23]
ta28	20	20	1591	1614	1995/2002	[28]	[23]
ta29	20	20	1514	1625	1995/1996	[28]	[32]
ta30	20	20	1472	1584	1995/2001	[28]	[23]
ta31	30	15	1764	1764	1994/1999	[32]	[35]
ta32	30	15	1774	1796	1994/2002	[32]	[29]
ta33	30	15	1778	1793	1995/2002	[28]	[23]
ta34	30	15	1828	1829	1994/2001	[32]	[23]
ta35	30	15	2007	2007	1995/1994	[28]	[32]
ta36	30	15	1819	1819	1995/1999	[28]	[35]
ta37	30	15	1771	1778	1994/2002	[32]	[23]
ta38	30	15	1673	1673	1994/2000	[32]	[29]
ta39	30	15	1795	1795	1995/1999	[28]	[35]
ta40	30	15	1631	1674	1995/2001	[28]	[23]
ta41	30	20	1859	2014	1995/2003	[28]	[29]
ta42	30	20	1867	1956	1995/2001	[28]	[23]
ta43	30	20	1809	1859	1995/2002	[28]	[23]

ta44	30	20	1927	1984	1995/2002	[28]	[23]
ta45	30	20	1997	2000	1995/2001	[28]	[23]
ta46	30	20	1940	2016	1994/2003	[32]	[29]
ta47	30	20	1789	1903	1995/2002	[28]	[23]
ta48	30	20	1912	1952	1995/2002	[28]	[23]
ta49	30	20	1915	1968	1995/2002	[28]	[29]
ta50	30	20	1807	1926	1995/2003	[28]	[36]
ta51	50	15	2760	2760	1994/1994	[32]	[32]
ta52	50	15	2756	2756	1994/1994	[32]	[32]
ta53	50	15	2717	2717	1994/1994	[32]	[32]
ta54	50	15	2839	2839	1994/1994	[32]	[32]
ta55	50	15	2679	2679	1994/1993	[32]	[23]
ta56	50	15	2781	2781	1994/1994	[32]	[32]
ta57	50	15	2943	2943	1994/1994	[32]	[32]
ta58	50	15	2885	2885	1994/1994	[32]	[32]
ta59	50	15	2655	2655	1994/1994	[32]	[32]
ta60	50	15	2723	2723	1994/1994	[32]	[32]
ta61	50	20	2868	2868	1994/1993	[32]	[23]
ta62	50	20	2869	2869	1995/2003	[28]	[36]
ta63	50	20	2755	2755	1994/1993	[32]	[23]
ta64	50	20	2702	2702	1995/1993	[27]	[23]
ta65	50	20	2725	2725	1994/1993	[32]	[23]
ta66	50	20	2845	2845	1994/1993	[32]	[23]
ta67	50	20	2825	2825	1995/1999	[28]	[35]
ta68	50	20	2784	2784	1995/1993	[27]	[23]
ta69	50	20	3071	3071	1994/1993	[32]	[23]
ta70	50	20	2995	2995	1994/1993	[32]	[23]
ta71	100	20	5464	5464	1994/1994	[32]	[32]
ta72	100	20	5181	5181	1994/1994	[32]	[32]
ta73	100	20	5568	5568	1994/1994	[32]	[32]
ta74	100	20	5339	5339	1994/1994	[32]	[32]
ta75	100	20	5392	5392	1994/1994	[32]	[32]

ta76	100	20	5342	5342	1994/1994	[32]	[32]
ta77	100	20	5436	5436	1994/1994	[32]	[32]
ta78	100	20	5394	5394	1994/1994	[32]	[32]
ta79	100	20	5358	5358	1994/1994	[32]	[32]
ta80	100	20	5183	5183	1994/1993	[32]	[23]

Fuente: Adaptación Cortes[24]

En el presente trabajo, se seleccionó la familia de instancias de Lawrence, estas instancias fueron propuestas por Stephen R. Lawrence en un artículo no publicado[37], las cuales luego fueron difundidas ampliamente por J. E [38].

Estas instancias se convirtieron en el punto de partida para muchas de las pruebas que realizan los autores en la literatura de este campo. Como se mencionó, estas instancias son 40 problemas de 8 diferentes tamaños propuestos por Lawrence, que Applegate y Cook [17] nombraron “LA”.

Estas instancias de Lawrence fueron propuestas para cualquier aplicación del problema de Job Shop que cumpliera con las especificaciones donde hay$n$el número de trabajos que deben recorrer $m$número de máquinas.

En conclusión esta familia fue la seleccionada para el desarrollo de este trabajo debido a que es una de las familias de instancias más conocidas en la literatura y además presenta gran heterogeneidad en cuanto a tamaño y complejidad. Algunos de los trabajos en la literatura que han usado esta familia de instancias para medir rendimientos en los algoritmos que dan solución al problema de Job Shop Scheduling son [24] [39] [7][40][41][42][43][44][45].

1.2 MÉTODOS DE SOLUCIÓN DEL PROBLEMA JSP

Las técnicas existentes para resolver el problema de Job Shop Scheduling se pueden clasificar básicamente en algoritmos exactos o aproximados. Los algoritmos

exactos intentan encontrar una solución óptima y demostrar que la solución obtenida es de hecho la óptima global; estos algoritmos incluyen técnicas como, por ejemplo: procesos de vuelta atrás (backtracking), Ramificación y poda (branch and bound), programación dinámica, etc.[46] [47]. Debido a que los algoritmos exactos muestran un rendimiento pobre para muchos problemas se han desarrollado múltiples tipos de algoritmos aproximados que proporcionan soluciones de alta calidad para estos problemas.

Los algoritmos aproximados se pueden clasificar en dos tipos principales: algoritmos constructivos y algoritmos de búsqueda local. Los primeros se basan en generar soluciones desde cero añadiendo componentes a cada solución paso a paso. Un ejemplo bien conocido son las heurísticas de construcción voraz o heurísticas greedy [48]. Su gran ventaja es la velocidad: normalmente son muy rápidas y, además a menudo devuelven soluciones razonablemente buenas. Sin embargo, no puede garantizarse que dichas soluciones sean óptimas con respecto a pequeños cambios a nivel local. En consecuencia, una mejora típica es refinar la solución obtenida por la heurística voraz utilizando una búsqueda local. Los algoritmos de búsqueda local intentan repetidamente mejorar la solución actual con movimientos a soluciones vecinas (con la esperanza de que sean mejores). El caso más simple son los algoritmos de mejora iterativos: si en el vecindario de la solución actual s se encuentra una solución mejor s', ésta reemplaza la solución actual y se continúa la búsqueda a partir de s'; si no se encuentra una solución mejor en el vecindario, el algoritmo termina en un óptimo local[49].

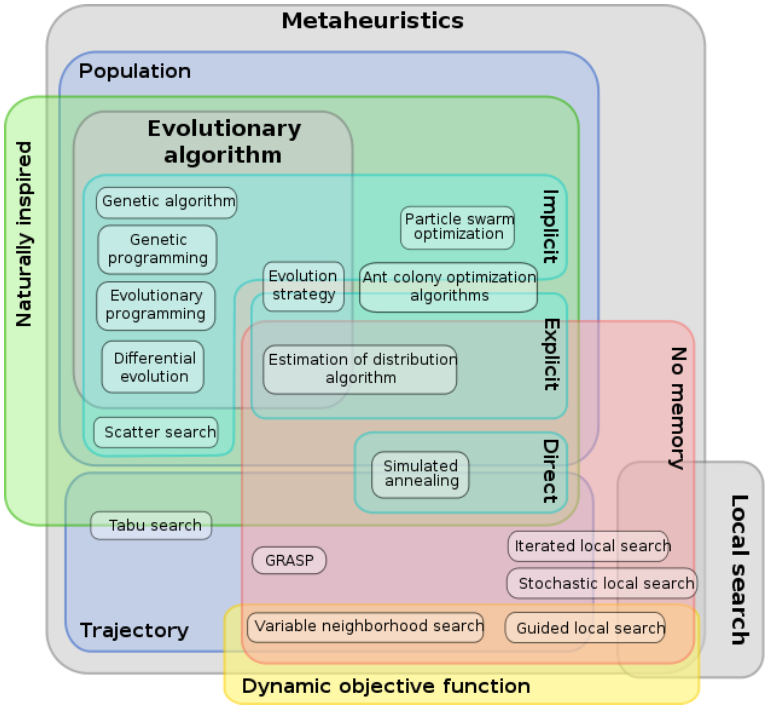
Desafortunadamente, los algoritmos de mejora iterativos pueden estancarse en soluciones de baja calidad (óptimos locales muy lejanos al óptimo global). Para permitir una mejora adicional en la calidad de las soluciones, la investigación en este campo en las últimas dos décadas ha centrado su atención en el diseño de técnicas de propósito general para guiar la construcción de soluciones o la búsqueda local en las distintas heurísticas. Estas técnicas se llaman comúnmente

metaheurísticas y consisten en conceptos generales empleados para definir métodos heurísticos.

Dicho de otra manera, una metaheurística puede verse como un marco de trabajo general referido a algoritmos que puede aplicarse a diversos problemas de optimización (combinatoria) con pocos cambios significativos si ya existe previamente algún método heurístico específico para el problema. De hecho, las metaheurísticas son ampliamente reconocidas como una de las mejores aproximaciones para atacar los problemas de optimización combinatoria [50][2].

Las metaheurísticas incorporan conceptos de muchos y diversos campos como la genética, la biología, la inteligencia artificial, las matemáticas, la física y la neurología, entre otras. Algunos ejemplos de metaheurísticas son: Enfriamiento simulado, búsqueda tabú, búsqueda local iterativa ("iterated local search"), algoritmos de búsqueda local con vecindario variable ("variable neighborhood search"), GRASP ("greedy randomized adaptative search procedures") y algoritmos evolutivos. En la figura a continuación se caracterizan los diferentes tipos de metaheurísticas.

Figura 6 Tipos de Metaheurísticas



Fuente: Johann Dréo [51]

ALGORITMOS BIOINSPIRADOS

La eficiencia de los sistemas biológicos se centra en la evolución alcanzada por los mismos en el transcurrir de millones de años, dicha eficiencia es una característica de suma importancia en la computación, a raíz que los recursos son limitados, esto fue lo que originó que dichos comportamientos y procesos de los sistemas biológicos quisieran ser emulados por los investigadores en el plano computacional.

La computación bioinspirada, se ha soportado en el creciente desarrollo tecnológico gracias a los grandes avances en la electrónica y hardware computacional, y ha venido avanzando sobre los métodos de computación clásicos, utilizando procedimientos que reproducen ciertas propiedades inspiradas en la biología con el fin de diversificar los resultados obtenidos en la medida que se obtengan mejores resultados. Esta disciplina, que está íntimamente vinculada al campo de la Inteligencia Artificial, engloba varios enfoques, tales como: los algoritmos evolutivos (AE)[52], la optimización de colonia de hormigas (ACO) [53], y la optimización de enjambre de partículas (PSO)[54].

Las áreas mencionadas han tomado un gran interés como campo de investigación en Ingeniería y Ciencias de la Computación y han tenido aplicación en diferentes campos del conocimiento que van desde aplicaciones comerciales e industriales (redes de distribución, planificación de recursos, optimización, robótica entre otras) a emulación de órganos y sus respectivas capacidades (visión artificial, sentidos artificiales, modelado del cerebro). A continuación se presentan diferentes trabajos realizados desde el enfoque de Bioinspirado para darle solución al problema de Job shop Scheduling.

En el trabajo de Mahdi Mobini et al. [55] se aborda el uso de un algoritmo inmune artificial de Selección Clonal para la minimización de makespan como objetivo de la solución del problema de Job Shop Scheduling. Los resultados obtenidos tanto en el análisis computacional, como en la comparación con otros algoritmos tales

como Búsqueda de dispersión y Algoritmo genético híbrido, evidencian un rendimiento aceptable y competitivo del algoritmo inmune propuesto en dicho trabajo.
El desarrollo de Castro et al. [56] se basa en adaptación de un modelo de red inmune artificial especialmente diseñado para resolver problemas de optimización multimodales. Se comparó con un algoritmo de selección clonal sobre un escenario de optimización multimodal, utilizando funciones de benchmarking.

En el trabajo de Coello et al. [57] se propone un algoritmo de Selección Clonal para solucionar Job Shop Scheduling. El algoritmo desarrollado en este trabajo se compara con otros 3 algoritmos, como lo son: Algoritmo Genético Hibrido (HGA), Algoritmo Genético Paralelo (PGA) y GRASP. Los resultados indican que el algoritmo inmune presenta un rendimiento superior del 0.23%, 0.74% y 0.28% con respecto a los algoritmos HGA, PGA y GRASP.

En el desarrollo de Quan Zuo y Shun Fan [58] se presenta un algoritmo de Selección Clonal para resolver el Problema de Job Shop Scheduling. El algoritmo inmune utiliza una técnica de nicho para evitar óptimos locales y cuenta con sistema de caos para mejorar la eficiencia de búsqueda. Como característica de dicho desarrollo es que Cuando el retardo es de 200 el algoritmo necesita 39.5 iteraciones para encontrar la mejor solución, sin embargo si el retardo es de 10 el algoritmo se optimiza a 2,2 iteraciones.

Chandrasekaran et al [39] desarrollo un algoritmo de Selección Clonal, el cual utilizo para resolver problemas de job shop scheduling con el objetivo de lograr la minimización del makespan. Los resultados son comparados con un algoritmo de Desplazamiento de Cuello de Botella y un algoritmo de búsqueda Tabú. Los resultados arrojan que el algoritmo inmune tiene el menor error relativo con 1,865% mientras que Búsqueda Tabú tiene 2,56% y Cuello de Botella 3,68%.

Cortés et al. [24], propone un algoritmo inmune artificial de Selección Clonal para resolver el problema de Job Shop Scheduling, adicional dos procedimientos de GRASP Aleatorizado y Adaptativo GRASP y GRASP+RT. Los resultados indican

que: el AIS mostró tener un porcentaje de desviación menor que GRASP pero mayor que GRASP+RT esto respecto a la mejor solución conocida.

1.3 SISTEMAS INMUNES ARTIFICIALES (AIS)

Los sistemas inmunes en la última década, han servido de inspiración para solucionar problemas computacionales complejos, puesto que cuentan con características como las de ser sistemas distributivos y adaptativos de naturaleza auto-organizativa junto a su memoria, aprendizaje, reconocimiento de patrones y capacidad de extracción.

En 1994 Forrest et al [59] propusieron un método llamado algoritmo de *selección negativa*, que está basado en la generación de células T en el sistema inmune. Este método fue aplicado a los problemas de detección de virus de computadores.

