Representaciones Indirectas en Algoritmos Genéticos y la alcanzabilidad de schedules semi-activos

Guillermo Ordoñez **Guillermo Leguizamón**

LIDIC* - Departamento de Informática
Universidad Nacional de San Luis
San Luis, Argentina
{legui,gordonez}@unsl.edu.ar

Resumen

El problema de Job-Shop Scheduling (JSS) es de gran importancia práctica. Sin embargo, su dificultad inherente lo convierte en uno de los miembros más duros de la clase de problemas $\mathcal{NP}$-Completos. Dada la inherente dificultad de los problemas de *scheduling* y en particular JSS, no siempre es posible aplicar métodos de búsqueda convencionales para encontrar soluciones cercanas al óptimo en un tiempo razonable. Esto ha llevado a un creciente interés en el uso de metaheurísticas, particularmente Algoritmos Evolutivos, como métodos de búsqueda alternativa. En este artículo se presenta un estudio comparativo de los resultados obtenidos a partir de la aplicación de un Algoritmo Genético (AG) al problema JSS. El diseño del AG incluye dos propuestas para atacar un problema detectado en estudios previos con respecto de la alcanzabilidad de *schedules* semi-activos usando representaciones indirectas. Una de las propuestas consiste en incrementar el espacio de búsqueda de manera tal de poder alcanzar los *schedules* semi-activos a través de una representación y *schedule builder* apropiados. La segunda propuesta, la cual es una extensión de la primera, reduce el espacio de búsqueda respecto al alcanzado con la primera propuesta, para cubrir sólo a los *schedules* semi-activos. El objetivo del estudio está centrado en mostrar las posibles mejoras en la performance de un AG a medida que la representación y el *schedule builder* incorporan variantes que permitan cubrir regiones adicionales del espacio de búsqueda. El estudio incluye comparaciones con representaciones indirectas previamente estudiadas.

Palabras Claves: *Job Shop Scheduling, Algoritmos Genéticos, Representaciones Indirectas, Schedules non-delay, activos y semi-activos.*

*El laboratorio es dirigido por el Dr. Raúl Gallard y subvencionado por la Universidad Nacional de San Luis y la ANPCyT (Agencia Nacional para la Promoción de la Ciencia y la Tecnología).

1 Introducción

El problema de Job-Shop Scheduling (JSS) es de gran importancia práctica. Sin embargo, su dificultad inherente lo convierte en uno de los miembros más duros de la clase de problemas $\mathcal{NP}$-Completos [6]. En general JSS [1] puede expresarse como sigue: Existen n jobs y m máquinas. Cada job comprende un conjunto de tareas u operaciones las cuales deben ser realizadas en un orden predefinido. Cada tarea debe ser realizada sobre una máquina específica y con una duración (tiempo) asignada dependiendo de la máquina en particular. Un scheduling es factible si no existe superposición de intervalos de tiempo correspondientes a distintos jobs (un job no puede ser procesado por dos máquinas al mismo tiempo) ni superposición de intervalos de tiempo de distintas máquinas (dos jobs no pueden ser procesados en una misma máquina al mismo tiempo). La formulación más común para este problema es minimizar el tiempo de procesamiento de la secuencia completa de jobs (makespan). JSS corresponde a un sistema de múltiples etapas [1], dado que los jobs involucran operaciones que se realizan sobre distintas máquinas. Para JSS cada job tiene una ruta preestablecida a través de las máquinas a ser usadas, pero puede variar de un job a otro.

Los Algoritmos Genéticos son algoritmos de búsqueda inspirados en la genética de las poblaciones naturales. Ellos mantienen una población de individuos que representan las soluciones candidatas. Dicha población evoluciona en el tiempo a través de la competencia entre los individuos y una variación controlada de los mismos. La aplicación de AGs incluye un amplio rango de problemas de optimización y aprendizaje de máquinas en variados dominios. Una descripción más detallada de este tipo de algoritmos puede ser consultada en [7, 9]. Los Algoritmos Genéticos han sido aplicados con diferente grado de éxito a problemas de scheduling de distintos tipos. Davis (1985) fue el primero en proponer el uso de AGs para resolver JSS [4]. Al mismo tiempo, diferentes versiones de AGs aplicados a un problema de ordenamiento, tal como el Problema del Viajante de Comercio, dieron lugar a representaciones avanzadas, muchas de ellas adecuadas para ser usadas en el contexto de problemas de scheduling. Un ejemplo de estas representaciones es la permutación de jobs la cual tiene asociado un conjunto considerable de operadores de crossover a ser aplicados: PMX, OX, CX, etc. [7, 9]. Estudios más recientes incluyen AGs que manipulan representaciones binarias para las soluciones en conjunción con algoritmos de interpretación y reparación [10]. Nuestro trabajo es una extensión de un estudio previo [11] sobre representaciones indirectas en el cual se consideraron dos representaciones diferentes. La primera de ellas es una representación simple que consiste en una permutación de jobs [7, 9]. La segunda, una propuesta que añade conocimiento del problema, intentaba corregir ciertos problemas presentados en la primera cuando dos o más jobs compiten por una misma máquina. Sin embargo, y a pesar de ciertas mejoras producidas, estas representaciones presentan algunas limitaciones en cuanto al tipo de *schedules* que pueden ser alcanzados una vez que la solución es decodificada. En el presente trabajo, este problema es considerado a través de una representación extendida que incluye información adicional que es usada por el *schedule builder* al momento de obtener el respectivo *schedule* (este término es usado como sinónimo de planificación). Dicha información permite extender el conjunto alcanzable de schedules válidos y por ende incrementar la probabilidad de encontrar mejores schedules para una instancia de JSS dada.

