Uso de un Sistema Inmune Artificial para Problemas de Calendarización

Daniel Cortés Rivera y Carlos A. Coello Coello

Resumen— **En este trabajo se propone un algoritmo basado en un sistema inmune artificial para resolver problemas de calendarización de tareas de los denominados *Job Shop Scheduling*. La propuesta se basa en el uso de selección clonal, hipermutación y bibliotecas de anticuerpos para construir las soluciones. También se utiliza un mecanismo de búsqueda local que trata de eliminar huecos entre las tareas a fin de mejorar la solución obtenida por el mecanismo de búsqueda del algoritmo. El algoritmo propuesto es comparado contra 2 versiones del Procedimiento de Búsqueda Ciega Aleatorizado y Adaptativo (GRASP por sus siglas en inglés) en algunos problemas tomados de la literatura especializada. Nuestros resultados indican que el algoritmo propuesto es altamente competitivo con respecto a los 2 algoritmos GRASP contra los que se le compara, a pesar de realizar un número considerablemente menor de evaluaciones de la función objetivo.**

Palabras clave— **sistema inmune artificial, job shop scheduling, calendarización**

I. Introducción

LOS problemas de calendarización se encuentran en todas la áreas. El propósito de la calendarización es asignar un conjunto de recursos (limitados) a determinadas tareas en un cierto periodo de tiempo [1]. La calendarización ha sido un área muy activa de investigación durante varios años, tanto en investigación operativa como en informática [2], [3]. Los problemas de calendarización de nuestro interés se centran fundamentalmente en encontrar formas de asignar tareas (u operaciones) a un conjunto de máquinas, satisfaciendo ciertas restricciones y minimizando un cierto coste (p.ej., el tiempo total requerido para completar las tareas).

Se han utilizado diversas heurísticas para resolver una amplia gama de problemas de calendarización (p.ej., job shop, flow shop, open shop, etc.): algoritmos evolutivos [4], [5], búsqueda tabú [6] y recocido simulado [7] entre otras. Nótese, sin embargo, que el uso de sistemas inmunes artificiales para la solución de problemas de calendarización de cualquier tipo ha sido escaso en la literatura (ver por ejemplo [8], [9], [10]).

CINVESTAV-IPN, Sección de Computación, Grupo de Computación Evolutiva, Av. IPN No. 2508, Col. San Pedro Zacatenco, México, D.F. 07300. email: dcortes@computacion.cs.cinvestav.mx, ccoello@cs.cinvestav.mx

Este trabajo introduce una nueva propuesta, basada en un sistema inmune artificial y el uso de una biblioteca de anticuerpos, para resolver problemas de los denominados de *Job Shop Scheduling*. El algoritmo propuesto es comparado con respecto a 2 versiones de GRASP (*Greedy Randomized Adaptive Search Procedure*) en varios problemas de prueba tomados de la literatura especializada. Nuestros resultados indican que el algoritmo propuesto es una alternativa viable para resolver problemas de Job Shop Scheduling.

II. Planteamiento del Problema

En este artículo, abordamos el problema denominado *Job Shop Scheduling* (JSSP), en el cual el objetivo general es minimizar el tiempo que toma finalizar la última tarea (u operación) disponible (makespan) de entre un conjunto a realizarse. En otras palabras, el objetivo es encontrar un plan de trabajo que tenga la mínima duración requerida para completar todos los trabajos requeridos [2].

Más formalmente, podemos decir que en el JSSP, tenemos un conjunto de n trabajos $\{J_i\}_{1\leq j\leq n}$, que tienen que ser procesados por un conjunto de m máquinas $\{M_i\}_{1\leq r\leq m}$. Cada trabajo tiene una secuencia que depende de las restricciones de precendencia para procesar cada uno de éstos. El procesamiento de un trabajo J_j en una máquina M_r es llamado operación O_jr. La operación O_jr requiere el uso exclusivo de M_r por un periodo ininterrumpido de tiempo p_jr (éste es el tiempo de procesamiento). Un plan de trabajo consta del conjunto de tiempos de procesamiento de cada operación $\{c_jr\}_{1\leq j\leq n,1\leq r\leq m}$ que satisface las condiciones previamente especificadas. La duración total de tiempo requerido para completar todos los trabajos (makespan) puede ser denominada L y el objetivo es minimizar precisamente este valor.

Garey y Johnson [11] demostraron que el JSSP es un problema **NP-duro**. Se han aplicado diversos algoritmos enumerativos basados en los métodos de ramificación y acotamiento (*Branch & Bound*) al JSSP. Hay, sin embargo, muchas otras propuestas algorítmicas. Una de las más populares de la actualidad se basa en el uso de reglas de prioridad (*priority rules*) y en la generación ac-

tiva de planes de trabajo (*active schedule generation*) [12]. Adicionalmente, otros algoritmos han resultado también muy efectivos (ver por ejemplo [13]). Es importante destacar también que se ha utilizado un número importante de heurísticas para resolver el JSSP, de entre las que destacan los algoritmos genéticos, la búsqueda tabú y el recocido simulado.

