Capítulo VI

MÉTODOS DE SOLUCIÓN PARA JOB SHOP SCHEDULING

6.1. HEURÍSTICAS CONVENCIONALES

El problema de job shop scheduling (JSSP) es un problema muy importante [69]; está entre los problemas de optimización combinatoria más duros, por lo tanto existen métodos de aproximación que producen soluciones aceptables empleando un tiempo razonable. Los procedimientos heurísticos para un problema de job shop se pueden clasificar en dos clases:

- ✓ *Heurísticas de una pasada*: construyen una única solución fijando una operación por vez en el schedule basándose en *reglas de prioridad de despacho*. Hay muchas reglas para elegir una operación de un subconjunto para ser asignada próximamente. Esta heurística es rápida.
- ✓ *Heurísticas de múltiples pasadas*: se construyen al repetir una heurística de una pasada para obtener schedules mejores incurriendo en algún costo extra de computación.

6.1.1. HEURÍSTICAS DE PRIORIDAD DE DESPACHO

Las reglas de prioridad son probablemente las reglas heurísticas aplicadas con mayor frecuencia para resolver problemas de scheduling, esto se debe a su fácil implementación y a su baja complejidad de tiempo. Los algoritmos de Giffler y Thompson se pueden considerar como la base de todas las heurísticas basadas en reglas de prioridad [133]. Giffler y Thompsom han propuesto dos algoritmos para generar schedules: procedimientos de generación de *schedules activos* y *schedules nondelay* [72]. Un schedule nondelay tiene la propiedad de que ninguna máquina permanece ociosa si hay un job disponible para su procesamiento. Un schedule activo tiene la propiedad de que ninguna operación se puede iniciar tempranamente sin retrasar otro

job. Los schedules activos forman un conjunto más grande que incluye como subconjunto a los schedules nondelay. El procedimiento de generación de Giffler y Thompson explora el espacio de búsqueda por medio de una estructura de árbol. Los nodos en el árbol representan los schedules parciales, los arcos representan las posibles elecciones, y las hojas del árbol son el conjunto de schedules. Para un schedule parcial, el algoritmo esencialmente identifica todos los conflictos de procesamiento, es decir, identifica operaciones que compiten por la misma máquina, y en cada etapa usa un procedimiento de enumeración para resolver esos conflictos bajo todas las formas posibles. En contraste, las heurísticas resuelven esos conflictos con reglas de prioridad de despacho; las heurísticas especifican una regla de prioridad para seleccionar una operación entre las operaciones en conflicto.

Los procedimientos de generación trabajan con un conjunto de operaciones planificables en cada etapa. Las operaciones planificables son aquellas que no se han asignado y que tienen predecesoras planificadas; este conjunto se puede determinar desde la estructura de precedencia. El número de etapas para un procedimiento de una pasada es igual al número de operaciones $m \times n$. En cada etapa, se selecciona una operación para agregar al *schedule parcial*. Los conflictos entre operaciones se resuelven por reglas de prioridad de despacho.

Para un schedule parcial activo dado, el posible tiempo de inicio σ_i se determina por el tiempo de finalización del antecesor directo de la operación i y el tiempo de finalización sobre la máquina requerida por la operación i. La mayor de esas dos cantidades es σ_i. El posible tiempo de finalización es simplemente $\sigma_i + t_i$, donde t_i es el tiempo de procesamiento de la operación i. Dada la siguiente notación:

- ✓ PS_t -un schedule parcial conteniendo t operaciones ya planificadas.
- ✓ S_t - el conjunto de operaciones planificables en la etapa t, correspondiente al PS_t dado.
- ✓ σ_i- el tiempo más temprano en el cual puede comenzar la operación $i \in S_t$.
- ✓ ϕ_i - el tiempo más temprano en el cual la operación $i \in S_t$ puede finalizar.

el procedimiento para generar un schedule activo consta de los siguientes pasos:

1. Sea $t = 0$ y comenzar con PS_t como el schedule parcial nulo. Inicialmente S_t incluye todas las operaciones sin antecesoras.

2. Determinar $\phi^*_t = \min_{i \in S_t} \{\phi_i\}$ y la máquina m^* sobre la cual se puede realizar ϕ^*_t.
3. Para cada operación $i \in S_t$ que necesita la máquina m^* y para la cual $\sigma_i < \phi^*_i$, calcular un índice de prioridad de acuerdo a la regla de prioridad. Hallar la operación con el índice más pequeño y agregarla a PS_t tan pronto como sea posible, creando así un nuevo schedule parcial PS_{t+1}.
4. Para PS_{t+1} actualizar los datos como sigue:
   1. Eliminar la operación *i* desde S_t.
   2. Formar S_{t+1} al adicionar el sucesor inmediato de la operación *j* a S_t.
   3. Incrementar *t* en uno.
5. Retornar al paso 2 hasta que se haya generado un schedule completo.

