PROBLEMA DE SCHEDULING RESUELTO MEDIANTE UN SISTEMA DE RESTRICCIONES TEMPORALES

Ibañez, F. *[A],[B]*
Forradellas, R *[A],[B]*
Toledo, F *[A],[B]*
Berlanga, R. *[A]*
Barber, F. *[A]*
Martin, G. *[A]*

[A] LISITT- Universidad de Valencia
C/Dr.Moliner 50 -46100 Burjassot -Valencia- España
FAX: +34-6-386 4568 Email:LISITT@EVALUN11.BITNET

[B] Universitat Jaume I
Campus de Penyeta Roja-Castellón-España
FAX: +34-64-345842

RESUMEN:

Partiendo de un problema de scheduling [4][5][6], el que puede expresarse en términos generales como tareas que deben ser procesadas utilizando determinados recursos, se presenta una solución a dicho problema mediante un sistema de restricciones temporales, el cual será descrito en el presente trabajo.

El sistema propuesto es una herramienta basada en CLP(Constraint Logic Programming) que brinda un entorno integrado para manejar restricciones temporales, las que pueden ser representadas básicamente de dos maneras: numérica y simbólica. En la mayoría de las aplicaciones reales, tanto las cantidades involucradas en las restricciones numéricas, como la información expresable mediante restricciones simbólicas, no están totalmente definidas a priori. El presente trabajo permite representar esta incertidumbre y reducirla como consecuencia de las restricciones definidas.

El trabajo incluye un análisis de diferentes métodos de optimización y su resolución mediante el sistema propuesto.

Palabras Clave: Scheduling, razonamiento basado en restricciones y razonamiento temporal.

Este trabajo fue parcialmente financiado por el Proyecto de la CEE Esprit 5291 "CHIC"

1. INTRODUCCIÓN

Se entiende como problema de scheduling el que surge cuando se busca una secuencia óptima de tareas que deben procesarse con un número finito de recursos. Esta clase de problemas pueden considerarse como combinatorios discretos ya que, al igual que otros problemas surgidos en áreas de aplicación de la IA, tienen como característica la búsqueda en un espacio finito y discreto de una solución que satisfaga un conjunto de restricciones fijadas a priori. En estos problemas hay que enfrentarse con una complejidad alta (generalmente NP-completitud), lo cual implica la no existencia de un algoritmo general y eficiente para resolverlos.

Habitualmente el medio más utilizado para la resolución de los problemas combinatorios discretos viene siendo la realización de programas especializados escritos en lenguajes procedurales. Estos programas son generalmente muy eficientes pero poco flexibles.

La programación lógica permite expresar los problemas combinatorios discretos debido a su no determinismo, su forma relacional y su semántica declarativa. Sin embargo, los métodos de resolución existentes en programación lógica para estos problemas son muy ineficientes [1][3].

La inclusión de técnicas de consistencia dentro de la programación lógica mejora substancialmente la eficiencia para la resolución de los problemas que estamos tratando [1]. La combinación de la programación lógica y de técnicas de consistencia ha dado lugar a los nuevos lenguajes lógicos denominados CLP (Constraint Logic Programming). Aunque estos lenguajes no llegan a alcanzar los niveles de eficiencia de los programas procedurales, sí proporcionan niveles bastante aceptables (de varios órdenes por encima de la programación lógica sin restricciones). Esta pérdida mínima de eficiencia viene compensada con la declaratividad ganada. Actualmente existen numerosos compiladores Prolog que incluyen el manejo de restricciones y añaden nuevos dominios de computación como los racionales, booleanos, dominios finitos naturales, enteros, etc... , algunos de éstos son CHIP, Prolog III, Pecos, etc ...

Dado el carácter temporal de los problemas de scheduling, algunos de estos pueden ser expresados mediante los diferentes modelos temporales desarrollados para el manejo de eventos temporales y sus relaciones. De todos los formalismos existentes destaca el modelo desarrollado por Detcher, Meiri y Pearl, ya que trabaja con información temporal métrica y el mantenimiento de la consistencia se realiza mediante técnicas CSP (Constraint Solving Problems) que le proporcionan muy buena eficiencia [12]. Desafortunadamente, este formalismo no puede expresar cualquier tipo de información temporal, ni siquiera disyunciones a nivel de intervalos que tan frecuentemente aparecen en problemas de scheduling. Además, al igual que el resto de formalismos, la obtención de soluciones suele ser bastante costosa debido a que como norma general el mantenimiento de la consistencia es parcial.