El presente artículo está organizado de la siguiente manera. En la Sección 2 se da una breve reseña de algunos tipos de schedules y su relación con el conjunto de schecules válidos y óptimos; en la Sección 3 se describen las representaciones usadas en nuestro Algoritmo Genético incluyendo una breve revisón de las representaciones usadas en trabajos previos. La Sección 4 presenta los experimentos realizados junto con el análisis de los resultados obtenidos. Finalmente las conclusiones según el presente reporte son consideradas.

2 Tipos de Schedules

Tal como fue introducido en la sección anterior, un problema de scheduling es un problema de distribución de recursos a lo largo del tiempo para el cual existe un conjunto de tareas a realizar (los jobs) las que para poder finalizar deben ocupar una serie de recursos, representados por máquinas, que sólo pueden atender una tarea a la vez.

Dependiendo de las características, un *schedule* (solución) asociado a un problema de scheduling se puede clasificar en:

1. Non-delay: en ningún momento ninguna máquina se encuentra ociosa cuando hay jobs esperando por alguna de éstas.

2. Activos: ninguna operación puede ser terminada antes sin retrasar ninguna otra operación (pero pudiendo ser alterado el orden de los jobs). Se puede ver que los *schedules* del tipo *non-delay* también son activos.

3. Semi-Activos: ninguna operación puede ser completada antes sin alterar el orden de los jobs. En consecuencia, los *schedules* activos son también semi-activos.

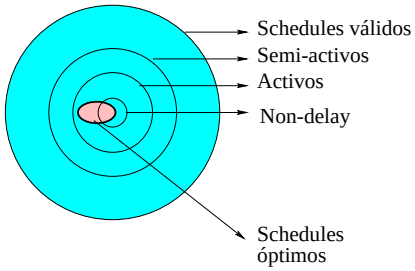

Figura 1: Diferentes tipos de *schedules*.

La Figura 1 representa la relación de inclusión entre los distintos tipos de *schedules* incluyendo los válidos y óptimos. Es importante destacar que la representación elegida para las soluciones y el respectivo *schedule* builder afectan directamente la performance del AG. Una de las razones es que no siempre es posible alcanzar todos los posibles *schedules* en el espacio de búsqueda. Adicionalmente, se deber tener en cuenta que las soluciones óptimas a un problema de scheduling se encuentran dentro del conjunto de los *schedules* activos, aunque no necesariamente dentro de los *non-delay*.

3 Representaciones consideradas

Estudios realizados sobre problemas de scheduling usando algoritmos basados en el enfoque evolutivo han considerado diferentes esquemas para la representación de las soluciones. Como fue mencionado previamente, Nakano y otros [10] hacen uso de una codificación binaria para representar indirectamente un *schedule*. Sin embargo, fue necesario desarrollar operadores especializados, como así también algoritmos de reparación con el objetivo de generar *schedules* válidos. En el otro extremo, Bruns [4] propone en su trabajo, el uso de una representación directa del *schedule*. Aunque una representación directa mantiene toda la información necesaria acerca del scheduling (la solución no codifica un schedule, sino que es el *schedule* en sí mismo), los operadores genéticos son muy complejos, debiendo considerar varios aspectos relacionados al conocimiento y restricciones del problema. La tendencia actual [8] y la más exitosa, es el uso de representaciones de soluciones entre dos enfoques extremos (representación binaria o representar directamente el schedule [4, 10]). En este sentido estamos haciendo referencia a representaciones indirectas que involucran cierto conocimiento del problema. Con este tipo de representación, el AG incorpora en la función de evaluación un algoritmo llamado *"schedule builder"*, el cual construye un *schedule* basándose en la solución (indirecta) dada como entrada. Muchos de los *schedule builder* son diseñados de acuerdo al método propuesto por Giffler & Thompson [8], el cual

genera *schedules* activos de acuerdo a una permutación de jobs dada. En nuestro trabajo, el *schedule builders* (Figura 2) está basado en el trabajo de Bagchi y otros [2], donde el tipo de *schedules* generados son *non-delay*. A continuación, se da una breve reseña de las representaciones previamente estudiadas en [11] (Sección 3.1) con el objeto de explicar la representación y su variante, usadas en el presente trabajo (Secciones 3.2 y 3.3).

