ARTÍCULO

Evolutionary Computation for CSP's

Camino R. Vela, Jorge Puente, Cesar L. Alonso, Ramiro Varela

Centro de Inteligencia Artificial
University of Oviedo. Campus de Viesques
Gijón, 33271

e-mail: {camino, puente, calonso, ramiro}@aic.uniovi.es

Scheduling problems are a paradigm of CSP. In this paper we present a number of techniques to solve them by means of Genetic Algorithms. Firstly we consider conventional Genetic Algorithms and then we demonstrate that their performance really improves when used together with other techniques such as dispatching rules, probability based heuristics or local search. We report results from an experimental study showing that a local search method is able to improve the performance of a Genetic Algorithm in solving the Job Shop Scheduling problem.

Computación Evolutiva para Resolución de CSPs

Camino R. Vela, Jorge Puente, Cesar L. Alonso, Ramiro Varela

Centro de Inteligencia Artificial
Universidad de Oviedo. Campus de Viesques
Gijón, 33271
{camino, puente, calonso, ramiro}@aic.uniovi.es
http:\\www.aic.uniovi.es

Resumen

Los problemas de Scheduling son un paradigma de la familia de problemas CSP. En este artículo presentamos algunas técnicas de resolución mediante Algoritmos Genéticos. Consideramos en principio la aplicación de Algoritmos Genéticos convencionales, y luego vemos como la eficacia de éstos se puede mejorar notablemente con la utilización conjunta de otras técnicas también clásicas como son las reglas de prioridad, los heurísticos basados en la probabilidad y la búsqueda local. En particular mostramos mediante un estudio experimental como un esquema de búsqueda local mejora el rendimiento de un Algoritmo Genético convencional en la resolución del problema Job Shop Scheduling.

Palabras clave: Algoritmos Genéticos, Scheduling, Búsqueda Local, Heurísticos, Reglas de Prioridad.

1. Introducción

En este artículo presentamos algunas ideas y experiencias sobre la aplicación de Algoritmos Evolutivos (AE) a la resolución de problemas CSP. En particular nos centramos sobre un tipo particular de AEs, los Algoritmos Genéticos (AG) y también sobre una familia de problemas CSP como son los problemas de scheduling.

Los AEs constituyen una estrategia adecuada especialmente para problemas CSP con optimización, como es el caso de los problemas de scheduling, en los que se trata de asignar tiempos de inicio a un conjunto de tareas que están sujetas a una serie de restricciones temporales, de forma que se minimice alguna medida regular de rendimiento, es decir, una función no decreciente en el tiempo de finalización de las tareas.

Para resolver CSPs generales existen AG genéricos que utilizan codificaciones binarias, o bien codifican directamente los valores de las variables en los cromosomas. En [Eib98, Cra00] se discuten varias aproximaciones de este tipo, en particular se propone una clasificación y se presentan algunos estudios experimentales. Sin embargo para el caso particular de los problemas de Scheduling, se suelen utilizar AGs con características diferentes a los AGs genéricos. Por ejemplo no resulta eficiente codificar directamente los valores de las variables, dado que los dominios son en general muy grandes, incluso infinitos en algunos casos, con lo que realizaríamos una búsqueda en un espacio de soluciones potenciales mucho mayor del que en la práctica resulta necesario. Por el contrario, otros tipos de representaciones, por ejemplo las que se basan en permutaciones y que describiremos en la sección 5, proporcionan espacios mucho más reducidos con lo que la búsqueda es más eficiente.

El resto de este artículo está organizado del siguiente modo. En la sección 2 presentamos una formulación de los problemas de Scheduling. En la

sección 3 comentamos uno de los métodos más simples de resolución: las reglas de prioridad. En particular describimos la regla DS/LTRP que se basa en un criterio heurístico y proporciona soluciones en un espacio restringido que puede no contener soluciones óptimas: el de planificaciones densas. En la sección 4 describimos otras alternativas para generar espacios de búsqueda completos: las planificaciones activas y semiactivas. En la sección 5 introducimos los AG aplicados a los problemas de Scheduling y discutimos varias estrategias de realización de las componentes más importantes, como la codificación, operadores genéticos y función de evaluación. En las secciones 6 y 7 presentamos dos métodos que permiten mejorar el rendimientos de un AG convencional: la búsqueda local, y la inicialización heurística. Aquí presentamos algunas ideas aplicables a problemas de Scheduling. La sección 7 se dedica a los estudios experimentales. Y por último en la sección 8 hacemos algunas consideraciones sobre nuevos problemas de Scheduling que pueden ser abordados mediante el uso de AGs.

2. Formulación de los problemas de *Scheduling*

Los problemas de scheduling son una subclase de problemas CSP que aparecen con frecuencia en la realidad. Podemos encontrar problemas de esta familia en muchas áreas de la industria, la administración y la ciencia; problemas que van desde la organización de la producción hasta la planificación de multiprocesadores. Los problemas de Scheduling son en general NP-duros por lo que para resolverlos se han aplicado prácticamente todas las técnicas de Inteligencia Artificial, con mayor o menor éxito. Así, podemos encontrar en la literatura (véase, por ejemplo, el número de la revista Artificial Intelligence 58 (1-3), 1992) soluciones con técnicas heurísticas [Mac93, Vae96, Sad96], Programación Lógica [Hen92, Var00], Redes Neuronales [Ado90], Aprendizaje Automático [Zwe92], Ramificación y Poda [Sab99], Búsqueda Local [Vae96, Yam97], Búsqueda Tabú [Now99], Enfriamiento Simulado [Kol99] y Algoritmos Genéticos [Bie95, Dor95, Sys91, Yam98], entre otros.

Uno de los problemas de scheduling más clásicos es el Job Shop Scheduling (JSS). El problema JSS aparece con profusión en la literatura científica como ejemplo de prueba de muchas técnicas de resolución de problemas. Hay también disponibles numerosos bancos de ejemplos de uso común entre los investigadores, lo que facilita el contraste de resultados experimentales. En [Jai99], Jain y Meeran ofrecen una clasificación de las principales técnicas que han sido aplicadas para resolver el problema JSS.