Por estas razones los modelos de razonamiento temporal no son frecuentes para la resolución de problemas de scheduling y tienen una aplicación muy limitada. De acuerdo con lo anterior CHIP, [2][9] es un lenguaje cuya eficiencia debe ser investigada para estas cuestiones.

2. SISTEMA DE RESTRICCIONES TEMPORALES

El modelo de razonamiento temporal que se describe a continuación se basa en la representación de información temporal de distintos formalismos y pretende ser una integración natural de los mismos. El mantenimiento de la consistencia, la obtención de soluciones y el añadido de nueva información (update) se realiza mediante restricciones sobre variables de dominio finito sobre naturales [1][2][9].

2.1. Formalismo

El sistema admite dos tipos de objetos temporales:

Puntos deslocalizados: Representan magnitudes cuya localización es incierta dentro de un dominio dado a priori. Estos son representados en el lenguaje CHIP mediante variables de dominio finito.

Se declaran mediante el operador predefinido ´::´, con la sintaxis siguiente:

X:: Min:Max % X es una variable cuyo dominio es el intervalo [Min,Max].

Si tenemos un conjunto de variables dominio que inicialmente parten con un mismo dominio, entonces podemos declararlas mediante una lista:

[X1,X2, ... ,Xn] :: Min : Max % las variables de la lista toman sus valores en [Min,Max]

Nótese que en relación con otros formalismos, los puntos deslocalizados pueden representar los eventos puntuales de los modelos de Kautz y Vilain [8] y Dechter, Meiri y Pearl [12].

Intervalos deslocalizados: Representan eventos temporales con cierta duración. Estos se representan en CHIP mediante una estructura de tres variables dominio que representarían el inicio, final y longitud del intervalo, opcionalmente, cualquiera de estas componentes puede ser constante.

En el sistema propuesto, los intervalos deslocalizados se declaran mediante el operador ´:>´ con la sintaxis siguiente:

nombre :> [Min,Max,Length]

donde, *´nombre´* es un átomo que actua como identificador del intervalo, *Min* y *Max* son dos números naturales que delimitan el espacio de ocupación del intervalo deslocalizado, es decir, los márgenes dentro de los cuales puede localizarse el intervalo. *'Length'* es la longitud del intervalo, que puede ser bien una constante o una variable dominio previamente declarada, bien un functor de la forma *up(M,N)* que permite definir la longitud del intervalo como un punto deslocalizado de dominio *[M,N]*, siendo *M* y *N* dos constantes naturales, sin una declaración previa.

Para acceder a los campos de un intervalo deslocalizado genérico *A* se definen las funciones *Start(A), End(A) y Length(A)*, que devuelven puntos deslocalizados que representan el inicio, final y longitud del intervalo *A*. Internamente, el sistema creará la restricción métrica que relaciona los bordes del intervalo con su longitud: *Start(A) + Length(A) #= End(A)*.

Nótese que los intervalos deslocalizados aquí definidos corresponderían con los intervalos del sistema propuesto por Allen [7], aunque en éste los intervalos no pueden expresarse de forma métrica ya que sólo admite relaciones de tipo simbólico. La representación de intervalos temporales en los modelos basados en puntos es incompleta puesto que las disyunciones a nivel de intervalos no pueden ser expresados con esos modelos. De hecho este es uno de los

problemas constantes en razonamiento temporal: la integración de las diferentes representaciones temporales [10][11].

En el Sistema que se propone, todas las relaciones entre eventos temporales, sean intervalo-intervalo, punto-punto o punto-intervalo, se transforman internamente en restricciones lineales predefinidas en CHIP. La resolución de estas restricciones se realiza a través de la regla de inferencia PLAIR (Partial Looking Ahead), la cual produce la poda parcial de valores inconsistentes de las variables dominio implicadas en la restricción (reducción de dominios) y propaga estas reducciones a todas las restricciones que contengan alguna de las variables cuyo dominio se haya reducido (restricciones activas). Un amplio tratamiento de esta regla de inferencia y otras definidas en CHIP puede encontrarse en [1] y [9].