3.1 Permutación de enteros (P)

Este es un ejemplo clásico de representación indirecta del cromosoma, ya que el cromosoma no codifica el *schedule*, sino, una permutación de n elementos. La posición de cada job dentro de la permutación, indica la prioridad que éste posee al momento de asignarle una máquina. Esta decisión es tomada por el *schedule builder* quien es el encargado de construir un *schedule* válido (Ver Figura 2). Una de las

```
=====================================================================
1. [Schedule Builder]
2.     SBuilder ( in Cromosoma : Perm(n) out objetivo)
3.     {
4.         Tiempo = 0;
5.         MIENTRAS no hayan terminado todos los jobs:
6.         {
7.             PARA CADA máquina libre:
8.                 Asigna el job de mayor prioridad que esté solicitando la máquina;
9.             Tiempo = menor de los tiempos de liberación de las máquinas;
10.            Libera todas las máquinas cuyo tiempo de liberación sea igual a Tiempo;
11.        }
12.            Objetivo = Tiempo (Equivale al makespan)
13.    }
=====================================================================
```

Figura 2: *Schedule builder* usado por el Algoritmo Genético.

anomalías que presenta este tipo de representación indirecta en el contexto de JSS está relacionada con la asignación de prioridades a los jobs. Debe tenerse en cuenta que la prioridad es la información usada por el *schedule builder* para construir un *schedule* válido. Dicha prioridad está dada por el orden en que aparece el job en la permutación y esta permanece fija independientemente de la máquina que se esté considerando para asignar un job. Es decir que cuando al menos dos jobs compiten varias veces por alguna de las máquinas en particular, ésta será asignada siempre al mismo job. El algoritmo de la Figura 2 (l'ınea 8) refleja esta situación; el job que aparece primero en la permutación siempre tiene mayor prioridad. Si bien esta asignación de prioridades podría ser efectiva para ciertas instancias del problema, pueden darse situaciones donde sea razonable asignar prioridades diferenciadas a cada job, dependiendo de la máquina.

La Tabla 1, muestra un ejemplo de una instancia de JSS para la cual se refleja el tipo de anomalía previamente descripta. Los pares en la segunda y tercera columna representan la máquina (M) y las unidades arbitrarias de tiempo (T).

Un cromosoma válido representando una posible secuencia podría ser la siguiente permutación (1,2,3). La aplicación del *schedule builder* entrega como salida el makespan relativo al *schedule*[1] de la Tabla 2 (izquierda), de acuerdo a las prioridades dadas en la permutación. Sin embargo, el *schedule* de

[1]El tiempo T en los *schedules* mostrados a través del art'ıculo, representa el tiempo de inicio.

Job	Ruta=(M, T)	
1	(1,1)	(2,1)
2	(1,1)	(2,1)
3	(2,2)	(1,2)

Tabla 1: Una instancia del problema JSS.

la Tabla 2 (derecha), también es válido, pero no es alcanzable (vía el *schedule builder*) para ninguna de las 3! permutaciones posibles.

T	0	1	2	3
M_1	1	2	3	3
M_2	3	3	1	2

T	0	1	2	3
M_1	1	2	3	3
M_2	3	3	2	1

Tabla 2: Dos *schedules* óptimos alternativos para la instancia de la Tabla 1.

Si bien para la instancia mostrada en la Tabla 1 es posible alcanzar otro *schedule* con un *makespan* óptimo, no es posible asegurar que exista este tipo de *schedules* alternativos para problemas más complejos. Tal como fue estudiado en [11], el conjunto de *schedules* alcanzables por esta representación es un subconjunto de los *schedules non-delay*. En consecuencia, para mejorar la performance del AG, se propuso ([11]) una representación denominada arreglo de permutaciones o AP, que solucionaba parcialmente el problema de alcanzabilidad, ya que con esta representación el conjunto de *schedules* alcanzables es incrementado hasta ser equivalente a los *schedules* de tipo *non-delay* (ver Figura 3).