Existen distintas variantes del problema *Job Shop Scheduling* (JSS) dependiendo del criterio de optimización. En este trabajo consideramos el más clásico: el problema conocido en la literatura como $J//C_{max}$. Se trata de planificar la ejecución de un conjunto de trabajos $\{J_1,...,J_n\}$ sobre un conjunto de máquinas $\{M_1,...,M_m\}$. Cada trabajo Ji está compuesto por un conjunto de tareas $\{t_{i1},...,t_{im}\}$ que deben ser ejecutadas secuencialmente. Cada tarea t_{ij} requiere el uso exclusivo e ininterrumpido de una de las máquinas durante su tiempo de procesamiento du_{ij}. Supondremos además que para cada trabajo hay un tiempo mínimo de inicio y un tiempo máximo de fin entre los cuales deben ser planificadas todas sus tareas. El objetivo es conseguir un tiempo de inicio para cada una de las tareas de modo que se satisfagan todas las restricciones del problema y que además se minimice el tiempo de finalización de todas las tareas, es decir el makespan.

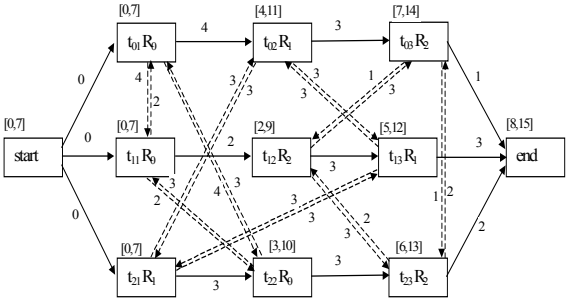

Figura 1. Representación gráfica de un problema con 3 trabajos de 3 tareas cada uno. El tiempo mínimo de inicio es 0 para todos ellos y el tiempo máximo de fin es 15. Cada nodo representa una tarea e indica el recurso que requiere. Los arcos continuos representan las restricciones de precedencia y los discontinuos las de capacidad. Cada arco está etiquetado con el coste de la tarea del nodo origen. Las tareas start y end tienen duración nula. Los intervalos asociados a cada nodo representan los tiempos de inicio de cada tarea restringidos por los tiempos mínimos de inicio y máximos de fin, así como por las restricciones de precedencia.

Una instancia del problema se puede representar mediante un grafo dirigido $G = (V, A \cup E)$ [Mat95]. Cada nodo del conjunto V representa una tarea, cada

arco del conjunto *A* representa una restricción de precedencia y los arcos del conjunto *E* representan las restricciones de capacidad. El conjunto *E* es de la forma $E = \cup_{i=1..m} E_i$, donde E_i representa las restricciones de capacidad asociadas a la máquina M_i. El subconjunto E_i incluye un arco *(v,w)* para cada par de tareas que requieren la máquina M_i. La Figura 1 muestra un ejemplo con 3 trabajos y 3 recursos. Una planificación válida se representa mediante un subgrafo acíclico *Gs* de *G*, *Gs=(V,A∪H)*, donde $H = \cup_{i=1..m} H_i$, siendo H_i un subgrafo hamiltoniano de E_i. El makespan de la planificación es el coste del camino más largo de *Gs*. De esta forma, encontrar una solución al problema consiste en encontrar subgrafos hamiltonianos compatibles, es decir planificaciones parciales para el conjunto de tareas que requieren a cada uno de las máquinas del problema tales que el grafo *Gs* resultante sea acíclico. La Figura 2 muestra una planificación válida para el problema de la Figura 1.

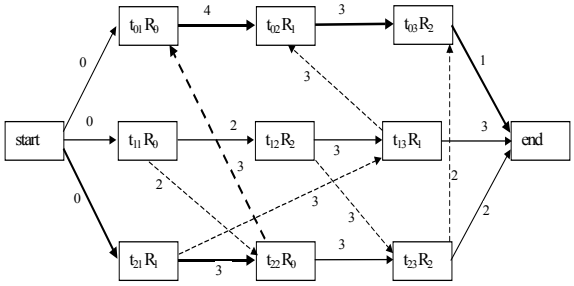

Figura 2. Una solución de coste 14 para el problema de la Figura 1

Dos variantes del JSS de gran interés son el Flow Shop Scheduling (FSS) y el Open Shop Scheduling (OSS). El primero es en realidad una simplificación del JSS en la que todos los trabajos utilizan las máquinas en el mismo orden. Mientras que el segundo es una generalización ya que cada trabajo puede utilizar las máquinas en cualquier orden. De este modo si para un problema OSS fijamos el orden en el que cada trabajo tiene que visitar las máquinas, tenemos un problema JSS.

Consecuentemente muchas de las soluciones desarrolladas para el problema JSS se pueden adaptar a los problemas FSS y OSS haciendo las simplificaciones oportunas en el primer caso o ampliaciones en el segundo.

En el caso del FSS existe a su vez una variante de interés: se trata del FSS con permutaciones (PFSS) en el que se exige que cada máquina sea visitada por los trabajos en el mismo orden. Es decir, en el problema FSS las primeras tareas de todos los trabajos utilizan la misma máquina, lo mismo sucede con las segundas tareas de todos los trabajos y así sucesivamente; sin embargo, el orden en el que las primeras tareas de los trabajos visitan su máquina no tiene que coincidir necesariamente con el orden en el que las segundas tareas visitan la suya. En cambio, en el problemas PFSS las segundas tareas de los trabajos visitan su máquina en el mismo orden que lo hicieron las primeras y lo mismo sucede con el resto. Existe también una variante del PSF con gran interés práctico: el Lot-Streaming FSS (LSFSS), en el que las tareas se pueden organizar en lotes de producción.

3. Reglas de Prioridad

Una de las estrategias más comunes para resolver problemas de Scheduling es el uso de reglas de prioridad (dispatching rules). Se trata en general de algoritmos voraces que toman decisiones basadas en algún criterio de optimización local. En la literatura podemos encontrar una amplia variedad de este tipo de estrategias heurísticas para problemas de Scheduling ([Raj01, Gué98, Lia98, Pin95]). En este trabajo describiremos en principio la regla DS/LTRP y una variante de esta denominada DS/RANDOM que han sido propuestas en [Gué98] para el problema OSS, pero que es aplicable también al problema JSS, y que a su vez están inspiradas en otras propuestas anteriores. Estas dos reglas se caracterizan por el hecho de que la solución que producen es *densa* (de ahí la etiqueta DS, *Dense Scheduling*). Una planificación es densa si nunca se da la situación de que una máquina esté inactiva a la vez que hay una operación disponible para poder ejecutarse en dicha máquina. Una operación se considera disponible cuando no se está realizando en ese instante otra operación del mismo trabajo. La etiqueta LTRP proviene de la regla "Longest Total Remaining Processing on other machine first" propuesta en [Pin95].