El usuario puede manejar directamente los objetos temporales y establecer sus relaciones mediante un lenguaje superior. Este lenguaje tiene varias partes: declaración de objetos temporales, vista anteriormente, restricciones temporales, consulta y obtención de soluciones.

2.2. Tipos de restricciones

Métricas:

Estas restricciones actuan siempre a nivel de puntos y son representadas mediante restricciones predefinidas en el lenguaje CHIP con la sintaxis siguiente:

<TL_1> <operador> <TL_2>

donde *TL_1* y *TL_2* representan términos lineales sobre variables dominio o puntos deslocalizados definidos como en [2].

<operador> es uno de los operadores de restricción predefinidos en CHIP: #< (menor), #> (mayor), #= (igual), #>= (mayor o igual), #<= (menor o igual) y ## (distinto).

Simbólicas a nivel de intervalos:

Estas restricciones actúan exclusivamente sobre intervalos deslocalizados y su sintaxis viene dada por:

<identificador_intervalo_1> {<relaciones>} <identificador_intervalo_2>

donde los miembros derecho e izquierdo de la expresión son los identificadores de dos intervalos deslocalizados previamente decalarados. El operador <relaciones> es un subconjunto de las relaciones simbólicas de Allen [7] las cuales conformarán una restricción disyuntiva en el caso en que dicho subconjunto tenga más de un elemento.

Ejemplo: La intersección temporal entre el periodo de comida y trabajo vendría expresado por:

```
comer :>[3,10,up(1,2)],
trabajar :>[3,10,up(2,4)],          %declaración de intervalos
comer {overlaps} trabajar,          %restricción simbólica no disyuntiva
End(comer) #= Start(trabajar) + 5   %restrición métrica
```

2.3. Obtención de soluciones

La obtención de soluciones en el sistema descansa en las potencialidades de CHIP, que asegura la completitud del sistema mediante la instanciación de las variables [1]. Contrariamente a otros sistemas basados en red [12], la obtención de soluciones en CHIP es muy eficiente debido a que las restricciones siempre permanecen activas gracias a las reglas de inferencia PLAIR y Forward Checking [1][9]. La obtención de valores de una variable dominio X genérica, se realiza mediante el predicado predefinido ***indomain(X)***. Este predicado es no de-

terminístico e instancia la variable *X* a cada uno de los valores de su dominio que satisfaga todas las restricciones donde participe la variable. Debemos hacer notar que los valores obtenidos a través de *indomain* puede que no pertenezcan a ninguna solución final debido a la parcialidad del mantenimiento de la consistencia en CHIP. La forma de obtener soluciones finales a partir de un conjunto de variables dominio *X1, ..., Xn* se realiza mediante el siguiente predicado:

```
labeling([]).
labeling([X|T]):-
        indomain(X),
        labeling(T).
```

Este predicado genera juegos de valores de las variables dominio pasadas en forma de lista, que satisfacen todas las restricciones, es decir, genera soluciones finales.

En el sistema propuesto, la obtención de soluciones a nivel de intervalos se realiza generando soluciones para las variables dominio que representan el comienzo y la longitud de cada intervalo. El predicado que realiza la generación de soluciones para intervalos se denomina *instanciate/1*, al cual se le pasa como parámetro una lista de los identificadores de los intervalos involucrados.

3. EL PROBLEMA GENERAL DE SCHEDULING

La terminología de la teoría de scheduling surge en numerosas aplicaciones industriales, es por ello que nos referiremos a *tareas* y a *máquinas* para nombrar a cualquier actividad y a cualquier procesador respectivamente [4]. En el planteamiento inicial del problema debemos conocer el número de tareas y el número de máquinas que permitirán realizar estas tareas.

3.1. Definiciones

Entendemos como **operación** al procesamiento de una tarea sobre una máquina. Denotaremos con $\mathbf{o_{ij}}$ a la operación de la i-ésima tarea ejecutada por la j-ésima máquina. Denotaremos con $\mathbf{p_{ij}}$ al tiempo que tarda en ejecutarse la operación $\mathbf{o_{ij}}$.

Las operaciones correspondientes a una tarea deben procesarse en un determinado orden, las restricciones que definen dicho orden son conocidas como *Restricciones Tecnológicas* (RT).