Los autores sólo conocen de otro intento por usar un sistema inmune artificial para resolver el JSSP. Se trata de la propuesta de Hart & Ross [9], [8]. En este caso, los autores utilizan un sistema inmune artificial en el cual un anticuerpo representa indirectamente un plan de trabajo, y un antígeno describe un conjunto de tiempos de llegada para cada tarea en el conjunto de recursos disponibles. En esta propuesta, los planes de trabajo se consideran dinámicos en el sentido de que los cambios repentinos en el ambiente requieren la generación de nuevos planes de trabajo. La propuesta de Hart & Ross presenta un desempeño aceptable (de acuerdo a sus autores) con respecto a un algoritmo genético, al usar problemas tomados de [14]. Sin embargo, los autores no proporcionan información suficiente para reproducir sus resultados (los problemas y resultados obtenidos no se incluyen en sus trabajos).

III. Descripción de nuestra propuesta

Nuestra propuesta se basa en dos mecanismos fundamentales del sistema inmune:

1. La forma en que son creadas las moléculas llamadas anticuerpos. Un anticuerpo es codificado en múltiples segmentos de genes distribuidos a lo largo de un cromosoma del genoma. Estos segmentos deben colocarse juntos para formar el anticuerpo. Para crear tal molécula (p.ej. un anticuerpo), los segmentos de gene deben concatenarse (vea la figura 1). Es importante hacer notar que otros autores han usado esta codificación de anticuerpos en sus propios modelos computacionales (p.ej., [15], [16], [17]).
2. El principio de selección clonal.

El enfoque propuesto en este trabajo puede verse como una variante de CLONALG, que es un sistema inmune artificial basado en el principio de selección clonal el cual ha resultado exitoso en diversas tareas de optimización [18]. CLONALG usa dos poblaciones: una de antígenos y otra de anticuerpos. Cuando se usa para optimización, la idea principal de CLONALG es reproducir individuos con una alta afinidad, aplicando mutación (o una variación) y seleccionar el descendiente mejorado. Nótese que "afinidad" se define en este caso, en términos de un mejor valor de la función objetivo en vez de hacerlo de términos de sus similitudes genotípicas (como se hace, por ejemplo, en las tareas de reconocimiento de patrones). Destaca también el hecho de que en tareas de optimización, el número de clones es el mismo para cada anticuerpo. Esto implica que CLONALG realmente no utiliza antígenos cuando se aplica a problemas de optimización, sino que más bien adopta la proximidad de cada anticuerpo al óptimo global (medido en términos relativos con respecto al conjunto de soluciones producidas) para definir la tasa de hipermutación a aplicarse. También es importante mencionar que CLONALG no usa bibliotecas para contruir anticuerpos como en el caso de nuestra propuesta.

trabajo	máquina (tiempo)			
1	1(2)	2(2)	3(2)	4(2)
2	4(2)	3(2)	2(2)	1(2)
3	2(2)	1(2)	4(2)	3(2)
4	3(2)	4(2)	1(2)	2(2)
5	1(2)	2(2)	3(4)	4(1)
6	4(3)	2(3)	1(1)	3(1)

TABLA I
Un problema de tamaño 6×4

Para poder aplicar un sistema inmune artificial (o cualquier otra heurística) al JSSP, es necesario utilizar una representación adecuada. En nuestro caso, cada elemento de la biblioteca representa la secuencia de trabajos a ser procesada por las máquinas. Un anticuerpo es una secuencia de trabajos a ser procesados por cada una de las máquinas (de longitud $m \times n$). Un antígeno es representado de la misma manera que un anticuerpo. La representación adoptada en este trabajo es llamada *permutación con repeticiones* y fue propuesta originalmente en [19].

Para ilustrar esta representación, podemos considerar el problema de 6×4 (6 trabajos y 4 máquinas) mostrado en la Tabla I.

Entre los datos de entrada se incluye información respecto a la máquina en que cada trabajo debe ser procesado junto con la duración de dicho trabajo en cada máquina. Los diagramas de Gantt son una muy buena herramienta para visualizar la solución obtenida al JSSP. Un ejemplo de un diagrama de Gantt que representa la solución obtenida al problema de 6×4 previamente indicado, se muestra en el paso 1 de la Figura 2. La figura 2 requiere de algunas explicaciones adicionales:

- La cadena en la parte inferior de la figura 2 corresponde a la solución que procederemos a decodificar.
- **Paso 1**: Muestra la decodificación antes de al-

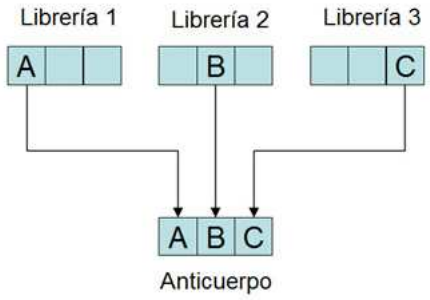

Fig. 1. Construcción de un anticuerpo a partir de las bibliotecas de genes.

canzar la segunda operación del trabajo 2.