Durante 1995 Hunt y Cooke [60] continuaron trabajando sobre modelos de red inmune en el cual Timmis y Neal [61] hicieron algunas mejoras redefiniendo y re-implementado este algoritmo. Estas obras fueron llamadas formalmente AINE (Artificial Immune Network).

En el año 2000 Castro y Zuben [62] propusieron el algoritmo de selección clonal, más tarde conocido como Clonalg, que está inspirado en la selección clonal de los sistemas inmunes la cual explica cómo los linfocitos B y T mejoran su respuesta ante los antígenos. A este tiempo se le conoce como tiempo de maduración de la afinidad.

En 2008, Dasgupta y Niño [63] presentaron un panorama de los conceptos fundamentales de la inmunología y algunos modelos inmunológicos teóricos de los procesos inmunes. También se presenta un compendio de los trabajos relacionados hasta esa fecha basadosen técnicas de inmunología computacional y describe una amplia variedad de aplicaciones.

1.3.1 ALGORITMO DE SELECCIÓN CLONAL PARA LA SOLUCIÓN DEL PROBLEMA JSP

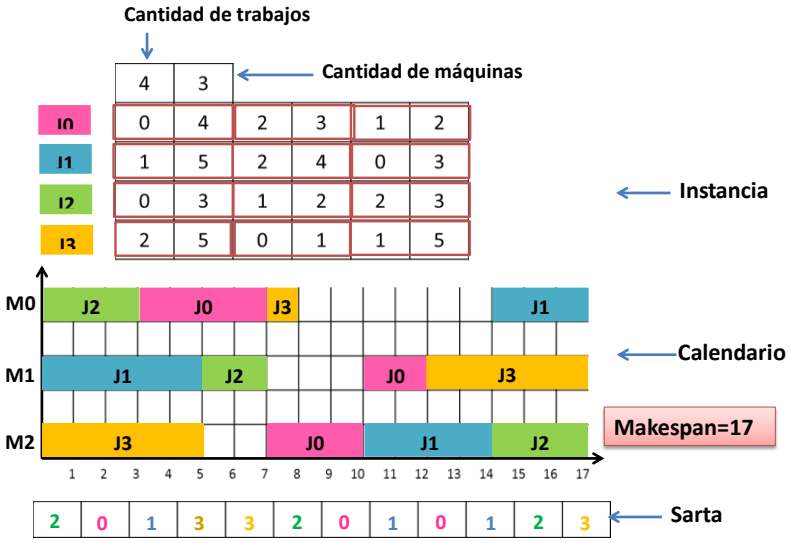
Castro y Von Zuben [62] desarrollaron un algoritmo inspirado en Sistemas Inmunes Artificiales basados en la abstracción de selección clonal, el cuál ha sido utilizado para realizar tareas de emparejamiento de patrones [64] [65] y optimización de funciones multimodal [12] [66], entre otras aplicaciones[67][62]. Este algoritmo se conoce como Clonalg [68] y la implementación computacional de este principio presenta la siguiente estructura general[69]:

1) Generar aleatoriamente un conjunto (P= $Ab_i\ i\ = 1, \dots, n$) de soluciones potenciales, dicho conjunto lo componen, haciendo la analogía a términos biológicos, la representación de las anticuerpos de las células de memoria M y la representación de los anticuerpos producidos por los demás linfocitos del organismo.

   En el algoritmo propuesto para la solución del problema Job Shop Scheduling, cada anticuerpo corresponde a una sarta de valores enteros que representa a un calendario candidato a ser solución de la instancia que se está evaluando durante dicha ejecución.

   Cada uno de los anticuerpos o calendarios candidatos se crean de manera aleatoria, y se garantiza su factibilidad mediante la verificación de las restricciones del problema durante el proceso de generación. En la figura a continuación se describe la representación de un calendario para una instancia específica, su correspondiente representación en un diagrama de Gantt y la sarta que se genera como anticuerpo.

Figura 7 Representación de anticuerpos para el problema de Job Shop Scheduling



Se observa que cada uno de los elementos de la sarta corresponde a los trabajos a desarrollarse, y el orden corresponde a la secuencia de ejecución de cada uno de ellos en dicho calendario , ese orden se establece bajo el criterio del menor tiempo que tarda en ser ejecutado dicho trabajo en una determinada máquina[70].

2) Determinar los *n* mejores individuos de la población de anticuerpos, midiendo el grado de afinidad de cada uno con la representación del antígeno (evaluando cada solución con respecto a la función a optimizar).

   Para el algoritmo propuesto la afinidad f de anticuerpos está dada por la siguiente ecuación:

$$f = 1 - \frac{makespan}{rango} \tag{3}$$

   Donde el rango está dado por:

$$rango = \max(makespan) - \min(makespan) \tag{4}$$

3) Reproduzca los *n* mejores individuos (dependiendo qué tan bien cada solución se comporta en relación con el valor de la función), generando una población temporal de clones (C).

   Para el algoritmo propuesto, la cantidad n de clones para cada anticuerpo corresponde a un parámetro del algoritmo, el cual está dado por la ecuación:

$$N_c = (\beta * N) \quad (5)$$

Donde β es un factor de clonación y N es la cantidad de anticuerpos.

Para el algoritmo desarrollado se tomó un rango entre 0 y 1 para realizar las pruebas del análisis de sensibilidad en lo referente al factor de clonación, después de dicho análisis se determinó como factor de clonación el valor de 0.1.

4) Someta a la población de clones a un proceso de hipermutación, haciendo que la variación sea inversamente proporcional a grado de afinidad del anticuerpo; este proceso da lugar a una nueva y mejorada población de clones (C*).

En el algoritmo propuesto los clones generados sufren un proceso de hipermutación la cual es inversamente proporcional a la afinidad del antígeno (entre mayor afinidad, menor es la tasa de mutación), dicha mutación está dada por:

$$p = e^{(-\rho * f)} \quad (6)$$

Donde, ρ es el factor de mutación y f la afinidad.

En la literatura los autores toman valores específicos en sus trabajos para el factor de mutación, producidos bajo un análisis de sensibilidad en las pruebas de sus desarrollos. Para el presente trabajo se determinó elaborar un análisis de sensibilidad a partir del rango entre 0 y 1 que era recurrente en la literatura, después de dicho análisis los mejores resultados para el parámetro de factor de mutación se consiguieron con 0.1.

Al fijar este parámetro se confirmó la noción planteada por Cortés [12] donde *"El impacto que tiene la mutación en un anticuerpo para generar un nuevo individuo es mínima, ya que se aplica de tal forma que sólo se realiza un cambio en la cadena".*

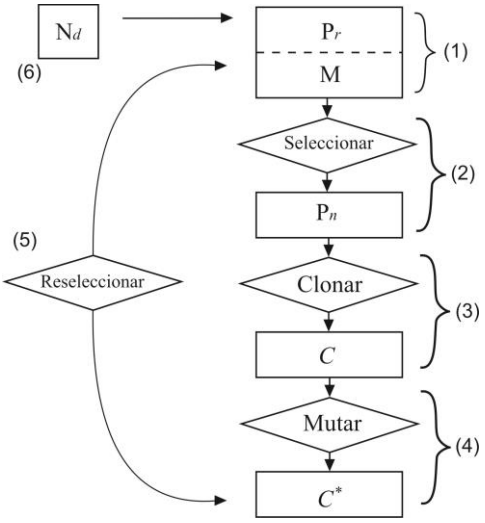
5) Seleccione los individuos mejorados del conjunto C* para estructurar un conjunto de "memoria".

6) Reemplace con los mejores individuos del conjunto C* a los *d* anticuerpos con menos afinidad de la población inicial.

Adicional se cuenta con el parámetro denominado número randómico de células, que corresponde al porcentaje de individuos respecto a la población que serían generados aleatoriamente con el fin de mantener la diversidad en la población.

El rango se encuentra entre 0 y 100% respecto a la población. El valor calculado que optimiza el makespan para la mayoría de instancias fue del 10%. Este valor cobra sentido pues valores pequeños en este parámetro no alteran negativamente la diversidad en la población, lo cual haría que el algoritmo se estancara en un espacio de búsqueda.

Figura 8 Diagrama general del algoritmo de selección clonal



Fuente: Castro y Von Zuben[71]**.**

1.4 ALGORITMOS DE OPTIMIZACIÓN POR COLONIA DE HORMIGAS

En los algoritmos de la familia ACO, 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. La hormiga artificial (k) se mueve de un nodo a otro (del nodo origen i al nodo destino j), construyendo paso a paso una solución que se va guardando en la memoria Tabú (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 definidos por el objetivo del problema.

Así, en cada iteración, las hormigas pueden construir soluciones aproximadas a problemas complejos como los de secuenciación, asignación, planificación o programación. Cada arista del grafo como se observa en la figura 13 tiene asociada dos tipos de información que guían el movimiento de la hormiga [47] y cuyos valores son modificados por las hormigas en cada iteración:

- η_{ij} Información heurística que 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 las soluciones encontradas por las hormigas para reflejar la experiencia adquirida por estos agentes.

Pseudocódigo de la metaheurística ACO [72]

$$
\begin{aligned}
&Procedimiento\ ACO\\
&\quad Establecer\ parámetros, Inicializar\ los\ rastros\ de\ feromona\\
&Actividades\ programadas\\
&Construcción\ de\ soluciones\ por\ hormigas\\
&Servidor\ de\ acciones\ (Opcional)\\
&Actualización\ de\ feromona\\
&End-Actividades\ programadas\\
&End-Procedimiento
\end{aligned}
$$

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 una 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 [73]

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. 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, 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.

1.4.1 ALGORITMO ELITISTA DE OPTIMIZACIÓN CON COLONIA DE HORMIGAS (EAS)

Esta versión de ACO 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.

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:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha} \left(\eta_{ij}\right)^{\beta}}{\sum_{l \in Tabu_k} (\tau_{il})^{\alpha} \left(\eta_{il}\right)^{\beta}} \qquad (7)$$

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

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). Se procura que las aristas con gran cantidad de feromona sean las más visibles, teniendo una probabilidad de transición mayor a las aristas de los otros nodos del conjunto de operaciones realizables. Para tener un algoritmo equilibrado (con un apropiado ajuste), los parámetros α y β deben tener valores adecuados, evitando valores cercanos a cero, porque si $\alpha = 0$, solo la información heurística indicaría que posibles elementos de la solución tendrán mayor probabilidad de ser seleccionados, lo que corresponde a un algoritmo greedy (voraz) estocástico, y si $\beta = 0$, solo será relevante la cantidad de feromona. En ambos casos las hormigas podrían estancarse en un óptimo local, generando la misma solución en cada iteración, 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 los relacionaremos 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:

$$\tau_{ij}(t) = \sum_{k=1}^{n} \Delta \tau_{ij}^{k} + \rho * \tau_{ij}(t-1) \quad (8)$$

Donde $\tau_{ij}^{k}(t)$ es la contribución de la hormiga k al total de 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, así:

$$\Delta \tau_{ij}^{k} = \frac{Q}{L_k} \quad (9)$$

Donde Q es una constante 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 el mismo 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:

$$\Delta \tau_{ij}^{k} = \frac{Q}{L_k} * e \quad (10)$$

1.4.2 ALGORITMO ELITISTA DE OPTIMIZACIÓN CON COLONIA DE HORMIGAS PARA EL PROBLEMA JSP.

La convergencia rápida de este algoritmo puede hacer que se pierda capacidad de exploración ya que las hormigas 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 hacen a las máquinas esperar (en pausa) unas unidades de tiempo para empezar su ejecución porque el trabajo correspondiente aún está activo en otra máquina. Pero esta operación que demora o retarda 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 problema de JSP[74]:

$Procedimiento$ EAS

1. $Inicializacion\ de\ los\ parametros$:
   $\propto, \beta, \rho, K\ Hormigas\ y\ C\ Ciclos$
2. $Para\ cada\ arista\ a_{ij}\ hacer$:
   $\tau_{0_{ij}} = c\ ;\ donde\ c\ es\ una\ constante$
   $\Delta\tau_{ij} = 0\ ;\ \ Acumulador\ de\ feromona\ actual$
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 $Para\ cada\ hormiga\ k\ hacer$:
$Obtener\ el\ makespan\ L_k\ del\ plan\ construido$
$Guardar\ el\ plan\ con\ el\ menor\ makespan\ del\ Ciclo\ C$
$Para\ cada\ arista\ a_{ij}\ hacer$:
$Calcular\ \Delta\tau_{ij}\ según\ ecuación\ 3\ o\ 4$ 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 $\alpha = 0.2,\ \beta = 0.8\ y\ \rho = 0.7$, el número de ciclos (o iteraciones) está fijado en 1000 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:

$$K = \frac{|J|}{2} \tag{11}$$

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 inicialmente visitables 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.

3.3. Mientras no termine de llenarse la memoria $tabu_k$, 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 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. Actualizar los rastros de feromona de las aristas visitadas utilizando un proceso conocido como *actualización en línea a posteriori*, que es una actualización global realizada fuera de línea (offline), es decir, 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.

CONTENDIENTE: PROCEDIMIENTO DE BÚSQUEDA MIOPE ALEATORIA Y ADAPTATIVA (GRASP).

Esta metaheurística es un método aproximado para problemas de optimización combinatoria[75]. El algoritmo GRASP implementa una búsqueda local, iniciando desde una construccion global básica diversa y de buena calidad, para asegurar el éxito posterior en los métodos de búsqueda local de manera iterativa.GRASP construye soluciones probabilísticas, por medio de dos fases que iteran un numero definido de veces hasta encuentrar la mejor solución [76].

El algoritmo GRASP se compone de dos fases:

Fase de construcción: En esta fase se busca combinar 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 definido por las restricciones, y se van ordenando los elementos bajo un criterio relacionado con la información heurística que tiene en cuenta la solución parcial que ya está disponible, esta información se denomina componente adaptativa, y permite determinar los componentes más prometedores denominados componentes miopes o voraces, con una función Greedy para cada nodo candidato, la cual encuentra el incremento del valor del makespan temporal al incluir el nodo candidato en la solución.

Dentro del algoritmo se incorpora 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.

Mientras se construye un calendario factible, no todas las operaciones pueden ser seleccionadas en una fase de construcción dada. Una operación O_k^j puede sólo ser asignada a un calendario si todas las operaciones a priori de los trabajos j han sido asignadas. Por tanto, en cada iteración de la fase de construcción, a lo mucho $|J|$ operaciones son candidatas a ser asignadas. Este conjunto de operaciones candidatas se denota como O_c.

Fase de búsqueda Local: En esta fase se realizan pruebas aleatorias con cada uno de los otros nodos de RCL para tratar de mejorar la solución inicial, utilizando la anterior regla de decisión miope aleatoria, 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, ésta se guarda y se explora su subespacio de vecindad en un nuevo ciclo de mejora.