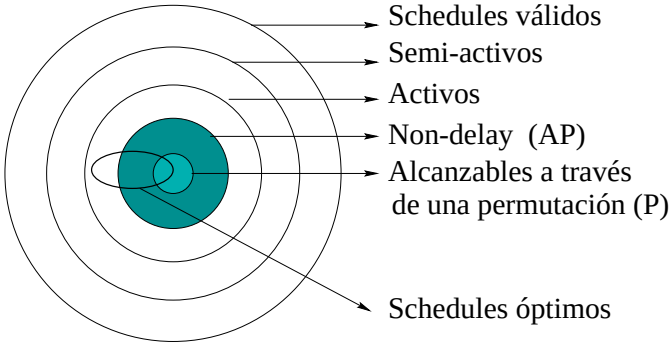

Figura 3: Schedules alcanzables desde las representaciones P y AP respectivamente.

3.2 Arreglo de Permutaciones + Matriz de Demora ($APMD$)

A fin de explicar la representación $APMD$ y el respectivo *schedule builder* usado, recordemos brevemente la representación AP. En el problema de JSS, como otros problemas de scheduling, existe la restricción de que a lo sumo cada job pasa una vez por cada máquina, entonces el problema mencionado anteriormente sólo puede pasar si un mismo grupo de jobs está compitiendo una vez por una máquina y luego por otra. Lo que se propuso en el trabajo anterior para resolver este problema consistió en modificar el cromosoma de forma tal que en vez de ser una permutación de enteros, sea un vector de m permutaciones, donde m es el número de máquinas de una particular instancia y la i-ésima componente del vector es una permutación que establece las prioridades de todos los jobs sobre la máquina i. Un

schedule builder para esta representación es muy similar al mostrado en la Figura 2. En principio, el argumento de entrada cambia por:

SBuilder (in $Cromosoma : array[1..m]$ of $Perm(n)$, out $Objetivo$)

Asimismo, el resto del algoritmo permanece igual excepto en la parte correspondiente a la selección del próximo job a procesar (Figura 2, linea 8). Aqu'ı la prioridad de los jobs involucrados en el conflicto depende de la máquina que se está analizando; en consecuencia, si la máquina corriente es la k, el job de mayor prioridad es el que se encuentre primero en la permutación $Cromosoma[k]$, es decir la permutación asociada a la máquina k.

La representación $APMD$ es un arreglo de permutaciones como AP, con la diferencia que tiene asociada información adicional representada por una matriz de demoras (MD) de m filas por n columnas. Dicha matriz contiene información acerca de eventuales demoras que pueden sufrir los jobs al momento de ser planificados (etapa de ejecución del *schedule builder*). $MD[i, j]$ representa la demora expresada en relación al número de intentos que serán considerados para el job j antes de asignarlo a la máquina i. La matriz MD es inicializada con valores aleatorios $x \in \{0, 1\}$ al momento de crear la población inicial. La variable x está distribuida de una manera no uniforme; en nuestro caso, $P_r(x = 0) = 0.98$ y $P_r(x = 1) = 0.02$. Sin embargo, $MD[i, j]$ puede alcanzar valores mayores que 1 durante la ejecución del AG.

El *schedule builder* para esta representación se muestra en la Figura 4. Puede observarse que se ha incorporado un argumento adicional correspondiente a la matriz de demoras. En el cuerpo del procedimiento (l'ıneas 11-12) la matriz de demora MD es consultada y eventualemente modificada durante el proceso de planificación de los jobs.

```
==========================================================================
1. [Schedule Builder]
2.    Sbuilder ( in Cromosoma: record AP : array[1..m] of Perm(n); MD : array[1..m, 1..n] end,
3.                 in out Objetivo)
4.    {
5.        Tiempo = 0;
6.        MIENTRAS no hayan terminado todos los jobs:
7.        {
8.            PARA CADA máquina i libre:
9.            {
10.               Busco el job j de mayor prioridad que esté solicitando la máquina;
11.               Si MD[i, j] > 0 decremento MD[i, j]
12.               En otro caso, asignar al job j la máquina i
13.           }
14.           Tiempo = menor de los tiempos de liberación de las máquinas;
15.           Libera todas las máquinas cuyo tiempo de liberación sea igual a Tiempo;
16.       }
17.       Objetivo = Tiempo (Equivale al makespan)
18.   }
==========================================================================
```

Figura 4: *Schedule builder* modificado.

El siguiente es un ejemplo que muestra sintéticamente el funcionamiento del *schedule builder* para la representación $APMD$ (Figura 4). Supongamos que nuestro problema es el que se muestra en Tabla 3.