- **Paso 2**: Muestra la forma en que la operación del trabajo 2 podría ser colocada en el plan de trabajo, en caso de que se utilizara una decodificación normal. Nótese que el trabajo 2 (J_2) se muestra en el extremo derecho de la máquina 3 (M_3).
- **Paso 3**: Nuestro algoritmo realiza una búsqueda local tratando de encontrar huecos en el plan de trabajo actual. Tales huecos deben cumplir con las restricciones de precedencia impuestas por el problema. En este caso, la figura muestra la operación del trabajo 2 colocada en uno de estos huecos de la máquina 3.
- **Paso 4**: En este caso, se aplica el proceso de búsqueda local (búsqueda de huecos disponibles) a otras máquinas. Este paso muestra la solución óptima para el problema en cuestión.

Nuestra propuesta extiende el algoritmo (basado en la teoría de selección clonal) propuesto en [20]. Entre las extensiones más importantes se encuentra el uso de un mecanismo de búsqueda local que se encarga de acomodar las operaciones de los trabajos usando los huecos de tiempo disponibles. Obviamente, este mecanismo cuida que no se viole ninguna de las restricciones del problema.

Nuestra propuesta se muestra en el algoritmo 1. Primero, generamos aleatoriamente una biblioteca de anticuerpos. Tal biblioteca consiste en un conjunto de cadenas que codifican diferentes secuencias de trabajos. Después generamos aleatoriamente un antígeno, que es una posible solución al problema. Luego entramos al ciclo principal del algoritmo que se cumplirá hasta alcanzar la condición de paro, la cual es que se haya realizado el número de iteraciones especificado en los parámetros de entrada. Ya dentro del ciclo lo primero que hacemos es generar un anticuerpo a partir de segmentos de la biblioteca de anticuerpos (esto se hace tal y como se describe en

Algoritmo 1 Nuestro sistema inmune artificial para resolver el *job shop scheduling*

Require: Fichero de entrada (en el formato adoptado en [21]).
Parámetros de entrada: #antigenos, #bibliotecas, porcentaje de mutación, semilla de aleatorios (opcional)
p - número de iteraciones
i - contador
Generar (aleatoriamente) la biblioteca de *anticuerpos.*
Generar (aleatoriamente) un *antígeno* y decodificarlo.
repeat
  Generar un *anticuerpo* con segmentos de la biblioteca de *anticuerpo*
  Generar los clones del *anticuerpo*
  Aplicar mutación a cada uno de los clones del *anticuerpo*
  for all Clones del anticuerpo **do**
    Decodificar al *anticuerpo* y aplicar búsqueda local para mejorar cada solución
    if Es mejor el *anticuerpo* que el *antígeno* **then**
      Cambiamos el *antígeno* por el *anticuerpo*, ahora el *anticuerpo* es el *antígeno*
    end if
    Actualizamos componentes de la biblioteca
  end for
until $i > p$
Reportamos la mejor solución encontrada, al *antígeno*

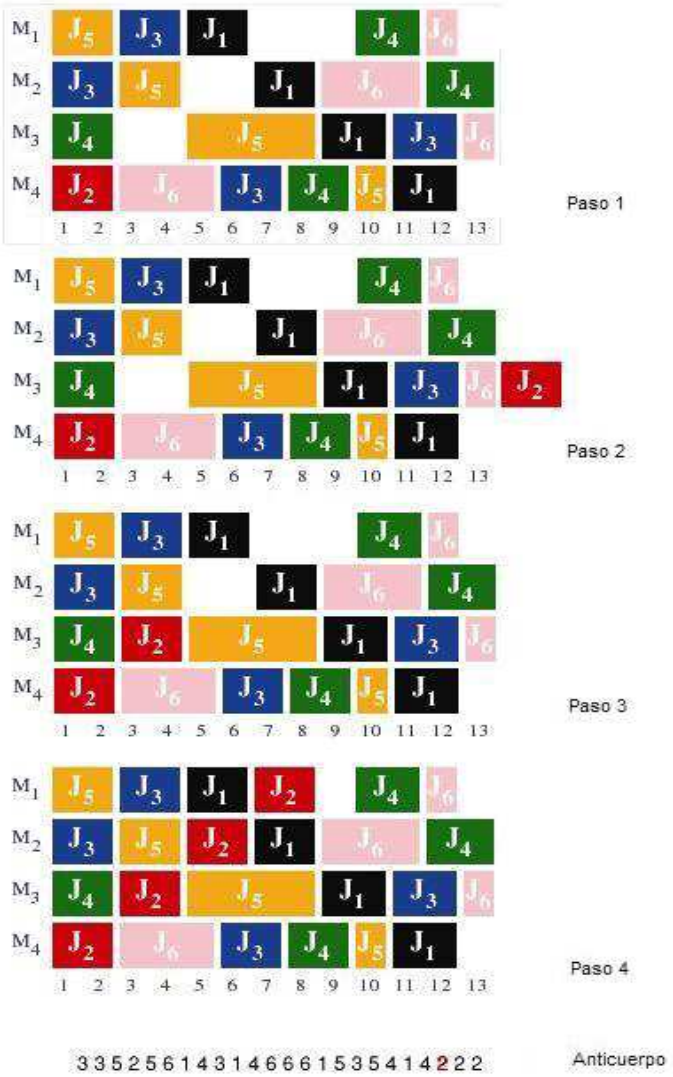

Fig. 2. Representación gráfica de una solución al problema de 6×4 mostrado en la Tabla I usando un diagrama de Gantt. La cadena en la parte inferior de la imagen indica el anticuerpo que está siendo decodificado. En el texto se proporciona una explicación por cada uno de los pasos.