En el caso del problema de JSP, se considera una sola operación como bloques de construcción. Se crea un calendario factible asignando operaciones individuales, una a la vez hasta que todas las operaciones son asignadas.
Sea O_k^j denota k-esima operación del trabajo j y está definido por (M_k^j, p_k^j), donde M_k^j es la máquina sobre la cual la operación O_k^j es realizada y p_k^j es el tiempo de procesamiento de la operación O_k^j.

Más de un algoritmo GRASP puede ser propuesto para el problema de JSP. Uno de los algoritmos selecciona las operaciones O_k^jque se traduce en el menor incremento en el makespan de los trabajos ya programados para el siguiente calendario. Sea la función de codicia $h(O)$ que denote el tiempo de ejecución de las tareas como resultado de la adición de operaciones O a las operaciones ya programadas.

La elección de GRASP es la siguiente operación de asignación de tareas:

$$\underline{O} = argmin(h(O)|O \in O_c) \quad (12)$$

Definiendo

$$\overline{O} = argmax(h(O)|O \in O_c) \quad (13)$$

$\underline{h} = h(\underline{O})$,Y $\overline{h} = h(\overline{O})$, la lista de candidatos restringidos (RCL) se define como:

$$RCL = \{(O \in O_c | \underline{h} \leq h(O) \leq \underline{h} + \alpha(\overline{h} - \underline{h}))\} \quad (14)$$

Donde α es un parámetro tal que $0 \leq \alpha \leq 1$.

La búsqueda local empleada en este GRASP para el problema de JSP es búsqueda local de dos intercambios basada en el grafo disyuntivo[77] [78]

$G = (V, A, E)$ es definido tal que:

$$V = \{O \cup \{0, |J| . |M| + 1\}\} \quad (15)$$

V: Es el conjunto de nodos, donde $\{0\}$ y $|J| . |M| + 1$ son fuentes y sumideros artificiales. $A = \{((v,w)|v,w \in O, v \prec w)\} \cup \{((0,w)|w \in O, \nexists v \in O \ni v \prec w)\} \cup \{((v, |J| . |M| + 1)|v \in O, \nexists w \in O \ni w \prec v)\}$ (16)

A: Es el de arcos dirigidos, conectando consecutivamente operaciones del mismo trabajo, y

$$E = \{((v,w)|M_v = M_w)\} \quad (17)$$

E: Es el conjunto de operaciones conectadas sobre la misma máquina.

Los vértices 0 y $|J| . |M| + 1$ tienen peso cero, mientras que el peso de los vértices $i \in \{1, \dots, |J| . |M|\}$ es el tiempo de procesamiento de las operaciones correspondientes al vértice i.

La entrada para GRASP incluye parámetros para ajustar el tamaño de la lista de candidatos y el número máximo de iteraciones de GRASP, y la semilla para el generador de números aleatorios. A continuación se presenta el seudocódigo de GRASP:

Pseudocódigo de la metaheurística GRASP [79]

$Procedimiento\ GRASP\ (Max\_Iteraciones, Semilla)$

$Establecer\ parámetros\ (\alpha)$

$Constructor\_Soluciones$

$Solución\ \leftarrow Construcción\_Miope\_Aleatoria(Semilla)$

$Solución\ \leftarrow Búsqueda\_Local(Solución)$

$Actualizar\_Solución\ (Solucion, Mejor\_Solución)$

$Fin - Constructor\_Soluciones$

$Fin - Procedimiento\ GRASP$

PRUEBAS Y ANÁLISIS DE RESULTADOS

Las funciones utilizadas para la prueba del algoritmo constan de 40 problemas pertenecientes a la clase LA. Esta clase maneja 8 tamaños diferentes de problemas propuestos por Lawrence: 10 × 5, 15 × 5, 20 × 5, 10 × 10, 15 × 10, 20 × 10, 30 × 10 y 15 × 15.

Con un análisis de sensibilidad se establecieron los valores de los parámetros de entrada para los tres algoritmos (EAS, CLONALG, GRASP).

Para el caso de EAS el parámetro alfa se definió de la siguiente manera $\alpha = 0.2$, el parámetro Beta $\beta = 0.8$ 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 el caso de CLONALG el parámetro de factor de clonación se definió en $\beta = 0.1$, el factor de mutación $\rho = 0.1$, la población inicial de 100 anticuerpos, el factor de numero aleatorio equivalente al 10% de la población y el número de generaciones en 300

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.

Para medir el desempeño de los algoritmos se realizaron comparaciones del número de evaluaciones de la función objetivo, que en el caso del problema corresponde al cálculo del makespan. Para medir la calidad de las soluciones, se realizó una comparación directa del makespan obtenido por el algoritmo propuesto en cada una de las instancias en 50 iteraciones y se comparó con el mejor conocido. Las

medidas estadísticas que se reportan son el promedio $\bar{x}$, la varianza S^2 y la desviación estándar

Las técnicas fueron programadas en lenguaje de programación Java utilizando los IDE Netbeans y Eclipse, además las pruebas se desarrollaron en un equipo portátil con sistema operativo Windows 7, procesador Intel Core I5 primera generación a 2.53 Ghz y 4Gb de memoria RAM. Estas características fueron suficientes y no se requirió de apoyo adicional en procesamiento ni memoria.

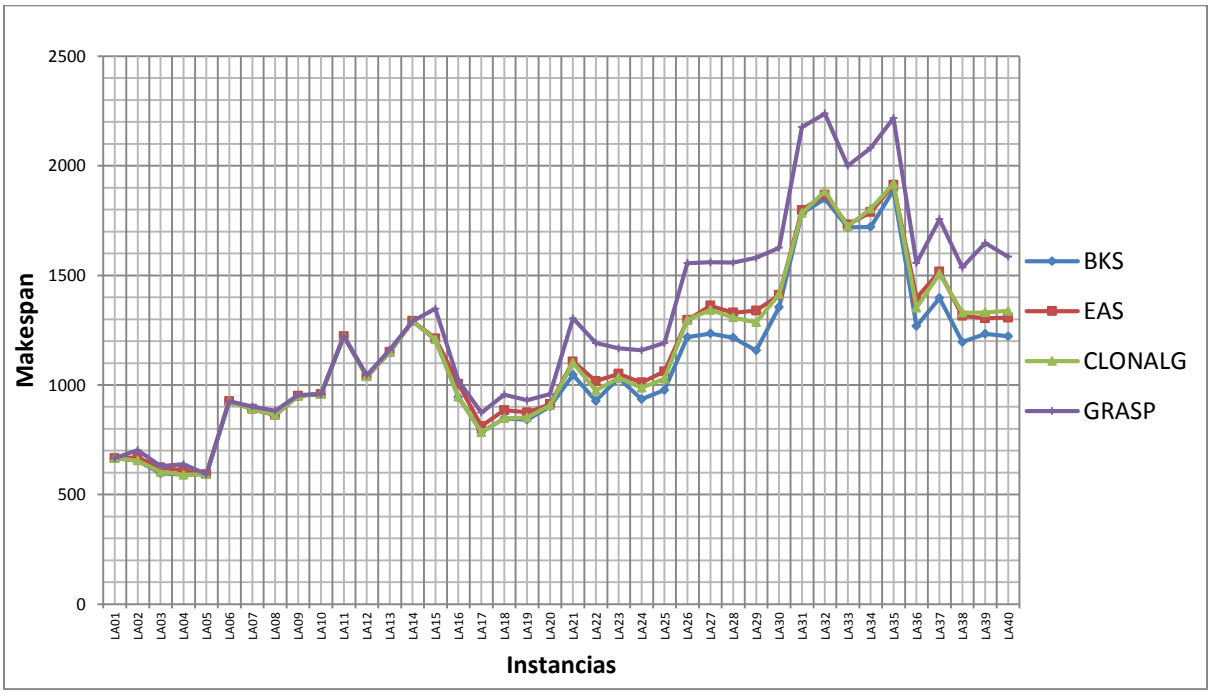
En la Tabla 9 y en la Figura 9, se observa el compendio de los resultados en cada instancia de Lawrence, se listan el mejor makespan C_{max} obtenido por cada algoritmo, el error relativo frente al makespan de la mejor solución conocida, además se incluye la mejor solución conocida y el tamaño de cada instancia.

Tabla 9 Resumen de errores relativos de los algoritmos para cada instancia del JSP (Best Know Solution [BKS], Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])

Instancia	Tamaño	BKS	EAS		CLONALG		GRASP	
			C_{max}	Error relativo %	C_{max}	Error relativo %	C_{max}	Error Relativo %
LA01	10 x 5	**666**	**666**	0,00	**666**	0,00	**666**	0,00
LA02	10 x 5	**655**	**669**	2,14	**655**	0,00	**702**	7,18
LA03	10 x 5	**597**	**617**	3,35	**603**	1,01	**630**	5,53
LA04	10 x 5	**590**	**595**	0,85	**590**	0,00	**638**	8,14
LA05	10 x 5	**593**	**593**	0,00	**593**	0,00	**593**	0,00
LA06	15 x 5	**926**	**926**	0,00	**926**	0,00	**926**	0,00
LA07	15 x 5	**890**	**890**	0,00	**890**	0,00	**902**	1,35
LA08	15 x 5	**863**	**863**	0,00	**863**	0,00	**883**	2,32
LA09	15 x 5	**951**	**951**	0,00	**951**	0,00	**954**	0,32
LA10	15 x 5	**958**	**958**	0,00	**958**	0,00	**958**	0,00
LA11	20 x 5	**1222**	**1222**	0,00	**1222**	0,00	**1222**	0,00
LA12	20 x 5	**1039**	**1039**	0,00	**1039**	0,00	**1045**	0,58
LA13	20 x 5	**1150**	**1150**	0,00	**1150**	0,00	**1160**	0,87
LA14	20 x 5	**1292**	**1292**	0,00	**1292**	0,00	**1292**	0,00
LA15	20 x 5	**1207**	**1212**	0,41	**1207**	0,00	**1348**	11,68

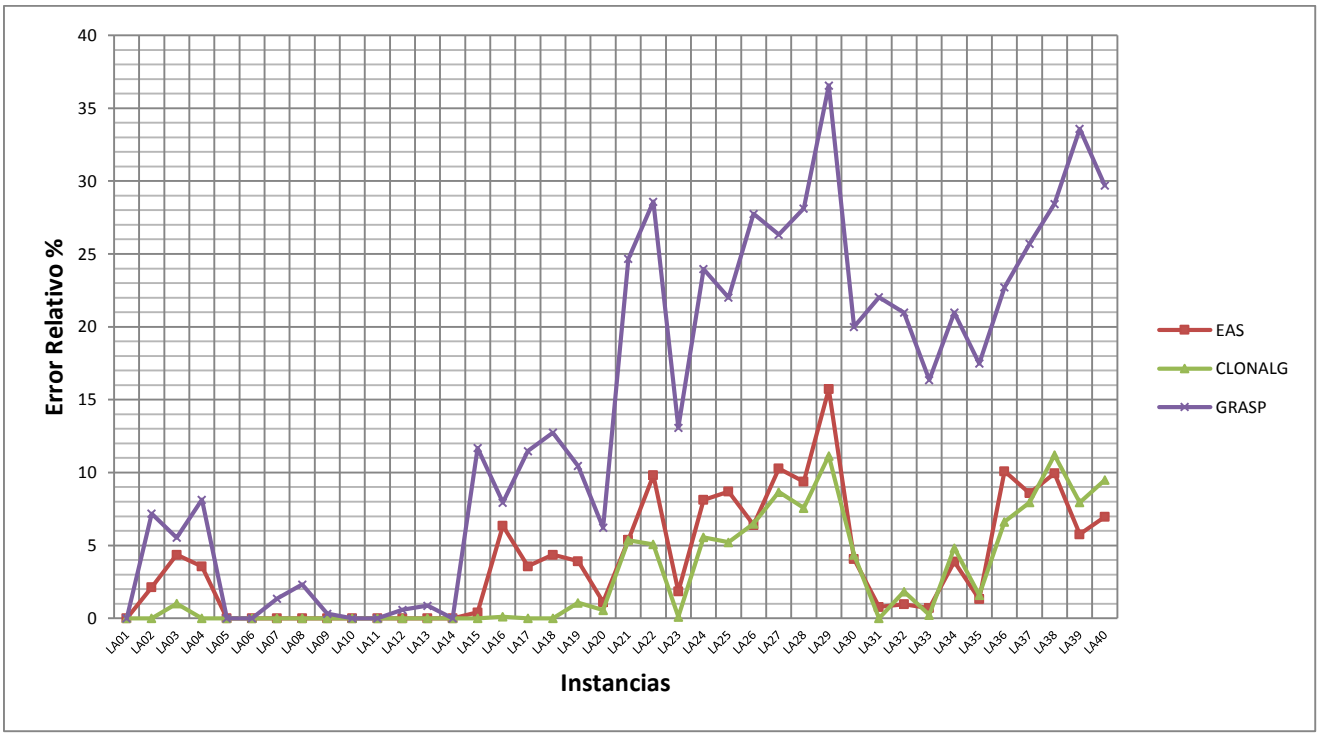
LA16	10 x 10	**945**	**996**	5,40	**946**	0,11	**1020**	7,94
LA17	10 x 10	**784**	**812**	3,57	**784**	0,00	**874**	11,48
LA18	10 x 10	**848**	**885**	4,36	**848**	0,00	**956**	12,74
LA19	10 x 10	**842**	**873**	3,68	**851**	1,07	**930**	10,45
LA20	10 x 10	**902**	**912**	1,11	**907**	0,55	**958**	6,21
LA21	15 x 10	**1046**	**1107**	5,83	**1102**	5,35	**1304**	24,67
LA22	15 x 10	**927**	**995**	7,34	**974**	5,07	**1192**	28,59
LA23	15 x 10	**1032**	**1049**	1,65	**1033**	0,10	**1167**	13,08
LA24	15 x 10	**935**	**1008**	7,81	**987**	5,56	**1159**	23,96
LA25	15 x 10	**977**	**1062**	8,70	**1028**	5,22	**1192**	22,01
LA26	20 x 10	**1218**	**1296**	6,40	**1297**	6,49	**1556**	27,75
LA27	20 x 10	**1235**	**1349**	9,23	**1342**	8,66	**1560**	26,32
LA28	20 x 10	**1216**	**1322**	8,72	**1308**	7,57	**1558**	28,13
LA29	20 x 10	**1157**	**1331**	15,04	**1286**	11,15	**1580**	36,56
LA30	20 x 10	**1355**	**1410**	4,06	**1414**	4,35	**1626**	20,00
LA31	30 x 10	**1784**	**1784**	0,00	**1784**	0,00	**2177**	22,03
LA32	30 x 10	**1850**	**1860**	0,54	**1884**	1,84	**2238**	20,97
LA33	30 x 10	**1719**	**1731**	0,70	**1723**	0,23	**2000**	16,35
LA34	30 x 10	**1721**	**1778**	3,31	**1804**	4,82	**2082**	20,98
LA35	30 x 10	**1888**	**1902**	0,74	**1918**	1,59	**2218**	17,48
LA36	15 x 15	**1268**	**1396**	10,09	**1352**	6,62	**1556**	22,71
LA37	15 x 15	**1397**	**1517**	8,59	**1508**	7,95	**1756**	25,70
LA38	15 x 15	**1196**	**1315**	9,95	**1330**	11,20	**1536**	28,43
LA39	15 x 15	**1233**	**1304**	5,76	**1331**	7,95	**1647**	33,58
LA40	15 x 15	**1222**	**1300**	6,38	**1338**	9,49	**1585**	29,71

