Búsqueda de Secuencias Óptimas de Montaje Mediante Programación Lógica con Restricciones

Carmelo Del Valle
Dept. Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Avda. Reina Mercedes s/n
41012 Sevilla, SPAIN
E-mail: carmelo@lsi.us.es

Eduardo F. Camacho
Dept. Ingeniería de Sistemas y Automática
Universidad de Sevilla
Camino de los Descubrimientos s/n
41092-Sevilla, SPAIN
E-mail: eduardo@cartuja.us.es

Miguel Toro
Dept. Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Avda. Reina Mercedes s/n
41012 Sevilla, SPAIN
E-mail: mtoro@lsi.us.es

Resumen

En este trabajo se presenta la utilización de la Programación Lógica con Restricciones en la obtención de secuencias óptimas de montaje para un sistema de varias máquinas de ensamblaje. Se trata de un problema que conlleva una mayor dificultad que otros de secuenciamiento, como el problema de *Job Shop Scheduling*, que han sido abordados mediante éstas y otras técnicas de optimización. Aparte de requerir la búsqueda de una ordenación óptima de las tareas de montaje, como sucede en los otros, también es necesario seleccionar las tareas a incluir en la solución a partir de un conjunto de posibles operaciones. Tal conjunto de tareas, así como las relaciones de precedencia entre ellas, viene representado mediante un grafo *And/Or*. En este trabajo se proponen algoritmos basados en técnicas de *branch and bound* que combina el tratamiento de las restricciones disyuntivas debidas a la utilización de recursos comunes con el de las relacionadas a la selección de las tareas, a través de una exploración del grafo *And/Or*. Se utilizan, además, distintas heurísticas encaminadas a mejorar la eficiencia de la búsqueda.

1 Introducción

Los problemas de tipo combinatorio representan una clase de problemas especialmente difícil de resolver, entre los que se incluyen los problemas de secuenciamiento. Muchos de estos problemas a su vez han sido estudiados utilizando técnicas de satisfacción de restricciones, como el problema de *Job Shop Scheduling* [Caseau and Laburthe, 1995] [Esquirol *et al.*, 1996].

En este artículo se presenta la utilización de algoritmos de *branch and bound* en conjunción con la Programación con Restricciones en la resolución del problema de la determinación de secuencias de ensamblaje de productos, un problema de planificación más amplio que los indicados anteriormente. En aquéllos, se trata de obtener una ordenación óptima de tareas, apareciendo en la solución todas ellas. En el caso que nos ocupa, se añade al problema de la ordenación de tareas la propia selección de las mismas, de entre un conjunto de tareas posibles, y de acuerdo a una serie de restricciones para componer un plan de ensamblaje factible.

La planificación del ensamblado es un problema de gran importancia, incluido en el de la fabricación de productos. En él se contemplan la identificación, selección y secuenciamiento de operaciones de ensamblaje, consideradas desde el punto de vista de su efecto sobre las partes a ensamblar. La identificación de las tareas de ensamblaje se aborda mediante el análisis de la estructura del producto, bien utilizando un sistema experto interactivo [Bourjault, 1984] [De Fazio and Whitney, 1987], o más recientemente, de forma automática a partir de modelos geométricos y relacionales [Homem de Mello and Sanderson, 1991a] y de modelos CAD y otras informaciones de tipo no geométrico [Ames *et al.*, 1995] [Romney *et al.*, 1995].

La identificación de las operaciones de ensamblaje conduce normalmente al conjunto de todos los planes de ensamblaje factibles. El número de ellos crece exponencialmente con el número de partes, y depende de otros factores, tales como la forma en que las partes individuales están interconectadas en

el ensamblaje completo, es decir, la estructura del grafo de conexiones. De hecho, este problema ha sido probado como NP-completo tanto en el caso bidimensional [Kavraki and Kolountzakis, 1995] como en el tridimensional [Kavraki *et al.*, 1995] [Wilson *et al.*, 1995].

En este contexto, la representación de planes de ensamblaje es un aspecto importante. El uso de grafos *And*/*Or* para este fin se está convirtiendo en un estándar para la representación de todos los planes de ensamblaje posibles. Puede ser obtenido estudiando el problema opuesto, el desensamblado, pero suponiendo las restricciones del ensamblado. La mayoría de los planificadores automáticos trabajan con esta estrategia. El resultado es una representación adecuada para un enfoque dirigido hacia el objetivo. Además, en [Homem de Mello and Sanderson, 1990] y [Wolter, 1992] se muestra que esta estructura es más eficiente en la mayoría de los casos que otras estructuras enumerativas.

La siguiente consideración es la obtención de un plan de montaje óptimo, seleccionado del conjunto de todos los planes de ensamblaje factibles. Se han usado diversidad de criterios para escoger un plan óptimo, que pueden encuadrarse en dos enfoques alternativos, uno cualitativo, en el que se usan reglas para eliminar planes que incluyan tareas difíciles de ejecutar, y otro cuantitativo, en el que se usan funciones de mérito para evaluar la bondad de los planes de ensamblaje. Varias de estas propuestas se recogen en distintos capítulos de [Homem de Mello and Lee, 1991], dedicado monográficamente al tema.

El criterio seguido en este trabajo es la minimización del tiempo total del ensamblado, supuestas unas estimaciones del tiempo necesario para la realización de cada posible tarea así como de los recursos necesarios para llevarlas a cabo. Este mismo enfoque es seguido en [Del Valle and Camacho, 1996] y [Del Valle and Camacho, 1999].