Job	Ruta=(M, T)		
1	(3,1)	(2,2)	(1,1)
2	(1,2)	(2,1)	(3,3)

Tabla 3: Una instancia de JSS con $m = 3$ y $n = 2$.

Un posible cromosoma utilizando esta representación podr'ıa ser el mostrado en Tabla 4 (izquierda). El *schedule builder* se comporta en forma similar para el caso de AP, la diferencia es que al llegar al punto especificado en el *schedule* parcial (Tabla 4, derecha) se toman diferentes decisiones. En el

Cromosoma			
AP		MD	
2	1	0	0
2	1	0	1
1	2	0	0

$\longrightarrow$

Schedule parcial		
T	0	1
M_1	2	2
M_2	-	(*)
M_3	1	-

Tabla 4: Cromosoma y su *schedule* en etapa de construcción.

tiempo $T = 1$, el job 1 está listo para ser ejecutado en la máquina 2 y ésta, está libre. En el caso de la representación AP asignar'ıa la máquina 2 al job 1, pero, en este caso como $MD[2, 1]$ es mayor que 0, se decrementa en una unidad de tiempo la demora respectiva. Como en $T = 1$ no es posible realizar ninguna otra operación, T se incrementa hasta el próximo tiempo de liberación de una máquina, en este ejemplo, hasta $T = 2$ ya que la máquina 1 se libera en $T = 2$. Ahora el job de mayor prioridad disponible para ser ejecutado en la máquina 2 es el job 2, y como $MD[2, 2] = 0$, se asigna la máquina 2 al job 2, de esta forma, MD y el *schedule* bajo construcción son modificados acordemente (Tabla 5). La 'Xén la construcción del *schedule* indica que sa ha diferido una asignación de un job a una máquina disponible. Así al terminar el proceso de decodificación, obtenemos el schedule en Tabla 6 (derecha).

Cromosoma			
AP		MD	
2	1	0	0
2	1	0	**0**
1	2	0	0

$\longrightarrow$

Schedule parcial			
T	0	1	2
M_1	2	2	-
M_2	-	X	2
M_3	1	-	-

Tabla 5: Cromosoma y su *schedule* en etapa de construcción (MD es modificada).

Cromosoma			
AP		MD	
2	1	0	0
2	1	0	**0**
1	2	0	0

$\longrightarrow$

Schedule parcial						
T	0	1	2	3	4	5
M_1	2	2	-	-	-	1
M_2	-	X	2	1	1	-
M_3	1	-	-	2	2	2

Tabla 6: Cromosoma y su *schedule* en etapa de construcción.

Con esta representación, para todo *schedule* semi-activo, es posible encontrar al menos un individuo que lo represente. Es decir, el conjunto de *schedules* semi-activos está incluido el conjunto de *schedules* que pueden ser representados por $APMD$. Esta es una inclusión estricta, donde hay un alto porcentaje de los cromosomas que representan *schedules* válidos que sin embargo, no son semi-activos.

3.3 Una variación de Arreglo de Permutaciones + Matriz de Demora ($APMD_2$)

Esta representación es igual a $APMD$ explicada en la sección previa, la diferencia radica en la inclusión de una *función de corrección* que se aplica sobre un individuo recientemente creado. Esta función tiene por objetivo reducir el espacio de búsqueda sólo a los *schedules* que son semi-activos, y al mismo tiempo, que cada *schedule* sólo sea representado por un cromosoma. La función de corrección consta de tres etapas: la primera consiste en aplicar el *scheduler builder* descripto en la Figura 4; la segunda obtiene la parte AP del nuevo cromosoma a partir del *schedule* construido; la última etapa completa la matriz de demora MD de manera tal que al evaluar el nuevo individuo, el orden de ejecución sea aquel encontrado en el individuo original. Las distintas etapas de la función de corrección son explicadas a través del siguiente ejemplo (Tabla 7).

Job	Ruta=(M,T)			
1	(3,1)	(2,2)	(1,1)	(4,1)
2	(1,1)	(2,2)	(3,1)	(4,1)
3	(4,2)	(2,1)	(1,4)	(3,2)

Tabla 7: Instancia de JSS com $m = 3$ y $n = 4$.

Una posible entrada para la función de corrección podría ser el cromosoma de la Tabla 8. El primer

Cromosoma					
AP			MD		
3	2	1	0	0	0
3	2	1	0	1	0
3	2	1	0	0	0
3	2	1	1	0	0

Tabla 8: Individuo generado por un operador genético.

paso es armar el *schedule* utilizando el *schedule builder* de la Figura 4, como se muestra en la Tabla 9.

