V Congreso de Ingeniería de Organización
Valladolid-Burgos, 4-5 Septiembre 2003

Índice

Desarrollo de un programador de tareas de un horno de carburización mediante programación basada en restricciones

Miguel Gutiérrez[1], Jorge del Pino[2]
[1] Ingeniero Industrial, mgfernan@ing.uc3m.es
[2] Ingeniero Industrial, jdelpino@results-toc.com
Área de Ingeniería de Organización. Universidad Carlos III de Madrid
Escuela Politécnica Superior. Avenida de la Universidad, 30. 28911 Leganés (Madrid)

RESUMEN

En esta comunicación se presenta el desarrollo de un programador de tareas, realizado esencialmente con el módulo de programación basada en restricciones y la utilidad de programación de tareas del entorno ILOG OPL Studio. El problema recoge la situación real de una planta de fabricación dedicada a la producción de barras de perforación para la industria minera. En el proceso de producción de esta planta, el horno de carburización constituye el cuello de botella. En él confluyen multitud de restricciones y posibilidades de carga, que dan como resultado un complejo problema de optimización combinatoria. Para su modelado se saca partido del enorme potencial de la programación basada en restricciones, una técnica de resolución de problemas de optimización con creciente utilización en la última década. La aplicación se realiza con el mencionado producto comercial de ILOG, que ofrece un entorno integrado para desarrollar aplicaciones de programación basada en restricciones, conjuntamente con modelos y algoritmos de programación lineal, y cuenta además con utilidades específicas para problemas de programación de tareas. También se ha hecho uso de la posibilidad que brinda el entorno de intercambio de datos en tiempo de ejecución con bases de datos externas, de manera que el programador que se presenta se puede integrar con el sistema de información de la empresa.

***Palabras clave**: Programación de la Producción, Programación Basada en Restricciones (Constraint Programming), ILOG OPL Studio.*

1. Introducción

El presente trabajo surge a raíz de una estancia en el *Department of Manufacturing and Operations Engineering* de la Universidad de Limerick (Limerick, Irlanda). Allí se tomó contacto con la problemática de una industria dedicada a la producción de material de perforación para la industria minera, y en particular con la planta de fabricación de barras endurecidas para ser adjuntadas a las cabezas perforadoras. En un estudio previo realizado en dicho departamento, habían llegado a la conclusión de que el causante principal de los problemas operativos de la planta es el horno de carburización, identificado como el cuello de botella del proceso de producción.

En el horno de carburización de esta industria, confluyen un gran número de restricciones y de posibilidades de carga, de manera que la programación de tareas da lugar a un problema de optimización combinatoria de los llamados *NP-hard* [1]; es decir, que el tiempo de resolución crece más que polinómicamente con el número de variables, en este caso con el número de pedidos de tubos a programar.

En el interés por explorar las posibilidades que ofrece la programación basada en restricciones para este tipo de problemas [2], y en particular el entorno comercial OPL Studio de la casa ILOG, se ha desarrollado un programador de tareas para el horno. Es importante notar que el desarrollo se ha realizado con la versión de evaluación del mencionado entorno, que tiene iguales posibilidades de modelado y algorítmicas que la versión comercial, pero con una limitación fundamental en el número de restricciones y variables a considerar. Por ello, se presenta el desarrollo íntegro hasta la etapa de verificación, pero sin llegar a la experimentación con problemas de tamaño real.

Se comienza por resumir las características principales de la programación basada en restricciones y seguidamente del entorno ILOG OPL Studio. A continuación se expone el problema a resolver y las restricciones fundamentales a tener en cuenta, para pasar a describir el desarrollo realizado con una breve explicación de los módulos principales. Finalmente, se comentan las conclusiones principales que se desprenden del trabajo.