El resto del artículo se estructura de la siguiente manera: en el siguiente apartado se describe el problema de la selección y secuenciamiento de tareas de ensamblaje. Una breve introducción a la Programación Lógica con Restricciones es presentada en la sección 3. En las siguientes secciones se muestra la adaptación del problema a CLP(R), un lenguaje de Programación Lógica con Restricciones, así como los esquemas algorítmicos propuestos. En la sección 7 se realiza un comentario sobre los resultados obtenidos, y finalmente en las conclusiones se realizan algunas puntualizaciones finales.

2 Descripción del problema

El proceso de unir partes para formar una unidad se conoce como ensamblaje. El proceso de unión produce la conexión de una parte con otras partes ya ensambladas. Un submontaje es un grupo de partes con la propiedad de poder ser ensambladas de forma independiente de otras partes del producto. Un plan de montaje es un conjunto de tareas de montaje parcialmente ordenadas. Cada tarea consiste en la unión de un conjunto de submontajes, habitualmente dos, que es lo que aquí se considerará, para formar un submontaje mayor. Una secuencia de montaje es una secuencia ordenada de tareas de montaje, que satisface las restricciones de orden de tareas. Cada plan de montaje se corresponde con una o más secuencias de montaje.

Un grafo *And*/*Or* es una representación del conjunto de todos los planes de montaje posibles para un producto. Los nodos *Or* se corresponden con submontajes, siendo el nodo raíz el que hace referencia al producto completo, y los nodos hoja a las partes individuales. Cada nodo *And* se corresponde con la tarea de montaje que une los submontajes de sus nodos hijos produciendo el submontaje de su nodo padre. Un árbol de ensamblaje es un camino

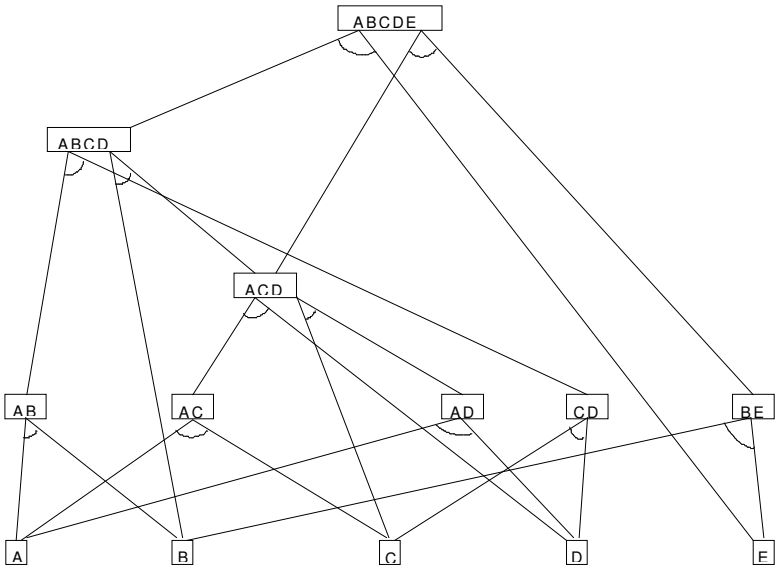

Figura 1. Grafo *And*/*Or* del producto ABCDE

del grafo *And*/*Or* que comienza en el nodo raíz y termina en los nodos hoja, y representa un plan de ensamblaje. Una importante ventaja de esta representación, usada en este trabajo, es que el grafo *And*/*Or* muestra la independencia de las tareas de ensamblaje que pueden ejecutarse en paralelo. La Figura 1 ilustra un ejemplo de esta representación, en la que los nodos *And* se han representado mediante hiperarcos *And*.

El problema planteado es la selección de un plan de montaje, uno de los árboles que componen el grafo *And*/*Or*, y el secuenciamiento de las tareas que lo componen, sujetas a las relaciones de precedencia de tareas impuestas por el árbol y a la utilización de recursos compartidos. Para ello, se parte de una estimación previa de los recursos necesarios y duración aproximada de todas las tareas de montaje correspondientes a los nodos *And*. Estos tiempos deberían incluir una estimación del tiempo necesario para otras operaciones auxiliares, como el transporte de partes y submontajes.

Otro factor tenido en cuenta en el presente trabajo es el tiempo necesario para realizar el cambio de herramientas en las máquinas, el cual es del mismo orden que el tiempo de ejecución de las tareas de ensamblaje, y que por tanto no puede despreciarse como sucede habitualmente en la planificación de otros procesos de fabricación.

3 Programación Lógica con Restricciones

La Programación Lógica presenta un modelo relacional que facilita la formulación de problemas de tipo combinatorio, así como la expresión de restricciones. Su indeterminismo permite liberar al programador de la búsqueda en árboles, usando mecanismos de backtracking que vienen incorporados en su motor de inferencia, en conjunción con técnicas de resolución y unificación. Sin embargo, los lenguajes de Programación Lógica se comportan de manera muy ineficiente al resolver problemas combinatorios debido a que usan las restricciones para verificar si las soluciones generadas las cumplen, aparte del comportamiento patológico de los esquemas de *backtracking* simples.

La Programación con Restricciones surgió a partir de la Programación Lógica mediante el uso de las restricciones para reducir el espacio de búsqueda de soluciones, introduciendo el concepto de dominio asociado a las variables frente a la unificación de la Programación Lógica, de manera que era posible aplicar técnicas de consistencia que mejoraran la eficiencia de búsqueda [Van Hentenryck, 1989][Jaffar and Maher, 1994].