El problema que resta es identificar una regla de prioridad efectiva. A continuación se listan algunas de las reglas más comúnmente usadas en la práctica [69]:

- ✓ **SPT** (*shortest processing time*): seleccionar una operación con el tiempo de procesamiento más corto.
- ✓ **LPT** (*longest processing time*): seleccionar una operación con el tiempo de procesamiento más largo.
- ✓ **MWR** (*most work remaining*): seleccionar una operación para el job con el mayor tiempo de procesamiento pendiente.
- ✓ **LWR** (*least work remaining*): seleccionar una operación para el job con el menor tiempo de procesamiento pendiente.
- ✓ **MOR** (*most operations remaining*): seleccionar una operación para el job con el mayor número de operaciones pendientes.
- ✓ **LOR** (*least operations remaining*): seleccionar una operación para el job con el menor número de operaciones pendientes.
- ✓ **EDD** (*earliest due date*): seleccionar un job con la fecha de entrega más temprana.
- ✓ **FCSF** (*first come, first served*): seleccionar la primer operación en la cola de jobs para la misma máquina.
- ✓ **RND** (*random*): seleccionar una operación en forma aleatoria.

6.1.2. HEURÍSTICA DE DESPACHO ALEATORIO

Mientras las heurísticas de una pasada se limitan a la construcción de una única solución, las heurísticas de multipasada (también denominadas heurísticas de búsqueda) tratan de obtener muchas soluciones al generar varias de ellas, usualmente a expensas de mucho tiempo de computación. Técnicas tales como branch-and-bound y programación dinámica pueden garantizar una solución óptima, pero no son prácticas para problemas a gran escala.

Las heurísticas aleatorias son un intento para proveer soluciones más exactas [12]. La idea de despacho aleatorio es comenzar con una familia de reglas de despacho. Por cada selección de una operación para ejecutar, se elige la regla de despacho en forma aleatoria, esto se repite a lo largo de la generación del schedule. El proceso se reitera varias veces y se elige el mejor resultado. El procedimiento para generar un schedule activo tiene los siguientes pasos:

0. Sea *BS* el mejor schedule, iniciarlo como un schedule nulo.
1. Sea $t = 0$ y PS_t el schedule parcial nulo. Inicialmente S_t incluye todas las operaciones sin antecesoras.
2. Determinar $\phi^*_t = \min_{i \in St} \{\phi_i\}$ y la máquina m^* sobre la cual se realizará ϕ^*_t.
3. Seleccionar una regla de despacho aleatoriamente desde la familia de reglas. Para cada operación $i \in S_t$ que necesita la máquina m^* y para la cual $\sigma_i < \phi^*_i$, calcular un índice de prioridad de acuerdo a la regla de prioridad específica. Hallar la operación con el índice más pequeño y agregar esta operación a PS_t tan pronto como sea posible, creando así un nuevo schedule parcial PS_{t+1}.
4. Para PS_{t+1} actualizar los datos como sigue:
    1. Eliminar la operación *i* desde S_t.
    2. Formar S_{t+1} al adicionar el sucesor directo de la operación *j* a S_t.
    3. Incrementar *t* en uno.
5. Retornar al paso 2 hasta que se haya generado un schedule completo.
6. Si el schedule generado es los pasos anteriores es mejor que el mejor hallado hasta el momento, copiar éste en *BS*. Regresar al paso 1 hasta que la cantidad de iteraciones iguale las veces predeterminadas.

Varios investigadores han tratado de obtener mejoras a la opción aleatoria. Un cambio es introducir un proceso de aprendizaje de modo que las reglas de despacho más exitosas tengan más chances de ser seleccionadas en el futuro.

6.2. ALGORITMOS EVOLUTIVOS PARA EL PROBLEMA DE JOB SHOP SCHEDULING

6.2.1. REPRESENTACIÓN

Por la existencia de restricciones de precedencia de las operaciones, en JSSP no es tan fácil determinar una representación natural como en el problema del viajante (*traveling salesman problem* - TSP). No hay una buena representación que utilice un sistema de desigualdades para las restricciones de precedencia. Así, la opción con penalidades no es de fácil aplicación para manejar esos tipos de restricciones. Orvosh y Davis [112] han mostrado que para muchos problemas de optimización combinatoria, es relativamente fácil reparar un cromosoma ilegal o no factible, y la estrategia de reparación realmente supera estrategias tales como la de rechazo o la de penalización. La mayoría de los investigadores de JSSP y de algoritmos genéticos prefieren usar la estrategia de reparación para manejar la no factibilidad o ilegalidad. Una muy buena opción para la construcción de un algoritmo evolutivo para el problema de job shop scheduling es inventar una representación apropiada de soluciones junto con operadores genéticos específicos del problema, de manera tal que todos los cromosomas generados en la fase inicial o en el proceso evolutivo produzcan schedules factibles. Durante los últimos años se han propuesto varias representaciones para el problema de job shop scheduling, las cuales se pueden clasificar en:

- ✓ *Opciones directas*: un schedule (la solución al JSSP) se codifica en un cromosoma, y el algoritmo genético se usa para evolucionar esos cromosomas a fin de determinar un schedule mejor. Dentro de esta clase se encuentran las siguientes representaciones:
  - Representación basada en operaciones.
  - Representación basada en jobs.
  - Representación basada en relación de pares de jobs.
  - Representación basada en tiempo de finalización.

– Representación random key.

✓ *Opciones indirectas*: tal como la representación basada en reglas de prioridad, dentro del cromosoma se codifica una secuencia de reglas de despacho, los algoritmos genéticos se usan para determinar una mejor secuencia de reglas de despacho. Un schedule se construye con la secuencia de reglas de despacho. A esta clase pertenecen las siguientes representaciones:

– Representación basada en lista de preferencias.