Consideraremos que las tareas están listas para ejecutarse a partir de un determinado tiempo, denominado **'ready time'**. Denotaremos con $\mathbf{r_i}$ al tiempo en el cual la tarea i-ésima está lista para ser procesada.

El problema general de scheduling es encontrar una secuencia de uso de las máquinas para cada cada tarea que sea *compatible* con las RT y *óptima* con respecto a algún criterio de performance.

3.2. Hipótesis

Enumeremos el conjunto de hipótesis de partida que se imponen generalmente en el planteamiento de un problema de scheduling:

1) No pueden procesarse simultaneamente ningún par de operaciones de una misma tarea.
2) Cada operación, una vez comenzada, no puede ser interrumpida.
3) Cada tarea tiene *m* operaciones distintas, una sobre cada máquina. Esto implica que un mismo trabajo no puede realizar más de un procesamiento sobre la misma máquina durante

el tiempo que dura el scheduling. Por otro lado implica que cada tarea utiliza todas las máquinas.
4) Los tiempos de procesamiento son independientes del scheduling. Esta hipótesis implica por un lado que el tiempo para preparar la máquina para su procesamiento es independiente de la última tarea procesada y por otro lado que los tiempos para mover las tareas entre las máquinas son despreciables.
5) Las tareas pueden esperar hasta que las máquinas estén disponibles.
6) Ninguna máquina puede procesar más de una operación al mismo tiempo.
7) Las máquinas nunca fallan y están siempre disponibles a través del periodo de scheduling.
8) Las RT son conocidas a priori y no pueden cambiar.
9) No existe incertidumbre. En particular el número de tareas y máquinas, los tiempos de procesamiento y los ready times son conocidos y fijos.

4. RESOLUCIÓN DEL PROBLEMA.

Supongamos, sin pérdida de generalidad, que tenemos dos tareas y tres máquinas. Las RT definen para cada tarea la secuencia en que serán utilizadas las máquinas. Por ejemplo, las RT:

Tarea 1: o_{12} o_{13} o_{11}
Tarea 2: o_{21} o_{23} o_{22}

implican que la tarea 1 utilizará primero la maquina 2, luego la 3 y finalmente la 1, mientras que la tarea 2 utilizará las máquinas en la secuencia 1, 3, 2.

Como paso inicial, expresaremos cada una de las operaciones o_{ij} como intervalos que en principio ocupan todo el espacio de soluciones del scheduling.

scheduling :-
o11:>[r1,n,p11], o12:>[r1,n,p12], o13:>[r1,n,p13],
o21:>[r2,n,p21], o22:>[r2,n,p22], o23:>[r2,n,p23)],

donde *n* es una constante suficientemente grande que actúa como cota superior de todo el scheduling, *r1* y *r2* son los 'ready time' de las tareas y p*ij* son los tiempos de procesamiento, definidos como en el apartado anterior.

Las RT se representan añadiendo las restricciones siguientes:

End(o12)#<=Start(o13), End(o13)#<=Start(o11), % para la tarea 1
End(o21)#<=Start(o23), End(o23)#<=Start(o22), % para la tarea 2

Luego se incluyen las restricciones disyuntivas para expresar el hecho de que una máquina no puede ser utilizada simultáneamente por dos operaciones (hipótesis 6). En nuestro caso:

o11 {before, after} o21,
o12 {before, after} o22,
o13 {before, after} o23,

Finalmente, se incluye una variable que indique el tiempo de finalización del scheduling del siguiente modo:

End_sch::0:n,
End(o11)#<= End_sch,
End(o22)#<= End_sch,

Hacemos notar que estas restricciones afectan solamente a las últimas operaciones de cada tarea.

La utilización del predicado predefinido *instanciate([o11,o12,o13,o21,o22,o23])* generará todos los schedulings factibles, es decir, que cumplen todas las restricciones.

5. TRATAMIENTO DE HIPÓTESIS MENOS RESTRICTIVAS.

Reconsideraremos las hipótesis descritas en la sección 3.1 con el objeto de dar mayor flexibilidad al tratamiento de scheduling.

Hipótesis 1: Si en una determinada aplicación necesitáramos permitir que un par de operaciones de una misma tarea (por ejemplo: *o12* y *o13*) puedan ejecutarse simultáneamente, sólo debemos excluir del programa anterior la restricción: *End(o12)#<=Start(o13)*. Si quisiéramos expresar además que la operación *o13* puede empezar hasta 2 unidades de tiempo antes de que finalice la operación *o12*, incluiríamos la restricción: *End(o12)#<=Start(o13)+2*.