Existen distintos lenguajes basados en los mismos principios, aunque difieren en diversos aspectos, como en los dominios asociados a las variables. El lenguaje usado en este trabajo es CLP(R) [Jaffar *et al.*, 1992], que usa el dominio de los reales. Utiliza un algoritmo basado en el Simplex para resolver las restricciones lineales, así como un módulo de eliminación de restricciones superfluas. El sistema usa además un mecanismo para retrasar las restricciones no lineales, hasta que la instanciación de variables las convierta en lineales.

4 Adaptación del Problema a CLP(R)

El problema a resolver puede representarse a través de una base de hechos que recoja la información relativa al grafo *And*/*Or*, los recursos y duraciones estimadas para cada tarea, las combinaciones permitidas para máquinas y herramientas y la duración de los posibles cambios de herramientas. En la figura 2 se muestra la representación del grafo *And*/*Or* de la figura 1.

Como se ha indicado antes, la búsqueda guiada por el objetivo a conseguir (el producto totalmente ensamblado) hace que pueda visualizarse el problema en sentido inverso, el desensamblado del producto. En cualquier caso, la correspondencia de uno y otro problemas es evidente y no ofrece más dificultad que invertir la solución obtenida en cuanto al origen de tiempos y a las secuencias de tareas. De esta forma, resultará más cómodo plantear las relaciones de precedencia entre tareas como si fueran de desmontaje en lugar de montaje. Obviamente los datos referidos a duraciones y recursos necesarios seguirán diendo los de las tareas de montaje equivalentes.

Según lo indicado en el párrafo anterior, definimos las relaciones de precedencia entre tareas mediante la relación `prec(T,L1,L2)`, donde `L1` y `L2` representan las listas de tareas correspondientes a los nodos Or izquierdo y derecho respectivamente a los que enlaza la tarea `T`. De

```
nodoOr(abcde,[1,2]).
nodoOr(abcd,[3,4]).
nodoOr(acd,[5,6]).
nodoOr(ab,[7]).
nodoOr(ac,[8]).
nodoOr(ad,[9]).
nodoOr(cd,[10]).
nodoOr(be,[11]).
nodoOr(a,[]).
nodoOr(b,[]).
nodoOr(c,[]).
nodoOr(d,[]).
nodoOr(e,[]).

nodoAnd(1,abcd,e).
nodoAnd(2,acd,be).
nodoAnd(3,ab,cd).
nodoAnd(4,b,acd).
nodoAnd(5,ac,d).
nodoAnd(6,c,ad).
nodoAnd(7,a,b).
nodoAnd(8,a,c).
nodoAnd(9,a,d).
nodoAnd(10,c,d).
nodoAnd(11,b,e).
```

Figura 2: Grafo *And*/*Or* en CLP(R).

esta relación debe interpretarse que, en el caso de que la solución contenga a la tarea `T`, también incorporará a una y sólo una tarea de cada una de las listas `L1` y `L2`, si éstas no son la lista vacía (lo cual indicaría que la tarea `T` añade una pieza a un submontaje). Además, lógicamente, se extraerá la propia relación de precedencia entre `T` y la(s) tarea(s) seleccionada(s) de `L1` y `L2`.

La solución al problema se va a representar mediante una secuencia de tareas por cada máquina, formando una lista. Por razones de eficiencia en el tratamiento de listas, aparecen al principio las tareas añadidas a la solución en último lugar. Para cada tarea se incluye su identificación, herramienta usada y los tiempos de inicio y finalización de la misma, expresadas mediante el término `t(IdTarea,Herramienta,TIni,TFin)`. Obsérvese que los datos relativos a los tiempos son valores numéricos que estarán involucrados en las restricciones que se planteen para ellos, lo cual conllevará que para la solución (o soluciones) obtenida esos valores estén totalmente determinado para las tareas sin holguras o restringidos a un ranfo de valores para las tareas que presenten holguras. Este aspecto es uno de los que nos beneficiamos al plantear la solución mediante técnicas de satisfacción de restricciones, ya que aparte de ofrecernos la solución en forma de secuencias de tareas, se nos informa de los valores asociados a los tiempos de comienzo y finalización en forma de restricciones cuando para alguna tarea se permite un intervalo de comienzo de la misma.

Las restricciones entre los tiempos asociados a las tareas vienen asociadas a dos circunstancias: las relaciones de precedencia definidas por el grafo *And*/*Or* y la utilización de recursos comunes por distintas tareas. Esto último es fuente de parte del carácter combinatorio del problema, ya que en general habrá que resolver restricciones de tipo disyuntivo: si no existe restricción de precedencia entre dos tareas y ambas usan el mismo recurso, hay que decidir cuál de ellas se realiza antes. Cualquiera de las dos alternativas son en principio válidas, lo cual obliga a contemplarlas por separado para la búsqueda de la solución óptima. Esta problemática ya ha sido abordada en problemas de secuenciamiento similares utilizando técnicas de propagación de restricciones, dando lugar a diferentes técnicas [Esquirol *et al.*, 1996] [Caseau and Laburthe, 1995], normalmente retrasando el tratamiento de las restricciones disyuntivas.