2. Programación basada en restricciones

En su artículo introductorio, Lustig y Puget apuntan que el origen de la programación basada en restricciones se puede encontrar en la década de los años 70 con los problemas de satisfacción de restricciones [3]. Un problema de satisfacción de restricciones se define como compuesto por un conjunto finito de variables, cada una de las cuales lleva asociado un dominio, y un conjunto de restricciones que reducen los valores que pueden tomar simultáneamente las variables. El objetivo consiste en asignar a cada variable un valor de su dominio, satisfaciendo todas las restricciones [4].

En los años 80, desde el ámbito de la inteligencia artificial, se produjo la evolución de la programación lógica, en particular del lenguaje Prolog, a la llamada programación lógica basada en restricciones. Estos lenguajes proporcionan funciones o predicados para describir las restricciones más habituales, y ayudan al usuario a resolver problemas generales aplicando técnicas surgidas de la investigación de los problemas de satisfacción de restricciones [5]. La principal innovación de la programación lógica basada en restricciones frente a la programación lógica, radica en la manipulación de las restricciones; esto es, el enfoque de los algoritmos de resolución. Las restricciones son ecuaciones entre términos, representadas internamente como ligaduras entre variables. La resolución de un problema se realiza por asignación de valores a todas las variables, de manera que a medida que se va seleccionando un valor para cada una de las variables, se van eliminando los valores inconsistentes de los dominios de las variables aún no asignadas. Si en el proceso de resolución alguno de los dominios queda con un único valor, su variable queda automáticamente asignada con ese valor. Si en el proceso de resolución alguno de los dominios queda vacío, el proceso ha fallado, y se produce una vuelta atrás hacia la última situación en que se asignó una variable, restaurando los dominios al estado previo a esa situación de fallo, y continuando la resolución con una selección de valor diferente.

En la década de los 90, el interés suscitado por la programación lógica basada en restricciones provocó el desarrollo, con la misma filosofía, de librerías y algoritmos en lenguajes de propósito general como el C++ de ILOG Solver, dando lugar, de acuerdo a Lustig y Puget, a la programación basada en restricciones. El ILOG Solver está incluido en el entorno de desarrollo utilizado en el presente trabajo, ILOG OPL Studio. ILOG es la empresa líder en

componentes software para empresa, y su utilización en los sistemas informáticos de gestión es muy amplia. Está contando con una aceptación creciente en el ámbito del soporte a la planificación y programación de la producción. Destaca su utilización con estos propósitos en el módulo Advanced Planning and Optimization (APO) de la empresa SAP [6].

3. Entorno ILOG OPL Studio

ILOG OPL Studio es un entorno de programación que integra diferentes módulos de ILOG, con el objetivo de facilitar el desarrollo rápido de herramientas para optimización de problemas de gestión. En el corazón del entorno, se encuentran dos motores de resolución algorítmica [7]:

- ILOG CPLEX: incluye algoritmos de programación lineal, entera y mixta.
- ILOG Solver: incluye el motor de resolución de programación basada en restricciones.

Además, el entorno incluye ILOG Scheduler [8], un módulo específico dedicado a problemas de programación de tareas con distintas utilidades y mejoras algorítmicas para este tipo de problemas.

Entre las ventajas del ILOG OPL Studio, destacan las siguientes:

- Utiliza un único lenguaje de modelado, OPL (*Optimization Programming Language*, lenguaje de programación para optimización) [9], cuya sintaxis permite integrar sentencias propias de programación lineal con sentencias de programación basada en restricciones, incluyendo en estas últimas las posibilidades específicas para problemas de programación de tareas.

- Incluye también el lenguaje OPL Script [10] para la generación de módulos de secuencias de comandos (*scripts*), que permite componer y controlar distintos modelos escritos en lenguaje OPL. Surge motivado por el interés de resolver, de una manera razonablemente sencilla, la necesidad frecuente de ejecutar múltiples ejemplos de un mismo problema (para análisis de sensibilidad por ejemplo), secuencias de distintos modelos, o ambos combinados. Además, facilita la comunicación con bases de datos y hojas de cálculo.

- Separa los modelos del problema por un lado y los datos por otro, de forma que se facilitan las pruebas, modificaciones, mantenimiento y ampliaciones.