El hecho de restringir la búsqueda de soluciones al espacio de las planificaciones densas tiene como principal ventaja que, en media, este tipo de planificaciones tienen un bajo coste. Sin embargo tiene un inconveniente importante, ya que en general no hay garantía de que entre las planificaciones densas se encuentre al menos una planificación óptima.

El algoritmo asociado a la regla DS/LTRP es el que se muestra en la figura 3. Como se puede observar el algoritmo tiene dos situaciones de indeterminismo (* y **) que se resuelven con el criterio de favorecer tanto a las máquinas como a los trabajos con mayor tiempo de procesamiento restante. La regla DS/RANDOM es una variante de la anterior en la que estas situaciones de no determinismo se resuelven aleatoriamente.

Algoritmo DS/LTRP

mientras *queden operaciones sin planificar* **hacer**

1. *Seleccionar la máquina M que pueda ser utilizada lo antes posible según la planificación parcial construida hasta el momento. (*) En caso de empate seleccionar la máquina con mayor tiempo de procesamiento restante.*
2. *Elegir entre las tareas disponibles para la máquina M aquella que pueda comenzar antes y asignarle el menor tiempo de inicio posible. (**) En caso de empate elegir la tarea que pertenezca al trabajo con mayor tiempo de procesamiento restante.*

finmientras;
fin.

Figura 3. Algoritmo de planificación DS/LTRP para los problemas OSS y JSS

4. Planificaciones Activas y Semiactivas

Como hemos adelantado en la sección anterior, el principal problema de las planificaciones densas es que no se puede garantizar que entre ellas exista al menos una solución óptima. Existen otros subconjuntos de planificaciones entre las que siempre se encuentra al menos una solución óptima (considerando el makespan como criterio de minimización): se trata de las planificaciones *semiactivas* y las planificaciones *activas*.

Una planificación es activa cuando no es posible adelantar la ejecución de una operación sin retardar la ejecución de al menos otra. La figura 4 muestra una solución óptima (en forma de gráfico de Gantt) para un problema OSS. Se trata de una planificación activa pero no densa ya que entre los instantes de tiempo 4 y 5 la máquina M_3 está inactiva a la vez que está disponible la correspondiente tarea del trabajo J_1.

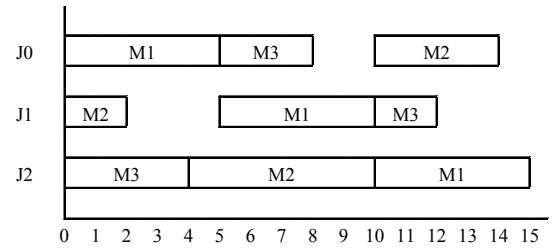

Figura 4. Una solución óptima para un problema OSS con 3 trabajos y 3 máquinas. Se trata de una planificación activa pero no densa.

Una planificación es semiactiva cuando para adelantar la ejecución de una tarea es preciso cambiar el orden relativo de ejecución de al menos dos tareas. Es fácil comprobar que una planificación densa es activa y que una planificación activa es también semiactiva. Además, la intersección del conjunto de planificaciones óptimas con el conjunto de las planificaciones activas es siempre no vacío, mientras que en general esto mismo no se puede garantizar en el caso de las planificaciones densas (también llamadas planificaciones non-delay). Esta relación se representa en el gráfico de la figura 5. En consecuencia, si queremos tener la seguridad de que el espacio de búsqueda contiene al menos una solución óptima, debemos restringir la búsqueda, como mucho, al espacio de planificaciones activas.

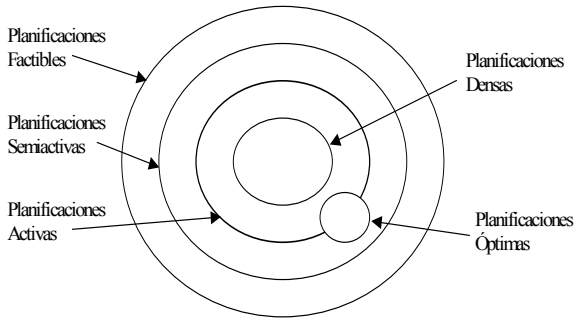

Figura 5 Relación entre los espacios de planificaciones factibles, semiactivas, activas, densas y óptimas**.**

Para el cálculo de planificaciones activas existen varios algoritmos. El más natural es aquel en el que el tiempo de inicio de cada operación se decide siguiendo un orden lineal (en principio cualquiera) entre las tareas del problema, de modo que para

cada operación este instante es el menor tiempo de inicio posible teniendo en cuenta las operaciones ya planificadas (es decir, las anteriores en el orden lineal) sobre a misma máquina y/o sobre el mismo trabajo, dependiendo de la familia de problemas que estemos considerando (JSS, OSS, FSS).

Una alternativa al método anterior es la que proporciona el conocido algoritmo G&T propuesto por Giffler y Thomson en [Gif60] en el año 1960 en principio para resolver el problema JSS, pero que se puede extender de forma natural al problema OSS. Se puede probar que el algoritmo G&T calcula planificaciones activas y que además considera todas las posibles planificaciones activas. Se trata de un algoritmo voraz no determinista que puede ser particularizado de muchas formas y adaptado a diferentes estrategias de búsqueda como los algoritmos genéticos y la búsqueda en espacios de estados, entre otros.

Una de las características más interesantes del algoritmo G&T es que permite utilizar técnicas complementarias de reducción del espacio de búsqueda. La idea es que en problemas grandes el espacio de búsqueda formado por todas las planificaciones activas tiene un tamaño excesivo y lo práctico es hacer una reducción de este espacio para acelerar la búsqueda, con la consiguiente pérdida de algunas propiedades: al no considerar todas las planificaciones activas es posible que sea imposible encontrar una solución óptima. Pero aun así la reducción suele ser efectiva sobre todo en problemas de un cierto tamaño.

Consideramos aquí una variante del algoritmo G&T está tomada de un trabajo de Ch. Bierwirth y D. Mattfeld [Bie99], y que está a su vez inspirada en otro trabajo de Storer [Sto92]. La figura 6 muestra este algoritmo que, siguiendo a Bierwirth y Mattfeld denominamos algoritmo G&T híbrido. En este algoritmo el parámetro δ permite controlar la reducción del espacio de búsqueda. Si $\delta=1$ no hay reducción, mientras que si $\delta=0$ la reducción es máxima. En los experimentos comentados en [Bie99] se observa que a medida que el tamaño del problema crece, el aumento del valor del parámetro δ se traduce en que el AG produce mejores resultados.