Pero existe otro factor que dota al problema de un carácter combinatorio: para cada nodo Or pueden existir varias tareas alternativas que permitan completar el montaje del grupo de piezas asociadas a él. Se trata en definitiva de la propia selección de las tareas que van a componer la solución. Para este tipo de disyunciones no existen tratamientos similares, debido a que ni siquiera puede hablarse de una estructura única del árbol de tareas, ya que para un nodo Or determinado, una tarea podría descomponerlo en dos submontajes con parecido número de piezas, y otra tarea podría hacerlo separando una única pieza del resto. La inexistencia de esa estructura única del árbol de tareas impediría en su caso utilizarla como plantilla donde sustituir las tareas elegidas, así como los valores asociados a los tiempos de comienzo y finalización de las mismas para las variables temporales asociadas con la plantilla.

Así las cosas, cabe plantear la resolución del problema propuesto de dos maneras: (a) extraer todos los árboles de tareas del grafo *And*/*Or* y resolver el problema planteado para cada uno de ellos, con lo que nos serviríamos de las técnicas conocidas para el scheduling disyuntivo; o (b) utilizar como base esquemas de *branch and bound* en los que en los pasos de ramificación se consideren como alternativas aquellas que nos vayamos encontrando en la exploración consiguiente del grafo *And*/*Or*. En el presente trabajo se desarrollan algoritmos que siguen ésta última línea.

5 Backtracking

El esquema más fácil de implementar en un lenguanje de programación lógica con restricciones es obviamente un esquema de *backtracking*. Como se ha indicado en la sección anterior, la solución se compone de una secuencia de tareas por cada máquina. El tratamiento de la solución que realizarán los algoritmos desarrollados en este trabajo requerirá que el orden que siguen las tareas en las secuencias se correspondan con el orden de ejecución de las mismas. En cada paso de expansión se elige una tarea para incluirla en el plan de montaje. Las tareas candidatas a ser incluidas en la solución serán aquellas cuyas predecesoras en el grafo *And*/*Or* hayan sido ya incluidas. De esta forma, las restricciones de precedencia de tareas definidas en el grafo *And*/*Or* se cumplen en cuanto al orden de inclusión de las tareas en la solución. Por otro lado, la decisión del orden de inclusión de tareas en la solución se hace corresponder con la decisión del propio orden de ejecución de las tareas que usan la misma máquina para su ejecución. Éste es pues el tratamiento que se hace para este tipo de restricciones disyuntivas.

Aparte de la manera de construir la solución, ésta se completa tras la resolución del conjunto de restricciones temporales asociadas a los tiempos de comienzo y finalización de las tareas que componen la solución. La forma en que se añaden estas restricciones depende de la propia naturaleza de ellas.

En la figura 3 se muestra la relación `incluirTareaPlanMaquina(T,D,M,H,TIni,TFin,LM,LM1)` en la que, aparte de incluir la tarea `T` a la secuencia `LM` asociada a la máquina `M` para obtener la nueva secuencia `LM1`, añade las restricciones para los tiempos de comienzo `TIni` y finalización `TFin` asociados a la tarea `T`. Se tienen en cuenta, además, la duración `D` de la tarea y el posible cambio de herramienta que podría suponer la ejecución de la tarea. A esta relación se llegará cada vez que una tarea deba ser incorporada a la solución, a través de la relación `incluirTareaPlanGlobal`.

```
% incluirTareaPlanMaquina:
%          incluye la tarea (T,D,M,H,TIni,TFin) en el plan de la máquina M

incluirTareaPlanMaquina(T,D,_M,H,TIni,TFin,[],[t(T,H,Tini,TFin)]) :-
  !, TFin = TIni + D.

IncluirTareaPlanMaquina(T,D,M,H,TIni,TFin, [t(T1,H1,TIni1,TFin1)|R],
                        [t(T,H,TIni,TFin), t(T1,H1,TIni1,TFin1)|R]) :-
  cambioHerramienta(M,H,H1,DCH), % 0 si H=H1
  TIni >= TFin1 + DCH,
  TFin = TIni + D.
```

Figura 3: Inclusión de restricciones asociadas a recursos compartidos.

La inclusión de las restricciones temporales asociadas a la precedencia de tareas se realiza al actualizar las tareas candidatas a ser incluidas en la solución. Antes de entrar en esos detalles, analicemos el esquema base de *backtracking* propuesto, que se muestra en la figura 4. En el programa, la relación `planBt` inicializa la solución y usa el algoritmo recursivo `completarPlanBt` para obtener las distintas soluciones al problema. Para el cálculo del tiempo total acumulado para el plan de montaje, se añaden restricciones adicionales (tiempo final mayor o igual que el acumulada en cada una de las máquinas) y se busca el menor valor que haga satisfacer todas las restricciones acumuladas.

Para determinar la siguiente tarea a incluir en la solución existen en principio dos puntos en donde escoger entre distintas alternativas: la elección del nodo *Or* a explorar y por otro lado le elección de la tarea (nodo *And*) para el nodo *Or* seleccionado. Esos puntos de retroceso para buscar soluciones alternativas se corresponden en el programa con las relaciones `escogeRama` y `escogeTarea` respectivamente. Así, las tareas *candidatas* a ser incluidas en la solución en cada paso de expansión

```
planBt(NodoOrRaiz,Plan,TFinal) :-
  numeroMaquinas(Nmaq),
  creaPlanInicial(NMaq,L),
  nodoOr(NodoOrRaiz,LTareasInicio),
  TIni >= 0,
  completarPlanMantenerRamasBt(L,Plan,[r(TIni,LTareasInicio)]),
  obtenerTFinal(Plan,TFinal).

% completarPlanMantenerRamasBt(L,Plan,Rcand)
%    completa el plan parcial L con las tareas seleccionadas de Rcand

completarPlanMantenerRamasBt(L,L,[]) :- !.
completarPlanMantenerRamasBt(LM,Plan,Rcand):-
  escogeRama(Rcand,Rama,RestoRamas),
  escogeTarea(Rama,Tarea,TIni),
  datosTarea(Tarea,Dur,Maq,Herr),
  incluirTareaPlanGlobal(Tarea,Dur,Maq,Herr,TIni,TFin,LM,LM1),
  incluirSucesoresEnCandidatos(Tarea,TFin,RestoRamas,RCand1),
  completarPlanMantenerRamasBt(LM1,Plan,RCand1).

obtener_TFinal([],TFinal) :- minval(TFinal).
obtener_TFinal([[]|R],TFinal) :-
  obtener_TFinal([R],TFinal).
obtener_TFinal([[t(_,_,_,TFin)|_]|R],TFinal) :-
  TFinal >= TFin,
  obtener_TFinal(R,TFinal).
```