- Proporciona conectividad con bases de datos (posibilidad de lectura de datos en tiempo de ejecución) y hojas de cálculo (posibilidad de combinar e intercambiar información).

- Genera objetos para C++, Visual Basic, Java.

4. Descripción del problema

La planta de fabricación que se considera, fabrica barras que posteriormente se adjuntarán a las cabezas perforadoras usadas para trabajos en minas y pozos. El proceso de producción parte de una barra de acero y sigue un esquema genérico en varias etapas: forjado, recocido,

soldado de extremo por fricción, roscado en torno, precarburización, carburización en horno, enfriamiento en torre, templado, enfriado al aire, enderezamiento, limpieza de óxido, revestimientos y acabado, y control de calidad.

A partir del proceso genérico, en función del tipo de barra, dimensiones y de diversas variantes en las etapas, surge el catálogo de productos que la empresa oferta. Las variantes son numerosas, de modo que llega a haber más de 800 tipos de ruta diferentes. Los productos se agrupan en cinco familias: barras de perforación, barras macho/hembra, extensiones, barras cónicas y tubos. Además, a excepción de los tubos, que siempre parten de barras de sección circular hueca, el resto de las barras pueden ser de sección hexagonal o redonda en cada una de las familias.

El interés del programador se centra en la etapa de carburización en el horno. La carga del horno se realiza por lotes de barras en las llamadas plantillas de carburización, que consisten en columnas de 4,6 metros de altura con una estructura adjunta de sección hexagonal, según se muestra en la figura 1. La estructura está dividida en seis secciones, dentro de cada una de las cuales se ponen bandejas en forma de panal a distintas alturas. La capacidad de las bandejas varía dependiendo del diámetro de los tubos que se introducen. Así, sólo se pueden poner 3 barras por bandeja si el diámetro es grande (66mm., 76mm., 87mm.), mientras que se pueden poner hasta 29 barras por bandeja si el diámetro es pequeño (19mm., 22mm., 25mm.). Por otra parte, se conoce el peso por longitud de barra para cada tipo de sección (hexagonal, redonda o tubo) y diámetro.

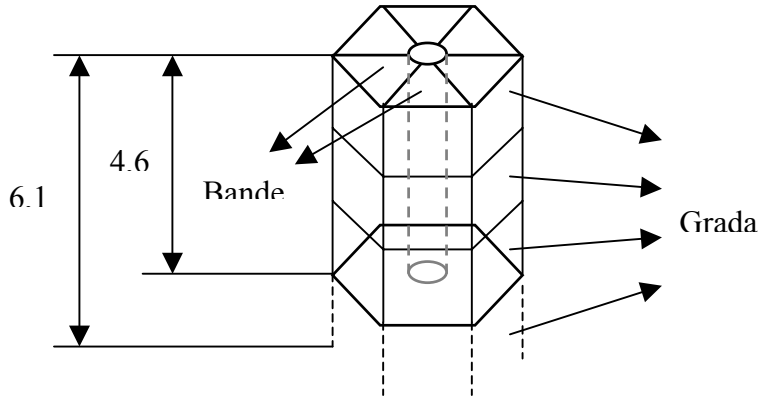

Figura 1: Columna de carburización

Las barras quedan colgadas verticalmente en cada sección, y el conjunto de cada bandeja con las barras y la estructura de unión con la columna se denomina grada. La empresa ofrece barras entre 30,5cm. (1 pie) y 6,1m (20 pies) de longitud, que es la altura del horno. El número de gradas varía entre 1 y 4, dependiendo de la longitud de las barras, de manera que queda espacio para barras de 1,5 m. (5 pies) o inferiores colgando de la grada inferior (la diferencia entre la altura del horno y la de la columna). De este modo, si la longitud de las barras a carburizar es superior a 3,05m (10 pies) sólo podrá ponerse una grada, y si la longitud es igual o inferior a 1,52m (5 pies) podrán ponerse 4 gradas.