Schedule										
T	0	1	2	3	4	5	6	7	8	9
M_1	2	-	-	3	3	3	3	1	-	-
M_2	-	X	3	2	2	1	1	-	-	-
M_3	1	-	-	-	-	2	-	3	3	-
M_4	3	3	-	-	-	-	2	-	-	1

Tabla 9: Schedule asociado al individuo de la Tabla 8 (primer etapa).

Para armar el nuevo cromosoma, primero se asignan las prioridades para cada máquina en el orden en que se realizaron (ver *schedule* en Tabla 9) y se inicia MD con 0's según se muestra en la Tabla 10.

Por último se arma el *schedule* a partir del nuevo cromosoma, y en los casos en que se esté por asignar una máquina a un job antes de habérsela asignado a alguno de sus predecesores, se incrementa el entero asociado a la subtarea que está por ejecutarse tempranamente. Por ejemplo, en la Tabla 11 se está contruyendo un *schedule*. En este punto la máquina 2 está por ser asignada al job 2, pero, como ésta aún no ha sido utilizada por el job 3 (ver segunda permutación en AP), entonces en vez de asignarle la

Cromosoma					
AP			MD		
2	3	1	0	0	0
3	2	1	0	0	0
1	2	3	0	0	0
3	2	1	0	0	0

Tabla 10: Obtención de la parte AP del individuo (segunda etapa).

Schedule parcial		
T	0	1
M_1	2	-
M_2	-	(*)
M_3	1	-
M_4	3	3

Tabla 11: Punto de decisión para el llenado MD determinado por las prioridades.

máquina, incrementa $MD[2,2]$. De esta forma al terminar de la función de corrección obtenemos un nuevo individuo y su correspondiente *schedule* (Tabla 12).

Cromosoma					
AP			MD		
2	3	1	0	0	0
3	2	1	0	**1**	0
1	2	3	0	0	0
3	2	1	0	0	0

$\longrightarrow$

Schedule									
T	0	1	2	3	4	5	6	7	8
M_1	2	-	-	3	3	3	3	1	-
M_2	-	X	3	2	2	1	1	-	-
M_3	1	-	-	-	-	2	-	3	3
M_4	3	3	-	-	-	-	2	-	1

Tabla 12: Individuo obtenido según la función de corrección y su schedule correspondiente.

De esta forma, al terminar no sólo obtenemos un cromosoma válido y su respectivo *schedule*, sino también el resultado de la evaluación. Hay dos casos a tener en cuenta: 1) el *schedule* original era semi-activo, 2) el schedule original no lo era. Si el cromosoma original representaba un schedule semi-activo, entonces la función de corrección lo convierte en otro cromosoma que representa el mismo *schedule* semi-activo, este nuevo cromosoma sólo depende del *schedule* y no del cromosoma original, por lo tanto, si existen dos individuos diferentes en nuestra población sabemos que representan dos planificaciones diferentes, lo cual es útil en el caso que sea necesario evaluar la diversidad genética.

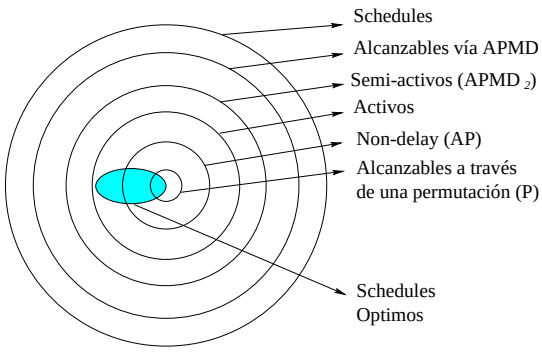

Figura 5: Schedules cubiertos por las distintas representaciones.

En el caso que el cromosoma no represente un *schedule* semi-activo, lo convierte en un *schedule* semi-activo, para el que se puede asegurar que cada subtarea va a terminar en un tiempo menor o igual que en el original. De esta forma no sólo se reduce el espacio de búsqueda del problema (principal objetivo al crear esta representación), sino que lo hace de una

Instancia	Optimo	E_m				$\bar{E}_m$			
		P	AP	$APMD$	$APMD_2$	P	AP	$APMD$	$APMD_2$
abz5	1234	3,82	2,45	2,06	1,28	4,12	2,57	2,76	1,86
ft10	930	6,72	3,53	5,01	3,23	7,65	4,90	6,76	4,78
la19	842	3,99	3,77	3,33	1,75	3,99	3,77	3,77	3,32
la25	977	7,57	4,50	5,15	3,65	8,91	6,34	8,95	5,98
la30	1216	12,33	10,65	13,33	10,26	14,60	12,11	14,93	11,65
la39	1233	6,52	2,30	2,76	2,14	6,80	3,38	5,96	5,63