Figura 4: Backtracking

```
incluirSucesoresEnCandidatos(T,_TFin,RCand,RCand) :-
  prec(T,[],[]), !.
incluirSucesoresEnCandidatos(T,TFin,RCand,[r(TI,L)|RCand]) :-
  (prec(T,L,[]); prec(T,[],L)), !,
  TI >= TFin.
incluirSucesoresEnCandidatos(T,TFin,RCand,[r(TI1,L1),r(TI2,L2)|RCand]) :-
  prec(T,L1,L2),
  TI1 >= TFin,
  TI2 >= TFin.
```

Figura 5: Restricciones por precedencia de tareas.

del algoritmo se recogen en una lista cuyos componentes son términos `r(Tini,ListaIdTareas)` correspondientes al conjunto de tareas asociadas a cada rama (nodo *Or*) que puede ser explorada a continuación. Obsérvese que, como sólo se podrá escoger para cada solución una tarea de cada rama, el valor asociado al comienzo de todas esas tareas puede tomarse asociado a una única variable `TIni`. Inicialmente, sólo se tendrá una rama, la correspondiente al nodo Or raíz (producto completo), y conforme se vayan escogiendo tareas, se incluirán en la lista de candidatas aquellas ramas ligadas a la tarea incluida. Se habrá completado el plan de montaje cuando la lista de ramas de tareas candidatas esté vacía.

Una vez visto el formato de las tareas candidatas a ser incorporadas en la solución, sólo falta mostrar la forma en que las restricciones debidas a las precedencias de tareas definidas en el grafo *And*/*Or* son agregadas para el cálculo de los tiempos en la solución (ver figura 5). Una vez introducida una tarea en la solución, y conforme se añaden sus tareas sucesoras a la lista de candidatas, se define el tiempo de comienzo de éstas de manera que sea posterior al de finalización de la tarea predecesora. La relación `incluirSucesoresEnCandidatos` actualiza la lista de ramas candidatas incorporando las tareas sucesoras de la última tarea incluida en la solución, y usando el tiempo de finalización de ésta para añadir las restricciones de precedencia.

6 Branch and Bound

El esquema de *backtracking* presentado en la sección anterior da lugar, como bien es conocido, a un algoritmo muy ineficiente en general. La adaptación de ese esquema a otro de *branch and bound* permite mejorar su comportamiento debido a dos factores: en la fase de ramificación (*branch*) se puede utilizar algún criterio más *inteligente* a la hora de elegir el orden de expansión de entre las diferentes alternativas que encuentra; por otro lado, se puede hacer uso de los valores obtenidos de las soluciones ya generadas para evitar la búsqueda por caminos que no van a proporcionar soluciones mejores (fase de acotación –*bound*–).

La acotación de soluciones se puede implantar, en una primera aproximación, añadiendo simplemente restricciones del tipo TFin < B al referirse a los tiempos de finalización de las tareas incorporadas a la solución, siendo B el valor de la mejor solución encontrada hasta el momento.

El uso de funciones que estimen los valores de las soluciones que pueden obtenerse a partir de un punto de la expansión ayudará a podar más el espacio de soluciones en la búsqueda del óptimo. Esas mismas estimaciones pueden servir, además, para elegir con mejor criterio el orden en el que se deben explorar las diferentes alternativas. Todo esto, sin embargo, requerirá un coste computacional añadido que, en algunos casos, puede no ser despreciable.

6.1 Heurísticas

Si se pretende realizar estimaciones de las soluciones para eliminar aquellas alternativas que no den lugar a soluciones mejores, tales estimaciones deben ser obligatoriamente optimistas, es decir, deben indicar un límite inferior (ya que se trata de un problema de minimización) del valor de todas las soluciones a las que se puede llegar expandiendo esa solución parcial. Obviamente, ya se está realizando una estimación optimista con lo qe ya tenemos, pero se está despreciando el efecto de las tareas que aún faltan por incluir en la solución, por lo que aquella estimación resulta demasiado optimista. Cuanto más ajustado sea el límite impuesto por la estimación, mejor se comportará el algoritmo en cuanto a soluciones despreciadas, pero existirá un compromiso con el coste computacional que conlleve el cálculo de esas estimaciones más realistas o mejor informadas.

En este trabajo se exponen varias funciones *heurísticas* que realizan estimaciones optimistas de las soluciones. Se basan en separar lo que pueda ser *pre*calculado *fuera de línea* de aquello que deba esperar hasta la ejecución del algoritmo de *branch and bound*. Con esto, se intenta minimizar el efecto negativo en el coste computacional.