Figura 9 Comportamiento de los Makespan de los algoritmos para el JSP (Best Know Solution [BKS], Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])



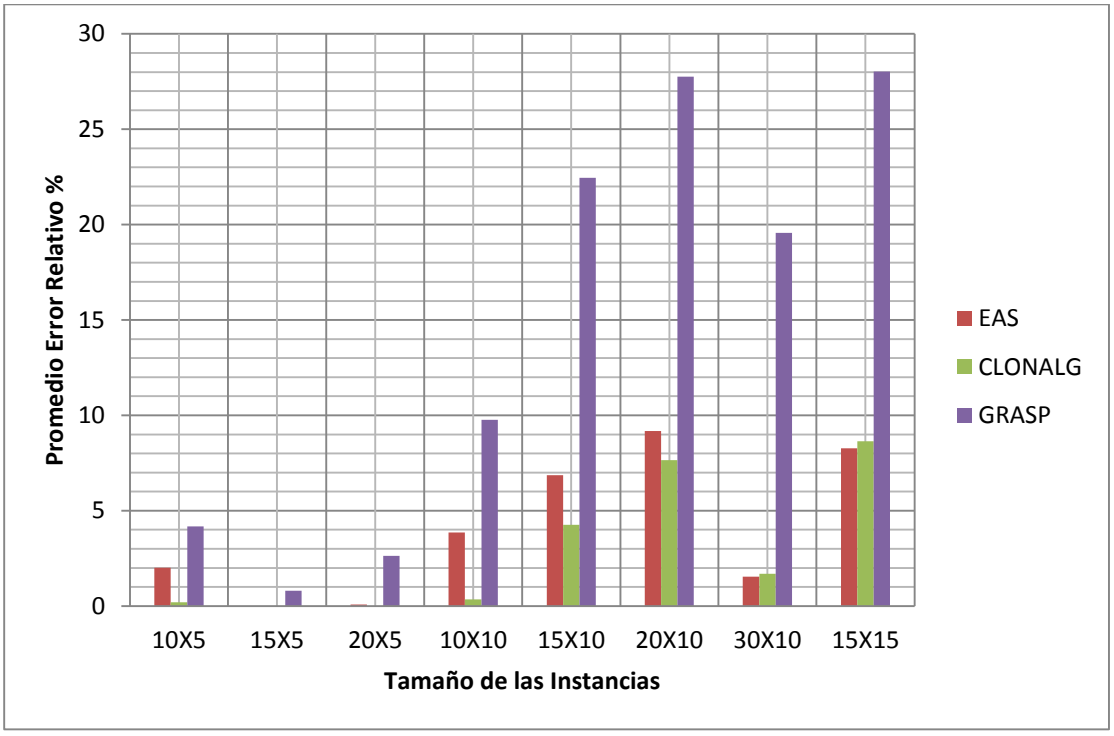
En la figura a continuación se observa el comportamiento del error a lo largo de las 40 instancias utilizando los 3 algoritmos, se observa como el algoritmo clonalg presenta el comportamiento más estable de los tres durante las primeras 20 instancias, de ahí en adelante su comportamiento con base al error relativo es similar al algoritmo de colonia de hormigas EAS, por otro lado el algoritmo Grasp presenta un comportamiento inestable y alejado a la mejor solución conocida. Se observa que conforme las instancias son más grandes y de mayor complejidad los errores crecen en la mayoría de los casos haciendo una excepción en la instancia 23 y el rango de la 30 a la 35, este comportamiento es similar en los 3 algoritmos.

Figura 10 Comportamiento del error relativo para las instancias de Lawrence (Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])



En la figura a continuación se observa la clasificación de los promedios de los errores relativos de cada algoritmo para cada uno de los diferentes tamaños de las instancias de Lawrence, 5 instancias corresponden a cada tamaño y en total son 8 tamaños diferentes por las 40 instancias. Se observa que los primeros cuatro tamaños presentan los promedios de los errores relativos más bajos, siendo el tamaño de 15X5 en del promedio de los errores relativos más bajo en los tres algoritmos y los promedios de los errores relativos más altos se encuentran en las instancias de 20X10 y 15X15. Es interesante observar que aunque en la mayoría de tamaños el mejor rendimiento lo obtuvo el algoritmo CLONALG para el caso de las instancias de 30x10 y 15X15 que tienen una alta complejidad presenta un mejor desempeño el algoritmo EAS.

Figura 11 Representación del error relativo vs tamaño de las instancias para cada algoritmo en el JSP(Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])



En la Tabla 11 se listan los errores relativos promedios, el mayor error la desviación estándar, y la varianza de cada uno de los algoritmos para el total de las 40 instancias,

Tabla 10 Medidas de calidad respecto al error relativo de los algoritmos (Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])

Algoritmos	EAS	CLONALG	GRASP
Promedio Error Relativo %	3,64	2,85	14,39
Mayor Error Relativo %	15,04	11,20	36,56
Desviación	3,90	3,66	11,42
Varianza	15,24	13,41	130,47

Se observa que el menor error y la menor dispersión de los datos fueron alcanzados por el algoritmo Inmune artificial, seguido muy cerca por el algoritmo de Hormigas, y los valores más altos en las medidas estadísticas aplicadas se encuentran en el algoritmo GRASP.

En la Tabla 12 se describen la cantidad de mejores soluciones alcanzadas por cada algoritmo y su porcentaje en referencia a las 40 instancias, se observa que el mejor rendimiento fue alcanzado por el algoritmo inmune artificial.

Tabla 11 Cantidad de Mejores Soluciones alcanzadas por los algoritmos para el JSP (Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])

Algoritmos	EAS	CLONALG	GRASP
Cantidad de BKS alcanzados	12	17	6
Porcentaje BKS alcanzado %	30	42,5	15

Para verificar los resultados se consultaron algunos desarrollos reportados en la literatura de técnicas bioinspiradas similares y otras no bioinspiradas; como indicadores de calidad se contrastaron en función del error promedio, adicional se validan la cantidad y porcentaje de las mejores soluciones alcanzadas. Los algoritmos que se listan en la Tabla 13 son los siguientes con sus respectivas referencias tomadas de los trabajos dirigidos por el profesor Coello [7] [24]:

- (**AS**) Este algoritmo es una variante del algoritmo Ant System basado en el comportamiento de forrajeo de las hormigas.[7]
- (**AIS**) Algoritmo Inmune artificial que se basa en el principio de selección clonal y utiliza expansión clonal, así como hipermutación somática [24]
- (**CULT**) Algoritmo cultural basado en teorías sociales y arqueológicas las cuales tratan de modelar la evolución cultural. [80]
- (**INSA**) Heurística utilizada por Búsqueda Tabú para la construcción de la solución inicial. [81]
- (**TS**)Búsqueda Tabú es una metaheurística designada para encontrar una solución vecina óptima en problemas de optimización combinatoria.[81]

- (**GRASPL**) Algoritmo GRASP procedimiento de Búsqueda Miope Aleatoria y Adaptativa Utiliza una representación de grafos. [82]
- (**GA**) Es un algoritmo genético simple que utiliza representación de llaves aleatorias.[83]

Tabla 12 Comparación resultados frente a algoritmos de la literatura para el JSP

Algoritmo	AS	AIS	CULT	INSA	TS	GRASPL	GA	EAS	CLONALG	GRASP
Error Relativo Promedio %	0,34	0,16	0,97	9,26	**0,05**	1,87	0,92	**3,64**	**2,85**	**14,39**
Cantidad de BKS alcanzados	22	31	25	4	**34**	22	21	**12**	**17**	**6**
Porcentaje BKS alcanzado %	55	77,5	62,5	10	**85**	55	52,5	**30**	**42,5**	**15**

Se observa que el algoritmo con un error relativo más bajo y el porcentaje más alto de BKS alcanzadas corresponde a la búsqueda Tabú, aunque los algoritmos propuestos en este trabajo alcanzan soluciones de calidad inferior, los errores relativos siguen siendo bajos y competitivos con varios trabajos como GRASP y el algoritmo INSA. Respecto al porcentaje de BKS el algoritmo CLONALG se encuentra cerca de la mayoría de trabajos con los que se está haciendo el contraste y que presentan mejor calidad en las soluciones.

Para medir el desempeño se compararon los tres algoritmos respecto a la cantidad de evaluaciones necesarias para llegar a la solución, una evaluación está definida como cada intento o iteración que el algoritmo realiza para verificar cómo se comporta una solución generada frente a la función objetivo o a una función que calcula la afinidad y que le sirve de referencia para aplicar estrategias de optimización. En la tabla 14 se presentan dichos resultados.

Tabla 13 Resumen de resultados con base a la cantidad de evaluaciones de los algoritmos para cada instancia del JSP (Best Know Solution [BKS], Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])

Instancia	Tamaño	BKS	EAS		CLONALG		GRASP	
		C_{max}	C_{max}	Eval	C_{max}	Eval	C_{max}	Eval
LA01	10 x 5	**666**	**666**	2375	**666**	6902	**666**	645
LA02	10 x 5	**655**	**669**	2809	**655**	18751	**702**	2809

LA03	10 x 5	**597**	**617**	2230	**603**	21321	**630**	4310
LA04	10 x 5	**590**	**595**	2257	**590**	13383	**638**	2000
LA05	10 x 5	**593**	**593**	101	**593**	4	**593**	101
LA06	15 x 5	**926**	**926**	531	**926**	3133	**926**	385
LA07	15 x 5	**890**	**890**	3443	**890**	26441	**902**	3409
LA08	15 x 5	**863**	**863**	2251	**863**	8182	**883**	505
LA09	15 x 5	**951**	**951**	391	**951**	3131	**954**	462
LA10	15 x 5	**958**	**958**	637	**958**	1121	**958**	250
LA11	20 x 5	**1222**	**1222**	1504	**1222**	6357	**1222**	900
LA12	20 x 5	**1039**	**1039**	1752	**1039**	3313	**1045**	1752
LA13	20 x 5	**1150**	**1150**	2952	**1150**	5257	**1160**	920
LA14	20 x 5	**1292**	**1292**	471	**1292**	1153	**1292**	340
LA15	20 x 5	**1207**	**1212**	3836	**1207**	27391	**1348**	3836
LA16	10 x 10	**945**	**996**	2700	**946**	18921	**1020**	2890
LA17	10 x 10	**784**	**812**	2401	**784**	27851	**874**	2401
LA18	10 x 10	**848**	**885**	2946	**848**	16662	**956**	2946
LA19	10 x 10	**842**	**873**	2394	**851**	21584	**930**	185
LA20	10 x 10	**902**	**912**	2496	**907**	27741	**958**	2496
LA21	15 x 10	**1046**	**1107**	3658	**1102**	79311	**1304**	3658
LA22	15 x 10	**927**	**995**	2938	**974**	86371	**1192**	2261
LA23	15 x 10	**1032**	**1049**	3826	**1033**	97121	**1167**	4914
LA24	15 x 10	**935**	**1008**	3097	**987**	64871	**1159**	2653
LA25	15 x 10	**977**	**1062**	3632	**1028**	58801	**1192**	3632
LA26	20 x 10	**1218**	**1296**	5955	**1297**	107181	**1556**	5955
LA27	20 x 10	**1235**	**1349**	4450	**1342**	112551	**1560**	7770
LA28	20 x 10	**1216**	**1322**	3938	**1308**	72101	**1558**	4410
LA29	20 x 10	**1157**	**1331**	4532	**1286**	112961	**1580**	2450
LA30	20 x 10	**1355**	**1410**	5186	**1414**	90321	**1626**	5186
LA31	30 x 10	**1784**	**1784**	7098	**1784**	121411	**2177**	8016
LA32	30 x 10	**1850**	**1860**	8016	**1884**	167801	**2238**	5415
LA33	30 x 10	**1719**	**1731**	5796	**1723**	149361	**2000**	5796
LA34	30 x 10	**1721**	**1778**	6811	**1804**	99241	**2082**	4335
LA35	30 x 10	**1888**	**1902**	7357	**1918**	102501	**2218**	3360
LA36	15 x 15	**1268**	**1396**	3405	**1352**	35502	**1556**	3405
LA37	15 x 15	**1397**	**1517**	2142	**1508**	66071	**1756**	1057
LA38	15 x 15	**1196**	**1315**	4051	**1330**	89221	**1536**	4051
LA39	15 x 15	**1233**	**1304**	3266	**1331**	115261	**1647**	3266
LA40	15 x 15	**1222**	**1300**	2655	**1338**	95121	**1585**	3927

En la tabla 14 se comparan los desempeños de los 3 algoritmos respecto a la cantidad de evaluaciones promedio por cada algoritmo y algunas medidas estadísticas descriptivas con base a la dispersión de los datos, se observa que el algoritmo EAS es el que presenta el mejor desempeño y la menor dispersión de datos, se observa que el comportamiento de los algoritmos CLONALG y GRASP en función de las evaluaciones es muy variable y el consumo de iteraciones de evaluación es alto.

Tabla 14 Medidas de desempeño respecto a las evaluaciones de los algoritmos (Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])

Algoritmo	EAS	CLONALG	GRASP
Promedio Evaluaciones	3124,83	54542	56471,78
Mayor número de Evaluaciones	8016	167801	131301
Menor número de Evaluaciones	101	4	9001
Desviación Evaluaciones	2557,49	47745,49	38324,51
Varianza Evaluaciones	6540744,866	2279631831	1468768307

En la figura a continuación se observa el comportamiento por tamaños de los promedios de las cantidades de evaluaciones consumidas por cada algoritmo, se puede denotar que la cantidad de evaluaciones requeridas por el algoritmo EAS para los diferentes tamaños es similar y relativamente bajo en comparación con los otros 2 algoritmos, a diferencia del algoritmo GRASP donde se evidencia que crece proporcionalmente al incremento de tamaño de la instancia, sin embargo el algoritmo CLONALG se comporta de manera inconstante porque no se observa un patrón claro o definido.