Hipótesis 2: Si necesitáramos descomponer una operación de una duración determinada ,o incluso indeterminada dentro de ciertos límites, en dos o más suboperaciones, sólo debemos agregar a las relaciones temporales deseadas para estas suboperaciones, una restricción que relacione las longitudes de las mismas. Por ejemplo, si la operación o11, de duración 20, debe descomponerse en las suboperaciones *o111* y *o112*, se debe incluir la restricción: *Length(o111)+Length(o112)#=20*.

Hipótesis 3: Si necesitáramos que una tarea realice más de una operación sobre una misma máquina, debemos realizar dos cambios en el programa general para reflejarlo; por un lado, especificar las relaciones temporales de cada una de las nuevas operaciones, y por otro, incluir nuevas restricciones disyuntivas de éstas. En el ejemplo, si después de la operación *o11* de la tarea 1, necesitáramos realizar otra operación sobre la máquina 3, sólo debemos agregar en el programa: *End(o11)#<=Start(otra_o13) , otra_o13{before,after}o23*.

Por otro lado, si abandonamos la hipótesis de suponer que todas las tareas ocupan todas las máquinas, la solución matemática es substancialmente diferente [4]. Sin embargo la estructura del programa, usando el sistema propuesto, es exactamente la misma. Por ejempo, si la tarea 1 no usara la máquina 3, sólo debemos excluir la declacración del intervalo *o13* y todas las restricciones en que estaba involucrado.

Aunque podría pensarse que una tarea que no usa una máquina podría ser implementada asignándole un tiempo de procesamiento igual a cero a la operación correspondiente a esa máquina, todavía tendríamos un problema: ¿Dónde serían ubicadas estas operaciones nulas en la secuencia de procesamiento de la tarea? A partir de la hipótesis 2 inicial, la tarea estaría esperando que se torne disponible una máquina que en realidad no usa.

Hipótesis 4: Supongamos que el tiempo de procesamiento de la operación *o11* es de 10 o de 13, dependiendo de que esta operación se realice antes o después de la operación *o21* (que es la otra operación que usa la misma máquina). El cambio necesario en el programa general puede expresarse declarando el intervalo correspondiente a la operación *o11* con duración inderminada entre los valores 10 y 13 y expresando las dos posibilidades mencionadas como 'puntos de elección' del siguiente modo:

o11:>[r1,n,up(10,13)],
(End(o11)#<=Start(o21), Length(o11)#=10 ;
End(o21)#<=Start(o11), Length(o11)#=13),

Sin embargo, el tratamiento necesario para casos más generales se torna más complejo e ineficiente.

Hipotesis 5: Supongamos que en una determinada aplicación, necesitáramos comenzar la operación *o13*, a lo sumo 5 unidades de tiempo después de finalizada la operación *o12*, la única restricción que debe incluirse en el programa es: *Start(o13) #<=End(o12)+5.*

Esta es una hipótesis no trivial. En algunos problemas el procesamiento de las tareas debe ser continuo de una operación a la otra.

Algunos casos particulares de las hipótesis no consideradas pueden resolverse en el sistema propuesto pero no son generalizables y excede el alcance de este artículo.

6. OPTIMIZACIÓN

En este apartado analizaremos los criterios de optimización usualmente planteados en scheduling, y mostraremos como son resueltos en el sistema propuesto. Existen muchos más criterios que los aquí descritos, pero generalmente estos se aplican a casos muy particulares y de difícil resolución, es por ello que no serán tratados en el presente artículo.

6.1. Minimización del tiempo de finalización del scheduling

Para cada tarea se define el tiempo de su finalización, denotado por C_i, y a partir de éstos definimos el tiempo de finalización del scheduling como $C_{máx}$ = *máximo*$\{C_i\}$ y el tiempo medio de finalización del scheduling C como la media aritmética de los C_i.

La variable *End_sch*, que representa $C_{máx}$, quedará reducida como consecuencia de las declaraciones y restricciones previas. Un intento razonable para minimizar $C_{máx}$ sería tomar el mínimo elemento del dominio al cual se redujo *End_sch*. Un scheduling óptimo puede encontrarse asignando a los comienzos de cada intervalo el valor más pequeño. Sin embargo este método sólo es válido para un scheduling sin disyunciones y con tiempos de procesamiento conocidos y fijos [1].