Heurística *hs*

La primera estimación significativa que vemos considera el tiempo mínimo necesario para ejecutar cada tarea *candidata* y sus sucesoras en el grafo *And*/*Or*, sin atender a

las posibles relaciones disyuntivas por la utilización de recursos comunes entre tareas de distintas ramas del grafo. Por simplicidad, tampoco se tienen en cuenta los posibles cambios de herramientas que debieran producirse en algunos casos.

De esta forma, el tiempo mínimo para ejecutar una tarea *T* y sus sucesoras viene dado por la expresión:

$$hs(T) = Dur(T) + \max(\min_{T_i \in Or1}(hs(T_i)), \min_{T_j \in Or2}(hs(T_j)))$$

en donde T_i y T_j se refieren a las tareas asociadas a los nodos *Or* enlazados por *T*. La operación de mínimo se corresponde con la elección de la tarea más favorable del nodo *Or* (sólo se ejecuta una), mientras que la operación de máximo se corresponde con la rama (nodo *Or*) más cargada según la estimación (al menos una tarea de cada rama hay que escoger para completar la solución).

El cálculo de estos valores puede realizarse antes de iniciar el algoritmo de búsqueda de la solución óptima, ya que es independiente de la solución parcial a que se haya llegado.

La estimación a la que se llegaría si se añadiera la tarea *candidata T* a una solución parcial cuyos tiempos acumulados para cada máquina son $TFin(M_i)$ sería

$$TFinMin = \max(TFin(M_i), TFin(M(T)) + DCH + (hs(T)))$$

siendo $M(T)$ la máquina que necesita la tarea *T* y *DCH* la duración del cambio de herramienta (0 si no lo hubiere).

Los cálculos anteriores permitirán excluir como candidatas a aquellas tareas para las que la solución tuviera un tiempo mínimo acumulado *TFinMin* no inferior al valor de la mejor solución encontrada hasta el momento. Además, si todas las tareas correspondientes a una misma rama han sido eliminadas, la solución parcial debe ser desechada y volver atrás en busca de otras soluciones alternativas.

Heurística *ht*

En la heurística anterior no se han tenido en cuenta los cambios de herramienta necesarios para ejecutar las tareas candidatas y sus sucesoras. El cálculo del tiempo mínimo necesario para ejecutar la tarea *T* y sus sucesoras se consigue adaptando la fórmula indicada antes para *hs*:

$$ht(T) = Dur(T) + \max(\min_{T_i \in Or1}(htc(T_i, T)), \min_{T_j \in Or2}(htc(T_j, T)))$$

siendo $htc(T_i,T) = ht(T_i) + \tau(T_i, M(T), H(T))$, con $M(T)$ y $H(T)$ la máquina y la herramienta requeridas para la ejecución de la tarea *T*, respectivamente. El nuevo factor $\tau(T,M,H)$ representa un tiempo adicional a la ejecución de *T* y sus sucesoras, que es debido a que, previo a la ejecución de *T*, la herramienta *H* se encuentra instalada en la máquina *M*. El valor de τ viene dado en función del caso de que se trate:

$\tau(T, M(T), H(T)) = 0$

$\tau(T, M(T), H) = DCH(M,H,H(T))$

$\tau(T, M, H) = \max(0, \tau_1(T, M, H))$, si $M \neq M(T)$, siendo

$$\tau_1(T, M, H) = \max(\min_{T_i \in Or1}(\tau_2(T, T_i, M, H)), \min_{T_j \in Or2}(\tau_2(T, T_j, M, H)))$$, y

$$\tau_2(T, T_i, M, H) = \tau(T_i, M, H) - (ht(T) - ht(T_i))$$

Nótese que, en el caso $M \neq M(T)$, puede darse un valor positivo para τ ($\tau_1 > 0$) si el cambio de herramienta necesario para la ejecución de alguna tarea sucesora tiene una duración mayor que la de la propia tarea *T*.

Utilizando ahora la heurística *ht*, la estimación a la que se llegaría si se añadiera la tarea *candidata T* a una solución parcial cuyos tiempos acumulados para cada máquina son $TFin(M_i)$ sería similar a la de *hs*:

$$TFinMin = \max(TFin(M_i), TFin(M(T)) + DCH + (ht(T)))$$

Nótese que ahora *DCH* coincide con el valor de $\tau(T, M(T), UH(M(T)))$, siendo $UH(M(T))$ la herramienta colocada en $M(T)$, correspondiente a la última tarea incluida en la solución parcial con esa máquina.

Los criterios para realizar la poda de soluciones serán evidentemente los mismos que para *hs*, sólo que ahora la estimación puede ser diferente.

Heurística *htm*

El uso de la función τ en la heurística *ht* sirvió para incorporar los retardos asociados a los cambios de herramientas. En las expresiones que la definían se requería que τ fuera no negativa, lo cual significaba que $ht(T) + \tau(T, M, H)$ representaba el tiempo mínimo necesario para ejecutar la tarea *T* y sus sucesoras, partiendo de que en la máquina *M* estaba colocada la herramienta *H*. Cuando *M* no era la máquina requerida por *T*, $\tau > 0$ indicaba que el cambio de herramienta necesario en *M* tardaba más que la propia duración de *T*, mientras que si ésto no ocurría, teníamos $\tau = 0$, lo cual indica que no se requería tiempo adicional (a la ejecución de *T*) para realizar cambios de herramienta en la máquina *M*. Las expresiones correspondientes a τ_1 que se calculaban podían dar lugar a valores negativos, lo que significa que *sobraba tiempo* para realizar el cambio de herramienta que debía realizarse en su caso. Esto quiere decir que, permitiendo valores negativos para τ podríamos estimar el momento en el que, para la ejecución de *T* y sus sucesoras, se necesita la herramienta *H* en la máquina *M*. La función τ seguiremos necesitándola tal como está definida, ya que la restricción de ser no negativa está ligada a la precedencia de tareas. Necesitaremos, entonces, definir otra función τ', en parecidos términos a τ, que sirva para estimar el momento en que se necesita una determinada máquina, teniendo instalada una determinada herramienta, para ejecutar la tarea *T* y sus sucesoras. De esta forma,