la sección anterior). Posteriormente aplicamos la clonación para el anticuerpo y luego aplicamos mutación a cada uno de los clones del anticuerpo. Después, cada uno de los clones mutados se compara contra el antígeno y si su valor de la función objetivo es mejor que éste (es decir, si el *makespan* del anticuerpo es mejor que el del antígeno), se convierte al anticuerpo en antígeno y pasará a ser entonces objeto de las mejoras producidas por el algoritmo. Luego viene el proceso de actualización de las bibliotecas. Esto implica que para cada una de las secuencias del anticuerpo en cada una de las máquinas, ésta debe compararse contra un segmento de la biblioteca elegido aleatoriamente. Si la secuencia del anticuerpo resulta mejor que la de la biblioteca, se realiza la actualización correspondiente a la biblioteca. Cabe mencionar que cuando se selecciona aleatoriamente un segmento, éste debe ser de la misma máquina. Este ciclo se vuelve a repetir hasta que se cumpla la condición de paro y una vez que se cumple ésta, reportamos la mejor solución encontrada. La mejor solución encontrada es el antígeno, que es quien codifica dicha solución.

Es importante mencionar que el número de clones oscila entre 10 y 100 para cada uno de los problemas, dependiendo del tamaño del problema a optimizar. Nótese que no se utiliza ninguna medida de afinidad. Esto es debido principalmente a la representación adoptada, la cual permite repeticiones. Dicha representación dificulta definir una medida de similitud entre dos secuen-

cias de trabajos y por lo tanto nuestra decisión de usar como afinidad los valores de la función objetivo (makespan).

IV. Comparación de Resultados

Para validar nuestra propuesta, usamos un conjunto de problemas que consta de 66 instancias de 5 diferentes clases de problemas del JSSP, los cuales se tomaron de la *OR-Library* [21]. Nuestro algoritmo fue comparado contra 2 versiones de GRASP (Greedy Randomized Adaptive Search Procedure) [22], [23]. GRASP en una metaheurística para encontrar soluciones de buena calidad, aunque no necesariamente óptimas, a problemas de optimización combinatoria. Se basa en la premisa de que soluciones iniciales diversas y de buena calidad juegan un papel importante en el éxito de métodos locales de búsqueda. GRASP en un método multi-arranque en el cual cada iteración consiste en la construcción de una solución ciega aleatorizada seguida de una búsqueda local usando la solución construida como el punto inicial de la búsqueda local. Este procedimiento se repite varias veces y la mejor solución encontrada sobre las iteraciones GRASP se devuelve como la solución aproximada al problema [24]. Todas nuestras pruebas fueron realizadas en un PC con un procesador Intel Pentium 4 a 2.6GHZ con 512MB de RAM usando Windows XP Professional, y el compilador Visual C++ .NET de Microsoft.

Los parámetros de nuestra propuesta son los siguientes:

- **Número de Bibliotecas**: Adoptamos entre 4 y 8. Cabe hacer notar que cada biblioteca es de la misma longitud que un anticuerpo y un antígeno.
- **Número de Antígenos**: Usamos solamente un antígeno en los experimentos reportados en este trabajo. Sin embargo, experimentamos con otros valores (hasta 8) pero no encontramos ninguna diferencia significativa en las pruebas realizadas.
- **Porcentaje de Mutación**: Este valor está en función de la longitud del anticuerpo y se define de tal forma que se lleven a cabo 2 mutaciones por cada anticuerpo. Usamos mutación por intercambio, pero tomando en cuenta que el intercambio que debe hacerse es con respecto a una operación que esté en la misma máquina, lo cual implica verificar las posiciones que se pueden cambiar y luego reordenar las operaciones para que sólo se afecte dicha operación. Es importante tomar en cuenta que no se debe modificar el orden de las demás operaciones en las demás máquinas. Para la mutación usamos una función $flip(P_m)$ (la función $flip(P_m)$ regresa CIERTO un P_m porcentaje de las veces que es invocada) para cada posición a lo largo de la cadena del anticuerpo y si el resultado devuelto es CIERTO aplicamos el intercambio de posición de la operación; la posición seleccionada es aleatoria.
- **Número de Clones**: Adoptamos valores entre 10 y 100, dependiendo de la complejidad del problema (este valor generalmente depende del tamaño del problema, 10 para problemas de tamaño a lo más de 10×10 y 100 para los restantes).