– Representación basada en reglas de prioridad.

– Representación basada en grafos disjunctivas.

– Representación basada en máquinas.

6.2.1.1.REPRESENTACIONES DIRECTAS

6.2.1.1.1. REPRESENTACIÓN BASADA EN OPERACIONES

Esta representación codifica un schedule como una secuencia de operaciones, cada gen representa una operación. Hay dos posibles formas de nombrar cada operación. Una forma natural es usar números naturales para nombrar cada operación. Desafortunadamente, por la existencia de restricciones de precedencia, no todas las permutaciones de números naturales definen schedules factibles. Gen, Tsujimura, y Kubota propusieron una forma alternativa: nombrar a todas las operaciones de un mismo job con el mismo símbolo y entonces interpretarlas de acuerdo al orden de ocurrencia en la secuencia para un cromosoma dado [30, 92]. Para un problema de n jobs y m máquinas, un cromosoma contiene $n \times m$ genes. Cada job aparece en el cromosoma m veces, y cada repetición no indica una operación concreta de un job sino que se refiere a una operación la cuál es dependiente del contexto. Es fácil ver que cualquier permutación del cromosoma siempre produce un schedule factible.

Consideremos el problema de tres jobs y tres máquinas utilizado en [69] y presentado en la tabla 6.1. Supongamos el siguiente cromosoma [3 2 2 1 1 2 3 1 3], donde 1 indica el job j_1, 2 el job j_2, 3 el job j_3. Como cada job tiene tres operaciones, tiene exactamente tres ocurrencias en el cromosoma. Por ejemplo, en el cromosoma hay tres 2, lo cual representa las tres operaciones del job j_2:

✓ el primer 2 corresponde a la primer operación del job j_2 la cual se procesará sobre la máquina 1,

	Tiempo de Procesamiento				Secuencia de Máquinas		
	Operaciones				Operaciones		
Job	1	2	3	Job	1	2	3
j_1	3	3	2	j_1	m_1	m_2	m_3
j_2	1	5	3	j_2	m_1	m_3	m_2
j_3	3	2	3	j_3	m_2	m_1	m_3

Tabla 6.1 Ejemplo de un problema de tres jobs y tres máquinas.

- ✓ el segundo 2 corresponde a la segunda operación del job j_2 la cual se procesará sobre la máquina 3, y
- ✓ el tercer 2 corresponde a la tercer operación del job j_2 la cual se procesará sobre la máquina 2.

Como se puede observar, todas las operaciones para el job j_2 se denominan con el mismo símbolo 2 y se interpretan acorde al orden de ocurrencia en la secuencia del cromosoma. Las relaciones de ocurrencia correspondientes de las operaciones de jobs y máquinas de procesamiento se muestran en la figura 6.1.

Acorde a esas relaciones, podemos obtener la lista de máquinas [2 1 3 1 2 2 1 3 3], como se muestra en la figura 6.2. De esta figura se puede ver que el orden de procesamiento de jobs para la máquina 1 es 2 – 1 – 3 (indicado con el sombreado), para la máquina 2 es 3 – 1 – 2, y para máquina 3 es 2 – 1 – 3. En la figura 6.3, se muestra un schedule factible.

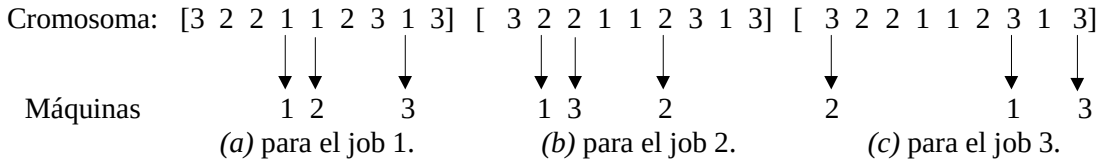


Figura 6.1 Operaciones de los jobs y correspondencia con las máquinas.

cromosoma	[3	2	2	1	1	2	3	1	3]
máquinas	2	1	3	1	2	2	1	3	3

Figura 6.2 El orden de procesamiento de los jobs sobre la máquina 1.

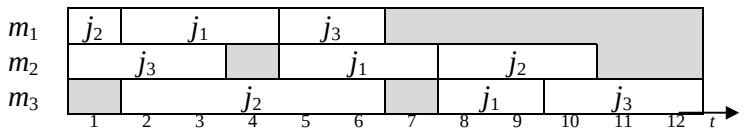
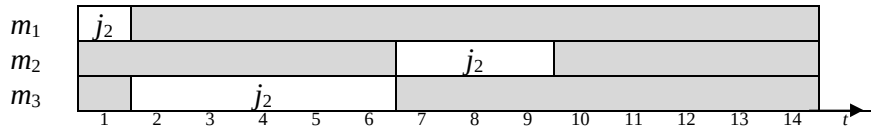


Figura 6.3 Un schedule factible.

