Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial
ISSN: 1137-3601
revista@aepia.org
Asociación Española para la Inteligencia Artificial
España

Morón Serna, Antonio; Sastrón Báguena, Francisco
Programación Basada en Restricciones: la Programación Simbólica al Servicio del Control de Producción Inteligente
Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial, vol. 4, núm. 10, verano, 2000, pp. 82-93
Asociación Española para la Inteligencia Artificial
Valencia, España

Programación Basada en Restricciones: la Programación Simbólica al Servicio del Control de Producción Inteligente

Antonio Morón Serna, Francisco Sastrón Báguena

DISAM – Universidad Politécnica de Madrid
amoron@disam.upm.es, sastron@disam.upm.es

El problema del control de la producción en ambientes de fabricación orientada a proyectos es el caso más general y complejo de los sistemas productivos discretos. En él se unen la singularidad y tamaño de los productos, la incertidumbre en la ingeniería y la fabricación, la cantidad de restricciones de los recursos productivos y la cantidad de restricciones derivadas de las condiciones de contratación por parte de los clientes. La programación simbólica se sitúa por delante de la programación procedural a la hora de definir y resolver los escenarios de programación y control de la producción, y la programación basada en restricciones es una especialización de los lenguajes de programación lógica que permite coexistir deducciones lógicas con ciertos conocimientos expertos para una reducción de los espacios de búsqueda de soluciones que sufren de explosiones combinatorias. Presentamos en este artículo los conceptos y ventajas más importantes de la programación basada en restricciones para la planificación, la programación y el control de la producción inteligente.

Introducción

El presente trabajo tiene como objetivo probar el potencial innovador de diferentes tecnologías en las aplicaciones informáticas de apoyo a la programación de la producción para la fabricación a medida. El trabajo se ha realizado en el marco del proyecto PPES, "Diseño Avanzado de Aplicaciones de Programación de la Producción para Ámbitos Específicos", financiado por la CICYT (TAP96-0604).

Se ha buscado la innovación en los tres componentes de una aplicación informática: los algoritmos, las interfases de usuario y los sistemas de información, materializándose los resultados en un prototipo.

De las tres áreas mencionadas, se presenta en este artículo lo referente al área algorítmica, donde se ha evaluado la utilización de la programación basada en restricciones en la gestión de la producción de la fabricación a medida. El orden de exposición es el siguiente: en el primer apartado se presentan las características de la gestión de la producción en ambientes de fabricación a medida, indicando cuales son las principales restricciones que podemos considerar para su planificación, programación y control; a continuación se presenta la programación basada en restricciones en el área algorítmica de la gestión de la producción; de este paradigma se presentará la herramienta de desarrollo elegida; después se detalla el conjunto de experimentos llevados a cabo con dichas herramientas; a continuación se hace una breve descripción del prototipo completo; y cerrando el documento se enumeran los resultados de la evaluación, finalizando con las conclusiones.

Gestión de Producción

La optimización de los programas de producción y el mantenimiento de los mismos en fabricación a medida es una tarea compleja debido a las continuas desviaciones de la realidad del taller con respecto a los planes

En la fabricación a medida, los productos se diseñan y producen a medida, por lo que en este tipo de fabricación la incertidumbre en los datos de planificación es máxima.

En algunos casos de fabricación a medida, como la fabricación de grandes bienes de equipo ocurre que esos equipos se componen de un gran número de piezas, que pueden no estar totalmente definidas después de iniciarse la fabricación física del producto. Esta característica hace que la programación y el control de la producción se parezcan más a la gestión de proyectos civiles que a la fabricación de bienes más o menos repetitivos basados en subcomponentes catalogados.

En otras palabras, no se puede explotar la lista de materiales para la planificación y la programación porque esta lista no se conoce hasta que se tiene el diseño completo del producto.

La gestión de producción en general requiere de sistemas de planificación, programación y control de los recursos productivos restringidos no sólo en número sino también en disponibilidad a lo largo del tiempo. Entre esos recursos cabe incluir los recursos humanos escasos debido al grado de especialización requerida, como por ejemplo los especialistas en soldaduras críticas. Además de las restricciones en los recursos, encontramos también restricciones en la disponibilidad de los materiales, en la calidad de los mismos, en las fechas y condiciones de entrega del producto, con sus penalizaciones por incumplimiento.

Podemos clasificar las restricciones en dos tipos: duras y blandas. Restricciones duras son aquellas que hay que cumplir en todo caso, como por ejemplo, las precedencias entre operaciones. Restricciones blandas son aquellas que pueden incumplirse o relajarse, si bien a un cierto coste, como por ejemplo la fecha de terminación de determinados pedidos.

La gestión de la producción en general plantea un problema de carácter temporal con un elevado número de variables y sobre ellas un elevado número de restricciones. En el caso particular de la fabricación a medida tenemos además un elevado grado de incertidumbre tanto en la definición de los productos como en la estimación de los datos de producción. Para hacer frente a las incidencias del taller se requiere mucha flexibilidad, no sólo para reasignar los recursos, sino también para reparar el programa de producción en base a un cambio de la ruta de producción, o incluso del propio proceso de producción.

La complejidad de la gestión de la producción se aborda mediante esquemas de tipo jerárquico, dividiéndola en niveles, que se distinguen por su horizonte temporal y por su grado de agregación de los datos manejados. Así se denomina planificación de producción al nivel más elevado, que contempla las actividades que componen un proyecto y los recursos requeridos a nivel agregado ; su objetivo es poder hacer estimaciones de carga-capacidad de la fábrica a nivel agregado, así como permitir el seguimiento del avance de cada proyecto, manteniendo las fechas adecuadas para los suministros de material más relevantes. Se denomina programación de producción al nivel más bajo, que contempla las operaciones de producción tal y como se gestionan y controlan en el taller. Por debajo de este nivel cabe incluir otro, la reprogramación, que se encarga de reparar los programas de produccción en respuesta a las incidencias del taller.