La comparación contra GRASP en sus 2 versiones se muestra en la Tabla II. Se reporta la siguiente información:

- **Problema:** Nombre del problema (tal y como se le denomina en la *OR-Library* [21]).
- **m:** Número de máquinas del problema.
- **n:** Número de trabajos del problema.
- **BKS:** Se refiere a la mejor solución conocida para cada problema (por sus siglas en inglés, *Best known solution*).
- **AIS:** Mejores soluciones obtenidas con nuestro Sistema Inmune Artificial (se realizaron 10 ejecuciones independientes por problema con el número de evaluaciones indicado de la función objetivo).
- **Evaluaciones:** El número de evaluaciones de la función objetivo realizadas por ejecución independiente de nuestro algoritmo.
- **Desviación:** Es el porcentaje que se desvía el resultado obtenido con respecto a la mejor solución conocida para cada problema. Este cálculo se realiza para los 3 algoritmos comparados.
- **GRASP:** Es el valor obtenido por la versión de GRASP reportada en [22].
- **Iteraciones:** Es el número de iteraciones de GRASP realizadas para obtener los resultados presentados.
- **GRASP + RT:** Es el valor obtenido por la versión de GRASP que utiliza "reenlace de trayectoria" (RT) [23].
- **Iteraciones:** Es el número de iteraciones de GRASP + RT realizadas para obtener los resultados presentados.

A. Discusión de Resultados

El primer aspecto importante que hay que discutir, es el costo computacional, ya que nosotros presentamos el número de evaluaciones que se realiza a la función objetivo, mientras que de las 2 versiones de GRASP usadas como referencia, sólo se muestra el número de iteraciones que se realizan para encontrar la solución. Es importante observar que en la mayoría de los problemas, nuestro AIS realiza un número considerablemente menor de evaluaciones que cualquiera de las 2 versiones de GRASP contra las que se

Problema	m	n	BKS	IAS	Evaluaciones	Desviación	GRASP	Iteraciones	Desviación	GRASP + RT	Iteraciones	Desviación
ft06	6	6	**55**	**55**	0.0001	0.00%	**55**	0.1	0.00%	**55**	0.00001	0.00%
ft10	10	10	**930**	936	0.25	0.65%	938	90.1	0.86%	**930**	2.5	0.00%
ft20	20	5	**1165**	1180	0.25	1.29%	1169	90.1	0.34%	**1165**	4.5	0.00%
abz5	10	10	**1234**	1239	0.25	0.41%	1238	20.1	0.32%	**1234**	1	0.00%
abz6	10	10	**943**	945	0.25	0.21%	947	20.1	0.42%	**943**	0.3	0.00%
abz7	20	15	**665**	696	0.25	4.66%	723	20.1	8.72%	692	50	4.06%
abz8	20	15	**670**	716	0.25	6.87%	729	20.1	8.81%	705	10	5.22%
abz9	20	15	**691**	735	0.25	6.37%	758	20.1	9.70%	740	1	7.09%
orb01	10	10	**1059**	1076	0.5	1.61%	1070	40.1	1.04%	**1059**	1.2	0.00%
orb02	10	10	**888**	889	0.25	0.11%	889	40.1	0.11%	**888**	1.1	0.00%
orb03	10	10	**1005**	**1005**	0.25	0.00%	1021	40.1	1.59%	**1005**	6.5	0.00%
orb04	10	10	**1005**	1023	0.25	1.79%	1031	40.1	2.59%	1011	100	0.60%
orb05	10	10	**887**	896	0.25	1.01%	891	40.1	0.45%	889	20	0.23%
orb06	10	10	**1010**	1013	0.25	0.30%	1013	40.1	0.30%	1012	3.5	0.20%
orb07	10	10	**397**	399	0.25	0.50%	**397**	10.1	0.00%	**397**	0.03	0.00%
orb08	10	10	**899**	**899**	0.25	0.00%	909	40.1	1.11%	**899**	1.6	0.00%
orb09	10	10	**934**	**934**	0.25	0.00%	945	40.1	1.18%	**934**	11.1	0.00%
orb10	10	10	**944**	952	0.25	0.85%	953	40.1	0.95%	**944**	0.3	0.00%
car1	11	5	**7038**	**7038**	0.05	0.00%	**7038**	0.1	0.00%	**7038**	0.001	0.00%
car2	13	4	**7166**	**7166**	0.05	0.00%	**7166**	0.1	0.00%	**7166**	0.001	0.00%
car3	12	5	**7312**	**7312**	0.05	0.00%	7366	240.1	0.74%	**7312**	50.7	0.00%
car4	14	4	**8003**	**8003**	0.05	0.00%	**8003**	0.1	0.00%	**8003**	0.01	0.00%
car5	10	6	**7702**	7720	0.05	0.23%	**7702**	20.1	0.00%	**7702**	0.5	0.00%
car6	8	9	**8313**	**8313**	0.05	0.00%	**8313**	10.1	0.00%	**8313**	0.01	0.00%
car7	7	7	**6558**	**6558**	0.05	0.00%	**6558**	0.1	0.00%	**6558**	0.001	0.00%
car8	7	7	**8264**	**8264**	0.05	0.00%	**8264**	0.1	0.00%	**8264**	0.02	0.00%