Como veremos en la sección 5, para adaptar el algoritmo G&T a un AG la idea es seleccionar la tarea θ* de B con un criterio que dependa de la codificación de un cromosoma. En el caso de la búsqueda en espacios de estados con algoritmos tipo A* y backtracking, la idea es considerar todas las tareas del conjunto B y con cada una de ellas generar un estado sucesor.

Algoritmo G&T híbrido

Sea A conjunto formado por la primera tarea de cada trabajo;

mientras $A \neq \emptyset$ **hacer**

Determinar la tarea $\theta' \in A$ tal que tiene el menor tiempo de fin posible en el estado actual, es decir $t\theta'+du\theta' \leq t\theta + du\theta, \forall\theta \in A$;

Sea M' la máquina requerida por θ', se construye el conjunto B formado por las tareas de A que requieren la máquina M';

Se eliminan de B aquellas tareas que no pueden comenzar antes de $t\theta'+du\theta'$;

/ De este modo sea cual sea la tarea de B elegida para ser planificada, se obtiene finalmente una planificación activa */*

Se determina la tarea $\theta'' \in B$ tal que es la que tiene el menor tiempo de inicio posible, sea este tiempo $t\theta''$;

/ Obviamente el tiempo de inicio de cualquier tarea de B si se elige la próxima está en el intervalo $[t\theta'', t\theta'+du\theta']$ */*

Se hace una reducción de B de tal forma que

$$B = \{ \theta \in B / t\theta < t\theta'' + \delta((t\theta'+du\theta') - t\theta'') \}, \text{ con } \delta \in [0,1];$$

/ De esta forma se reduce el intervalo a $[t\theta'', t\theta'' + \delta((t\theta'+du\theta') - t\theta'')]$ y en consecuencia ya no consideramos todas las planificaciones activas posibles */*

Seleccionar θ* de B **con algún criterio** y planificarla;

Borrar θ* de A y añadir a A la sucesora de θ* en caso de que exista, es decir si no es la última tarea de su trabajo;

finmientras;

fin.

Figura 6. Algoritmo G&T híbrido para el problema JSS

La figura 7 muestra una adaptación del algoritmo G&T híbrido al problema OSS. En este caso se obtiene además una estrategia determinista al combinarlo con la regla de prioridad LTRP.

Algoritmo G&T/LTRP

mientras *queden operaciones sin planificar* **hacer**

1. *Seleccionar la tarea Oij no planificada que antes puede terminar si es planificada en el estado actual. Sea Tfin el tiempo de terminación de esta tarea si es planificada en este estado. (*) En caso de empate elegir la tarea Oij tal que*

$$max(TR(J_i), TR(M_j)) = \max_{O_{kl}\ no\ planificada} (max(TR(J_k), TR(M_l)))$$

2. Elegir entre las tareas disponibles para la máquina Mj y entre las tareas no planificadas del trabajo Ji aquellas que puedan comenzar en un tiempo estrictamente menor que tfin, sea B el conjunto formado por todas estas tareas, y sea tini el tiempo de inicio más temprano para la tarea de B que antes puede comenzar en el estado actual.
3. Reducir el conjunto B de modo que solamente se mantienen aquellas tareas que pueden comenzar en el intervalo

$$\left[t_{ini}, t_{ini} + \delta(t_{fin} - t_{ini})\right]$$

siendo δ un parámetro con valor en el intervalo [0..1].

4. (**) Seleccionar la tarea Opq del conjunto B tal que

$$max(TR(J_p), TR(M_q)) = \max_{O_{kl} \in B} (max(TR(J_k), TR(M_l)))$$

y asignarle el menor tiempo de inicio posible.

finmientras;

fin.

***Figura* 7** Algoritmo de planificación G&T/LTRP. *TR(J)* y *TR(M)* se refieren respectivamente al tiempo de proceso restante del trabajo *J* y de la máquina *M* en el estado actual

5. Algoritmos Genéticos para problemas de *Scheduling*

Los Algoritmos Genéticos (AG) constituyen una estrategia de búsqueda de propósito general que permite resolver problemas de naturaleza combinatoria. Fueron propuestos por J. Holland en el año 1975. En esencia, utilizan una estrategia de búsqueda estocástica en un espacio de soluciones potenciales de un problema, que trata de modelar las leyes de la evolución natural, en particular la herencia genética y la adaptación al entorno. En líneas generales, la estrategia operativa de un algoritmo genético consiste en partir de una población inicial de individuos, generada aleatoriamente, cada uno de los cuales representa una solución potencial del problema. Estos individuos se evalúan mediante una función *(fitness)* que indica la calidad de la solución o grado de adaptación del individuo al entorno. A partir de esta situación inicial se realizan una serie de iteraciones en cada una de las cuales se simula la creación de una nueva generación de individuos a partir de la generación anterior. Este proceso consiste en aplicar los operadores genéticos de selección, cruce y mutación sobre los individuos. Si el algoritmo converge adecuadamente el valor medio de los individuos irá mejorando en las sucesivas generaciones. La solución que ofrece es el mejor individuo encontrado después de un determinado número de generaciones.

La aplicación de los AGs a problemas de naturaleza combinatoria ha interesado a muchos investigadores debido a que son capaces de tratar con los enormes espacios de búsqueda que aparecen en este tipo de problemas. Esto es debido a que en la práctica realizan una búsqueda estocástica multidireccional sobre todo el espacio de búsqueda, pero de un modo más intensivo sobre aquellas regiones más prometedoras. Por su naturaleza, los AGs no siempre encuentran la solución óptima de un problema, pero son capaces de encontrar soluciones razonables con mucha rapidez. La Figura 7 muestra el esquema de un AG convencional.

Algoritmo Genético

Generar una población inicial aleatoriamente, o bien con algún criterio heurístico;

Evaluar la población actual;

mientras *no se satisfaga la condición de terminación* **hacer**

1. *Aplicar el criterio de Selección a los cromosomas de la población actual;*
2. *Aplicar los operadores genéticos de Cruce y Mutación a los cromosomas seleccionados en el paso 1.-*
3. *Evaluar los cromosomas generados en el paso 2.-*
4. *Aplicar el criterio de Aceptación al conjunto de cromosomas formado por los cromosomas seleccionados en el paso 1.- junto con los generados en el paso 2.*

finmientras;

Devuelve el mejor de todos los cromosomas evaluados;

fin.