El enfoque que presentamos incluye la consideración integrada de los diferentes niveles, para un control adecuado del avance de la producción. Esta integración se realiza fácilmente gracias a la programación basada en restricciones. La planificación de la producción incluirá la consideración de limitación de capacidad de recursos y las restricciones de requerimientos de material. La programación de la producción incluirá la realimentación de las incidencias y avances de la fábrica. También se considerará la optimización de funciones objetivo, bien de tipo operativo o económico.

En cuanto al control de producción se pretende efectuar acciones correctoras inmediatamente después de la notificación de eventos inesperados en alguna de las operaciones de producción programadas. En este ambiente de producción llamamos inmediatez a los minutos que siguen a determinado evento, antes de que se vean retrasadas otras operaciones de producción a ejecutar dentro del mismo proyecto o dentro del mismo recurso productivo.

La importancia de la reprogramación es la de reparar los programas existentes manteniendo el control de la actividad de la fábrica. Y para ello es fundamental encontrar la solución que minimice los movimientos de las operaciones en el tiempo. Las opciones de utilización de heurísticos son limitadas en este sentido, mientras que la programación basada en restricciones lo consigue de una manera limpia y rápida.

La Programación Basada en Restricciones

La programación basada en restricciones proviene de las investigaciones en demostradores automáticos de los años 60 [Robinson 65]. La programación lógica se aplicó con éxito en numerosos campos como el diseño hardware, las bases de datos o los sistemas expertos [VRoy 94].

En los lenguajes lógicos como Prolog, un programa es un predicado que describe una relación entre valores. Los predicados se definen mediante reglas mutuamente recursivas, y ejecutar un programa lógico significa probar una proposición por generación de ligaduras entre las variables tales que conviertan en cierta una relación. Si existen múltiples ligaduras que satisfacen la relación, la vuelta atrás o los mecanismos de concurrencia pueden llegar a generar todas las posibles soluciones. Conceptualmente, los programas lógicos no distinguen entre parámetros de entrada y parámetros de salida.

La programación lógica sin embargo no era capaz de razonar directamente con expresiones numéricas. Por su parte la programación basada en restricciones (PBR) es una extensión de los lenguajes de programación lógica donde el dominio de discurso no son sólo las estructuras de términos, sino también los dominios numéricos, admitiendo como parámetros de los programas las expresiones aritméticas.

Los dominios de las variables proporcionan la misma información que los predicados en los lenguajes lógicos, pero esta información es manejada a nivel de unificación en lugar de a nivel de resolvente. Por tanto, estas variables son entendidas como variables que toman valores sobre una relación unaria. Alcanzar una solución consiste ahora en hallar una asignación de valores para cada una de las variables del problema.

Nos concentraremos en los problemas de satisfacción de restricciones que pueden ser formulados como sigue [Tsang 93]:

- *tenemos un conjunto de variables, un dominio discreto y finito para cada variable y un conjunto de restricciones, cada restricción está definida sobre un subconjunto de variables y limita la combinación de valores que las variables en este conjunto pueden tomar; el objetivo es encontrar una asignación a las variables tal que se satisfagan todas las restricciones. En algunos problemas el objetivo es encontrar todas las asignaciones.*

La programación lógica proporciona un lenguaje potente para la formulación de problemas combinatorios. Su forma relacional y sus variables lógicas son perfectamente adecuadas para situar los problemas de forma declarativa, y su computación no determinista libera al usuario de la programación de árboles de búsqueda. Sin embargo, hasta ahora, los lenguajes de programación lógica se han usado para resolver sólo pequeños problemas combinatorios, debido a la utilización de búsqueda por prueba y error en los árboles.

La programación basada en restricciones y las técnicas de garantía de consistencia proporcionan paradigmas de resolución de problemas más potentes. La programación basada en restricciones utiliza técnicas de propagación de valores locales combinadas con una computación dirigida por demonios. Las técnicas de consistencia incluyen algoritmos de búsqueda como la propagación de restricciones o la ordenación de variables. La idea general es usar las restricciones para podar el árbol de búsqueda a priori, antes de la generación de valores, en lugar de usarlas como comprobación de error.

Van Hentenryck [VH 89] estudió la introducción de la resolución de restricciones y las técnicas de consistencia dentro de la propia programación lógica, creando CHIP (Constraint Handling In Prolog), que ahora es una herramienta comercial. CHIP es un lenguaje de programación lógica basado en el concepto de restricciones activas. CHIP es comparable en eficiencia con los lenguajes procedurales y permite abordar una serie de problemas que imposibles de resolver con `PROLOG`. Van Hentenryck utilizó CHIP para la resolución de un sencillo problema de programación de tareas [DSVH 90].

La aplicación de la programación basada en restricciones a la programación de la producción comienza a ser abordada de forma realista en los trabajos de los desarrolladores de ILOG Solver [Puget 94][LP 94][NuijAarts 96], en trabajos de la Universidad de Toronto [Fox 93] y de la Universidad Carnegie Mellon [ChengSmi 97], tal y como se presenta en [Jain 98][Jain 99].

La PBR y las bibliotecas ILOG

La compañía ILOG comercializa unas bibliotecas de clases C++ para la creación de programas basados en restricciones. Estas bibliotecas implementan un motor de inferencia para la resolución de restricciones basado en una máquina abstracta. Estas bibliotecas tienen su origen en las creadas en Lisp bajo el paradigma de orientación a objeto, por J.F. Puget, del grupo de investigación de computación simbólica del INRIA [Puget 94].

El modelo de ejecución es sencillo. Un almacén contiene todas las variables lógicas y las restricciones. Cuando el valor de una variable no se conoce se le asocia un dominio de valores.