TABLA II

RESULTADOS EXPERIMENTALES DE LOS PROBLEMAS PERTENECIENTES A LAS CLASES *ft*, *abz*, *orb* Y *car*. LOS RESULTADOS OBTENIDOS POR NUESTRO AIS, SE COMPARAN CONTRA GRASP Y GRASP + RT.

comparan resultados. Otra ventaja que podemos observar de nuestro AIS sobre cualquiera de las 2 versiones de GRASP contra las que se le comparó, es que en nuestro caso se utilizó una representación (permutaciones con repetición) tal que siempre se generan soluciones válidas. Esto no ocurre con los algoritmos de GRASP analizados, los cuales tienen una etapa en la cual construyen una solución válida.

Es importante mencionar que en el caso de nuestro algoritmo, se están reportando evaluaciones de la función objetivo, mientras que en los 2 trabajos con GRASP usados como referencia se reportan iteraciones de GRASP, cada una de las cuales tiene un costo computacional más elevado que cada iteración de nuestro sistema inmune artificial.

En la Tabla IV se muestra la desviación de nuestro sistema inmune artificial (AIS) y los 2 GRASP evaluados con respecto a la mejor solución conocida para cada una de las clases de problemas bajo estudio. En esta tabla se observa que nuestro AIS le gana a GRASP, pero con respecto a la versión híbrida de GRASP (en la cual utilizan Reenlace de Trayectoria), nuestro enfoque tiene un desempeño ligeramente más pobre. Esto sucede debido a que la versión híbrida

Familia	AIS	GRASP	GRASP + RT
la	0.85%	1.97%	0.49%
ft	0.98%	0.56%	0.00%
abz	3.05%	4.57%	2.64%
orb	0.64%	1.01%	0.11%
car	0.03%	0.09%	0.00%

TABLA IV

PORCENTAJE DE DESVIACIÓN CON RESPECTO A LA MEJOR SOLUCIÓN CONOCIDA (BKS).

de GRASP dota a este algoritmo de memoria, lo cual lo hace más poderoso. Adicionalmente, es importante destacar que esta segunda versión de GRASP se implementó en paralelo y que los resultados reportados para la misma hacen uso de dicho procesamiento paralelo. En contraste, nuestro algoritmo sólo ha sido implementado de manera secuencial.

En la Tabla V presentamos la comparación de resultados finales. En dicha tabla se indica en cuántos problemas obtuvimos mejores, iguales o peores resultados con nuestro AIS, al comparar éstos con respecto a las 2 versiones de GRASP usadas como referencia. Puede verse que nuestro AIS le gana a GRASP en 26 casos, pero sólo

Problema	m	n	BKS	IAS	Evaluaciones	Desviación	GRASP	Iteraciones	Desviación	GRASP + RT	Iteraciones	Desviación
la01	10	5	666	666	0.001	0.00%	666	0.1	0.00%	666	0.0001	0.00%
la02	10	5	655	655	0.01	0.00%	655	0.1	0.00%	655	0.004	0.00%
la03	10	5	597	597	0.01	0.00%	604	50.1	1.17%	597	0.01	0.00%
la04	10	5	590	590	0.001	0.00%	590	0.1	0.00%	590	0.001	0.00%
la05	10	5	593	593	0.001	0.00%	593	0.1	0.00%	593	0.0001	0.00%
la06	15	5	926	926	0.001	0.00%	926	0.1	0.00%	926	0.0001	0.00%
la07	15	5	890	890	0.001	0.00%	890	0.1	0.00%	890	0.0001	0.00%
la08	15	5	863	863	0.001	0.00%	863	0.1	0.00%	863	0.0003	0.00%
la09	15	5	951	951	0.001	0.00%	951	0.1	0.00%	951	0.0001	0.00%
la10	15	5	958	958	0.001	0.00%	958	0.1	0.00%	958	0.0001	0.00%
la11	20	5	1222	1222	0.001	0.00%	1222	0.1	0.00%	1222	0.0001	0.00%
la12	20	5	1039	1039	0.001	0.00%	1039	0.1	0.00%	1039	0.0001	0.00%
la13	20	5	1150	1150	0.001	0.00%	1150	0.1	0.00%	1150	0.0001	0.00%
la14	20	5	1292	1292	0.001	0.00%	1292	0.1	0.00%	1292	0.0001	0.00%
la15	20	5	1207	1207	0.001	0.00%	1207	0.1	0.00%	1207	0.0002	0.00%
la16	10	10	945	946	0.01	0.11%	946	50.1	0.11%	945	1.3	0.00%
la17	10	10	784	784	0.01	0.00%	784	20.1	0.00%	784	0.02	0.00%
la18	10	10	848	848	0.01	0.00%	848	20.1	0.00%	848	0.05	0.00%
la19	10	10	842	842	0.01	0.00%	842	10.1	0.00%	842	0.02	0.00%
la20	10	10	902	907	0.25	0.55%	907	50.1	0.55%	902	17	0.00%
la21	15	10	1046	1056	0.25	0.96%	1091	50.1	4.30%	1057	100	1.05%
la22	15	10	927	962	0.25	3.78%	960	50.1	3.56%	927	26	0.00%
la23	15	10	1032	1032	0.25	0.00%	1032	10.1	0.00%	1032	0.01	0.00%
la24	15	10	935	958	0.25	2.46%	978	10.1	4.60%	954	125	2.03%
la25	15	10	977	1002	0.25	2.56%	1028	10.1	5.22%	984	32	0.72%
la26	20	10	1218	1218	0.2	0.00%	1271	10.1	4.35%	1218	3.5	0.00%
la27	20	10	1235	1281	0.5	3.72%	1320	10.1	6.88%	1269	10.5	2.75%
la28	20	10	1216	1256	0.25	3.29%	1293	10.1	6.33%	1225	20	0.74%
la29	20	10	1157	1203	0.25	3.98%	1293	10.1	11.75%	1203	50	3.98%
la30	20	10	1355	1355	0.1	0.00%	1368	10.1	0.96%	1355	3	0.00%
la31	30	10	1784	1784	0.005	0.00%	1784	10.1	0.00%	1784	0.01	0.00%
la32	30	10	1850	1850	0.025	0.00%	1850	10.1	0.00%	1850	0.0001	0.00%
la33	30	10	1719	1719	0.025	0.00%	1719	10.1	0.00%	1719	0.001	0.00%
la34	30	10	1721	1721	0.01	0.00%	1753	10.1	1.86%	1721	0.05	0.00%
la35	30	10	1888	1888	0.05	0.00%	1888	10.1	0.00%	1888	0.01	0.00%
la36	15	15	1268	1301	0.25	2.60%	1334	11.2	5.21%	1287	51	1.50%
la37	15	15	1397	1432	0.25	2.51%	1457	11.2	4.29%	1410	20	0.93%
la38	15	15	1196	1226	0.25	2.51%	1267	11.2	5.94%	1218	20	1.84%
la39	15	15	1233	1253	0.25	1.62%	1290	11.2	4.62%	1248	6	1.22%
la40	15	15	1222	1249	0.25	2.21%	1259	11.2	3.03%	1244	2	1.80%