Figura 8. Estructura de un AG convencional

En la literatura existen numerosas aplicaciones de los AGs a la resolución de problemas de Scheduling. Entre las aportaciones más destacables podemos citar los trabajos de [Bie95, Sys91, Mat96, Bie99]. En todos los casos se pone de manifiesto que la eficiencia de un AG convencional es limitada para tratar con el problema y se proponen mejoras de los distintos elementos que producen resultados comparables a los métodos más competitivos en algunos casos. En esta línea, el esfuerzo de los investigadores ha ido encaminado principalmente a la búsqueda de codificaciones que permitan reducir el espacio de búsqueda y que sean adecuadas para las operaciones genéticas, funciones de evaluación más precisas, formas de organizar la población y estrategias de búsqueda local. Asimismo, otra forma de mejorar el rendimiento de una AG consiste en generar cromosomas heurísticos en la población inicial. En esta sección describimos las componentes esenciales de un AG convencional.

La estrategia evolutiva debe mantener un equilibrio entre la presión selectiva y la diversidad de la población, por ello, en el caso de algoritmos de reemplazamiento generacional normalmente se opta por realizar una selección proporcional al fitness para elegir a los progenitores que serán sometidos a los operadores de cruce y mutación, y luego los hijos resultantes sustituyen a los padres en la nueva población. O bien los padres se seleccionan aleatoriamente y luego se aplica un criterio de aceptación que consiste en elegir los mejores entre los padres y los hijos para formar parte de la nueva población.

Codificación de los cromosomas

Para problemas JSS uno de los esquemas de codificación más contrastados es la codificación basada en *permutaciones con repetición* [Bie95, Fan93, Sys91]. De acuerdo con esta representación, un individuo se codifica, en principio, mediante una permutación del conjunto de tareas del problema en la que las tareas de los distintos trabajos mantienen su orden relativo. Por ejemplo, para el problema de la Figura 1 una permutación "ordenada" de sus tareas *es* (t_{21} t_{31} t_{22} t_{11} t_{32} t_{12} t_{13} t_{23}). Esta ordenación de las tareas debe representar una solución potencial del problema, es decir una planificación de las mismas que eventualmente satisfará todas las restricciones del problema y en este caso será una solución real. La planificación que representa un individuo se obtiene asignando tiempos de inicio a las tareas, a partir de su ordenación en la codificación del individuo, mediante alguna estrategia voraz, como veremos más adelante al hablar de la función de evaluación.

Para facilitar las operaciones de cruce y mutación, se realiza una simplificación sobre la codificación anterior. Se trata de reducir el identificador de cada tarea al identificador del trabajo al que pertenece. Así, el individuo anterior se transforma en *(2 3 2 1 3 1 1 2)*. Esta codificación debe entenderse como que el primer 2 representa a la primera tarea del trabajo 2, el segundo 2 a la segunda tarea del mismo trabajo, y así sucesivamente. La ventaja principal que presenta este esquema de codificación es que permite diseñar operadores genéticos eficientes, de cruce y mutación, que producen cromosomas factibles.

En el caso de PFSS, dado que el orden en que cada recurso procesa todos los trabajos es idéntico para todos los recursos, cada planificación se puede obtener trivialmente a partir de una permutación de los trabajos. Por ello la codificación natural de los individuos del Algoritmo Genético será la de una permutación *([1], ..., [n])* de los trabajos del problema. No se trata, sin embargo, de una simple codificación natural; en [Cot92] se realiza un estudio exhaustivo, mediante un análisis de formas y una posterior confirmación empírica, tanto de distintas representaciones como de una amplia variedad de operadores de recombinación para este problema. La conclusión de este trabajo, en lo que a la representación se refiere es que las características de una solución dependen más de las posiciones absolutas de sus tareas en el cromosoma que de sus órdenes relativos o de sus relaciones de adyacencia, y por ello, la mejor representación es la basada en la posición.

En el caso del problema OSS la representación basada en permutaciones, en este caso del conjunto de las tareas del problema, también resulta ser la más adecuada. Una permutación debe ser entendida de modo que expresa ordenes parciales entre las tareas de cada trabajo y de cada recurso. Evidentemente no todos estos ordenes parciales son compatibles entre si, dado que pueden dar lugar a ciclos en las precedencias de las tareas. De este modo, la función de evaluación deberá modificar alguno de estos órdenes para obtener una solución válida.

En realidad la representación basada en permutaciones es adecuada para otros problemas, por ejemplo el coloreamiento de grafos o el problema del viajante de comercio (TSP). No obstante la semántica de una permutación hay que interpretarla en el contexto de cada problema concreto. Es decir, en unos casos es importante el orden relativo, como en el caso del problema OSS; en otros sin embargo lo que es realmente importante

es la posición absoluta, como parece suceder en el problema PFSS. En otras ocasiones la permutación debe interpretarse como una estructura circular, como ocurre en el problema TSP. En definitiva, el modo de interpretar la permutación influye no solamente en el diseño de la función de evaluación, sino también en la elección de los operadores de cruce y mutación.

Función de evaluación

Para evaluar la calidad de un cromosoma (su *fitness*) se calcula en primer lugar el coste de la planificación asociada y luego el fitness como el inverso de este coste. Además, para obtener una presión selectiva adecuada, en el caso de realizar selección proporcional al fitness, se suele hacer un escalado de la función fitness. Para decidir el tipo de escalado, se debe tener en cuenta la forma de la función fitness, en particular las diferencias relativas entre los mejores y peores cromosomas. En los problemas de Scheduling se suele utilizar un escalado lineal en el que a cada cromosoma de una población se le resta el valor mínimo del fitness de esa población.

Por otra parte, para calcular la solución que representa un determinado cromosoma, si restringimos la búsqueda al espacio de planificaciones activas, se puede utilizar cualquiera de los algoritmos descritos en la sección 3. En el primer caso asumiendo como orden lineal entre las tareas el orden que expresa el cromosoma; y en el segundo caso (algoritmo G&T) eligiendo como siguiente tarea a planificar aquella del conjunto B que aparezca más a la izquierda en la codificación del cromosoma. En este segundo caso, se puede realizar además una reducción del espacio de búsqueda por medio del parámetro δ que, a pesar de que es posible que deje fuera del espacio de búsqueda a todas las soluciones óptimas, en la práctica suele ofrecer buenos resultados.