$$htm(T, M, H) = ht(T) + \tau'(T, M, H)$$

indica el tiempo mínimo adicional al acumulado en la máquina M, si tiene instalada la herramienta H, para completar la ejecución de T y sus sucesoras. Por otro lado, $\tau'(T, M, H)$ viene definida por las expresiones:

$$\tau'(T, M(T), H(T)) = 0$$
$$\tau'(T, M(T), H) = DCH(M,H,H(T))$$

Para $M \neq M(T)$ se tendrá:

$$\tau'(T,M,H) = \max(\min_{T_i \in Or1}(\tau'_2(T,T_i,M,H)), \min_{T_j \in Or2}(\tau'_2(T,T_j,M,H))), \text{con}$$

$$\tau'_2(T,T_i,M,H) = \tau'(T_i,M,H) - (ht(T) - ht(T_i))$$

Ahora la estimación del tiempo mínimo requerido por cuaquier solución que provenga de añadir la tarea candidata T a una solución parcial en la que los tiempos acumulados para cada máquina sean $TFin(M_i)$ vendrá dada por

$$TFinMin = \max_i(TFin(M_i) + htm(T, M_i, UH(M_i)))$$

siendo $UH(M_i)$ la última herramienta usada en la máquina M para esa solución parcial.

Como se puede apreciar, con esta heurística ampliamos la estimación a todas las máquinas, mientras que en las anteriores sólo se realizaba para la requerida por la tarea candidata en cuestión.

6.2 Orden de exploración de alternativas

Los valores obtenidos para las estimaciones de las soluciones que se alcanzarían a partir de cada tarea candidata no sólo puede servir para podar el espacio de búsqueda, tal como se ha mostrado en el subapartado anterior, sino que además pueden ser usados como criterio para escoger el camino más prometedor para encontrar el óptimo con el menor esfuerzo computacional posible.

Sin olvidar que mantenemos la búsqueda en profundidad debido a las limitaciones (o facilidades) impuestas por la Programación Lógica, el objetivo es escoger una tarea candidata de entre las que nos hemos encontrado tras realizar el último paso de expansión. Según se vio en el esquema de *backtracking* de la figura 3, debían realizarse dos elecciones, una para la rama (o nodo *Or*) y otra para la tarea de la rama seleccionada.

El criterio de elección debe ser parecido al que se toma normalmente para variables y valores en los problemas de satisfacción de restricciones [Van Hentenryck. 1989]. Para éstos se escoge en primer lugar la variable más restringida, es decir, la que puede hacer fracasar más fácilmente la búsqueda de la solución, con objeto de que los fracasos se detecten lo antes posible. Por otro lado, el valor que se escoge para la variable es el más prometedor, es decir, aquel que tenga más posibilidades para que se satisfagan todas las restricciones.

En nuestro caso, las ramas hacen las veces de las variables, y las tareas de cada rama serían los valores. Así pues, debemos comenzar con las ramas más *críticas*, es decir, las que tienen una estimación superior del valor de la solución. A su vez, para una determinada rama, debemos comenzar por las tareas más prometedoras, aquellas con una estimación inferior. De esta forma, las tareas candidatas pertenecientes a una misma rama deben ser ordenadas en orden creciente de su estimación. Por otro lado, las ramas deben escogerse en orden decreciente de la mejor estimación existente en ellas.

7 Resultados y discusión

Los algoritmos presentados han sido implementados en CLP(R), que ofrece las facilidades de la Programación Lógica para la expresión de este tipo de problemas, así como para la formulación de algoritmos basados en *backtracking*.

Los resultados obtenidos son, sin embargo, poco alentadores. Para problemas de tamaño pequeño/mediano (unas 15 piezas), aunque el número de nodos (soluciones parciales) generados para las heurísticas más avanzadas resultaba menor que para esquema de *backtracking*, el coste computacional añadido para evaluar las estimaciones no compensaba el ahorro en el espacio de búsqueda explorado. Teniendo en cuenta que buena parte de la estimación se realiza *fuera de línea*, se encaminaron esfuerzos para mantener las estimaciones de etapas anteriores cuando fuera posible, así como evitar de esta forma volver a ordenar las tareas completamente. Aunque se produjeron progresos, no fueron lo suficiente como para mejorar el comportamiento en tiempo de computación del esquema de *backtracking* para los tamaños del problema señalados. Para tamaños superiores, para los que se prevé que pueda invertirse el comportamiento de los algoritmos, el tiempo de ejecución es desmesurado.

Cabe atribuir estos pobres resultados a la enorme ineficiencia que este tipo de lenguajes presenta para realizar determinados tratamientos sobre estructuras de datos, lo que ha podido dar lugar a un excesivo tiempo en la ordenación de las tareas candidatas. En cualquier caso, no se ha podido llegar a resultados concluyentes en cuanto a la influencia de realizar o no la ordenación de las tareas para buscar un mejor orden de exploración de soluciones.