El sistema de restricciones se simplifica incrementalmente borrando los valores inconsistentes de los dominios de acuerdo con las restricciones. A esto se le llama propagación de restricciones y puede llevar a la resolución del problema, a una contradicción o a un nuevo estado intermedio. Se van hallando unificadores de acuerdo a un programa y una meta.

Un programa es una colección de reglas de la forma a←b1,...,bn donde a es un átomo, o predicado sobre un conjunto de términos, y los bi son átomos o restricciones. Llamamos cabeza de la regla a 'a' y cuerpo de la regla a b1,...,bn. Una meta es una regla que sólo tiene cuerpo, es decir una conjunción de restricciones y átomos. Cumplir una meta es dar valores a las variables que intervienen.

La propagación de restricciones es incompleta en el sentido de que no es capaz de descubrir todas las inconsistencias [BessFreu 99], pero en cambio permite disponer de soluciones en tiempo polinomial por medio de búsquedas no deterministas. Las soluciones se buscan aplicando repetidamente los pasos siguientes:

- Poner un punto de selección.
- Asignar un valor a una variable de dominio.
- Aplicar el algoritmo de propagación de restricciones.

Si esto nos lleva a una contradicción se produce una vuelta al punto de selección, deshaciendo la asignación y sus consecuencias para probar otra asignación.

Prolog ofrece algunas construcciones que no están presentes en C++:

- Variables lógicas.
- Gestión de memoria como una pila.
- Mecanismos de retardo utilizados para codificar la propagación de restricciones.
- No determinismo.

Para codificar estas construcciones en C++:

- Una variable lógica es un objeto que incluye el valor de la variable cuando se conozca, la lista de restricciones que le afectan y la representación del dominio cuando no se conozca su valor.
- La gestión de memoria se realiza automáticamente por la biblioteca de funciones que reclama cualquier zona de memoria reservada para alguna rama del árbol de búsqueda que ha sido abandonada.
- Una restricción es también un objeto, que se añade a la lista de las variables afectadas. Están implantadas todo tipo de restricciones aritméticas.
- La búsqueda se implanta con algoritmos no deterministas mediante la definición de objetos llamados metas. Una meta es un objeto con un método que especifica como debe ser ejecutada. Una meta fallará si se encuentra alguna inconsistencia. La búsqueda se implanta finalmente por medio de funciones que mantienen durante su ejecución información del estado de todas las variables en cada punto de selección al cual se puede volver. Para ello se usa un mecanismo de rastreo de todos los objetos que pueden ser modificados.

Para nuestro problema de asignación de instantes de comienzo de las actividades y operaciones de producción definimos objetos que contienen variables de dominio que representan el instante de comienzo de cada una de ellas.

Una vez definidas las variables, se definen las restricciones que actúan sobre ellas, algunas de las cuales se expresan en forma de predicados y otras en forma de expresiones aritméticas, después se definen las metas de resolución que indican al motor de búsqueda como se deben ordenar las decisiones que nos llevan a una solución. Si lo que necesitamos es optimizar una determinada función objetivo, la búsqueda de soluciones en el árbol continúa, después de hallar una solución, restringiendo el valor que puede tomar la función objetivo. Cuando el algoritmo no es capaz de reducir más la función objetivo habremos encontrado el óptimo.

Por ejemplo, si nuestro problema consiste en instanciar dos actividades con la restricción de que una de ellas preceda a la otra y queremos minimizar el instante de finalización de las actividades tendríamos el problema resuelto mediante el programa siguiente:

```
1: IlcManager manager(IlcEdit);
2: IlcInt horizonte = 365;
3: IlcInt duracionPrimera = 4;
4: IlcInt duracionSegunda = 6;
5: IlcIntVar finalizacionActividades;
6: IlcSchedule programa(manager, 0, horizonte);
7: IlcIntervalActivity actividadPrimera(programa,
8:         duracionPrimera);
9: IlcIntervalActivity actividadSegunda(programa,
10:        duracionSegunda);
11: finalizacionActividades =
12:        actividadSegunda.getEndVariable();
13: manager.post(actividadSegunda.startsAfterEnd(
14:        actividadPrimera));
15: IlcMinim(IlcAnd(
16:        IlcScheduleOrPostpone(actividadPrimera),
17:        IlcScheduleOrPostpone(actividadSegunda))
18:        ,finalizacionActividades);
```

Si no se quisiera minimizar ningún índice, la búsqueda de una única solución se lograría sustituyendo las últimas cuatro líneas por:

```
15: IlcSolve(IlcAnd(
16:        IlcScheduleOrPostpone(actividadPrimera),
17:        IlcScheduleOrPostpone(actividadSegunda))
18: );
```

La función de Ilog que se encarga de implementar la optimización de un índice es la IlcMinim, a la cual se le pasa como parámetros la meta a conseguir y la variable que representa la función objetivo que se desea optimizar. En el transcurso de la búsqueda y tras cada nueva solución obtenida para esa meta, el mecanismo de búsqueda vuelve a empezar, pero con una nueva restricción añadida, la cual obliga a la variable a optimizar a ser menor que el valor que se obtuvo en la solución previa.

En el caso de las bibliotecas Ilog la función que llama al motor de inferencia para la ejecución de cualquier meta es la IlcSolve, a la cual se le pasan las metas como parámetros. Estas metas que pasamos como parámetros pueden venir enlazadas mediante otro tipo de metas propias de Ilog, son la IlcAnd y la IlcOr, que como se puede deducir nos permiten pasar a la IlcSolve una más grande que pretendemos se cumpla si se cumplen todas o si se cumple simplemente alguna de ellas. Además se pueden anidar tantas llamadas a la función IlcSolve como se desee, de forma que el cumplimiento de una de ellas venga condicionada por el de otras llamadas a IlcSolve durante el transcurso de su resolución.