Operadores genéticos

El objetivo de un operador de cruce es hacer una combinación de dos o más cromosomas para obtener otros nuevos con características heredadas de sus progenitores. Para que un operador de cruce transmita características de los padres a los hijos debe tomar en consideración cuáles son las características relevantes que están codificadas en un cromosoma. Es decir si lo importante es el orden relativo de los genes, su posición absoluta, etc. Este extremo en general no es fácil de determinar de forma precisa, sino que hay que basarse en intuiciones que pueden resultar más o menos acertadas y que por supuesto dependerán del problema concreto. En base a distintas intuiciones existe una gran variedad de operadores de cruce específicamente diseñados para problemas scheduling: GOX, JOX, GPX, LOX, etc. Todos ellos asumen que tanto el orden relativo entre los genes como su posición son relevantes. Por ejemplo el operador LOX (Linear Order Crossover) consiste en lo siguiente: dados dos padres, se selecciona una subcadena del primero y esta subcadena se copia al hijo manteniendo la posición que tiene en el padre. Luego se insertan los genes restantes en las posiciones aun libres manteniendo el orden relativo que estos genes tienen en el segundo padre.

El objetivo de los operadores de mutación es introducir diversidad genética en la población, para ello realizan pequeños cambios de forma aleatoria en la estructura de los cromosomas. En el caso de las representación basada en permutaciones la mutación suele consistir en reordenar de forma aleatoria una subcadena de dos o tres genes, por ejemplo.

La elección de los operadores genéticos más adecuados se suele realizar de forma experimental. No obstante existen algunos métodos más formales. Por ejemplo en [Cot98], C. Cotta propone un método denominado análisis de formas.

A través del análisis de formas o mediante estudios experimentales se puede determinar cuál es el mejor operador a utilizar en cada caso. Por ejemplo, en [Cot98] se propone el PMX, o en general operadores basados en la posición, como operadores que exhiben un mejor rendimiento para problemas PFSS, en [Lia00] se utiliza el LOX para el OSS.

6. Generación heurística de cromosomas

Una de las posibilidades de mejorar el rendimiento de un AG es la generación de cromosomas que incorporen algún tipo de conocimiento heurístico, como alternativa a la generación de cromosomas puramente aleatoria. Lógicamente cabe esperar que si el AG parte de una población inicial compuesta por soluciones buenas, finalmente alcanzará soluciones mejores que si parte de soluciones iniciales no tan buenas. No obstante el sesgo que supone la generación de poblaciones iniciales heurísticas puede producir resultados poco satisfactorios ya que es posible que el AG derive la búsqueda hacia óptimos locales como consecuencia de la dominancia de soluciones semi-óptimas en la población. Para evitar este problema es preciso asegurar que las poblaciones, en particular la inicial, tengan un grado de diversidad aceptable.

La forma más habitual de generación de cromosomas heurísticos consiste en utilizar las soluciones que ofrece algún algoritmo voraz, como por ejemplo las reglas de prioridad. Para obtener una variedad de soluciones lo que se puede hacer es cambiar alguna de las acciones deterministas del algoritmo voraz por acciones probabilistas basadas en algún criterio heurístico. Esta es la aproximación que se propone en [Pue03] para el problema OSS, donde se explota la regla DS/LTRP comentada en la sección 3. En [Val03] se presenta un algoritmo probabilista de generación de cromosomas heurísticos para un algoritmo genético que resuelve el problema de planificación de secuencias de ensamblaje. En este caso se parte de una representación del conjunto de secuencias factibles mediante un grafo *Y/O*, y se calculan secuencias de modo que para cada nodo *O* se elige uno de sus sucesores de tipo *Y* teniendo en cuenta el valor que proporciona una función de estimación de coste optimista. En concreto se seleccionan los nodos para formar parte de una secuencia inicial con una probabilidad inversamente proporcional al valor del coste estimado.

Otra posibilidad es el uso de modelos probabilistas como los heurísticos de ordenación de variables y valores propuestos por N. Sadeh en [Sad96] para el problema JSS. Este modelo permite hacer estimaciones sobre lo prometedora que es a priori una planificación parcial de las tareas que comparten el mismo recurso, y a partir de estas planificaciones parciales se pueden obtener cromosomas heurísticos. Esta es la idea de las aproximaciones que se presentan en [Var03, Pue03] y que ofrecen buenos resultados para una familia de problemas para los que los heurísticos proporcionan información relevante.

7. Búsqueda Local

Con frecuencia los AGs convencionales producen resultados moderados, especialmente en problemas con una gran complejidad. Este hecho es debido en ocasiones a que los operadores genéticos son incapaces de combinar las características de los buenos cromosomas para generar otros de mejor calidad, o al menos de una calidad similar. En estos casos también es frecuente que la eficacia del AG pueda mejorar introduciendo alguna estrategia de búsqueda local, dando lugar así a lo que se suele denominar un Algoritmo Memético.

Para llevar a cabo un método de búsqueda local, el primer paso es definir un criterio de vecindad en el espacio de búsqueda. La vecindad de un punto es el conjunto de cromosomas alcanzables a través de la aplicación de una determinada regla. Una vez generado un cromosoma, este se reemplaza por uno de sus vecinos que satisfaga un determinado criterio de aceptación (normalmente que tenga un fitness mejor). La búsqueda local a partir de un punto finaliza después de un determinado número de iteraciones, o bien cuando en una iteración ninguno de los vecinos cumple el criterio de aceptación.

En el caso del problema JSS, hay una serie de propuestas, por ejemplo las que se presentan en [Mat96; Vae96] que se fundamentan en el concepto de bloque crítico. Un bloque crítico es un conjunto de tareas que aparecen de forma consecutiva en un camino crítico. Y un camino crítico es un camino de longitud máxima entre los nodos *start* y *end* de un grafo solución (figura 2). A partir de un bloque crítico se pueden definir distintos tipos de vecindades que consisten en sustituir al bloque crítico en la solución por una permutación de sus nodos. En [Vae96] se prueba que cualquiera de estas transformaciones proporciona una solución válida y que para que la nueva solución mejore a la original es preciso que al menos uno de los nodos extremos del camino crítico cambie su posición.

8. Estudio experimental

Dado que los AGs son un mecanismo de carácter estocástico y no exacto, su validez como método de búsqueda de soluciones debe ser realizada de forma experimental. En general se deben evaluar no solamente la eficacia y la eficiencia, como en cualquier otro método de búsqueda, sino también la estabilidad por tratarse de un método de naturaleza estocástica.