TABLA III

RESULTADOS EXPERIMENTALES DE LOS PROBLEMAS PERTENECIENTES A LA CLASE *la*. LOS RESULTADOS OBTENIDOS POR NUESTRO AIS SE COMPARAN CONTRA GRASP Y GRASP + RT.

	GRASP			GRASP + RT		
	Ganó	Empató	Perdió	Ganó	Empató	Perdió
AIS	26	33	7	2	38	26

TABLA V

COMPARACIÓN DE RESULTADOS FINALES.

puede ganarle en 2 casos a GRASP+RT.

Podemos mencionar algunos problemas en los que parece interesante resaltar la comparación:

- En el problema *orb04*, nuestro AIS realiza solamente 250,000 evaluaciones de la función objetivo, mientras que GRASP efectúa 40.1 millones y GRASP + RT realiza 100 millones de evaluaciones. Los resultados obtenidos (con respecto a la mejor solución conocida) son: AIS: 1023/1005, GRASP: 1031/1005 y GRASP + RT : 1011/1005. Puede verse entonces que éste es un problema difícil para los 3 algoritmos comparados, pero que nuestro AIS produce resultados altamente competitivos con respecto a los obtenidos con GRASP+RT y a un costo computacional muy inferior.
- En el problema *abz9*, AIS encontró 735/691 con tan sólo 250,000 evaluaciones de la función objetivo, mientras que GRASP y GRASP + RT realizaron 20.1 y 1.0 millones de iteraciones, respectivamente para obtener resultados de 758/691 y 740/691 en cada caso.

Este patrón se repite en varios problemas más, en los cuales nuestro AIS aproxima de manera muy cercana (o incluso mejora marginalmente) los resultados obtenidos por los 2 algoritmos de GRASP comparados, pero a una fracción del costo computacional empleado por éstos.

V. CONCLUSIONES Y TRABAJO FUTURO

Hemos introducido una nueva propuesta basada en un Sistema Inmune Artificial para resolver

el problema del *Job Shop Scheduling*. La propuesta utiliza el principio de selección clonal (extiende ideas de CLONALG[18]) y adopta una representación de permutaciones que permite repeticiones. Nuestro algoritmo también incorpora una biblioteca de anticuerpos que se usa para construir nuevas soluciones del problema. La comparación se hace contra 2 versiones de GRASP: una sencilla [22] y otra hibridizada que hace uso de paralelismo y que además agrega otra técnica llamada Reenlace de Trayectoria para mejorar las soluciones encontradas [23]. Nuestro algoritmo tiene un comportamiento bastante bueno con respecto a estas 2 versiones de GRASP, si bien resulta inferior al GRASP+RT. Cabe indicar, sin embargo, que nuestro algoritmo no utiliza paralelismo (en su versión actual). A pesar de esto, nuestro algoritmo logra producir aproximaciones razonablemente buenas de las soluciones obtenidas por GRASP+RT, a una fracción de su costo computacional.