La lógica general de los programas basados en restricciones es pues la disyunción, adición y anidación de metas lógicas que han de cumplirse dentro de una solución factible. Por sucesivas vueltas atrás se obtendrá aquella de las soluciones factibles que optimiza la variable definida como función objetivo. Sin la propagación de restricciones la búsqueda en el árbol se hace computacionalmente prohibitiva. Gracias a esa propagación es posible encontrar soluciones factibles en tiempos razonables.

Lo más destacable es que la estructura de las variables de dominio nos permite acotar drásticamente las ramas del árbol para el caso en que sólo queramos reparar una zona del programa. Esto es así en el control reactivo. Será tan sencillo como asignar valores a las variables fuera de esa zona, mientras que se le vuelven a asignar dominios a las variables correspondientes a la zona del conflicto por incidencia o evento en el taller. Y la programación orientada a metas lógicas nos facilita la codificación de esta nueva búsqueda sin realizar complicados cambios, sino simplemente intentando la consecución de metas distintas, mientras el algoritmo de búsqueda y propagación de restricciones permanece intacto.

Aplicación de ILOG para la Gestión de la Producción

Para validar el uso de Ilog en el problema de la gestión de la producción se han construido una serie de programas y unos casos de estudio de tamaño real. La generación de los casos de estudio se lleva a cabo con generadores aleatorios de proyectos [DeDoHe 93], con sus actividades y operaciones componentes, y unas duraciones y requerimientos de capacidad igualmente aleatorios. Para la construcción de los programas se ha diferenciado entre la definición de las metas y la forma en que estas se ejecutan. Con el objetivo de la construcción de un primer prototipo de aplicación de gestión de la producción se construyo inicialmente una aplicación con el núcleo algorítimico y la interfase gráfica imprescindible para la configuración de la búsqueda y la visualización de resultados. Posteriormente se integró con unas interfases gráficas más amigables para el usuario final.

El experimento principal consta de 20 pedidos de 20 operaciones cada uno para un total de 400

operaciones. Las características de estos proyectos son las siguientes.

- Red de precedencias aleatoria.
- La prioridad de los proyectos será en el 50% de los casos de 1, y en el resto de los casos estará entre 1 y 6.
- El comienzo de los proyectos estará entre 0 y la duración media de los proyectos que son tres meses.
- La duración de las operaciones estará distribuida uniformemente entre 1 y 10.
- La fecha de entrega más ajustada de los proyectos estará entre la duración media de los proyectos y la duración media de los proyectos más cuatro veces la duración media de las operaciones.
- El flujo de caja de las operaciones será cero en el 82,5% de las operaciones y para el resto estará entre 0 y 2.000.000 de pesetas.
- Para el nivel de planificación las operaciones se agregan de cuatro en cuatro en actividades que pueden tener requerimientos de varios recursos para su ejecución.

La función objetivo a optimizar es una variable restringida más, que se define en función de las variables de comienzo y finalización de las operaciones del problema. De todas las propuestas habitualmente por la bibliografía, nos vamos a fijar en dos de ellas.

- Función de Retraso Ponderado: basada en las penalizaciones por unidad de tiempo de retraso en los trabajos [Morton 93]:

  $$\text{Retraso Ponderado} = \sum_{l=1}^{n} \left(tard_l \cdot \text{Max}(0, C_l - dd_l) \right)$$

  donde tard es la penalización por retraso, C es la fecha de finalización y dd la fecha de entrega comprometida

- Función de Valor Actual Neto: basado en los flujos de caja asociados a la terminación de determinadas operaciones caso de existir y siguiendo la formulación de [BreMye 96] para el caso de capitalización continua:

  $$\text{Valor Actual Neto} = \sum_{l=1}^{n} \sum_{j=1}^{m} \left(fluj_{lj} \cdot e^{-r.st_{lj}} \right)$$

  donde fluj es el flujo de caja de cada actividad, r la tasa de interés y st la finalización de la actividad

Así pues con los datos de los proyectos y las funciones objetivo quedarían definidas las variables del problema. El siguiente paso es la imposición de las restricciones que limitan esas variables. La forma en que imponemos esas restricciones nos distingue las mismas entre las de cumplimiento obligado o restricciones duras y las de cumplimiento deseado o restricciones blandas.

La forma en que imponemos las restricciones duras es anterior al lanzamiento de la búsqueda de soluciones propiamente dicha. Esto quiere decir que la imposición de estas restricciones no tiene vuelta atrás como el resto de la búsqueda.

Por otro lado queda la imposición de las restricciones blandas o relajables, estas se intenta que se cumplan en la solución, pero si se llega a la conclusión de que no existen soluciones factibles, se intentará la búsqueda de soluciones sin la imposición de esas restricciones. De esta forma entramos en la parte de los programas lógicos que conllevan disyunciones y por tanto puntos de decisión.

De esta forma nuestro programa principal resumido será el siguiente:

```
1: Planificacion objetoProg (horizonteDeProg);
2: objetoProg.DefinirProblema(funcionObjetivo);
3: if(vincularNiveles)
4:    IlcSolve(RestriccionesNivelPlanificacion());
5: if(IlcSolve(
6:       IlcAnd(
7:          UsoCapacidadMaxima(capMax),
8:          UsoCapacidadFinSemana(capFinD),
9:          UsoTurnos(usoTurnos),
10:         AsignarPeriodosSinServicio(),
11:         Retraso(tipoRetraso)
12:      )
13:   )
14: ){
15:  IlcMinim(IlcAnd(
16:         PonerTiemposDeActividades(Heuristico()),
17:         EscribirSolucionBD (CosteFinGlobal)
18:      ),
19:      CosteFinGlobal);
20: };
```

Primero se crea el objeto contenedor de todas las variables y todas las restricciones (línea 1). En la definición del problema (2), dentro de ese objeto, se definen las variables. Después hay una imposición condicionada (3) de las restricciones del nivel superior (4), que básicamente limitan la ventana temporal en la cual se puede asignar cada operación en función de la planificación agregada realizada en el nivel superior. Tras ella la imposición de las restricciones duras (5), sin las cuales (7-11) no se

inicia la búsqueda de soluciones (15). La búsqueda de soluciones (16) se hará finalmente dentro de una función de minimización del coste final (19) que irá guardando cada una de las soluciones que se vayan hallando (17). Lo cual permite mantener siempre la mejor solución completa subóptima. Esta meta de minimización continúa la búsqueda de soluciones tras la consecución de cada nueva solución restringiendo el valor de la variable CosteFinGlobal que representa la función objetivo a optimizar.