Figura 12 Comportamiento del número de evaluaciones promedio para los diferentes tamaños de las instancias de Lawrence (Best Know Solution [BKS], Algoritmo Colonia de Hormigas Elitista [EAS], Algoritmo de Selección Clonal [CLONALG], Procedimiento de busqueda Miope [GRASP])

Para finalizar se comparó respecto a la cantidad de evaluaciones que requerían los algoritmos reportados anteriormente en literatura[7] para llegar a las soluciones de cada instancia del JSP, las medidas estadísticas que se contrastaron fueron el promedio, la desviación estándar y los mayores y menores datos alcanzados

Tabla 15 Comparación desempeño respecto a número de evaluaciones frente a algoritmos de la literatura para el JSP

Algoritmo	Evaluaciones promedio	Desviación	Max	Min
AS	3564	2880,82	11583	386
SIA	175058	219287,11	600328	27
CULT	454,525	656763,39	1800000	2000
TS	11108	17283,17	64473	0

EAS	3124,83	2557,49	8016	101
CLONALG	54542	47745,49	167801	4

Se observa que en función del número de evaluaciones, se obtiene un desempeño destacado por los algoritmos desarrollados en el presente trabajo tanto para los trabajos similares como para el total de algoritmos presentados, la menor cantidad promedio de evaluaciones y las mejores medidas fueron alcanzados por el algoritmo EAS.

Como análisis general se concluye que el rendimiento respecto a la calidad de las soluciones el algoritmo del presente trabajo que se destaco fue el algoritmo CLONALG, logrando un nivel competitivo frente a la cantidad de mejores soluciones conocidas alcanzadas frente a los algoritmos publicados en la literatura. Sin embargo respecto al desempeño relacionado con el número de evaluaciones, los mejores valores fueron alcanzados por el algoritmo EAS superando los resultados de los algoritmos publicados en la literatura.

Los comportamientos de los algoritmos implementados en referencia a las instancias de Lawrence permiten evidenciar que a mayor tamaño de las instancias la calidad de las soluciones decrece, adicional para las primeras 20 instancias se presentan los mejores resultados, en este segmento se ubican la mayor cantidad de BKS alcanzadas. Respecto al número de evaluaciones el comportamiento del algoritmo CLONALG presenta una mayor dispersión y baja estabilidad, a diferencia del comportamiento del algoritmo EAS que se mantiene continuidad sin importar las diferencias de tamaño de las instancias de Lawrence.

APROXIMACIÓN A UN ESCENARIO DE APLICACIÓN DEL JSP EN EL ÁREA DE LA COMPUTACIÓN.

Uno de los escenarios de la computación en que el que tiene aplicabilidad el problema de planificación de recursos es el de la computación distribuida en donde una cantidad definida de computadores se agrupan a través de clústeres y grids, para procesar una serie de tareas solicitadas por usuarios. Estos sistemas pueden realizar una gran cantidad de cálculos en corto tiempo, por lo que son denominados como sistemas de computación de alto rendimiento HPC (high-performance computing), estos sistemas han sido diseñados para satisfacer a gran escala la creciente la demanda de la comunidad informática que requiere cada vez más potencia de cálculo [84]. Según Alberto Lafuente y Mikel Larrea[2] "Un sistema distribuido es una colección de equipos independientes que comparten un estado y que ofrecen a sus usuarios la visión de un sistema único".

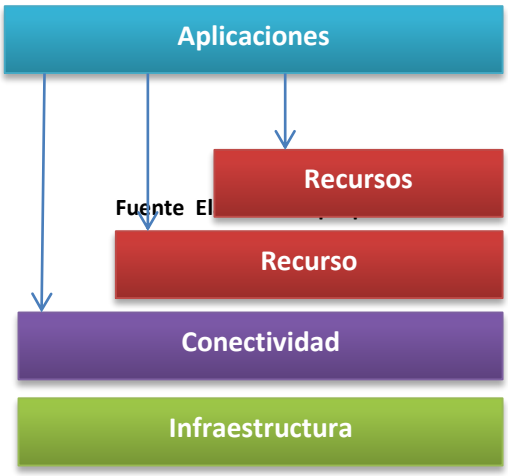
En los sistemas distribuidos uno de los fundamentos principales corresponde a la capacidad de disponer de una serie de recursos para diferentes usuarios ubicados en diferentes lugares siendo transparente para ellos la infraestructura desde la cual están accediendo. Dichos recursos puedes ser equipos de cómputo, redes de telecomunicaciones , instrumentos o herramientas científicas y datos o información[85].

En la figura a continuación se observa la arquitectura de los sistemas distribuidos grid, que está conformada por 4 capas, la capa de inferior denominada de infraestructura donde se encuentran los recursos sobre los que se realiza el acceso a la grid, la siguiente capa denominada de conectividad es en la que convergen las definiciones de los protocolos y autenticaciones para las transacciones de la red , la siguiente capa corresponde a la capa de recursos donde se realiza la gestión, inicialización, monitorización, control y aprovechamiento de las operaciones

[2] Tomado de la Web http://www.sc.ehu.es/acwlaalm/sdi/introduccion-slides.pdf, Consultado el 15 de Mayo de 2013

realizadas mediante el acceso compartido a los recursos individuales que componen la grid y por último la capa de aplicaciones que corresponde al grupo de programas que los usuarios operan y consumen dentro de la grid.

Figura 13 Arquitectura Grid

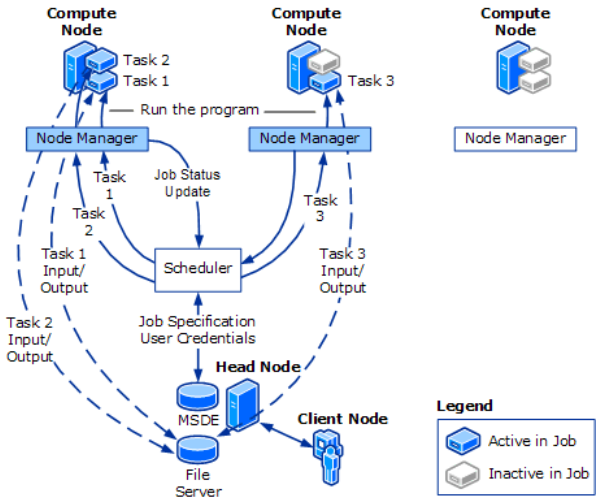


Fuente El

La administración de los recursos de un clúster es realizada por un Job Scheduler, que gestiona la distribución de las tareas en los nodos de cómputo para su ejecución, en el orden en que aparecen en la lista de tareas factibles a ejecutarse. Para despachar una tarea, el Job Scheduler pasa la tarea al nodo designado, que puede ser cualquiera de los nodos asignados al trabajo. A menos que se hayan especificado las restricciones, las tareas son despachadas con la siguiente prioridad, el primero en llegar, primero en ser ejecutado. También se pueden aplicar otras reglas de prioridad, como la de atender primero al trabajo más corto que genera una cola de tareas con tiempo de procesamiento creciente, o al contrario, atender de primero al trabajo más largo. Estas dos últimas reglas son similares a la regla de tiempo de proceso corto y la regla de tiempo de proceso largo aplicada en el algoritmo de optimización para definir la visibilidad de las operaciones candidatas según la extensión de su tiempo de procesamiento.

En la figura 19 se muestra un ejemplo de despacho de tareas secuenciales (como se debe realizar en JSPP), 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 14 Ejemplo de ejecución de tareas secuenciales en una arquitectura Grid



Fuente: technet.microsoft.com[3]

Los sistemas distribuidos requieren de métodos de optimización combinatoria adecuados para realizar la planificación de trabajos (Job scheduling), que permitan optimizar el tiempo y el uso de recursos en forma paralela. Bajo el enfoque de algoritmos bioinspirados, se encuentran trabajos de optimización por colonia hormigas que han abordado el problema de calendarización del flujo de trabajos en entornos distribuidos, es el caso del trabajo de W.N Chen [86], este trabajo, utiliza se basa en un algoritmo de colonia de hormigas que implementa una actualización local de la feromona (en cada paso de las hormigas) para la planificación a gran escala de los flujos de trabajo con diferentes parámetros de QoS. Este algoritmo

[3] http://technet.microsoft.com/en-us/library/cc720125(v=ws.10).aspx consultado 23 de abril de 2013

permite a los usuarios especificar sus preferencias de calidad de servicio, así como definir umbrales mínimos QoS para una determinada aplicación.

Otro trabajo en esta área fue desarrollado por Lorpunmanee [87], este autor aborda el problema mediante un algoritmo de optimización colonia de hormigas que permite la generación de planes de la red de la arquitectura distribuida, utilizando la información dinámica de la misma, logrando mejorar la toma de decisión de la planificación de la arquitectura de red.

Otro trabajo es el de Chang [88] que utiliza un algoritmo de optimización de Colonia de Hormigas Balanceada (BACO) para la planificación de tareas en el entorno de red. El principal aporte de este trabajo consiste en equilibrar toda la carga del sistema y tratar de minimizar el makespan de un conjunto dado de puestos de trabajo. En el trabajo de Liu et all [89] se utiliza el concepto Súper Scheduling aplicado al entorno de red. Se utiliza un algoritmo de Colonia de Hormigas para optimizar la asignación de los recursos de red.

Otro trabajo interesante fue desarrollado por Li [90], en este artículo se presenta un mecanismo de planificación de tareas que permite la adaptación de los puestos de trabajo. La técnica utilizada fue un algoritmo de colonia de hormigas que usa las propiedades de la inteligencia de enjambres.

Por ultimo en esta revisión se encuentra el trabajo de Ku-Mahamud [91], que propone un algoritmo de colonia de hormigas para la planificación de tareas en el entorno de red. El algoritmo se centra en la actualización del sendero de feromona local y los valores límite del rastro. La matriz de resultados se utiliza para registrar el estado de la disposición de cada uno de los recursos de red.

En el escenario de aproximación a una instancia real, el objetivo está orientado a optimizar la planificación de las tareas de cada usuario, 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, el algoritmo EAS busca el plan con el menor makespan. En la tabla a continuación se caracteriza el escenario de aplicación

Tabla 16 Caracterización de los elementos de un escenario aproximado de clúster

ELEMENTO	DEFINICIÓN
Trabajo	Corresponde a un conjunto de tareas secuenciales y es definido por un usuario.
Operación	Corresponde a una tarea que puede ser independiente o depender de otras tareas, y no se permite operaciones simultáneas del mismo trabajo para JSPP.
Máquina	Corresponde a un recurso computacional que es un nodo del clúster (que tiene múltiples procesadores).
Tiempo de procesamiento	Corresponde al tiempo solicitado en la reserva por el usuario, que es igual para todas las tareas del trabajo correspondiente.

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 sobre esta instancia se pueden ver en la Tabla 4.

Tamaño de la instancia: 10 trabajos x 21 máquinas

Nombre de la Instancia: Aproximación clúster

Tabla 17 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 validar la instancia y los resultados obtenidos se utilizó la herramienta Lekin[4] que es un software de planificación desarrollado en la Stern School of Business de la Universidad de Nueva York, esta aplicación permite leer 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.

La herramienta Lekin permite generar planes de trabajos, para lo cual usa heurísticas como el algoritmo Generic Shifting Bottleneck desarrollado por Nutthapol Asadathorn. Para la instancia planteada, la calidad de la solución alcanzada por la heurística corresponde a un makespan de 76 unidades de tiempo.

En la tabla a continuación se plasman los resultados de la ejecución del algoritmo de optimización por colonia de hormigas utilizando 50 iteraciones para la instancia antes planteada.

Tabla 18 Resultados de EAS y CLONALG con la instancia del clúster

Algoritmo	EAS		CLONALG	
Iteración	Makespan	Evaluaciones	Makespan	Evaluaciones
1	72	2505	72	1898
2	72	2150	72	6456
3	72	2325	72	3131
4	72	3980	72	2504
5	73	3140	72	115
6	72	640	72	1133
7	72	1815	72	6182
8	73	4895	72	42637
9	72	3100	72	3915
10	72	2220	72	6168
11	75	3940	72	1449
12	72	3430	72	5748
13	72	1265	72	44803
14	73	3975	72	7915
15	72	2995	72	3274
16	72	860	72	3176

[4] Flexible Job-Shop Scheduling System. http://community.stern.nyu.edu/om/software/lekin/ Consultado el 12 agosto de 2012

17	72	1890	72	82820
18	72	2745	72	6199
19	72	3150	72	4143
20	72	1310	72	4850
21	72	2000	72	2265
22	72	4350	72	4745
23	73	3430	72	4257
24	72	1580	72	4569
25	75	1825	72	8309
26	72	685	72	1111
27	75	4515	72	6247
28	72	1630	72	2131
29	73	4930	72	11239
30	72	1260	72	4287
31	72	2750	72	6225
32	76	20	72	4574
33	72	1060	72	6531
34	72	3465	72	2226
35	73	3020	72	1234
36	76	25	72	14231
37	72	4180	72	2352
38	73	3860	72	4151
39	72	4750	72	5497
40	72	715	72	87864
41	75	2300	72	11480
42	75	1860	72	1111
43	76	20	72	7069
44	75	3675	72	15796
45	72	4990	72	1716
46	76	15	72	4363
47	72	1025	72	16325
48	72	2405	72	5350
49	76	15	72	3537
50	72	3820	72	3142
Promedio Makespan	72.9		72	
Varianza:	2.01		0	
Desviación estándar	1.41		0	
Mejor Makespan	72		72	
Mejor Numero de Evaluaciones	640		115	

Promedio Evaluaciones	2450.1	9849
Mejores Soluciones alcanzadas	32	50
% de Mejores soluciones alcanzadas	64	100

La solución del makespan alcanzada por los algoritmos bioinspirados es superior a la alcanzada por Lekin, y corresponde a un plan de 72 horas, para el caso del algoritmo de optimización por hormigas con 640 evaluaciones y para el caso del algoritmo inmune con 115 evaluaciones. Para observar el desempeño se realizaron 50 iteraciones de las cuales en el algoritmo de colonia de hormigas se alcanzó el valor de 72 horas en 32 ocasiones, lo que corresponde al 64% del total de iteraciones, para el caso del algoritmo inmune el rendimiento fue superior alcanzando en las 50 iteraciones el valor de 72 en el makespan, Adicional se observa que el promedio de la cantidad de evaluaciones es menor en el algoritmo de optimización por colonia de hormigas, que en el caso del algoritmo inmune.