Como parte de nuestro trabajo futuro, planeamos mejorar la inicialización de la biblioteca de anticuerpos, para poder tener mejores soluciones desde el principio, haciendo uso de una heurística adicional. Pensamos agregar un mecanismo que verifique que la mutación tenga realmente un efecto sobre el objetivo a minimizar y no que sólo intercambie de posición las operaciones sin tener ningún cambio. Adicionalmente, planeamos tener una medida de afinidad que funcione con nuestra codificación. Finalmente, también planeamos trabajar con una versión multiobjetivo del problema del *Job Shop Scheduling*.

Agradecimientos

El segundo autor agradece el apoyo recibido a través del proyecto CONACyT No. 34201-A.

Referencias

[1] M. Pinedo, *Scheduling—Theory, Algorithms, and Systems*, Prentice Hall, Englewood Cliffs, 1995.

[2] Kenneth R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York, 1974.

[3] E.G. Coffman Jr., *Computer and Job Shop Scheduling Theory*, John Wiley and Sons, 1976.

[4] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms: I. Representation," *Computers and Industrial Engineering*, vol. 30, pp. 983–997, 1996.

[5] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms: II. Hybrid genetic search strategies," *Computers and Industrial Engineering*, vol. 36, no. 2, pp. 343–364, 1999.

[6] J.W. Barnes and J.B. Chambers, "Solving the Job Shop Scheduling Problem using Tabu Search," *IIE Transactions*, vol. 27, no. 2, pp. 257–263, 1995.

[7] Olivier Catoni, "Solving Scheduling Problems by Simulated Annealing," *SIAM Journal on Control and Optimization*, vol. 36, no. 5, pp. 1539–1575, September 1998.

[8] Emma Hart, Peter Ross, and J.Ñelson, "Producing robust schedules via an artificial immune system," in *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, Anchorage, Alaska, 1998, pp. 464–469, IEEE Press.

[9] Emma Hart and Peter Ross, "The Evolution and Analysis of a Potential Antibody Library for Use in Job-Shop Scheduling," in *New Ideas in Optimization*, David Corne, Marco Dorigo, and Fred Glover, Eds., pp. 185–202. McGraw-Hill, London, 1999.

[10] Xunxue Cui, Miao Li, and Tingjian Fang, "Study of Population Diversity of Multiobjective Evolutionary Algorithm Based on Immune and Entropy Principles," in *Proceedings of the Congress on Evolutionary Computation 2001 (CEC'2001)*, Piscataway, New Jersey, May 2001, vol. 2, pp. 1316–1321, IEEE Service Center.

[11] David S. Johnson Michael R. Garey, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W H Freeman & Co., June 1979, ISBN 0-7167-1045-5.

[12] Albert Jones and Luis C. Rabelo, "Survey of Job Shop Scheduling Techniques," NISTIR, National Institute of Standards and Technology, 1998.

[13] J. Adams E. Balas and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management science*, vol. 34(3):391-401, 1988.

[14] Thomas E. Morton and David W. Pentico., *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management.*, Wiley Series in Engineering & Technology Management. John Wiley & Sons, 1993.

[15] R. Hightower, S. Forrest, and A. S. Perelson, "The evolution of emergent organization in immune system gene libraries," in *Proceedings of the 6th. International Conference on Genetic Algorithms*, L. J. Eshelman, Ed. 1995, pp. 344–350, Morgan Kaufmann.

[16] A. Perelson, R. Hightower, and S. Forrest, "Evolution and Somatic Learning in V-Region Genes," *Research in Immunology*, vol. 147, pp. 202–208, 1996.

[17] Mihaela Oprea, *Antibody Repertories and Pathogen Recognition: The Role of Germline Diversity and Somatic Hypermutation*, Ph.D. thesis, University of New Mexico, Albuquerque,NM, 1999.

[18] Leandro Nunes de Castro and Fernando José Von Zuben, "Learning and Optimization Using the Clonal Selection Principle," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 239–251, 2002.

[19] Takeshi Yamada and Ryohei Nakano, "Job-shop scheduling," in *Genetic Algorithms in Engineering Systems*, A.M.S. Zalzala and P.J. Fleming, Eds., chapter 7, pp. 134–160. The Institution of Electrical Engineers, 1997.

[20] Leandro Nunes de Castro and Jonathan Timmis, *Artificial Immnue System: A New Computational Intelligence Approach*, Springer Verlang, Great Britain, September 2002, ISBN 1-8523-594-7.

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

[22] S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende, "A GRASP for Job Shop Scheduling," in *Essays and Surveys in Metaheuristics*, Celso C. Ribeiro and Pierre Hansen, Eds., pp. 59–80. Kluwer Academic Publishers, Boston, 2001.

[23] S. Binato, R.M. Aiex, and M.G.C. Resende, "Parallel grasp with path relinking for job shop scheduling," 2002.

[24] Mauricio G.C. Resende and José Luis Gonzalez Velarde, "Grasp: Procedimientos de búsqueda miopes aleatorizados y adaptativos," 2003.