De todas las metas que podemos observar en el programa la mayoría son de imposición de restricciones, caso de las metas UsoCapacidadMaxima, UsoCapacidadFinSemana, UsoTurnos, AsignarPeriodosSinServicio, Retraso. El uso de capacidad máxima nos ofrece dos posibilidades, la primera es el uso de la capacidad nominal de cada recurso, y la segunda el uso de la capacidad máxima disponible para cada recurso bien por rendimiento o por puesta en marcha de recursos alternativos. El uso de capacidad de fin de semana ofrece la posibilidad de considerar el funcionamiento de los recursos en sábado y domingo como si se tratara de día laborable. El uso de los turnos viene impuesto por la siguiente meta y limitará el uso de cada recurso a las horas definidas en los turnos de trabajo de cada uno de ellos. La asignación de períodos sin servicio incorpora a la búsqueda las restricciones de no - disponibilidad de recursos en períodos definidos como fuera de servicio por motivos de mantenimiento o disfunción de los mismos. La meta de retrasos consta de diferentes opciones, entre las que podemos citar, no retrasar ningún pedido con respecto a su fecha de entrega contractual, no retrasar más allá de lo permitido contractualmente, retrasar todo lo que sea necesario para encontrar una solución, o retrasar como máximo un número de unidades temporales que pueden ser cincuenta, cien o las definidas en la aplicación.

Pero sin embargo la realmente importante es la que se denomina PonerTiemposDeActividades, la cual implementa las decisiones acerca de cual es el orden en que se va a intentar instanciar las variables del sistema. Dado el carácter combinatorio del problema este orden es crucial para una correcta exploración del árbol en tiempo computacional aceptable, más aún en el caso de la reprogramación para el control de la producción.

La meta PonerTiemposDeActividades es en realidad un caso de anidación de metas con llamadas recursivas hasta la instanciación de todas las variables de forma factible. Es decir, esta meta sólo será cierta si se completa la recursividad siguiente:

```
PonerTiemposDeActividades (){
SeleccionarActividadMasPrioritaria(Heuristico);
return(IlcAnd(
  IlcOr(
    Instanciar(actividadSeleccionada),
    Posponer(actividadSeleccionada)
  ),
  this)
)}
```

Tras cada intento de instanciación se vuelve a intentar la selección de otra actividad. Dicho intento se realiza con las metas Instanciar y Posponer, la primera intenta instanciar la actividad a su instante de comienzo más temprano, la segunda pospondrá la instanciación a otro punto de selección caso de haber fallado el intento de instanciación. Para que se complete la meta se han de instanciar todas las variables. En la selección de variables es donde se aplica todo el conocimiento de heurísticos de ordenación procedente de la Investigación Operativa. Sin estos heurísticos gran parte de la potencia del nuevo paradigma de programación se perdería, dado que no se estaría guiando eficientemente la búsqueda de soluciones a través del árbol. Los heurísticos de ordenación utilizados en nuestra aplicación son los siguientes:

- MINENDMIN: selecciona la variable con menor límite superior de dominio.
- EDD: selecciona la variable con menor fecha de entrega de su proyecto.
- SPT: selecciona la variable con menor duración de su actividad.
- MINRWK: selecciona la variable con menor duración de sus actividades sucesoras.
- MAXRWK: selecciona la variable con mayor duración de sus sucesoras.
- RMMIOPE: selecciona la variable con menor índice miope de [Morton 88].
- RMGLOBAL: selecciona la variable con menor índice global de [Morton 88].

De su uso se concluye la importancia de los heurísticos en la eficiencia de la búsqueda dado el mal comportamiento del primero de los heurísticos que no tiene en cuenta parámetros operativos de las actividades. Con respecto a los demás su comportamiento dentro de la PBR es similar al observado en el despacho heurístico tradicional, pero con la mejora de las sucesivas soluciones que se obtienen gracias a las técnicas de vuelta atrás. Además hemos de decir que con la implementación de distintos heurísticos queda a disposición del usuario la posibilidad de la simulación de diferentes

escenarios de búsqueda de soluciones, a partir de los cuales se tomen decisiones.

Para el control inteligente de la producción, se le añaden otras metas a este programa principal de manera que primero se tenga en cuenta el estado del programa en ejecución dentro del taller y después se añadan las nuevas condiciones impuestas por la incidencia procedente del taller. Estas últimas serán de nuevo restricciones duras a imponer antes de iniciar la búsqueda (líneas 5-14). Y por el otro lado lo que se relajará será la situación temporal de las operaciones que rodean el instante de tiempo que afectó al programa en ejecución en el taller. Así pues a las líneas de código antes mostradas basta con añadir las siguientes:

```
5: IlcSolve(SituarActSim(objetoProg,
6:          instanteIncidencia,
7:          intervaloDeReprog,
8:          identificadorSimulacion));
```

Lo que se hace es imponer ventanas de tiempo a las operaciones cercanas a la incidencia por medio de intervalos centrados en el instante en el cual se encontraban en el programa inicial. La imposición de esas ventanas se relajará en la medida en que el algoritmo no sea capaz de encontrar una reparación a la incidencia. Se modificará el parámetro intervaloDeReprog. Si la ampliación de intervalos tampoco es suficiente se continuará relajando mayor número de operaciones, alejándonos en mayor medida del punto temporal de la incidencia.