Hay diferentes tipos de tratamientos en el horno preestablecidos, que varían principalmente en el tiempo de carburización. Aunque no todos los tratamientos son aplicables a todas las familias de barras, a cada familia se le pueden aplicar varios tratamientos y cada tratamiento

es aplicable a varias familias. Cuando el cliente hace el pedido, se determina cuál de los tratamientos tabulados requieren las barras de dicho pedido, y se almacena el identificador de tratamiento correspondiente como atributo del pedido en la tabla de la base de datos. Es posible mezclar barras diferentes en el horno, siempre que el tratamiento sea el mismo.

En cuanto a los turnos de operación, la planta trabaja de acuerdo a tres turnos de 8 horas diarias de lunes a jueves, y tres turnos de 7 horas el viernes. Los recursos trabajan más o menos turnos en función de su carga. El horno entra dentro de los que trabajan los tres turnos de lunes a viernes, haciendo horas extraordinarias los fines de semana cuando el cumplimiento de los pedidos lo exige. Uno de los problemas que motivan al desarrollo del programador, es el elevado número de horas extras en que se incurre actualmente.

5. Definición del problema

5.1 Requerimientos

El requerimiento funcional básico de partida consiste en que la aplicación desarrollada, a partir de los datos de los pedidos recibidos por la empresa (cantidad de barras, tipo de barra, diámetro, longitud, tipo de tratamiento, fecha de entrega) proporcione la programación del horno carga a carga, de manera que se sepa en todo momento el tipo de barra a introducir, la cantidad y la posición en la columna (grada).

Como requerimientos funcionales adicionales destacan dos: por un lado, a partir de la programación del horno, se debe obtener el momento en el que hay que introducir en el proceso productivo cada barra para que llegue a tiempo (*backward scheduling*); por otro, la aplicación debe ser capaz de integrarse con el sistema de información de la empresa, por lo que los datos de los pedidos y los relativos a las características de los tubos deben leerse de una base de datos en tiempo de ejecución.

5.2 Restricciones

El modelo considera dos tipos de restricciones:

- Restricciones técnicas. Provienen de las limitaciones del horno y de las plantillas de carburización en cuanto al peso máximo de carga (4.500 kg), número de gradas (depende de la longitud de las barras), número máximo de barras por bandeja (depende del diámetro de las barras; las bandejas sirven para varios diámetros parecidos) y tratamiento de carburización (diferente según el pedido).

- Restricciones temporales. Provienen de la imposición de cumplimiento de las fechas de entrega. No todos los tubos de un pedido deben ser procesados necesariamente en el mismo lote y se permite almacenamiento previo al envío.

5.3 Objetivo

La función a optimizar consiste en la minimización de horas extra. No se permite el retraso, por lo que no hay penalizaciones por retraso y se considera despreciable el coste de almacenamiento frente a las horas extra.

5.4 Hipótesis simplificadoras

- Dado que el interés del trabajo se centra en el horno, que según el mencionado estudio previo constituye el cuello de botella y, de acuerdo al mismo estudio, tiene un tiempo de proceso significativamente superior a las demás operaciones, se ha considerado que el resto de recursos tiene capacidad infinita. Esta consideración afecta a la programación hacia atrás (*backward scheduling*), que se hace a partir de la programación del horno con objeto de conocer la fecha en la que hay que comenzar la primera operación de un pedido, y a la programación hacia delante (*forward scheduling*) que permite conocer la fecha de finalización de las operaciones correspondientes a un pedido. Se trata por tanto de una simplificación para, con la programación del horno, establecer la programación de toda la planta. La realidad es más compleja y los resultados sólo tienen carácter orientativo.

- Se ha considerado que el horno, trabajando a máxima capacidad (con el máximo de horas extras), puede procesar todos los pedidos sin incurrir en retrasos. Se asume que la empresa no compromete fechas imposibles. Esta simplificación se podría eliminar mediante una posibilidad que brinda la programación basada en restricciones, de definir como "restricciones suaves" (*soft constraints*) las de cumplimiento de fechas de entrega, de manera que el algoritmo tratase de cumplirlas, pero pudiera relajarlas en caso de necesidad. En la función objetivo se incluirían penalizaciones por retraso, que podrían tener ponderaciones en función del pedido.