8 Conclusiones

En este trabajo se han presentado algoritmos basados en *branch and bound* para la obtención de secuencias óptimas de montaje, en sistemas con varias máquinas de ensamblaje. Este problema conlleva no sólo la ordenación óptima de las tareas de montaje, sino la selección de aquellas que deban formar parte de la solución. Aparecen dos tipos de restricciones disyuntivas en este problema: aquellas relacionadas con el uso compartido de recursos, y aquellas otras vinculadas a la comentada selección de

tareas alternativas. Frente a la opción de enumerar todos los conjuntos de tareas que pueden dar lugar a la solución, y resolver cada problema generado por separado, usando estrategias conocidas de tratamiento de restricciones disyuntivas, aquí se proponen algoritmos que realizan un tratamiento conjunto de todas las restricciones disyuntivas mediante *backtracking*, con objeto de aprovechar en lo posible los cómputos comunes para las distintas soluciones que deban ser generadas.

Para buscar un comportamiento mejor de estos algoritmos, se han propuesto distintas heurísticas encaminadas a estimar los valores de las soluciones que puedan obtenerse a partir de las soluciones parciales alcanzadas. Tales estimaciones se han usado para estrechar el espacio de búsqueda, así como para buscar una orden de exploración de alternativas más prometedor.

Los resultados obtenidos en CLP(R) parecen indicar que no se trata de la mejor plataforma para resolver este tipo de problemas de manera eficiente.

Referencias

[Ames *et al.*,1995] A.L. Ames, T.L. Calton, R.E. Jones, S.G. Kaufman, C.A. Laguna and R.H. Wilson. Lessons Learned from a Second Generation Assembly Planning System. *Proc. 1995 IEEE Intl. Symp. on Assembly and Task Planning*, pp. 41-47.

[Bourjault, 1984] A. Bourjault. *Contribution à une Approche Méthodologique de l'Assemblage Automatisé: Elaboration Automatique des Séquences Opératoires*. Thèse d'état, Université de Franche-Comté, Besançon, France, 1984.

[Caseau and Laburthe, 1995] Y. Caseau and F. Laburthe. Improving Branch and Bound for Jobshop Scheduling with Constraint Propagation. Proceedings of the 8th Franco-Japanese 4th Franco-Chinese Conference CCS'95.

[De Fazio and Whitney, 1987] T.L. De Fazio and D.E. Whitney. Simplified Generation of All Mechanical Assembly Sequences. *IEEE J. Robotics and Automat.*, Vol. 3, No. 6, 1987, pp. 640-658. Also, Corrections, Vol. 4, No. 6, pp. 705-708.

[Del Valle and Camacho, 1996] C. Del Valle and E.F. Camacho. Automatic Assembly Task Assignment for a Multirobot Environment. *Control Eng. Practice*, Vol. 4, No. 7, 1996, pp. 915-921.

[Del Valle and Camacho, 1999] C. Del Valle and E.F. Camacho. Binary vs Symbolic Chromosomal Encoding in GA-based Selection of Optimal Assembly Sequences. CSCC'99. Athenes, Greece, 1999.

[Esquirol *et al.*, 1996] P. Esquirol, H. Fargier, P. Lopez, T. Schiex. Constraint programming. Belgian Journal of Operations Research, Statistics and Computer Sciences, 1996.

[Homem de Mello and Sanderson, 1990]L.S. Homem de Mello and A.C. Sanderson. And/Or Graph Representation of Assembly Plans. *IEEE Trans. Robotics Automat*. Vol. 6, No. 2, 1990, pp. 188-199.

[Homem de Mello and Sanderson, 1991]L.S. Homem de Mello and A.C. Sanderson. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Trans. Robotics Automat*. Vol. 7, No. 2, 1991, pp. 228-240.

[Homem de Mello and Lee, 1991] L.S. Homem de Mello and S. Lee, *eds*. *Computer-Aided Mechanical Assembly Planning*. Kluwer Academic Publishers, 1991.

[Jaffar *et al.*, 1992] J. Jaffar, S. Michaylov, P.J. Stucley and R.H.C. Yap. The CLP(R) Language and System. *ACM Transaction on Programming Languages and Systems*. Vol.14, No. 3, July 1992, pp. 339-395.

[Jaffar and Maher, 1994] J. Jaffar and M.J. Maher. Consraint Logic Programming: A Survey. *The Journal of Logic Programming*, 1994: 19, 20, pp. 503-581.

[Kavraki *et al.*, 1993] L. Kavraki, J.C. Latombe and R.H. Wilson. On the Complexity of Assembly Partitioning. *Information Processing Letters*. Vol. 48, 1993, pp. 229-235.

[Kavraki and M. Kolountzakis, 1995] L. Kavraki and M. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Information Processing Letters*. Vol. 55, 1995, pp. 156-165.

[Romney *et al.*, 1995] B. Romney, C. Godard, M. Goldwasser, G. Ramkumar. An Efficient System for Geometric Assembly Sequence Generation and Evaluation. *Proc. 1995 ASME International Computers in Engineering Conference*, pp. 699-712.

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

[Wilson *et al.*, 1995] R.H. Wilson, L. Kavraki, T. Lozano-Pérez and J.C. Latombe. Two-Handed Assembly Sequencing. *International Journal of Robotic Research*. Vol. 14, 1995, pp. 335-350.

[Wolter, 1992] J. Wolter. A Combinatorial Analysis od Enumerative Data Structures for Assembly Planning. *Journal of Design and Manufacturing*. Vol 2, No. 2, June 1992, pp. 93-104.