Si existiera un conocimiento experto acerca de como solucionar determinada incidencia, este conocimiento se puede añadir en forma de meta lógica a la búsqueda de una solución reparadora. Y esto se realizará sin un excesivo trauma dentro del programa, pues las metas, como hemos visto, se añaden por medio de los operadores lógicos AND y OR. Así no se interfiere el resto de la búsqueda. Por ejemplo, a la incidencia conocida de avería de un recurso crítico como pueda ser un horno, se le puede asignar la meta de resolución de subcontratación de las operaciones de tratamientos térmicos. Nos quedaría:

```
15:  IlcMinim(IlcAnd(
16:          IncidenciasConocidas(),
17:          PonerTiemposDeActividades(Heuristico()),
18:          EscribirSolucionBD (CosteFinGlobal))
19:       ,CosteFinGlobal);
```

Donde la meta IncidenciasConocidas ejecutaría la llamada para la instanciación de las operaciones a subcontratar:

```
IncidenciasConocidas(){
  if(averíaHorno)
  return SubcontTratamTerm(intervaloIncidencia);
}
```

PPES, Prototipo de Prueba

El principal objetivo de este proyecto de investigación residía en poder dar soporte informático a la gestión de producción dentro del ambiente de fabricación bajo pedido.

En la primera figura podemos observar, representado en el lenguaje unificado de modelado (UML [Fowler 97]), como es el ciclo de vida de planificación, programación y control de un producto dentro de la fábrica. Por columnas se representan los actores que intervienen. Cada eje vertical representa hacia abajo el eje temporal. Los rectángulos sobre esos ejes representan la activación de los diferentes actores. Y finalmente, las flechas indican, situadas en el tiempo, cuales son los mensajes que se transfieren entre los diferentes actores. Si dichas flechas van con líneas de puntos indica que se trata de la respuesta a un mensaje recibido por parte de otro actor u objeto de la aplicación.

Con el núcleo algorítmico descrito en el epígrafe anterior se ha construido un prototipo de gestión de la producción que además de dicho núcleo dispone de una interfase gráfica de usuario y un sistema de información preparado para el funcionamiento sobre una red TCP-IP.

Las interfases gráficas de usuario se han diseñado con el objetivo de soportar mejor los procesos asociados a la planificación de una empresa fabricante dentro de este ambiente. De esta forma se han diseñado no solo los menús y los controles sino también todo lo que se refiere a las secuencias más importantes de actuación en la creación y mantenimiento de los planes y programas.

El sistema de información comprende el sistema gestor de la base de datos relacional central, así como las configuraciones de los equipos más destacados dentro del entorno de gestión de la producción, esto es incluyendo los nodos correspondientes a jefe de taller, jefe de proyecto, operarios, suministradores y clientes finales. Este sistema de información se ha concebido con el objetivo de una integración de los actores y de una fácil escalabilidad de los componentes presentes en cada nodo o en nuevos nodos de lo que pueda suponer una empresa virtual.