Tabla 13: Instancias del grupo uno, se refleja un incremento en la performace del AG usando $APMD_2$.

forma en la que ayuda al AG al mejorar al individuo. Por último, si observamos que todo *schedule* semi-activo alcanzado por la representación anterior (y la representación anterior alcanzaba todos los *schedules* semi-activos) también va a ser alcanzado por ésta, y que el *schedule* que no sea semi-activo va a ser convertido en otro semi-activo, podemos ver que el conjunto de *schedules* alcanzables es equivalente al conjunto de los *schedules* semi-activos. El gráfico de la Figura 5 resume las alcanzabilidad de cada una de las representaciones mostradas. Las demostraciones sobre la alcanzabilidad de cada una de las representaciones no son incluidas en este trabajo por problemas de espacio [12].

4 Experimentos y resultados

Las instancias consideradas en los experimentos realizados han sido obtenidas de OR Library [3]. El conjunto de instancias incluye, para muchas de ellas, los valores óptimos conocidos obtenidos por algún método de optimización. Los operadores genéticos usados en las representaciones $APMD$ y su variante $APMD_2$ son similares a los usados para AP en [11] para la primera parte del cromosoma: crossover OX y mutación SWAP en cada permutación del arreglo. Adicionalmente, se usaron los siguientes operadores para la segunda parte (MD): crossover uniforme y mutación *little creep* (pequeñas variaciones sobre la componente a mutar). Los restantes parámetros del AG son los siguientes: tamaño de población, 100; probabilidad de crossover y mutación, 0.15 y 0.20 respectivamente. En todos los casos el AG incluyó una estategia elitista para mantener el mejor individuo a través de las sucesivas generaciones.

Las siguientes tablas muestran los resultados para un conjunto de instancias de JSS. En cada tabla, se especifica el nombre de la instancia (Instancia), el valor óptimo conocido (Optimo) y la calidad de los resultados expresada en término de los respectivos valores del error m'ınimo (E_m) y m'ınimo promedio ($\bar{E}_m$) de 10 corridas para cada una de las representaciones consideradas, a saber: permutación de enteros (P), arreglo de permutaciones (AP) y las propuestas, arreglo de permutaciones con la matriz de demoras ($APMD$) y su extensión $APMD_2$ que incluye la función de corrección. El error m'ınimo es un porcentaje calculado como ($E_m = \frac{MejorValorObtenido - Optimo}{Optimo} * 100$).

El conjunto de instancias consideradas están separadas en cuatro grupos según la calidad de los resultados obtenidos. El primer grupo está formado por las instancias mostradas en Tabla 13. Para estas seis instancias vemos que los resultados de la representación $APMD_2$ son superiores a los de las otras representaciones analizadas, aunque no mejoran lo suficiente como para alcanzar los óptimos respectivos. Es imporante destacar el uso de la función de corrección a fin de mejorar los valores obtenidos por $APMD$.

El segundo grupo de instancias se muestra en la Tabla 14. Los resultados de este grupo son muy parecidos a los del primer grupo; sin embargo, los valores arrojados por el AG usando $APMD_2$ mejoran

Instancia	Optimo	E_m				$\bar{E}_m$			
		P	AP	$APMD$	$APMD_2$	P	AP	$APMD$	$APMD_2$
ft06	55	5,17	3,51	0,00	0,00	5,17	3,51	0,66	2,58
la04	590	3,44	3,44	3,44	0,00	3,87	3,44	3,51	2,60
la28	1252	3,62	0,24	3,10	0,00	6,08	3,68	5,81	2,77
la34	1721	2,71	0,12	2,44	0,00	4,87	2,77	5,13	0,55

Tabla 14: Grupo de instancias para las cuales el AG con $APMD_2$ alcanza los valores óptimos.

lo suficiente como para alcanzar el óptimo en relación al resto de las representaciones.

El tercer grupo de instancias (Tabla 15) está formado por aquellas instancias para las cuales todas las representaciones encontraron los valores óptimos. Este resultado muestra que la performance del AG no es degradada cuando la representación incluye la matriz de demoras ($APMD$ y $APMD_2$).

Instancia	Optimo	E_m				$\bar{E}_m$			
		P	AP	$APMD$	$APMD_2$	P	AP	$APMD$	$APMD_2$
la05	593	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
la08	863	0,00	0,00	0,00	0,00	0,00	0,00	0,06	0,00
la09	951	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
la10	958	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
la14	1292	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
la15	1207	0,00	0,00	0,00	0,00	0,41	0,00	0,24	0,00
la35	1888	0,00	0,00	0,26	0,00	1,34	0,47	2,42	0,14

Tabla 15: Tercer grupo. Todas la representaciones consideradas obtienen valores óptimos.