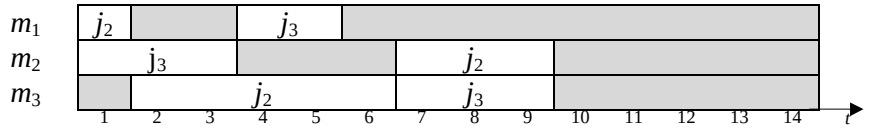
6.2.1.1.2. REPRESENTACIÓN BASADA EN JOBS

Esta representación consiste de una lista de *n* jobs y un schedule se construye en función de la secuencia de jobs. Para una secuencia de jobs dada, todas las operaciones del primer job se planifican primero, luego se consideran las operaciones del segundo job, y así sucesivamente. La primer operación del job en cuestión se asigna en el mejor tiempo de procesamiento disponible en la máquina que la operación necesite; luego se asigna la segunda operación, y así sucesivamente, hasta que se asignan todas las operaciones del job. El proceso se repite con cada uno de los jobs en la lista considerada en la secuencia apropiada. Toda permutación de jobs corresponde a schedules factibles.

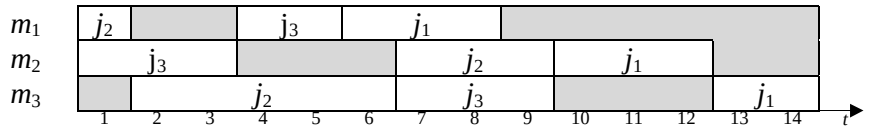
Usando el mismo ejemplo del problema de tres jobs y tres máquinas enunciado en la tabla 6.1, se supone el cromosoma [2 3 1]. El primer job a ser procesado es el job j_2. Las restricciones de precedencia de operaciones para j_2 son [m_1, m_3, m_2] y el tiempo de procesamiento correspondiente a cada máquina es [1 5 3]. El job j_2 se asigna como se muestra en la figura 6.4(a). Luego se procesa al job j_3, su precedencia de operaciones en las máquinas es [m_2, m_1, m_3] y los tiempos de procesamiento correspondientes en cada máquina son [3 2 3]. Cada una de sus operaciones se planifican en el mejor tiempo de procesamiento disponible en las máquinas que las operaciones necesitan, como se muestra en la figura 6.4(b). Finalmente, se planifica el job j_1 como se muestra en la figura 6.4(c).



(a) para el primer job j_2.



(b) para el segundo job j_3..



(c) para el último job j_1.

Figura 6.4 Deducir un schedule desde una codificación basada en jobs.

Precedencia de Operaciones				Schedule Factible			
job	Secuencia de Máquinas			máq.	Secuencia de Jobs		
j_1	m_1	m_2	m_3	m_1	j_2	j_1	j_3
j_2	m_1	m_3	m_2	m_2	j_3	j_1	j_2
j_3	m_2	m_1	m_3	m_3	j_2	j_1	j_3

Tabla 6.2 Ejemplo de un problema de tres jobs y tres máquinas.

6.2.1.1.3. REPRESENTACIÓN BASADA EN LA RELACIÓN ENTRE PARES DE JOBS.

Nakano y Yamada usan una matriz binaria para codificar un schedule, la cual se determina sobre la base de la relación de precedencia entre pares de jobs sobre las máquinas correspondientes [108].

Considerando el problema de scheduling de tres máquinas y tres jobs (tabla 6.1), en la tabla 6.2 se muestran las restricciones de precedencia de las operaciones de los jobs y un schedule factible para el problema.

Se define una variable binaria para indicar la relación de precedencia entre un par de jobs:

$$x_{ijm} = \begin{cases} 1, & \text{si el job } i \text{ es procesado antes al job } j \text{ sobre la máquina } m \\ 0, & \text{en cualquier otro caso.} \end{cases}$$

Examinando la relación de precedencia para el par de jobs (j_1 , j_2) sobre las máquinas (m_1, m_2, m_3), de acuerdo al schedule dado, se tiene (x_{121} x_{122} x_{123}) = (0 1 0). Para el par (j_1 , j_3) se tiene (x_{131} x_{132} x_{133}) = (1 0 1). Para un par de jobs (j_2 , j_3) la relación de precedencia sobre las máquinas (m_1, m_3, m_2) son (x_{231} x_{233} x_{232}) = (1 0 1). El orden de la variable x_{ijm} para un par de jobs deberá guardar consistencia con el orden de las operaciones del primer job *i*. Por ejemplo, para el par de jobs (j_2 , j_3), la secuencia de operaciones del job j_2 es (m_1, m_3, m_2), por lo tanto la secuencia de las variables relativas son (x_{231} x_{232} x_{233}). Para resumir los resultados, se muestra la matriz binaria para el schedule factible:

$$\begin{matrix} (j_1, j_2) \text{ sobre } (m_1, m_2, m_3) \\ (j_1, j_3) \text{ sobre } (m_1, m_2, m_3) : \\ (j_2, j_3) \text{ sobre } (m_1, m_3, m_2) \end{matrix} \begin{pmatrix} x_{121} & x_{122} & x_{123} \\ x_{131} & x_{132} & x_{133} \\ x_{231} & x_{233} & x_{232} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Esta representación es tal vez la más compleja. Es altamente redundante [53]. Además de su complejidad, los cromosomas producidos por procedimientos de inicialización o por operadores genéticos son, en general, ilegales.