CONCLUSIONES

El algoritmo inmune artificial presenta una mejor calidad de la solución que el algoritmo de colonia de hormigas para las instancias de Lawrence planteadas, debido a que presenta el menor error relativo y el mayor porcentaje de mejores soluciones conocidas alcanzadas, aunque el algoritmo de optimización por colonia de hormigas presenta un mejor desempeño en referencia a la cantidad de evaluaciones requeridas para encontrar la solución y una menor desviación estándar de las mismas lo que corresponde a un menor grado de dispersión de ese conjunto de datos. Ambos algoritmos superaron en ambos aspectos a la implementación realizada del algoritmo de búsqueda miope.

Los soluciones generadas por el algoritmo inmune artificial son competitivas frente a resultados presentados en la literatura especialmente para el caso de porcentaje de mejores soluciones conocidas alcanzadas, además los resultados de desempeño alcanzados por el algoritmo de optimización por colonia de hormigas son destacados, lo que permite que se hagan mejoras e incorporaciones de procedimientos de búsqueda local sobre dicho algoritmo manteniendo aun un buen desempeño en relación a la cantidad de evaluaciones.

Se realizó una aproximación de un escenario real en el área computacional, alcanzando una solución de buen desempeño frente a la herramienta Lekin que permite la solución de escenarios reales de planificación en diferentes problemas. Debido a las restricciones que plantea el problema JSP se exploró el caso de las arquitecturas de sistemas distribuidos secuenciales que en la literatura no han sido tan explorados.

RECOMENDACIONES

Se sugiere la implementación de un método que mejore la decisión de las hormigas, discriminando el conjunto de operaciones factibles a solo las más adecuadas, de este modo en cada uno de sus pasos de las hormigas se reduce el la diversidad cada vez se está más cerca de la solución, esto permite disminuir el tiempo en el que las máquinas están ociosas, adicional se propone incorporar una heurística para realizar una búsqueda Local específicamente para el servidor de operaciones que en este trabajo se omitió. Se debe aprovechar el bajo número de evaluaciones que realiza EAS, realizando una búsqueda local extensa para por lo menos igualar los resultados de las mejores técnicas que han abordado el JSP

Para el algoritmo inmune artificial se propone la construcción de los individuos a partir de librerías de anticuerpos o mediante una heurística que permita generar anticuerpos con buena afinidad, reduciendo de este modo el espacio de búsqueda de donde parten los anticuerpos candidatos a entrar a la población y además esto aportaría a aumentar la convergencia rápida de la solución y a disminuir la cantidad de evaluaciones realizadas por el algoritmo

Como trabajo futuro se propone aplicar las técnicas Bioinspiradas a arquitecturas distribuidas de flujo de tareas paralelas, realizando una optimización de carácter multi-objetivo en la que se tenga en cuenta otras medidas de desempeño como el tiempo consumido por todos los trabajos y la suma de todos los retrasos de los trabajos, buscando disminuir las esperas de los usuarios que mejoraría la calidad de la disponibilidad de los recursos de computo.

IMPACTO

El presente trabajo genero los siguientes productos de Investigación:

Ponencia Evento Nacional

Wilfredo Ariel Gómez Bueno, Nelson Eduardo Díaz Díaz, Leydy Luna Martínez, Lola Xiomara Bautista Rozo, Ponencia "*Enfoque Inmune Artificial Aplicado Al Problema De Job Shop Scheduling*" En: Colombia. 2012. *Evento:* 1 Congreso Nacional de Diseño e Ingenierías Fisicomecánicas.

Articulo Revista Internacional

Flórez, Edson. Gómez, Wilfredo. Bautista, Lola. "*An Ant Colony Optimization Algorithm For Job Shop Scheduling Problem"*. International Journal of Artificial Intelligence & Applications (IJAIA), May 2013.

Ponencias Evento Internacional

Flórez, Edson. Gómez, Wilfredo. Bautista, Lola Ponencia: "*Algoritmo Elitista de Optimización por Colonia de Hormigas aplicado al problema de planificación de Tareas*" En: Argentina. 2013. *Evento:* 42° Jornadas Argentinas de Informática Simposio Argentino de Inteligencia Artificial 2013

Gómez, Wilfredo. Diaz, Nelson E, Bautista, Lola X Ponencia: " *Algoritmo Inmune de Selección Clonal para el problema de Job Shop Scheduling*" En: Chile. 2013. *Evento:* IV Congreso Internacional de Computación e Informática del Norte de Chile.

Trabajos de Grado Codirigidos

Flórez, Edson "*Implementación De Un Algoritmo De Colonia De Hormigas Aplicado En El Área De Planificación De Recursos*" Universidad Industrial De Santander, 2013.

Diaz, Nelson Martínez, Leydy “*Implementación De Un Algoritmo Inmune Artificial Aplicado En El Área De Planificación De Recursos*” Universidad Industrial De Santander, 2012.

BIBLIOGRAFÍA

[1] J. Greensmith, A. Whitbrook, and U. Aickelin, “Artificial Immune Systems,” pp. 1–29.

[2] C. Blum and A. Roli, “Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison,” vol. 35, no. 3, pp. 268–308, 2003.

[3] G. W. Flake, *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*, vol. 107, no. 6. MIT Press, 2000, p. 514.

[4] O. G. Dur, G. Nasini, and A. N. Aguilera, “´ n Combinatoria y Grafos Optimizaci o,” pp. 95–107.

[5] D. De Inform, T. Universidad, and S. Mar, “Estado del Arte del Job Shop Scheduling Problem Introducci ´ on Definici ´ on del Problema,” pp. 1–8, 2006.

[6] G. Mailing, “Algoritmos heurísticos y el problema de job shop scheduling Algoritmos heurísticos y el problema de job shop scheduling,” 2003.

[7] Emanuel Téllez Enriquez, “Uso de una Colonia de Hormigas on de Horarios Resumen,” Laboratorio Nacional de Informática avanzada, 2007.

[8] C. S. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to Algorithms*. 2001, p. 1202.

[9] J. F. B. Villalpando, “Análisis asintótico con aplicación de funciones de landau como método de comprobación de eficiencia en algoritmos computacionales,” *e-Gnosis Universidad de Guadalajara, Guadalajara, México*, vol. 1, 2003.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979, p. 340.

[11] Tapan P. Bagchi, *Multiobjective Scheduling by Genetic Algorithms*, Springer. 1999, p. 358.

[12] N. C. Cortes and C. A. C. Coello, “Multiobjective Optimization using Ideas from the Clonal Selection Principle.”

[13] D. Cort, “Un Sistema Inmune Artificial para resolver el problema del Job Shop Scheduling.”

[14] D. Applegate y W. Cook, “A study of the Job shop scheduling problem.pdf,” *ORSA Journal on Computing*, vol. 3, pp. 149–156, 1991.

[15] J. and B. Adams and D. las, Egon and Zawack, “The shifting bottleneck procedure for job shop scheduling,” *Management science*, vol. 34, pp. 391–401, 1988.

[16] and R. S. S. H. Matsuo, C. J. Suh, “A controlled search simulated annealing method for the general jobshop scheduling problem,” *Austin, TX: Grad. School of Bus., Univ. of Texas*. Austin, 1988.

[17] D. Applegate and W. Cook., “A computational study of the job-shop scheduling instance,” *ORSA Journal on Computing*, vol. 3, pp. 149–156, 1991.

[18] J. K. Van Laarhoven, Peter JM and Aarts, Emile HL and Lenstra, “Job shop scheduling by simulated annealing,” *Operations research*, vol. 40, pp. 113–125, 1992.

[19] E. Pinson, “A practical use of Jackson’s preemptive schedule for solving the job shop problem,” *Annals of Operations Research*, vol. 26, pp. 269–287, 1991.

[20] J. K. Vaessens, Robert Johannes Maria and Aarts, Emile HL and Lenstra, “Job shop scheduling by local search.,” *INFORMS Journal on Computing*, vol. 8, pp. 302–317, 1996.

[21] E. Carlier, Jacques and Pinson, “Adjustment of heads and tails for the job-shop problem,” *European Journal of Operational Research*, vol. 78, pp. 146–161, 1994.

[22] D. B. Martin, Paul Douglas and Adviser-Shmoys, *A time-oriented approach to computing optimal schedules for the job-shop scheduling problem*. New York: Cornell University, 1996.

[23] C. Nowicki, Eugeniusz and Smutnicki, “A fast taboo search algorithm for the job shop problem,” *Management science*, vol. 42, pp. 797–813, 1996.

[24] D. Cortés Rivera, “Un Sistema Inmune Artificial para resolver el problema del Job Shop Scheduling,” CINVESTAV, 2004.

[25] D. B. Martin, Paul and Shmoys, “A new approach to computing optimal schedules for the job-shop scheduling problem,” in *Integer Programming and Combinatorial Optimization*, Springer, 1996, pp. 389–403.

[26] W. BrinkkÄotterandPeterBrucker., “Solving open benchmark problems for the job shop problem,” 1999.

[27] A. Balas, Egon and Vazacopoulos, “Guided local search with shifting bottleneck for job shop scheduling,” *Management Science*, vol. 44, pp. 262–275, 1998.

[28] R. Vaessens, “Operations research library of problems,” *Management School, Imperial College London, Anonymous FTP site at ftp://mscmga. ms. ic. ac. uk/pub/jobshop1. txt*, 1996.

[29] A. Henning., “Praktische Job-Shop-Scheduling-Probleme,” Friedrich-Schiller-University Jena, 2002.

[30] S. Thomsen, “Metaheuristics combined with branch & bound.,” Copenhagen, Denmark, 1997.

[31] R. Storer, Robert H and Wu, S David and Vaccari, “Problem and heuristic space search strategies for job shop scheduling,” *ORSA Journal on Computing*, vol. 7, pp. 453–467, 1995.

[32] E. D. Taillard, “Parallel taboo search techniques for the job shop scheduling problem.,” *ORSA journal on Computing*, vol. 6, pp. 108–117, 1994.

[33] M. Wennink., “Personal communitation to e. taillard.” [Online]. Available: http://www.idsia.ch/»eric/problemes.dir/ordonnancement.dir /job-.

[34] R. Schilham., “Personal communication (to e. taillard).” 2000.

[35] and R. S. E. Aarts, Huub ten Eikelder, J. K. Lenstra, “An adaptive mem- ory programme embedding a taboo search algorithm with ns neighbourhood and constant time ta.” Personnal communication, 1999.

[36] J. P. Caldeira., “hybrid evolutionary-tabu algorithm.” communication to E. Taillard, 2003.

[37] S. R. Lawrence, “Resource-Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques,” Carnegie-Mellon University, 1984.

[38] J. E. Beasley, “OR-Library: Distributing Test Problems by Electronic Mail,” *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[39] M. Chandrasekaran, P. Asokan, S. Kumanan, T. Balamurugan, and S. Nickolas, “Solving job shop scheduling problems using artificial immune system,” *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 5–6, pp. 580–593, Jan. 2006.

[40] T. Yamada and R. Nakano, “Job-shop scheduling,” pp. 134–160, 1997.

[41] T. Yamada and R. Nakano, “Genetic Algorithms for Job-Shop Scheduling Problems,” no. March, pp. 67–81, 1997.

[42] J. K. Lenstra, “Job Shop Scheduling by Local Search 1 The job shop scheduling problem,” pp. 1–28.

[43] M. Ventresca and B. M. Ombuki, “Ant Colony Optimization for Job Shop Scheduling Problem,” no. February, 2004.

[44] A. Udomsakdigool and V. Khachitvichyanukul, “Ant colony algorithm for multi-criteria job shop scheduling to minimize makespan , mean flow time and mean tardiness ∗,” vol. 6, no. 2, pp. 117–123, 2011.

[45] O. D. Castrill, J. Alberto, and W. A. Sarache, “Solución de un problema Job Shop con un,” pp. 75–92, 2009.

[46] C. H. Papadimitriou y K. Steiglitz., *Combinatorial. Optimization - Algorithms and Complexity*, Prentice H. 1982.

[47] F. H. Sergio Alonso, Oscar Cordón, Iñaki Fernández de Viana, “La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques.,” Universidad de Granada, España, 2004.

[48] S. Binato, W. J. Hery, and D. M. Loewenstern, “A GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE,” no. December, 2000.

[49] L. Aarts y J. K. Lenstra, *Local Search in Combinatorial Optimization.* .

[50] E. A. Silver, “AN OVERVIEW OF HEURISTIC SOLUTION METHODS An Overview of Heuristic Solution Methods,” 2002.

[51] J. Dréo, “Different classifications of metaheuristics,” 2007. [Online]. Available: http://nojhan.free.fr/metah/. [Accessed: 06-Apr-2013].

[52] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2003.

[53] M. Dorigo and G. Di Caro, “Ant colony optimization: a new meta-heuristic,” in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999, vol. 2, p. -1477 Vol. 2.

[54] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995, vol. 4, pp. 1942–1948 vol.4.

[55] M. Mobini, Z. Mobini, and M. Rabbani, “An Artificial Immune Algorithm for the project scheduling problem under resource constraints,” *Applied Soft Computing*, vol. 11, no. 2, pp. 1975–1982, 2011.

[56] L. N. de Castro and J. Timmis, “An artificial immune network for multimodal function optimization,” *Proceedings of the 2002 Congress on Evolutionary Computation. CEC’02 (Cat. No.02TH8600)*, vol. 1, pp. 699–704, 2002.

[57] C. A. C. Coello, “Job Shop Scheduling using the Clonal Selection Principle,” no. 1.

[58] X. Zuo and Y. Fan, “SOLVING THE JOB SHOP SCHEDULING PROBLEM BY AN IMMUNE ALGORITHM,” no. August, pp. 18–21, 2005.

[59] S. Forrest, a. S. Perelson, L. Allen, and R. Cherukuri, “Self-nonself discrimination in a computer,” *Proceedings of 1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 202–212, 1994.

[60] J. E. Hunt and D. E. Cooke, “Learning using an artificial immune system,” *Journal of network and computer applications*, vol. 19(2), pp. 189–212., 1996.

[61] M. Neal, J. Hunt, and J. Timmis, “Augmenting an artificial immune network,” in *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, 1998, pp. 3821–3826.

[62] L. N. De Castro, C. Brazil, and F. J. Von Zuben, “The Clonal Selection Algorithm with Engineering Applications 1,” no. July, pp. 36–37, 2000.

[63] D. Dasgupta and L. F. Niño, *Immunological Computation: Theory and Applications*. 2008.

[64] W. A. Gómez Bueno, M. I. Cuadrado Morad, and H. Arguello Fuentes, “Algoritmo Basado en Una Red Inmune Artificial para La Alineación de Patrones de Puntos,” *Revista De La Escuela Colombiana De Ingeniería*, vol. v.86, pp. 25 – 33, 2012.

[65] U. Garain, M. P. Chakraborty, and D. Dasgupta, “Recognition of Handwritten Indic Script using Clonal Selection Algorithm,” pp. 1–12.