- Se supone una capacidad de almacenaje infinita al final del proceso, de manera que no hay limitación a terminar los pedidos antes de la fecha de entrega.

6. Diseño e implementación

El desarrollo se ha realizado con la versión OPL Studio 3.5, que contiene el Solver 3.5, Scheduler 5.1 y el CPLEX 7.1 (este último no utilizado). El programador consta de los siguientes modelos, que funcionan de acuerdo al esquema de la figura 2:

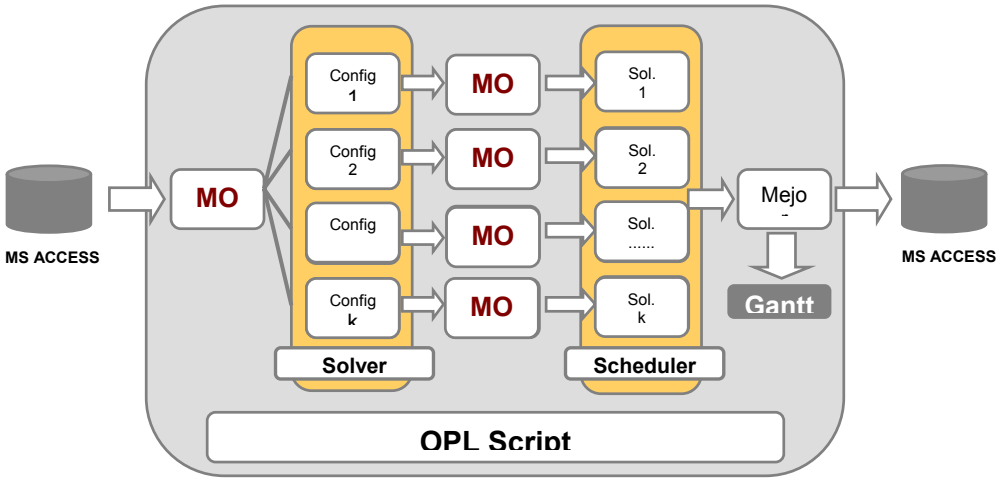

Figura 2: Esquema general del programador

- Modelo de generación de configuraciones (MOD 1). Está implementado en lenguaje OPL, utilizando el módulo de ILOG de programación basada en restricciones (*Solver*). Una configuración es una carga factible del horno en cuanto a restricciones técnicas. El módulo genera un conjunto de configuraciones, de manera que todas las barras a entregar pertenecen a alguna, especificando cuántas barras de cada tipo se deben incluir en cada grada.

- Modelo de secuenciación (MOD 2). Está implementado en lenguaje OPL, utilizando la utilidad de ILOG específica para programación de tareas (*Scheduler*). A partir de cada conjunto de configuraciones, genera la secuencia óptima de manera que se cumplan las restricciones de entrega y los turnos, y buscando minimizar el número de horas extra. Permite realizar las horas extra como continuación del tercer turno del viernes (5:00 AM) o como un nuevo turno el sábado (8:00 AM), con interrupción de tres horas. Para el guiado de la búsqueda, las tareas se han ordenado siguiendo la regla heurística de "menor holgura total", restando a la fecha de entrega, la fecha de comienzo más temprana posible y la duración del proceso. En caso de igualdad, se toma la menor fecha de entrega.

- Modelo global. Es el modelo en lenguaje OPL Script que integra los dos anteriores e interacciona con la base de datos, según se ilustra en la figura 3. Calcula las operaciones que hay que realizar antes y después del horno y permite personalizar la salida de datos.

- Modelo Gantt. Genera un diagrama de Gantt con la carga de trabajo del horno a partir de la solución almacenada en la base de datos. En la figura 4 se muestra un ejemplo, en el que también se puede observar el entorno de desarrollo OPL Studio y las facilidades que ofrece en cuanto a navegación por los módulos.