Por último (Tabla 16) observamos los resultados para el cuarto grupo. Aunque $APMD$ y $APMD_2$ arrojaron mejores resultados que la permutación (P), la representación AP [11] obtuvo los mejores resultados para este grupo.

Instancias	Optimo	E_m				$\bar{E}_m$			
		P	AP	$APMD$	$APMD_2$	P	AP	$APMD$	$APMD_2$
la24	935	7,15	3,81	6,12	4,88	7,94	6,08	8,07	6,01
la29	1142	9,44	6,78	8,93	9,08	11,30	9,77	12,70	10,37
la40	1222	6,43	3,78	5,56	4,83	7,63	5,30	7,59	5,52

Tabla 16: AP alcanza los mejores valores, aunque $APMD_2$ se mantiene por encima de P.

5 Conclusiones

En el este trabajo hemos presentado dos propuestas para incrementar el conjunto de *schedules* alcanzables. Una de ellas, $APMD$ logra incrementar el espacio de búsqueda pero degradando la performance del AG en comparación con las representaciones previamente estudiadas. Una extensión de $APMD$ incluye una función de corrección con el objetivo de disminuir el conjunto de *schedules* alcanzables solamente a los *schedules* semi-activos. Esta propuesta, denominada $APMD_2$, evita la generación de individuos diferentes que representen el mismo *schedule*, es decir, se impide la generación de soluciones superfluas en la población, evitando as'ı una convergencia prematura del AG. De esta manera, $AMPD_2$ logró mejorar la performace del AG en la mayor'ıa de las instancias consideradas. Por ejemplo, para el primer y segundo grupo (50% de las instancias consideradas) los resultados obtenidos por

$APMD_2$ fueron superiores al los obtenidos con las otras representaciones (P y AP). En el tercer grupo (35% de las instancias) los resultados obtenidos fueron tan buenos como para el resto de las representaciones. Sin embargo, para el último grupo (15% de las instancias) se obtuvieron mejores resultados utilizando la representación AP, aunque por la relación de inclusión, los *schedules* alcanzados por AP podr'ıan ser alcanzados por $APMD_2$. En función de esta observación, una extensión al presente trabajo estará dirigido a investigar operadores genéticos alternativos que exploten en mayor medida la información codificada en cada cromosoma según $APMD_2$.

Referencias

[1] Aarts, E.& Lenstra, J.K. - Editores, (1997) - "Local Search in Combinatorial Optimization". John Wiley & Sons.

[2] Bagchi, S.; Uckun, S.; Miyabe, Y. & Kawamura, K. (1991) - "Exploring problem-specific recombination operators for job shop scheduling". Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, Morgan Kaufmann.

[3] Beasley,J. (1990) - "OR Library: distributing test problems by electronic mail". WWW site at "http://www.ms.ic.ac.uk/info.html"

[4] Bruns, R. (1993) - "Direct representation and advanced genetic algorithms for production scheduling". Proceedings of the Fifth International Conference on genetic Algorithms. San Mateo, Morgan Kaufmann.

[5] Davis, L. (1985) - "Job Shop Scheduling with Genetic Algorithms. Proceedings of the First International Conference on Genetic Algorithms", pages 136-140. San Mateo; Morgan Kaufmann.

[6] Garey, M. & Johnson, D. (1979) - "Computers and Intractability -A guide to the theory of NP-Completness". Freeman.

[7] Goldberg, (1989) - "Genetic Algorithms in Search, Optimization and Machine Learning". Addison-Wesley, Reading, MA.

[8] Hart, E. (1997) - "The State of the Art in Evolutionary Approaches to Timetabling and Scheduling", Departament of Artificial Intelligence, University of Edinburgh. http://www.dcs.napier.ac.uk/evonet.

[9] Michalewicz, Z. (1996) - "Genetic Algorithms + Data Structures = Evolution Programs". Third, revised and extended version. Springer-Verlag, USA.

[10] Nakano, R. & Yamada, T. (1991) - "Conventional genetic algorithm for Job-Shop Scheduling". Proceedings of the Fourth International Conference on genetic Algorithms. San Mateo, Morgan Kaufmann.

[11] Ordoñez, G. & G. Leguizamón. (1999) - "Representaciones Indirectas en Algoritmos Genéticos aplicados a un problema de Scheduling". IV Congreso Argentino de Ciencias de la Computación. Octubre de 2000, Tandil, Argentina.

[12] Ordoñez, G. & G. Leguizamón. (2001) - "Schedules alcanzables a través de representaciones indirectas" (En preparación).