6.2.1.1.4. REPRESENTACIÓN BASADA EN TIEMPOS DE FINALIZACIÓN

Yamada y Nakano propusieron la representación basada en tiempo de finalización [148]. Un cromosoma es una lista ordenada de tiempos de finalización de las operaciones. Para el mismo ejemplo dado en la tabla 6.1, el cromosoma se puede representar como:

$$[c_{111}\, c_{122}\, c_{133}\, c_{211}\, c_{223}\, c_{232}\, c_{312}\, c_{312}\, c_{333}\,]$$

donde c_{jir} indica el tiempo de finalización para la operación *i* del job *j* sobre la máquina *r*. Tal representación no es adecuada para la mayoría de los operadores genéticos porque se producirán schedules ilegales. Yamada y Nakano diseñaron un operador de crossover especial para esta representación.

6.2.1.1.5. REPRESENTATION RANDOM KEY

La representacion *random key* fue introducida por Bean [14]. Con esta técnica, las operaciones genéticas pueden producir hijos factibles sin producir overhead adicional en una amplia variedad de problemas de optimización y de secuenciamiento. Norman y Bean generalizaron con éxito la opción para el problema de job shop scheduling [109, 110].

Esta representación codifica una solución con un *número random*. Esos valores se utilizan como *claves* de ordenamiento para decodificar la solución. Para un problema de scheduling de *n* jobs y *m* máquinas, cada gen consiste de dos partes: un entero en el conjunto {1, 2, . . ., *m*} y una fracción generada aleatoriamente en el rango (0,1). La parte entera de cualquier clave random se interpreta como la asignación de máquina para ese job. El ordenamiento de las partes fraccionarias provee la secuencia de jobs en cada máquina. Consideremos el ejemplo dado en la tabla 6.1. Supongamos que el cromosoma es:

[1.34 1.09 1.88 2.66 2.91 2.01 3.23 3.21 3. 44]

Al ordenar las claves (dentro de cada máquina) en orden ascendente se obtienen las siguentes secuencias de jobs en cada máquina:

- ✓ máquina 1: secuencia de jobs $2 \rightarrow 1 \rightarrow 3$,
- ✓ máquina 2: secuencia de jobs $3 \rightarrow 1 \rightarrow 2$,
- ✓ máquina 3: secuencia de jobs $2 \rightarrow 1 \rightarrow 3$.

El cromosoma se puede trasladar a una única lista de operaciones ordenados como sigue:

$$[\, o_{21}\ o_{11}\ o_{31}\ o_{32}\ o_{12}\ o_{22}\ o_{23}\ o_{13}\ o_{33}]$$

donde o_{jm} indica el job j sobre la máquina m. La secuencias de jobs dados puede violar las restricciones de precedencia.

6.2.1.2. REPRESENTACIONES INDIRECTAS

6.2.1.2.1 REPRESENTACIÓN BASADA EN LISTAS DE PREFERENCIA

Esta representación fue propuesta en primer término por Davis para una clase de problema de scheduling [26]. Falkenauer y Bouffoix usaron esta representación para tratar un problema de job shop scheduling con tiempos de vencimiento [53]. Croce, Tadei y Volta aplicaron esta representación al problema de scheduling clásico [23].

Para un problema de job shop scheduling de n jobs y m máquinas, un cromosoma consiste de m subcromosomas, uno por máquina. Cada subcromosoma es un string de símbolos de longitud n, cada símbolo identifica una operación a ser procesada sobre la máquina correspondiente. Los subcromosomas no describen una secuencia de operaciones sobre la máquina sino que son *listas de preferencia*, cada máquina tiene su propia lista de preferencia. El schedule actual se deduce del cromosoma a través de una simulación, la cual analiza el estado de la cola de espera frente a cada máquina y, si es necesario, usa la lista de preferencia para determinar el schedule; es decir, se elige aquella operación que aparece primero en la lista de preferencia.

En el siguiente ejemplo se muestra cómo deducir un schedule desde un cromosoma tomando como base el ejemplo presentado en la tabla 6.1. Se supone que el cromosoma es [(2 3 1) (1 3 2) (2 1 3)]. El primer gen (2 3 1) es la lista de preferencia para la máquina m_1, (1 3 2) es la lista para la máquina m_2 y (2 1 3) para la máquina m_3. Desde esas listas de preferencia se puede ver que las primeras operaciones preferenciales son: job j_2 sobre la máquina m_1, j_1 sobre m_2, j_2 sobre m_3. Acorde a las restricciones de preferencia de las operaciones sólo se puede planificar j_2 sobre m_1, de este modo el primer schedule es el que se muestra en la figura 6.5(a). La próxima operación planificable es j_2 sobre la máquina m_3 como se muestra en la figura 6.5(b). En este punto, las operaciones preferenciales son j_3 sobre la máquina m_1 y j_1 sobre m_2 y m_3. Como no son planificables en este momento, se toma la segunda operación preferencial

en cada lista: j_1 sobre m_1 y j_3 sobre m_2.y m_3. Las operaciones a planificar son j_1 sobre m_1 y j_3 sobre m_2 y Se asignan como se muestra en la figura 6.5(c). Las próximas operaciones planificables son j_3 sobre m_1 y j_1 sobre m_2, como lo muestra la figura 6.5(d). Luego, las operaciones planificables son j_2 sobre m_2 y j_1 sobre m_3 como se muestra en la figura 6.5(e). La última operación planificable es j_3 sobre m_3 (figura 6.5(f)). Ahora se completa la deducción de un schedule desde el cromosoma y se obtiene un schedule factible con un makespan de 12 unidades de tiempo. Con el procedimiento de decodificación, todos los posibles cromosomas producen un schedule factible.