En cuanto a los datos, se ha utilizado las dos posibilidades que ofrece el entorno. Por un lado, los datos del horno, que son por naturaleza más estables, se han almacenado en un fichero que lee el modelo global al ejecutarse. Por otro, se ha desarrollado una base de datos en MS Access 2000, cuyo diseño tiene en cuenta los distintos tipos de barras y procesos, y que incluye un formulario de introducción de nuevos pedidos. El acceso a la base de datos, tanto para lectura como para escritura de la solución, se realiza a través del módulo global, usando el interfaz estándar para conexiones con bases de datos ODBC (*Open Data Base Connectivity*), versión 3.1. La ventaja de utilizar este estándar consiste en la posibilidad que se genera de utilizar el desarrollo con otro sistema gestor de bases de datos relacional, los más extendidos (Oracle, SQL Server, etc.), lo que hace factible la integración del programador con el sistema de información de la empresa.

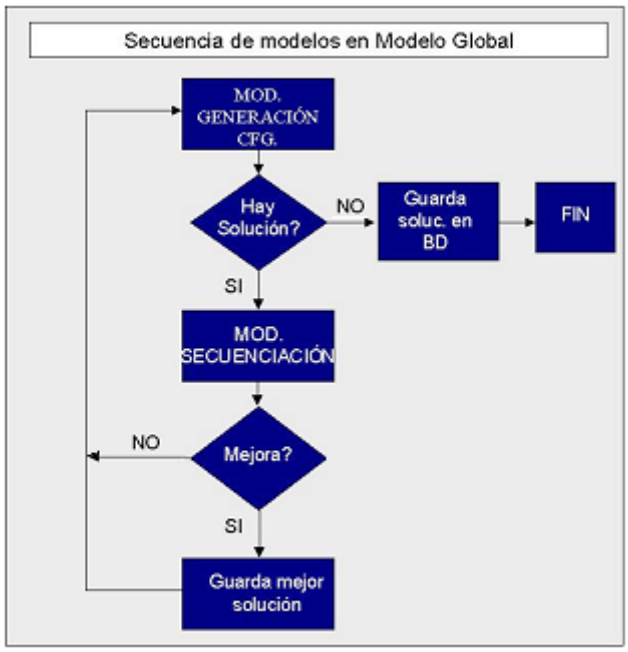

Figura 3: Representación esquemática del modelo global

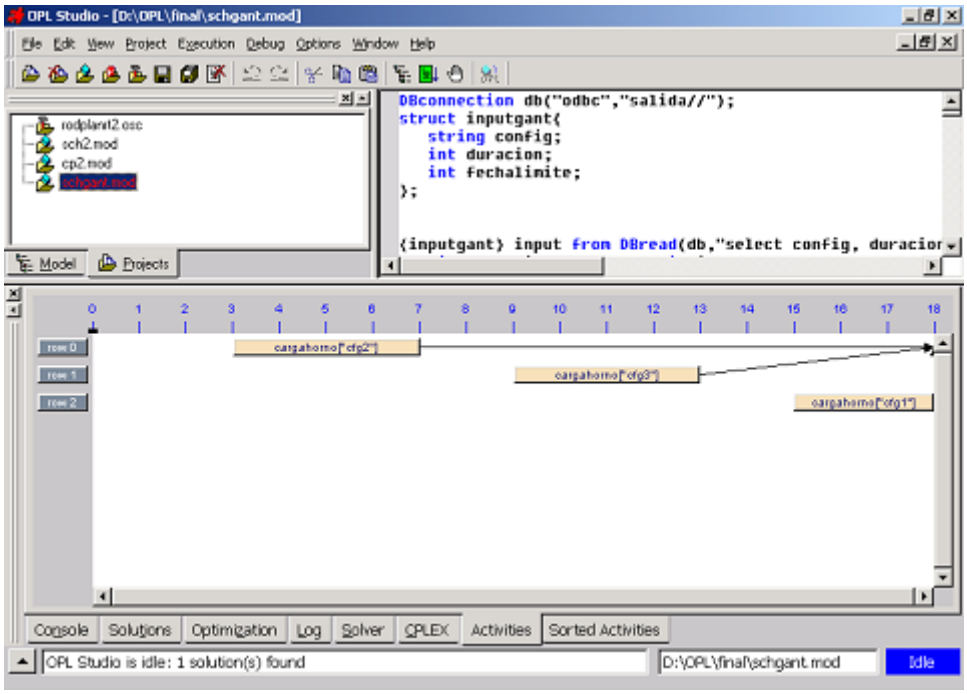

Figura 4: Módulo Gantt en el entorno OPL Studio

7. Conclusiones

Se ha presentado el desarrollo de un programador de carga de un horno de carburización presente en una planta dedicada a fabricar barras de acero para perforación en minas. Se ha

utilizado el entorno ILOG OPL Studio, que ha demostrado gran potencia en cuanto a facilidad y flexibilidad en el modelado. La posibilidad de integración con bases de datos externas añade atractivo a esta herramienta. Se ha comprobado la ventaja que supone la característica de la programación basada en restricciones de separar la definición del problema del algoritmo de resolución. En tiempo real se generan las restricciones necesarias en función de los datos que se leen de ficheros o de la base de datos. Las limitaciones de licencia no han permitido experimentar con problemas de tamaño real, si bien la experiencia con la programación basada en restricciones hace augurar un comportamiento algorítmico pobre al aumentar el número de pedidos a programar. No obstante, siempre se cuenta