Un modo de completar la solución óptima válida para todos los casos es agregar al programa anterior:

indomain(End_sch),
instanciate([o11,o12,o13,o21,o22,o23]).

el cual obtiene como primera solución la óptima. Notar que la instanciación de los intervalos podría hacer fallar las restricciones y necesitar considerar el segundo valor de la variable *End_sch*, como consecuencia de *indomain(End_sch).*

6.2. Minimización del flujo máximo.

Se define el flujo de la tarea i-ésima, denotado por F_i, como la diferencia entre el tiempo de finalización de la tarea y su 'ready time'. Se define el flujo máximo del scheduling como el máximo de todos los flujos de las tareas: $F_{máx}$ = *máximo*$\{F_i\}$.

En el ejemplo: *F1=End(o11) - r1 , F2=End(o22) - r2*

El criterio que seguimos para obtener la solución óptima es similar al apartado anterior pero aplicado a la variable $F_{máx}$:

```
[F1,F2,Fmax] :: 0 : n,          %declaración de los flujos
F1+r1 #= End(o11),
```

F2+r2 #= End(o22),
F1 #<= Fmax, F2 #<= Fmax,
indomain(Fmax), instanciate([*o11,o12,o13,o21,o22,o23*]).

6.3. Minimización del flujo medio

Definimos flujo medio **F** como la media aritmética de los flujos de cada tarea. La solución que minimiza **F** es la misma que minimiza **C** [4] (aunque **F** y **C** sean diferentes).

En el ejemplo, debemos minimizar *(C1+C2)/2* que es lo mismo que minimizar *End(o11) + End(o12)* lo cual puede expresarse como:

C :: 0 : n,
C#=End(o11)+End(o12),
indomain(C), instanciate([*o11,o12,o13,o21,o22,o23*]).

6.4. Minimización de promedios pesados

En general deseamos minimizar una expresión de la forma $\Sigma\alpha_i\ C_i$, donde α_i es una constante natural que representa el peso asociado a la finalización de la tarea i-ésima C_i.

En nuestro caso $\alpha 1\ C1+\alpha 2\ C2 = \alpha 1\ End(o11) + \alpha 2\ End(o22)$, por lo tanto escribiremos:

P :: 0 : n,
*P #= α1*End(o11)+α2 * End(o22),*
indomain(P), instance([*o11,o12,o13,o21,o22,o23*]).

6.5.Minimización del retraso máximo

Se define el '*due time*' de la tarea i-ésima, denotado por $\mathbf{d_i}$, como el tiempo en el cual debe estar finalizada esa tarea.

Se define el *retraso* de la tarea i-ésima, denotado por $\mathbf{L_i}$, como la diferencia entre la finalización de una tarea y su 'due time'.

En el ejemplo $L_1 = (End(o_{11}) - d_1)$ y $L_2 = (End(o_{22}) - d_2)$. Tanto L_1 como L_2 pueden resultar negativos para el caso en que la tarea finalice antes del 'due time'. Para estos casos, puesto que CHIP trabaja con variables dominio sobre naturales, sumaremos una constante *c* suficientemente grande de manera que obtengamos un retraso "desplazado" positivo:

L1despl= (End(o11) - d1) + c, *L2despl= (End(o22) - d2) + c*

Puesto que minimizar *Lmax* es lo mismo que minimizar *Lmax* desplazado, la solución óptima la podemos obtener de un modo similar a los programas anteriores:

[L1despl,L2despl,Lmax_despl] :: 0 : n,
L1despl+d1#=End(o11)+ c,
L2despl+d2#=End(o22)+ c,
L1despl#<=Lmax_despl,
L2despl#<=Lmax_despl,
indomain(Lmax_despl), instance([*o11,o12,o13,o21,o22,o23*]).

7. CONCLUSIONES

CHIP presenta ciertas ventajas en términos de declaratividad y proporciona una eficiencia cercana a la lograda con programas de propósitos particulares, escritos en lenguajes procedurales [1][3]. Sin embargo, existe una pérdida de expresividad en el manejo de información

temporal debido a la transformación que hay que realizar sobre las relaciones temporales, para expresarlas en restricciones predefinidas en CHIP. En el sistema propuesto el usuario declara los objetos temporales y especifica las relaciones entre los mismos, realizando el sistema las transformaciones necesarias.