6.2.1.2.2 REPRESENTACIÓN BASADA EN REGLAS DE PRIORIDAD

Dorndorg y Pesch propusieron un algoritmo evolutivo basado en reglas de prioridad [32], un cromosoma se codifica como una secuencia de reglas de despacho para una asignación de jobs. Un schedule se construye con heurísticas de despacho de prioridad en base a la secuencia de reglas de despacho. Los algoritmos evolutivos se usan para evolucionar esos cromosomas que ayudarán a producir una mejor secuencia de reglas de despacho.

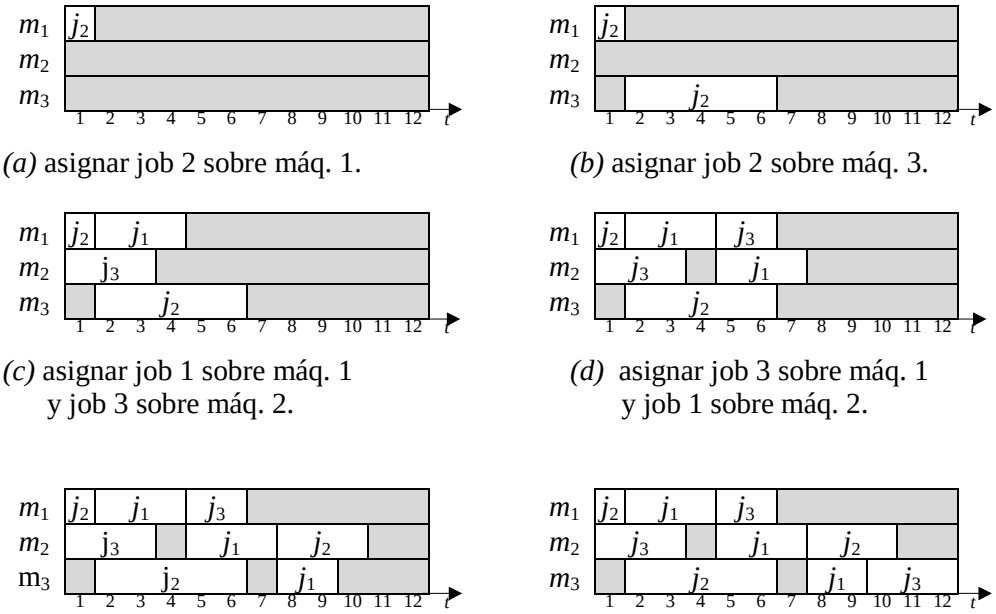


Figura 6.5 Deducir un schedule desde una codificación basada en listas de preferencias.

Las reglas de prioridad de despacho son las heurísticas que se usan con mayor frecuencia para resolver problemas de scheduling en la práctica, debido a su fácil implementación y a su bajo tiempo de complejidad. Los algoritmos de Giffler y Thompson se pueden considerar como la base para todas las heurísticas de reglas de prioridad [72, 133]. El problema es identificar una regla de prioridad efectiva.

Para un problema de n jobs y m máquinas, un cromosoma es un string de $n \times m$ entradas ($p_1, p_2, \ldots, p_{nm}$). Una entrada p_i representa una regla de un conjunto de reglas de prioridades de despacho. La entrada en la posición i-ésima indica que un conflicto en la iteración i-ésima del algoritmo de Giffler y Thompson se resolverá usando la regla de prioridad p_i. Una operación desde el conjunto de conflicto se selecciona por la regla p_i; en caso de existir más de una operación dentro de ese conjunto de conflicto, la decisión se hace en forma aleatoria.

Dada la siguiente notación:

- ✓ PS_t - un schedule parcial conteniendo t operaciones asignadas.
- ✓ S_t - el conjunto de operaciones asignables en la etapa t, correspondiente al PS_t.
- ✓ σ_i - el tiempo más temprano en el cual se deberá iniciar la operación $i \in S_t$.
- ✓ ϕ_i -el tiempo más temprano en el cual se deberá completar la operación $i \in S_t$
- ✓ C_t-el conjunto de operaciones en conflicto en la iteración t

el procedimiento para deducir un schedule desde un cromosoma dado ($p_1, p_2, \ldots, p_{nm}$) es el siguiente:

1. Sea $t = 1$ y comenzar con PS_t como el schedule parcial nulo. Inicialmente S_t incluye todas las operaciones sin predecesoras.
2. Determinar $\phi^*_t = \min_{i \in S_t} \{\phi_i\}$ y la máquina m^* sobre la cual se realizará ϕ^*_t. Si existe más de una máquina, la selección se determina en forma aleatoria.
3. Formar el conjunto C_t el cual incluye todas las operaciones $i \in S_t$ que necesita la máquina m^*. Seleccionar una operación de C_t usando la regla de prioridad p_t y adicionar esta operación a PS_t tan pronto como sea posible, creando así un nuevo schedule parcial PS_{t+1}. Si existe más de una operación acorde a la prioridad p_t, entonces esta situación se resuelve en forma aleatoria.
4. Actualizar PS_{t+1} al sacar la operación seleccionada desde S_t y agregar el sucesor directo de la operación j a S_t. Incrementar t en uno.
5. Retornar al paso 2 hasta que se haya generado un schedule completo.