En el caso de los problemas de Scheduling existen bancos de ejemplos de uso común entre los investigadores, lo cual facilita la comparación de distintos métodos de resolución. La mayoría de estos problemas son accesibles a través del conocido repositorio *OR-library* (http://www.ms.ic.ac.uk/info.html). La tabla I muestra una serie de resultados obtenidos en un estudio experimental, de una versión del AG descrito en las secciones anteriores para el problema JSS, sobre una selección de problemas que son considerados no triviales. Concretamente se trata de un AG con un esquema de selección proporcional al fitness y con un criterio de aceptación según el cual los hijos sustituyen siempre a los padres. Además los parámetros del AG son los siguientes: la probabilidad de cruce es 0,7; la probabilidad de mutación de un cromosoma es 0,2; el tamaño de la población es de 100 cromosomas; y el número de generaciones es 140, de modo que en cada ejecución se evalúan aproximadamente 10.000 cromosomas.

Además se consideran dos variantes del AG el primero sin búsqueda local y el segundo utilizando uno de los esquemas propuestos en [Mat96], concretamente el denominado N_3. En todos los casos se ha utilizado el algoritmo de decodificación G&T híbrido con un valor 0,5 para el parámetro δ. El tiempo aproximado de una ejecución varía según el tamaño del problema: para problemas de tamaño 10×10 es de 2,8 segundos y para problemas de tamaño 15×15 es de 6 segundos. Si no se utiliza búsqueda local, estos tiempos disminuyen aproximadamente un 10%. Estos tiempos han sido tomados en un procesador Pentium III a 750 Mhz.

Para cada uno de los problemas mostramos su identificador en el repositorio, el tamaño, el coste de la solución óptima que es conocida en todos los casos, y a continuación los resultados obtenidos al ejecutar el AG un número de 50 veces para cada problema, cambiando únicamente la semilla del generador de números aleatorios. En primer lugar mostramos el coste de la mejor solución obtenida en las 50 ejecuciones, y lo que es más significativo el error medio de la mejor solución obtenida en cada ejecución, lo cual es una medida de la eficacia del AG, y la desviación estándar obtenida en las 50 ejecuciones que es una buena medida de la estabilidad. La eficiencia del AG se puede medir por el tiempo de una ejecución sobre una determinada máquina, pero también es interesante indicar el número total de cromosomas evaluados en una ejecución, ya que la función de evaluación suele ser el componente más costoso del algoritmo.

Además de valorar el resultado final, en el caso de los AGs es conveniente estudiar la forma en que el algoritmo converge hacia una solución final. Para ello se suelen monitorizar los valores de los costes medio y mejor de las soluciones en cada generación. Es importante que en media se produzca una mejora de estos valores al ir avanzando en las sucesivas generaciones. En la práctica si el AG no está bien parametrizado, lo que suele ocurrir es que no se produce convergencia, o bien que se produce una rápida convergencia inicialmente, seguida de un estancamiento como resultado de que el AG es incapaz de obtener mejores soluciones. Este último fenómeno se conoce como convergencia prematura.

Tabla I. Resultados obtenidos con el AG descrito sobre una selección de problemas JSS no triviales. En cada caso se muestra la mejor solución encontrada en la realización de 50 experimentos con cada problema, así como el error medio porcentual EM con respecto a la mejor solución conocida, y la desviación estándar porcentual.

				Sin Búsqueda Local			Con Búsqueda Local		
Problema	Tamaño		Mejor Solución Conocida	Mejor	EM%	DE%	Mejor	EM%	DE%
	N	M							
FT10	10	10	930	949	5,54	1,45	949	4,38	1,55
FT20	20	5	1165	1178	5,27	2,02	1178	4,19	1,58
abz7	20	15	665	685	2,23	1,03	693	2,16	0,67
abz8	20	15	670	707	8,04	1,12	708	7,36	0,84
abz9	20	15	686	725	13,17	1,87	735	13,38	1,40
la21	15	10	1046	1083	5,57	1,05	1074	4,89	1,15
la24	15	10	935	963	5,73	1,30	950	4,80	1,56
la25	15	10	977	990	4,34	1,20	993	4,00	1,00
la27	20	10	1235	1271	5,19	1,10	1277	5,54	1,10
la29	20	10	1153	1215	8,21	1,13	1217	8,29	1,26
la38	15	15	1196	1288	10,32	1,49	1253	8,07	1,45
la40	15	15	1222	1252	6,05	1,22	1252	4,51	1,00

9. Consideraciones finales

Los algoritmos evolutivos, en particular los algoritmos genéticos, son una de las estrategias clásicas en la resolución de problemas CSP. Aquí hemos comentado algunas experiencias de aplicación a una familia particular de estos problemas: los problemas de Scheduling.

En los últimos años se han publicado numerosos trabajos que muestran que la combinación de los algoritmos evolutivos con otras estrategias heurísticas como la búsqueda local, el enfriamiento simulado, la búsqueda tabú, la ramificación y poda, etc, deben considerarse entre las aproximaciones más competitivas a problemas como el FSS, el OSS y el JSS. Recientemente (véase por ejemplo el número especial de la revista European Journal of Operational Research, número 149 (2003) titulado Sequencing and Scheduling, editada por P. Brucker y S. Knust) aparecen nuevos problemas de interés que suponen nuevos retos para los investigadores. Se trata de problemas muy próximos a la realidad y que tienen una complejidad mucho mayor todavía que problemas como el JSS clásico. Un paradigma de estos nuevos problemas es el conocido como Resource Constrained Project Scheduling Problem (RCPSP). Se trata de una generalización del problema JSS en la que, por ejemplo, hay recursos renovables (como una herramienta) y no renovables (como la energía), o que las actividades de los trabajos pueden realizarse en máquinas distintas y dependiendo de la máquina utilizada el consumo de recursos no renovables puede variar. Además existen restricciones temporales más fuertes que las simples restricciones de precedencia y capacidad.

Sin duda, estos y otros problemas similares, por su gran interés actual, proporcionan nuevas líneas de investigación para las que la aplicación de los algoritmos evolutivos se presenta como una de las aproximaciones más prometedoras.

Referencias Bibliográficas

[Ado90] Adorf, H. M. and Johnston, M. D. A discrete stochastic neural network algoritm for constraint satisfaction problems. Proc. of the International Joint Conference on Neural Networks. San Diego. 1990.

[Bie95] Bierwirth, C.: A Generalized Permutation Approach to Jobshop Scheduling with Genetic Algorithms. OR Spectrum, Vol. 17 (1995) 87-92.