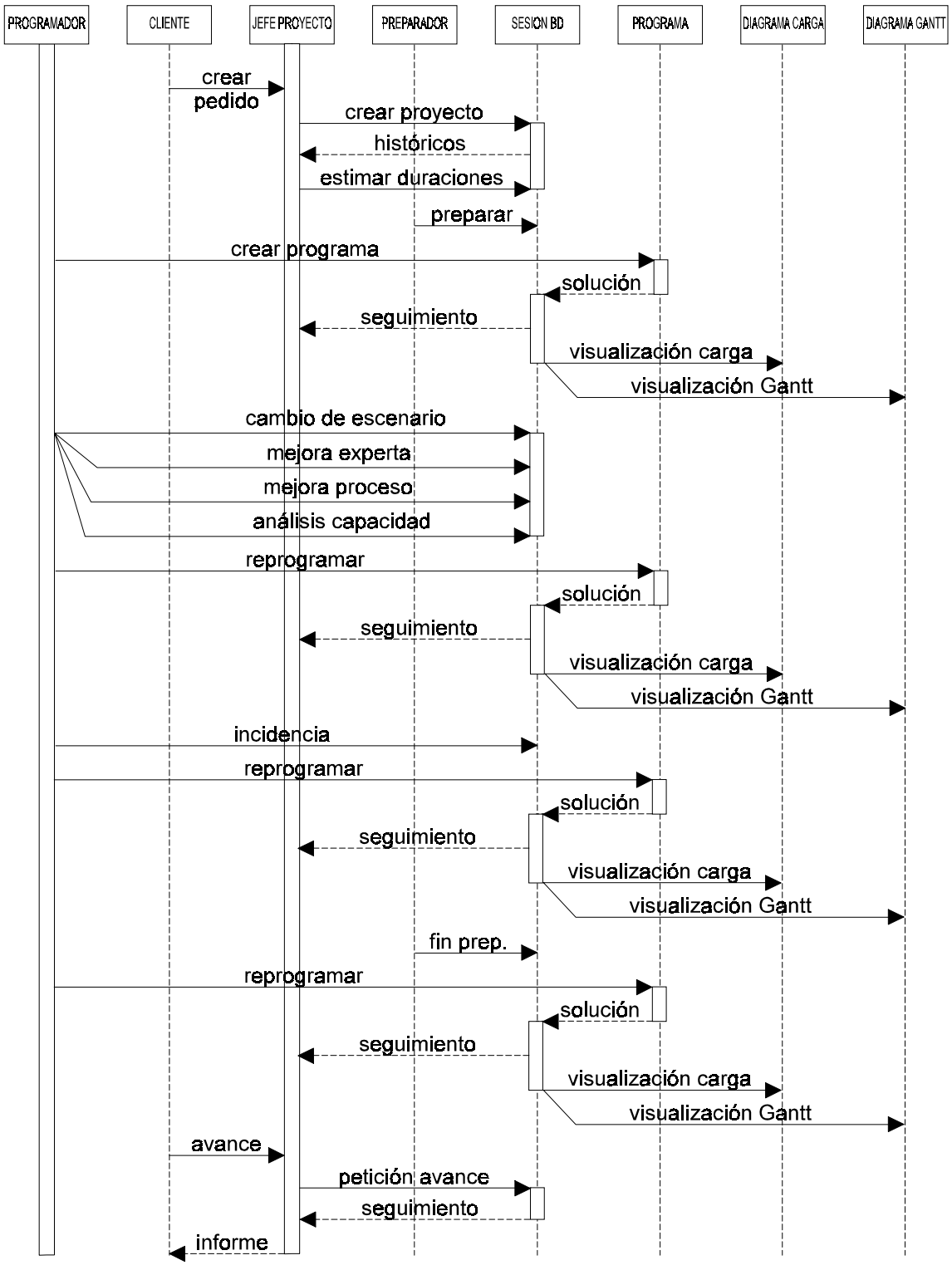
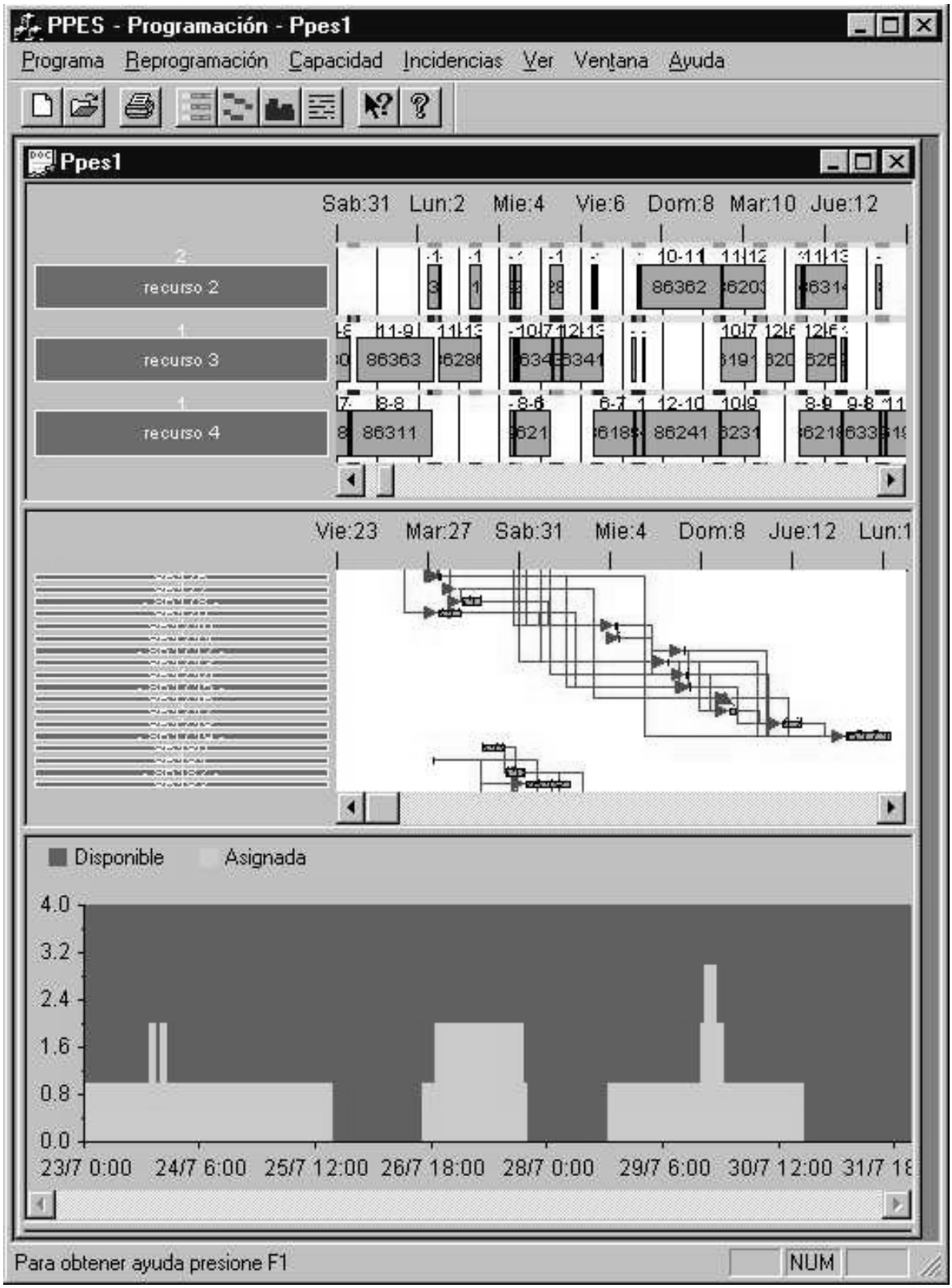


Diagrama de Secuencia de Programación y Control



Prototipo PPES. Pantalla principal

Evaluación de las Herramientas

Algunas de las ventajas de la programación basada en restricciones han quedado citadas en apartados anteriores pero en este volveremos a recopilarlas.

El uso de las restricciones del problema está implícito en la definición y en la resolución del mismo.

La optimización de funciones objetivo se hace posible dada la reducción de los tiempos de búsqueda de soluciones.

El desacoplamiento de la resolución del problema frente a la definición permite investigar sin interferencia ambas fases del problema. Se puede mejorar el conocimiento de los escenarios sin modificar el algoritmo de búsqueda. Se puede mejorar el algoritmo de búsqueda sin modificar la manera en que se representan las variables del problema.