con una solución factible, cuya obtención se puede guiar mediante el uso de heurísticos. En la implementación se ha probado una estrategia simple de ordenación de las barras por su holgura, pero quedan por explorar, como futura ampliación del trabajo, otras posibilidades que ofrece el lenguaje OPL en este sentido [11].

Agradecimientos

La realización del presente trabajo ha sido posible gracias a la información aportada por el profesor Dr. Cathal Heavey del Department of Manufacturing and Operations Engineering de la Universidad de Limerick (Limerick, Irlanda).

Referencias

[1] Garey M.R. y D.S. Johnson, (1979) Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman.

[2] Baptiste, P., C. Le Pape y W. Nuijten, (2001) Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems, Kluwer Academic.

[3] Lustig, I.J. y J.F. Puget (2001) "Program does not equal program: Constraint Programming and its relationship to Mathematical Programming", *Interfaces*, vol. 31, no. 6, pp. 29-53.

[4] Tsang, E.P.K., (1993) *Foundations of Constraint Satisfaction*, Academic Press.

[5] Jaffar, J. y M.J. Maher, (1994) "Constraint Logic Programming: A Survey", *Journal of Logic Programming*, vol. 19/20, pp. 503-581.

[6] Schierholt, K., (2002) "A working partnership", *SAP INFO*, no. 96, http://www.sap.info/resources/RFILE225743d2ae2750f53d.pdf.

[7] ILOG S.A., (2001) ILOG Optimization Suite White Paper, http://www.ilog.com/products/optimization/tech/optimization_whitepaper.pdf.

[8] Nuijten, W.P.M y C. Le Pape, (1998) "Constraint-Based Job Shop Scheduling with ILOG Scheduler", *Journal of Heuristics*, vol. 3, no. 4, pp. 271-286.

[9] Van Hentenryck, P., (1999) *The OPL Optimization Programming Language*, The MIT Press.

[10] Van Hentenryck, P. y L. Michel, (2000) "OPL Script: Composing and Controlling Models". En: K.R. Apt, A.C. Kakas, E. Monfroy, F. Rossi (Eds.), *New Trends in Constraints*, Lecture Notes in Artificial Intelligence (LNAI 1865), Springer Verlag, pp. 75-90.

[11] Van Hentenryck, P., L. Perron, J.F. Puget, (2000): "Search and strategies in OPL", ACM Transactions on Computational Logic, vol. 1, no. 2, pp. 282-315.