[66] F. O. de França, F. J. Von Zuben, and L. N. de Castro, “An artificial immune network for multimodal function optimization on dynamic environments,” *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO ’05*, p. 289, 2005.

[67] L. N. de Castro and F. J. Von Zuben, “Learning and optimization using the clonal selection principle,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, Jun. 2002.

[68] J. Brownlee, “Clonal Selection Algorithms.”

[69] N. E. D. DIAZ and L. J. L. MARTÍNEZ, “IMPLEMENTACIÓN DE UN ALGORITMO INMUNE ARTIFICIAL APLICADO EN EL ÁREA DE PLANIFICACIÓN DE RECURSOS,” Universidad Industrial de Santander, 2012.

[70] T. Y. and R. Nakano, *Genetic algorithms in engineering systems*, IEE contro. The Institution of Electrical Engineers, 1997, pp. 134–160.

[71] L. N. De Castro and F. J. Von Zuben, “The Clonal Selection Algorithm with Engineering Applications,” in *Proceedings of GECCO*, 2000, vol. 3637, no. July, pp. 36–37.

[72] K. Dorigo, Marco and Socha, “An introduction to ant colony optimization,” *Handbook of approximation algorithms and metaheuristics*, pp. 26–1, 2006.

[73] T. Dorigo, Marco and Birattari, Mauro and Stutzle, “Ant colony optimization,” *Computational Intelligence Magazine, IEEE*, vol. 1, pp. 28–39, 2006.

[74] E. FLÓREZ, “IMPLEMENTACIÓN DE UN ALGORITMO DE COLONIA DE HORMIGAS APLICADO EN EL ÁREA DE PLANIFICACIÓN DE RECURSOS,” Universidad Industrial de Santander, 2013.

[75] M. G. C. Resende and C. C. Ribeiro, “Greedy randomized adaptive search procedures,” 2002.

[76] S. Alonso, O. Cordón, I. F. de Viana, and F. Herrera, “La Metaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques,” in *Mejora de Metaheurísticas mediante Hibridación y sus Aplicaciones*, Universidad de Granada., Ed. .

[77] M.Tupia, “Un algoritmo GRASP para resolver el problema de la programación de tareas dependientes en máquinas diferentes (task scheduling),” San Marcos, 2005.

[78] B. Roy and B. Sussmann, “Les problemes d’ordonnancement avec constraints disjonctives,” *SEMA*. Paris, France, 1964.

[79] M. R. y C. Ribeiro, “Greedy randomized adaptive search procedures,” in *State of the Art Handbook in Metaheuristics*, .

[80] R. Landa and C. A. Coello, “A cultural algorithm for solving the job shop scheduling problem.,” *Knowledge Incorporation in Evolutionary Computation*, no. Springer, pp. 37–55, 2005.

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

[82] S. Binato, W. J. Hery, and D. M. Loewenstern, “A grasp for job shop scheduling,” no. March, pp. 1–17, 2000.

[83] B. Bean., A. Norman., and C. J., “A genetic algorithm methodology for complex scheduling problems.,” *Nav. Res. Logist*, vol. 46(2), pp. 199–211, 1999.

[84] Y. Ishida, “Fully distributed diagnosis by PDP learning algorithm: towards immune network PDP model,” in *IEEE International Joint Conference on Neural Networks*, 1990.

[85] Á. F. Casaní, “ARQUITECTURAS GRID orientadas a la gestión de recursos,” 2004.

[86] W. N. Chen and J. ZHANG, “Ant Colony Optimization Approach to Grid Workflow Scheduling Problem with Various QoS,” *IEEE Transactions on Systems, Man, and Cybernetics--Part C: Applications and Reviews*, vol. Vol. 31 No, pp. pp.29–43.

[87] 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,*, vol. 1(4), pp. 207–214,.

[88] R.-S. Chang, J.-S. Chang, and P.-S. Lin, “An ant algorithm for balanced job scheduling in grids,” *Future Generation Computer Systems*, vol. Volume 25, no. 1, pp. 20–27, 2009.

[89] L. Liu, Y. Yang, L. Li, and W. Shi, “Using Ant Colony Optimization for SuperScheduling in Computational Grid,” in *Services Computing, 2006. APSCC ’06. IEEE Asia-Pacific Conference on*, 2006, pp. 539 – 545.

[90] Y. Li, “A Bio-inspired Adaptive Job Scheduling Mechanism on a Computational Grid,” vol. 6, no. 3, pp. 1–7, 2006.

[91] K. R. Ku-Mahamud and H. J. A. Nasir, “Ant Colony Algorithm for Job Scheduling in Grid Computing,” *2010 Fourth Asia International Conference on Mathematical/Analytical Modelling and Computer Simulation*, pp. 40–45, 2010.

[92] L. N. De Castro, “Artificial Immune Systems : Theory and Applications,” vol. 2000, 2000.

[93] M. Ayara, J. Timmis, R. de Lemos, L. de Castro, and R. Duncan, “Negative selection:How to generate detectors,” *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)*, vol. 1, pp. 89–98, 2002.

[94] M. S. y J. P. M. Galiani, “Inmunología en línea.” [Online]. Available: http://www.inmunologiaenlinea.es/index.php?option=com_content&view=article&id=97:tolerancia-y-regulacion-sistema-inmune&catid=47:tolerancia&Itemid=126. [Último acceso: 20 Octubre 2012].

[95] I. Karonen, “Wikimedia Commons.” .

[96] N. K. Jerne, *Towards a Network Theory of the Immune System*. 1974, pp. 373–389.

[97] N. P. and A. P. J.D. Farmer, “The immune system, adaptation and machine learning,” *Physica D*, vol. 2, p. 187—204., 1986.

[98] J. Timmis and C. Edmonds, “A Comment on opt-AiNET : An Immune Network Algorithm for Optimisation 1 Introduction 2 Artificial Immune Systems in Optimisation.”

[99] P. Matzinger, “Tolerance, Danger, and the Extended Family,” *Annual Review of Immunology*, vol. 12, no. 1, pp. 991–1045, 1994.

[100] D. Dasgupta, “Advances in Artificial Immune Systems ©,” no. November 2006, pp. 40–49.

[101] E. Hart and J. Timmis, “Application Areas of AIS : The Past , The Present and The Future,” pp. 483–497, 2005.

[102] L. N. De Castro and J. Timmis, “Artificial Immune Systems : A Novel Paradigm to Pattern Recognition,” pp. 67–84, 2002.

[103] O. Nasraoui, F. Gonzalez, C. Cardona, and C. Rojas, “A Scalable Artificial Immune System Model for Dynamic Unsupervised Learning.”

[104] J. M. Goss, Simon and Aron, Serge and Deneubourg, Jean-Louis and Pasteels, “Self-organized shortcuts in the Argentine ant,” *Naturwissenschaften*, vol. 76, pp. 579–581, 1989.

[105] M. Perretto and S. Lopes, “Reconstruction of phylogenetic trees using the ant colony optimization paradigm.,” *Genetics and Molecular Research*, vol. 4(3), pp. 581–589, 2005.

ANEXO A

INSPIRACIÓN BIOLÓGICA SISTEMAS INMUNES ARTIFICIALES

El Sistema Inmune Natural de los vertebrados es un sistema de alta complejidad, robusto y adaptativo que se presenta como la barrera de protección frente patógenos extraños. Es capaz de clasificar todas las células (o moléculas) dentro del cuerpo como elementos propios o elementos no propios. Los vertebrados han desarrollado órganos específicos como la medula ósea y el timo que se encargan de la generación y posterior maduración de las células inmunológicas[92].

Existen dos tipos de inmunidad, la inmunidad innata y la inmunidad adaptativa

- El sistema inmune "*innato*", se denomina así debido a que todo individuo nace con la habilidad de reconocer una gran cantidad de agentes patógenos y destruirlos inmediatamente después que ingresan al organismo. La información y funcionamiento de éste tipo de inmunidad viene codificada genéticamente para cada especie[63].
- El sistema inmune "*adaptativo*", es un mecanismo de defensa que se va desarrollando a lo largo de la vida del individuo; es aquel capaz de generar, a partir de procesos fisicoquímicos aleatorios, células específicas capaces de neutralizar la presencia de antígenos particulares[63].

En la tabla a continuación se comparan las diferencias entre los dos tipos de inmunidades

Tabla 19 Comparación inmunidad innata vs inmunidad adaptativa

INMUNIDAD INNATA	INMUNIDAD ADAPTATIVA
Respuesta de antígeno independiente	Respuesta de antígeno dependiente
Respuesta inmediata	Respuesta retardada (Fase de latencia)

Antígeno no específico	Antígeno específico
No tiene memoria inmunológica	Memoria inmunológica

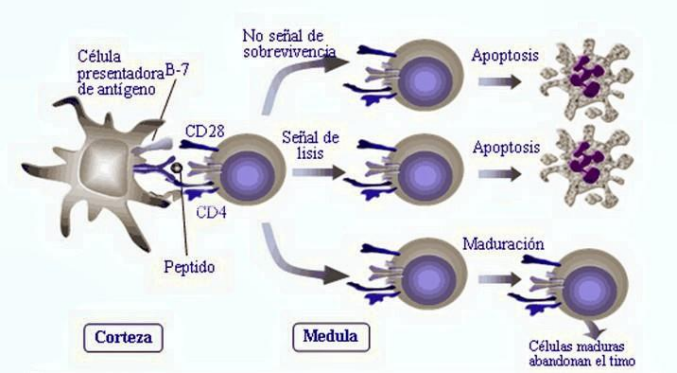
Gracias a este tipo de inmunidad se explica que un organismo durante el padecimiento de una enfermedad ocasionada por algún tipo de agente infeccioso, desarrolla defensas especializadas para combatir a los agentes patógenos, generando una especie de memoria inmunológica que perdura después de haberse curado y que lo protege generando una respuesta más efectiva en caso de enfrentarse nuevamente a el mismo tipo de antígeno o alguno que posea similar estructura.

MODELOS TEÓRICOS DEL SISTEMA INMUNOLÓGICO

Selección Negativa: Para que el sistema inmunológico funcione de manera apropiada, es necesario que éste posea la capacidad de diferenciar entre las moléculas que comúnmente se encuentran normalmente en el organismo (*propias*), y las moléculas pertenecientes a antígenos externos (*no propias*), que en últimas son las que constituyen una amenaza para el organismo[59].

Para que lo anterior se lleve a cabo de manera perfecta, existe un proceso importante al cual son sometidos los linfocitos T, en su etapa de maduración dentro del timo. Antes de ser liberados para que realicen su trabajo, los linfocitos son expuestos a manera de prueba a un repertorio de antígenos que el organismo tiene almacenados; a partir de las interacciones que se generan en esta prueba, se busca controlar la generación de células que se activen ante células propias del individuo y que no representan peligro. Si dichos linfocitos son liberados, podrían generar una respuesta de autoinmunidad ocasionando que estructuras del cuerpo sean reconocidas como agentes perjudiciales [93], en la figura siguiente se describe el proceso de selección negativa.

Figura 15 Representación Selección Negativa



Fuente: Galiani, Santamaría y Peña [94]

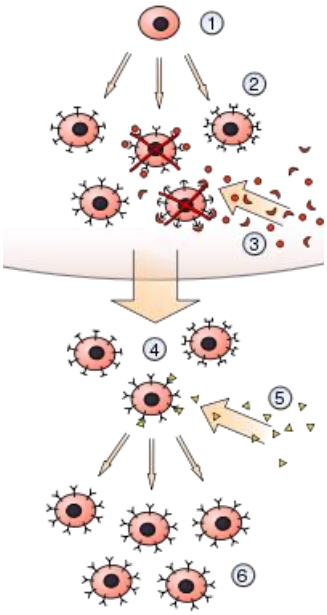
Selección Clonal: El principio de selección clonal es el proceso mediante el cual se describe las características básicas de una respuesta inmunológica ante un estímulo antigénico. Establece la premisa de que solo aquellas células capaces de reconocer la estructura molecular del antígeno, serán seleccionadas por encima de aquellas que no reaccionen ante la presencia del agente externo.

El proceso de selección clonal, se aplica sobre los dos distintos tipos de linfocitos, conocidos como células T y células B [71]. Cuando un organismo es expuesto a un antígeno extraño, la población de linfocitos responde produciendo anticuerpos para defender al individuo. Cada linfocito secreta un y solo un tipo de anticuerpos, haciendo que cada célula sea específica para contrarrestar a un tipo determinado de antígeno.

Aquellas células cuyos anticuerpos reconozcan y limiten la acción del antígeno, son estimuladas para su reproducción (clonación) y posterior evolución (diferenciación) a células más especializadas, como las células plasma encargadas de secretar anticuerpos en altas concentraciones, o las células de memoria cuya función es resguardar la información de la estructura molecular del anticuerpo desarrollado, en

caso que se presente una nueva amenaza del mismo antígeno. Cuando el organismo es expuesto ante un antígeno cuya estructura es igual o muy similar a otro previamente neutralizado, el sistema inmune recurre nuevamente a la selección clonal para desarrollar anticuerpos específicos para la nueva amenaza [67] en la siguiente figura se especifica de manera gráfica el anterior proceso.

Figura 16 Selección Clonal



Fuente: Karonen [95]

Debido a que existe un “registro” de la estructura de un antígeno similar, almacenado en un ataque previo, se espera que aun haya anticuerpos que tengan afinidad con el antígeno, pero no en el grado que se necesita para eliminarlo. Las células que produzcan este tipo de anticuerpos que reaccionen en cierto grado con el antígeno, son sometidas a mecanismos de hipermutación.

La mutación, es inversamente proporcional al grado de afinidad de los anticuerpos, siempre buscando que la población de células sea capaz de eliminar al antígeno;

aquellas células que presenten mayor afinidad presentarán una reorganización mínima en su estructura, hasta que sus anticuerpos reconozcan de manera óptima el antígeno, y ante éste se activen e inicien la respuesta inmunológica.

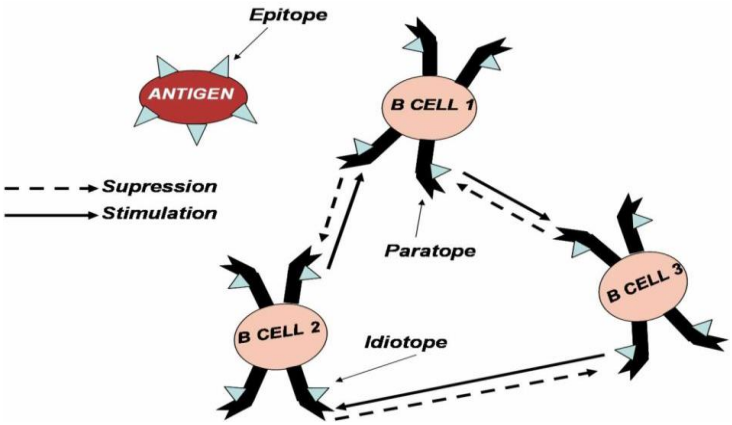
Paralelo al proceso de selección clonal, el sistema inmune realiza constantemente procesos de selección negativa previniendo que durante la ejecución de la mutación de las células, accidentalmente se generen linfocitos que produzcan anticuerpos "prohibidos" o autoinmunes.