Ejemplo. Dado [69]: las reglas SPT, LPT, MWR, LWR, con identificadores 1, 2, 3 y 4 respectivamente, el cromosoma [1 2 2 1 4 4 2 1 3] e indicando las operaciones como o_{jom} (donde el índice *jom* se refiere a job, operación, máquina). En el paso inicial (t = 1), se tiene:

$S_1 = \{ o_{111} , o_{211} , o_{312} \}$

$\phi_1^* = \min \{3, 1, 3\} = 1$

$m^* = 1$

$C_1 = \{ o_{111} , o_{211} \}$

Las operaciones o_{111} y o_{211} compiten por m_1. Como el primer gen del cromosoma es 1 (el cual significa la regla de prioridad SPT), la operación 211 es planificada en m_1, como se muestra en la figura 6.6(a). Después de actualizar tenemos (t =2),

$S_2 = \{ o_{111} , o_{223} , o_{312} \}$

$\phi_2^* = \min \{4, 6, 3\} = 3$

$m^* = 2$

$C_2 = \{ o_{312} \}$

Se planifica la operación o_{312} en m_2 como se muestra en la figura 6.6(b). Después de actualizar tenemos (en t =3),

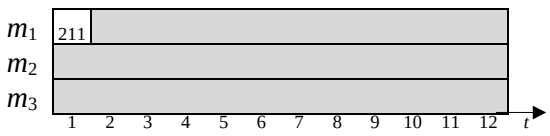
$S_3 = \{ o_{111} , o_{223} , o_{321} \}$

$\phi_3^* = \min \{4, 6, 5\} = 5$

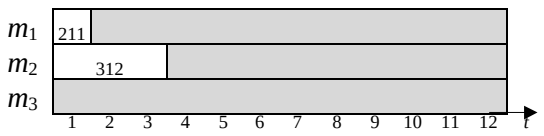
$m^* = 1$

$C_3 = \{ o_{111} , o_{321} \}$

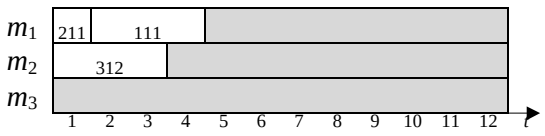
Las operaciones o_{111} y o_{321} compiten por m_1. Como el tercer gen del cromosoma es 2 (el cual representa a la regla de prioridad LPT), la operación o_{111} gana y se la planifica en m_1, como se muestra en la figura 6.6(c). Estos pasos se repiten hasta que se forma el schedule desde el cromosoma (figura 6.6(d)).



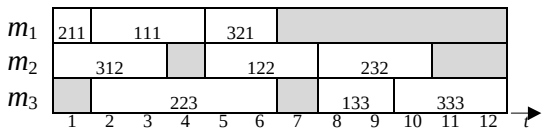
(a) planificación de la operación 211 en m_1.



(b) planificación de la operación 312 en m_1.



(c) planificación de la operación 211 en m_1.



(d) schedule final.

Figura 6.6 Deduciendo un schedule desde una codificación basada en reglas de prioridad.

6.2.1.2.3 REPRESENTACIÓN BASADA EN GRAFOS DISJUNCTIVOS.

Tamaki y Nishikawa proponen una representación basada en grafos disjunctivos [135], la cual se puede considerar como una clase de la representación basada en pares de jobs. El problema de job shop scheduling se puede representar con un grafo disjunctivo [13, 121]. El grafo disjunctivo $G = (N, A, E)$ se define como:

- ✓ N: contiene los nodos representado todas las operaciones.
- ✓ A: contiene los arcos que conectan operaciones consecutivas del mismo job.
- ✓ E: contiene los arcos disjuntos que conectan las operaciones a ser procesadas por la misma máquina.

Las restricciones disjunctivas se representan por un eje en E. Se puede establecer un arco disjunctivo por sus dos posibles orientaciones. La construcción de un schedule establece las orientaciones de todos los arcos disjunctivos así como también determina la secuencia de operaciones sobre la misma máquina. Una vez que se determina una secuencia para una máquina se reemplazan los arcos disjunctivos por arcos conjunctivos normales. La figura 6.7, muestra el grafo disjunctivo para el ejemplo de tres jobs y tres máquinas introducido en la tabla 6.1. Un cromosoma consiste de un string binario correspondiente a una lista de orden de arcos disjunctivos en E como se muestra en la figura 6.8, donde e_{ij} indica el arco disjunctivo entre los nodos i y j y se define como sigue:

$$e_{ij} = \begin{cases} 1, & \text{establecer la orientación del arco disjunctive desde el nodo } j \text{ al nodo } i \\ 0, & \text{establecer la orientación del arco disjunctive desde el nodo } i \text{ al nodo } j \end{cases}$$