La definición en forma de predicados y la implementación en forma de objetos C++ facilita el mantenimiento de las aplicaciones y habilita su escalabilidad. Igualmente reduce la curva de aprendizaje del uso de las aplicaciones.

La utilización de variables de dominio y la propagación de restricciones permite la combinación de técnicas de la investigación operativa basada en heurísticos con métodos de aseguramiento de la consistencia de las soluciones parciales.

La resolución basada en metas lógicas y variables de dominio permite obtener siempre soluciones factibles en el período de búsqueda de óptimos de manera que se puede abortar cualquier búsqueda para quedarnos con la mejor solución hasta ese momento.

La utilización de variables simbólicas favorece la integración de los diferentes niveles de control de la producción y muy especialmente el control reactivo por realimentación de eventos en el taller.

La programación basada en restricciones se puede aplicar a otros problemas de carácter combinatorio dentro de la logística de la producción, que contribuya a una mayor integración de los diferentes puntos de tomas de decisión en la empresa de fabricación.

Frente a estas ventajas vamos a citar también algunas de las desventajas presentes en esta programación.

La curva de aprendizaje de la programación basada en restricciones se superpone a la ya considerable curva de aprendizaje de la programación orientada a objeto. Esto hace que los resultados interesantes tarden en llegar desde el punto de vista del desarrollo.

El coste de las licencias de desarrollo y uso de las bibliotecas de programación basada en restricciones supera ampliamente el costo medio de dos hombres·año de programador de software.

Conclusiones

Hemos presentado como un tipo de programación simbólica como es la PBR potencia enormemente el mantenimiento de aplicaciones para la gestión de la producción y como se hace posible un control de la producción inteligente en el cual la actividad de la fábrica de bienes de equipo no se ve trastornada con sus continuos incidentes, debidos a su incertidumbre y dinamismo.

El control puede ser así rápido, flexible, optimizado, experto y con soporte para la decisión. Los mismos algoritmos son al mismo tiempo útiles para la simulación de escenarios. La elaboración de ofertas y la firma de contratos para nuevos pedidos toma así en cuenta el estado actual de la carga de trabajo del taller de fabricación. Y todo ello unido nos lleva a un aumento de los beneficios y de la satisfacción de los clientes.

Bibliografía

[BessFreu 99] Using Constraint Metaknowlegde to Reduce Arc Consistency Computation
Christian Bessière, Eugene C. Freuder, Jean-Charles Rêgin
to appear in Artificial Intelligence, 1999

[BörRos 94] The WAM - Definition and Compiler Correctness
Egon Börger, Dean Rosenzweig
In Logic Programming: Formal Methods and Practical Aplications,(eds. C. Beierle and L. Plümer), Elsevier Science Publishers 1994.

[BreMye 96] Fundamentos de Financiación Empresarial
Richard A. Brealey, Stewart C. Myers
Mc Graw Hill, 1996.

[ChengSmi 97] Applying Constraint Satisfaction Techniques to Job Shop Scheduling
Cheng-Chung Cheng, Stephen F. Smith.
Annals of Operations Research, vol 70, 327-357, 1997.

[DeDoHe 93] A Random Activity Generator
E. Demeulemeester, B. Dodin, W. Herroelen
Operarions Research,41, Nº 5 (1993), 972-980.

[DSVH 90] Solving Large Combinatorial Problems In Logic Programming.
Mehmet Dincbas, Helmut Simonis, Pascal Van Hentenryck.
The Journal of Logic Programming (1990) 8, 75-93

[Fowler 97] UML Distilled. Applying the Standard Object Modelling Language
Martin Fowler, Kendall Scott
Addison-Wesley (1997)

[Fox 93] The Integrated Supply Chain Management System
Mark S. Fox, John F. Chionglo, Mihai Barbuceanu
Technical Report, University of Toronto (1993)

[Jain 98] A State-of-the-Art Review of Job-Shop Scheduling Techniques
Jain, A. S. and Meeran, S.
Journal of Heuristics (submitted, 1998)

[Jain 99] Deterministic Job-Shop Scheduling; Past, Present and Future
Jain, A. S. and Meeran, S.
European Journal of Operational Research (to appear in volume 113, 1999)

[LP 94] Scheduling as Intelligent Control of Decision-Making and Constraint Propagation
Claude Le Pape
In Intelligent Scheduling. Morgan Kaufmann Publishers 1994.

[Morton 88] SCHED-STAR. A Price-Based Shop Scheduling Module
Thomas E. Morton, S. R. Lawrence, S. Rajagopolan, S. Kekre
Journal of Manufacturing and Operation Management (1988), 1, 131-181.

[Morton 93] Heuristic Scheduling Systems with Aplications to Production System and Project Management
Thomas E. Morton, D.W. Pentico
Jonh Wiley & Sons, 1993.

[NuijAarts 96] A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling
W.P.M. Nuijten, E.H.L. Aarts
European Journal of Operational Research 90, 269-284, 1996

[Puget 94] A C++ Implementation of CLP.
Jean-Francois Puget.
Technical Report 94-01, ILOG S.A., Gentilly, France, 1994.

[Robinson 65] A Machine-Oriented Logic Based on the Resolution Principle.
J. A. Robinson.
Journal of the ACM 12, 23-41. 1965

[Tsang 93] Foundation of Constraint Satisfaction
Edward Tsang
Academic Press, 1993

[VH 89] Constraint Satisfaction in Logic Programming.
Pascal Van Hentenryck.
The MIT Press 1989.

[VRoy 94] 1983-1993: The Wonder Years of Prolog Sequencial Implementation
Peter Van Roy
Journal of Logic Programming, Tenth Aniversary Issue, 1994