Los procesos fisiológicos del sistema inmune pueden ser comparados con aquellos presentes en fenómenos como la evolución, en los cuales los organismos con mejores características para adaptarse al ambiente, son los que se mantienen y evolucionan para alcanzar el mejoramiento de las especies[63].

Redes Inmunes: Una de las características del sistema inmunológico que ha despertado gran interés es la capacidad de reaccionar de manera más rápida y con mayor efectividad en contra de antígenos con los cuales ya se ha enfrentado antes. Eso ha llevado a definir el concepto de memoria inmunológica, y muchas teorías se han planteado en torno a este concepto.

Una de las teorías más conocidas es la llamada Red inmune Idiotópica desarrolladas por Jerne[96] y Perelson[97]. Sus trabajos explican la memoria y la dinámica del sistema inmune a través de un mecanismo de realimentación. Proponen que el sistema inmunológico no solo es estimulado por antígenos externos, sino también por elementos propios del sistema inmunológico. Las células no solo se estimulan entre ellas sino que también desencadenan procesos de supresión, manteniendo así un control sobre la estimulación de las células para conservar una memoria estable. En la figura a continuación se observa un esquema del concepto de red de Jerne.

Figura 17 Modelo de red inmune de Jerne



Fuente: Jerne[96]

En el modelo de Jerne se encuentran los siguientes elementos[98]:

Células B: Éstas realizan el papel de nodos de la red, interactuando entre sí y con los antígenos; el grado de estimulación de las células depende de dos factores clave: de una estimulación, que puede ser interna o externa, y de una supresión:

1. La estimulación interna es producida por los mismos elementos de la red, esto se define como el proceso en el cual una célula B puede detectar a otra.

2. La estimulación externa en una célula B es generada cuando ésta detecta un antígeno.

3. La supresión en una célula B es producida cuando ésta es detectada por otra célula B.

Paratopos: son parte de las células B, y se encargan de la detección de los elementos alcanzables por la célula. Es el único medio que tiene la célula para incrementar su estimulación.

Idiotopos: estos son los elementos de la célula que son detectados por los paratopos de otras células. Cuando uno de ellos es detectado por alguna célula B, se produce una disminución de la estimulación en la célula a la cual pertenece. Esta disminución de la estimulación es también llamada supresión.

Epitopos: son elementos que están presentes en los antígenos. Cuando se dice que una célula B detecta un antígeno, realmente sucede que un paratopo de la célula detecta un epitopo del antígeno.

***Células dendríticas*:** Polly Matzinger [99] propuso la teoría del peligro en 1994, la cual ha ganado mucha popularidad entre los inmunologistas en años recientes como una explicación para el desarrollo de tolerancia periférica (tolerancia a los agentes que se encuentran fuera del huésped). La teoría del peligro establece que los APCs (células presentadoras de antígeno), se activan a sí misma a través de una alarma: señales de peligro. Las señales de peligro son emitidas por células ordinarias del cuerpo que han sido lesionadas debido a un ataque por parte de un patógeno. Por ejemplo, el contenido intracelular liberado debido a una muerte no controlada de una célula (necrosis) que podría proveer tal señal. Estas señales son detectadas por células innatas inmunes especializadas células dendríticas que parecen tener tres modos de operación: inmaduro, semi-maduro y maduro. Las células dendríticas son capaces de integrar ese tipo de señales para decidir si el ambiente es seguro o peligroso. Si es seguro la célula dendrítica se convierte en semi madura al presentar el antígeno a las células T. Si es peligroso las células dendríticas maduran y causan que las células T se vuelvan más reactivas en la presentación de antígenos.

CARACTERÍSTICAS DE SISTEMA INMUNE BIOLÓGICO

Entre las características encontradas en la literatura citada a continuación están[100][101]:

1) *La extracción de características* Hart, Timmis[101]: En el sistema inmune natural hay células que se encargan de detectar la presencia de cuerpos extraños. Esta característica se ha tratado de simular en su contraparte artificial en implementaciones como la detección de anomalías.

Un caso particular se muestra en el trabajo de Ayara, Timmis, Lemos y Forrest[93] donde buscan implementar un sistema de detección de errores en los cajeros automáticos. La idea es "Extraer los errores genéricos" que ocurren en los cajeros, simulando el proceso de inmunización a través de las vacunas donde el sistema inmune detecta los cuerpos extraños, extrae sus características y genera anticuerpos para los agentes patógenos.

2) *Debe exhibir Homeóstasis* Dasgupta, Niño, [63]: Esta característica se refiere a la capacidad del sistema inmune natural de autorregularse y auto organizarse. Cortes en su trabajo se refirió particularmente a la situación donde el sistema inmune natural produce anticuerpos suficientes para destruir a los antígenos. El sistema tiene que deshacerse de estos, ya que una vez que cumplieron con su propósito no le sirven para nada al sistema.

Pero no se eliminan por completo sino que se conservan unos algunos anticuerpos para un posible uso futuro. Es decir, en caso de un nuevo ataque provocado por el mismo antígeno, el sistema sabe cómo defenderse.

En su contraparte artificial, la redes neuronales artificiales buscan implementar esta función natural, mapeando las entradas de señales contra resultados particulares, generando respuestas puntuales de auto regulación como lo hace el cuerpo humano.

3) *Es adaptativo,* Hart y Timmis [101]: La dinámica de interacción entre los dos sistemas (innato y adaptativo) genera una respuesta contundente, ya que el sistema innato es un tipo de respuesta predeterminada inmediata del sistema inmune ante

cuerpos extraños, y si esta respuesta no elimina los patógenos, la interacción del innato con el adaptativo genera una nueva respuesta particularizada ante las necesidades que presenta el problema identificadas por el sistema adaptativo. De ahí se dio el desarrollo del algoritmo de células dendríticas.

4) *Memoria y reconocimiento de patrones,* De Castro y Timmis[102]: El sistema inmune reconoce y clasifica patrones y generando respuestas particulares a los ataques. Además, mantiene un registro de aquello que atacó al sistema para así clonar el anticuerpo respectivo cuando sea necesario, y responder efectivamente si se genera un nuevo ataque.

De Castro y Timmis [102] muestran entre varios algoritmos, cómo se replicó esta característica para el sistema inmune artificial con el algoritmo de selección negativa. El sistema compara las señales con sus registros (P) con una serie de candidatos generados al azar (C) y si son identificados los descarta; crea un detector (M) y lo registra. Esto genera un patrón de reconocimiento y una memoria para monitorear el sistema en búsqueda de patrones extraños.

5) *Distribución y diversidad,* Cortes [12] : El sistema inmune puede verse como un sistema distribuido donde las células propias circulan por todo el cuerpo y cuando identifican la presencia de algún cuerpo extraño avisan al resto del sistema.

En su contraparte artificial los autores implementaron unas mascaras de permutación dentro del algoritmo de selección negativa que monitorean señales aleatorias en búsqueda de patrones extraños. Aunque aclaran que tiene limitaciones. Pues hay posibilidades de “huecos” donde no se filtren apropiadamente los cuerpos extraños.

6) *Aprendizaje,* Nasraoui, González, Cardona, Rojas, Dasgupta[103] : La capacidad de aprendizaje permanente del sistema inmune natural le permite sobrevivir ante ambientes que cambian, evolucionan, y presentan continuos riesgo para el sistema.

Hay muchos casos de esfuerzos por implementar esta característica pero una buena ilustración que estos autores ofrecen muestra la limitación que tiene su contraparte artificial para conocer y analizar todos clusters en un conjunto de datos dinámicos (que cambian en el tiempo).

Por ejemplo uno de los modelos que usan para mostrar este punto, es el modelo dinámico artificial de células-B basado en pesos robustos. Debido a la cantidad de datos que se manejan, que tienen cambios constantes en el tiempo, solo se pueden analizar con robustez clusters medianos y pequeños.

Este sistema de procesamiento de la información biológica notable ha llamado la atención de la informática en los últimos años, por lo que una nueva técnica de inteligencia computacional ha surgido inspirada en la inmunología, y se conoce como sistemas inmunes artificiales. Varios conceptos de la inmunología han sido extraídos y aplicados a la solución real del mundo de la ciencia y la ingeniería.

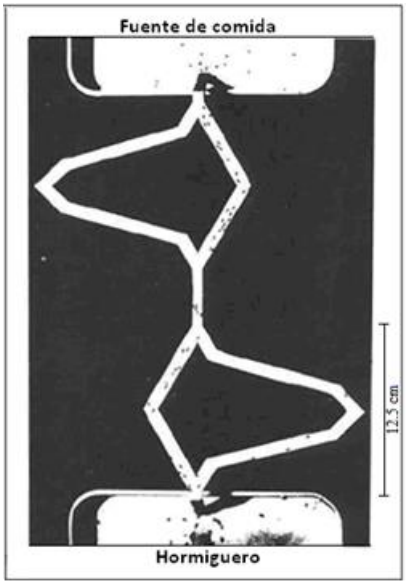
ANEXO B

INSPIRACIÓN BIOLÓGICA COLONIA DE HORMIGAS

Las hormigas o formícidos son insectos sociales del orden de los himenópteros que viven en colonias y que, debido a la interacción y comunicación de sus individuos, son capaces de mostrar comportamientos complejos y realizar tareas de gran dificultad desde la óptica un solo individuo. Una característica relevante que tienen muchas especies de hormigas es su habilidad para encontrar los caminos más cortos entre su hormiguero y las fuentes de alimento. Este hecho es especialmente interesante si se tiene en cuenta que gran parte de las especies de hormigas no cuentan con el sentido de la visión de manera desarrollada, lo que evita el uso de pistas visuales o referencias en el entorno. En el recorrido entre el hormiguero y la fuente de alimento, gran parte de las especies de hormigas para poder comunicarse, depositan una sustancia química denominada feromona. Si no se encuentra ningún rastro de feromona, las hormigas se mueven de manera básicamente aleatoria, pero cuando existe feromona depositada, tienen mayor tendencia a seguir el rastro.

En un experimento sobre la auto-organización de las hormigas Argentinas realizado en 1989 [104], se observó el comportamiento de alimentación de una colonia de hormigas, que lograron encontrar las ramas más cortas de un puente entre el hormiguero y la comida (Fig. 12), por medio del rastro de feromona que van dejando al moverse.

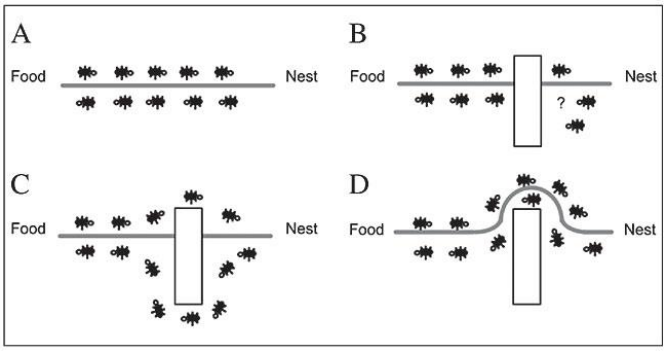
Figura 18 Fotografía de una colonia de hormigas que muestra su inteligencia de enjambre para encontrar el camino más corto a la comida (en solo 8 minutos).



Fuente: J. M. Goss, Simon and Aron, Serge and Deneubourg, Jean-Louis and Pasteels [104]

Las hormigas se mueven inicialmente de forma aleatoria en búsqueda de alimento y a lo largo de su camino de regreso a la colonia depositan la feromona. Si otra hormiga encuentra este rastro, lo más probable es que siga este camino, aumentando la cantidad de feromona, lo que estimula aún más a otras hormigas a seguir esa trayectoria (Fig. 13), y a disminuir el flujo de otras hormigas a trayectorias donde la cantidad de feromona es menor. Con el paso del tiempo rastro de la feromona comienza a evaporarse y se reduce su fuerza atractiva, haciendo que solo las trayectorias más usadas sean las más atractivas, lo que provoca la convergencia a una solución óptima que será la única trayectoria que finalmente siguen la mayoría de hormigas. Por el camino largo se acumula menos feromona porque la frecuencia con la que pasan las hormigas será menor, al tardar más tiempo para completar un recorrido, en la siguiente grafica se especifica este proceso.

Figura 19 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

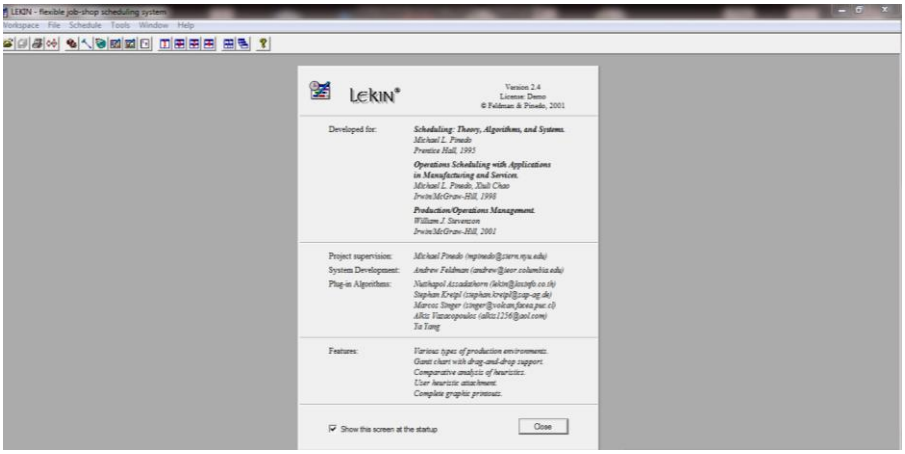


Fuente: Perretto y Lopes[105]**.**

ANEXO C

LEKIN

Lekin ® fue creado como una herramienta educativa, con el objetivo principal de la introducción de los estudiantes a la teoría de la planificación y sus aplicaciones. Además de eso, del sistema permite para utilizarlo en el desarrollo de algoritmos y validar los nuevos desarrollos. El proyecto ha sido dirigido por el profesor Michael L. Pinedo, profesor Xiuli Chao y el profesor Joseph Leung. Este desarrollo ha sido parcialmente financiado por la National Science Foundation.



Interfaz del software Lekin, representación de un plan de la instancia de 5X10