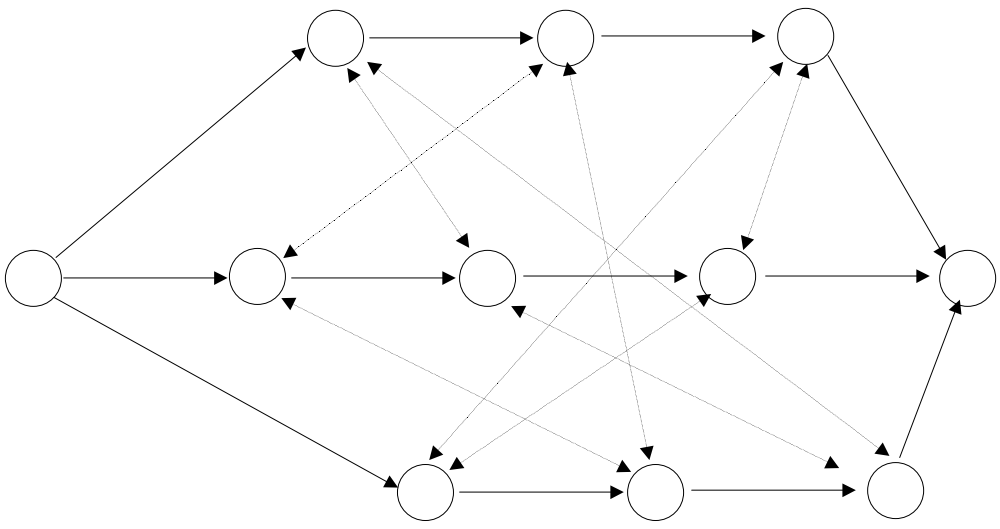


Figura 6.7 Grafo disjunctivo para el problema de tres máquinas y tres jobs.

Lista de orden de arcos disjunctivos	e_{15}	e_{19}	e_{59}	e_{24}	e_{28}	e_{48}	e_{36}	e_{37}	e_{67}
cromosoma	0	0	1	1	0	0	0	1	1

Figura 6.8 Representación basada en grafos disjunctivos.

El problema de job shop es encontrar un orden en las operaciones sobre cada máquina, es decir, fijar la orientación de los arcos disjunctivos tal que el grafo resultante no contenga ciclos para garantizar la no existencia de conflictos de precedencia entre las operaciones. Un cromosoma arbitrario puede provocar un grafo cíclico, lo cual significa que el schedule no es factible. Durante el proceso de deducción, cuando ocurre un conflicto de dos nodos (operaciones) sobre una máquina, se usa el bit correspondiente del cromosoma para fijar el orden de procesamiento de dos operaciones, es decir, fijar la orientación del arco disjunctive entre dos nodos.

6.2.1.2.4 REPRESENTACIÓN BASADA EN MÁQUINAS

Dornforf y Pesch propusieron un algoritmo evolutivo basado en máquinas [32] donde un cromosoma se codifica como una secuencia de máquinas y un schedule se construye con una heurística basada en corrimientos del cuello de botella sobre la secuencia.

La heurística de corrimiento del cuello de botella, propuesto por Adams, Balas y Zawack [1], es probablemente el procedimiento más poderoso entre todas las heurísticas para el problema de job shop scheduling. Las máquinas se secuencian una a una, sucesivamente, tomando aquella máquina identificada como el cuello de botella entre las máquinas aún no secuenciadas. Cuando se secuencia una nueva máquina, todas las secuencias establecidas previamente se reoptimizan localmente. Tanto los procedimientos de identificación del cuello de botella y de reoptimización local se basan en la resolución repetida de un cierto problema de scheduling de una única máquina que es una simplificación del problema original. La principal contribución de esta opción es la forma en que se usa esta simplificación para decidir respecto del orden en el cual se secuencian las máquinas.

Las heurísticas de corrimiento del cuello de botella se basan en la idea de dar prioridad a las máquinas con cuellos de botella. La calidad de los schedules obtenidos

por la heurística depende de la secuencia de máquinas identificadas como cuello de botella. Adams, Balas y Zawack también propusieron una versión enumerativa de la heurística para considerar diferentes secuencias de máquinas.

En lugar de una búsqueda enumerativa en un árbol, Dorndorf y Pesch propusieron una estrategia genética para determinar la mejor secuencia de máquinas para la heurística de corrimiento de cuello de botella. Un cromosoma es una lista de máquinas ordenadas. Los algoritmos genéticos evolucionan esos cromosomas que hallarán una mejor secuencia de máquinas que la heurística. La diferencia entre la heurística de corrimiento de cuello de botella y el algoritmo evolutivo es que no se usan más los cuellos de botella como criterio de decisión para elegir la próxima máquina, la cual se controla por el cromosoma dado.

Sea M_0 el conjunto de máquinas secuenciadas y $[m_1, m_2, \ldots, m_m]$ el cromosoma. El procedimiento para deducir un schedule desde el cromosoma es el siguiente:

1. Sea $M_0 \leftarrow \{\phi\}$ y $i \leftarrow 1$ y $[m_1, m_2, \ldots, m_m]$ el cromosoma.
2. Secuenciar óptimamente la máquina m_i. Actualizar el conjunto $M_0 \leftarrow M_0 \cup \{ m_i \}$.
3. Reoptimizar la secuencia de cada máquina crítica $m_i \in M_0$, mientras se toma las otras secuencias fijas.
4. $i \leftarrow i + 1$. Entonces si $i > m$, parar; en caso contrario ir al paso 2.