[Bie99] Bierwirth, C. and Mattfeld D. C. *Production Scheduling and Reschedunling with Genetic Algorithms*. Evolutinary Computation 7(1): 1-17. 1999.

[Bru97] Brucher, P., Hurink, J., Jurish, B. and Wostmann, B., A branch and bound algorithm for the open-shop problem, Discrete Applied Mathematics 76 (1997) 43-59.

[Cra00] Craenen B.G.W., Eiben A.E., Marchiori E. Slñoving Constrain satisfaction problems with heuristic-based Evolutionary Algorithms. In Proceedings of CEC'2000, pp.1571-1577, IEEE Press (2000)

[Cot98] Cotta C., Troya J.M, Genetic Forma Recombination in Permutation Flowshop Problem. Evolutionary Computation, 6(1), pp. 25-44, 1998

[Dor95] Dorndorf, U., and Pesch, E., Evolution based learning in a job shop scheduling environment, Computers & Operations Research, Vol. 22 25-40. 1995.

[Eib98] Eiben A. E:, van Hermet J.J., Marchiori E., Steenbeek A. G., Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function. In A. E. Eiben, Th. Bäck, M. Schoenauer, and H.P. Schwefel editors. Proceedings of the 5th Conference on Parallel Problem Solving from Nature, n. 1498 in LNCS, pp. 196-205, Springer (1998).

[Fan93] Fang, H.L., Ross, P., and Corne, D., A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems, in: Proceedings of the Fifth International Conference On Genetic Algorithms, 375-382, 1993.

[Gon76] Gonzales, T. and Sahni, S. Open shop scheduling to minimize finish time, Journal of the Association of Computing Machinery 23 (4), 665-679 (1976).

[Gif60] Giffler, B. Thomson, G. L.: Algorithms for Solving Production Scheduling Problems. Operations Reseach 8 (1960) 487-503.

[Gué98] Guéret, Ch. and Prins, Ch., Classical and new heuristics for the open-shop problem: A computational study. European Journal of Operational Research 107 (1998) 306-314.

[Hen92] Hentenryck, H. Simonis and M. Dincbas. Constraint satisfaction using constraint logic programming. Artificial Intelligence 58, 113-159. 1992.

[Jai99] Jain A. S. and Meeran S. *Deterministic job-shop scheduling: Past, present and future.* European Journal of Operational Research 113 (390-434). 1999.

[Kol99] Kolonko, M., Some new results on simulated annealing applied to the job shop scheduling problem, European Journal of Operational Research 113, (123-136), 1999.

[Lia98] Liaw, Ch-F., An iterative improvement approach for the nonpreemtive open shop scheuling problem, European Journal of Operational Research 111 (1998) 509-517.

[Lia00] Liaw, Ch-F., A hybrid genetic algorithm for the open shop scheduling problem, European Journal of Operational Research 124 (2000) 28-52.

[Mac93] Maccarthy, B.L. and Liu, J., Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. International Journal Production Research, Vol. 31, No 1, 59-79. 1993.

[Mat95]Mattfeld, D. C., Evolutionary Search and the Job Shop. Investigations on Genetic Algorithms for Production Scheduling. Springer-Verlag, November 1995.

[Now99] Nowicki E.; "The Permutation flow shop with buffers: A tabu search approach" European Journal of Operational Research 116, 205-219 (1999)

[Pin95] Pinedo, M., Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, Enblewoodd Cliffs, NJ. (1995).

[Pue03] Puente J., H. R. Díez, H. R., Varela, R. , Vela, C. R., y Hidalgo, L. P. Combinación de Reglas Heurísticas y Algoritmos Genéticos para el Problema Open Shop Scheduling. Aceptado en CAEPIA'03. San Sebastián, Nov. 2003.

[Raj01] Rajendran, Ch., Ziegler, H. A performance Análisis of Distpaching rules and heuristic in static flowshops with missing operations of jobs. European Journal of Operational Research, 131, 622- 34. 2001.

[Sab99] Sabuncuoglu, I: and Bayiz, M., Job shop scheduling with beam search. European Journal of Operational Research, 118, 390-412. 1999.

[Sad96] Sadeh N. and Fox M. S., Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Artificial Intelligence 86, pp. 1-41. 1996.

[Sto92] Storer, R.H., Wu, S.D., and Vaccari, R., *Local Search in Problem Space for* Sequencing *Problems*, In: Fandel, G, Gulledge, T., and Jones, A., New Directions for Operations Research in Manufacturing, Springer Verlag, Berlin, Heidelberg, 1992, 587-597.

[Sys91] Syswerda, G., Schedule Optimizacion Using Genetic Algorithms, in: Handbook of Genetic Algorithms, Ed. L. Davis, Van Nostrand Reinhold, New York, 332-349, 1991.

[Tai93] Taillard, E.: Bechmarcks for basic scheduling problems. European Journal of Operational Research, Vol 64 (1993) 278-285.

[Vae94] Vaessens, R. J. M., Aarts, E. H. L., Lenstra, J. K., Job Shop Scheduling by Local Search. Memorandum COSOR 95-95, Eindohoven University of Technology, Netherlands.

[Val03] Del Valle, C., Gasca, R. M., Toro, M., A Genetic Algorithm for Assembly Sequence Planning, LNCS, Springer, pp. 345-352 (2003).

[Var00] Varela, R.; Vela, C.R.; Puente, J.; Alonso, C.L.*, Parallel Logic Programming For Problem Solving*, International Journal of Parallel Programming, Vol 28, Num 3, pp. 275-319, Junio 2000.

[Var03] Varela, R., Vela, C. R., Puente, J., Gómez A.: A knowledge-based evolutionary strategy for scheduling problems with bottlenecks. European Journal of Operational Research, Vol 145 (2003) 57-71.

[Yam97] Yamada, T., Reeves C.R., "Permutation Flowshop Scheduling by Genetic Local Search" Proceedings of 2nd IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems (GALESIA'97), 232-238 (1997);

[Yam98] Yamada T., Reeves-C.R., "Solving the Csum Permutation Flowshop Scheduling Problem by Genetic Local Search" Proceedings of IEEE International Conference on Evolutionary Computations (ICEC'98), 230-234 (1998)

[Zwe92] Zweben, M. Davis, E., Daun, B., Drascher, E., Deale, M. and Eskey, M., Learning to improve constraint-based scheduling. Artificial Intelligence 58, pp. 271-296. 1992.