El principal problema de eficiencia surge con el tratamiento de las disyunciones que son expresadas y tratadas mediante puntos de elección dentro de un programa lógico. Este método es ampliamente tratado por P. Van Hentenryck en [1] y consiste básicamente en hacer uso del no determinismo de Prolog para crear un punto de elección que permita evaluar una tras otra las restricciones de la disyunción. El principal inconveniente de este método es el elevado costo que supone la evaluación de cada disyunción, ya que cada vez que se evalúa una nueva restricción de la disyunción, se deben restaurar los valores de todas las variables afectadas por la restricción anterioriormente evaluada. Nótese además que las disyunciones así tratadas, no permanecen activas en su conjunto, sino que sólo estarán disponibles en el momento de obtener soluciones. Este problema está parcialmente resuelto en nuestro enfoque mediante el tratamiento de disyunciones a nivel de variables dominio (ver apéndice). Sin embargo, las disyunciones que involucren a más de dos intervalos, seguirán siendo tratadas mediante el método del punto de elección.

En cuanto a las aplicaciones, existe una extensa literatura donde se muestran distintos sistemas reales expresados en el modelo presentado [4][5][6]. Sin embargo, no existe un sistema flexible unificado y eficiente para representar los distintos casos particulares de este modelo. Además, la complejidad matemática varía considerablemente para cada caso particular. El presente trabajo está orientado a la representación de este modelo en un sistema que permita expresar los distintos casos de un modo unificado y susceptible a modificaciones.

AGRADECIMIENTOS

Parte de la investigación realizada en el presente trabajo ha sido realizada mientras los autores trabajaban en el Proyecto Esprit 5291 de la CEE "CHIC" (Constraint Handling in Industry and Commerce). Organizaciones colaboradoras son: ECRC y Siemens (Alemania), ICL e Imperial College (Gran Bretaña), Bull, Reanault, CERT/ONERA y Dassault (Francia), Braghenti y AIS (Italia), CMSU (Grecia) y OCT (España).

APÉNDICE

El método que desarrollamos consiste básicamente en expresar las disyunciones entre dos intervalos a nivel de las variables dominio, en vez de expresarlo a nivel de restricciones. De este modo, se eliminan los puntos de elección y las disyunciones permanecen en todo momento activas.

Para realizar la disyunción de variables dominio es preciso definir una nueva restricción denominada *unión de variables dominio* cuya sintaxis sería:.

$$D = D1 \text{ U } D2 \text{ U } ..... \text{ U } Dn$$

donde *D, D1, D2, ... , Dn* son variables dominio sobre naturales.

Semánticamente, *D* toma como dominio la unión de los dominios de las variables *D1,..,Dn*. Esta restricción hace uso de la regla de inferencia PLAIR.

A modo de ejemplo, sean las variables dominio *D1*::0:3, *D2*::0:5 y *D3*::7:9, sobre las cuales imponemos la restricción

$$D = D1 \text{ U } D2 \text{ U } D3 \text{ (I)},$$

entonces, D tomará sus valores en el dominio definido por la unión de los dominios de *D1*, *D2* y *D3*, esto es *D*=[0,5] U [7,9]. Supongamos ahora que se produce una reducción en la variable *D*, por ejemplo introduciendo la restricción unaria *D#<8*, ello provocará una propagación hacia la restricción (I) que producirá una reducción en el dominio de D3 únicamente, *D3*=[7,8], el resto de variables no se ven afectadas. Si hubiesemos introducido la restricción *D#>7*, entonces las variables *D1* y *D2* se desconectarían de la unión.

Como paso previo a la transformación de las disyunciones entre intervalos se debe realizar una

transformación de todas las relaciones simbólicas implicadas en la disyunción a sus respectivas métricas. Para esta transformación se realiza un mapa de distancias entre los dos intervalos implicados y se extraen las restricciones que deben cumplir las distancias L1, L2, LA y LB (ver figura 1). Todas las relaciones simbólicas cumplirán las siguientes restricciones:

$$Start(B) \#= L1 + End(A) \quad \text{(R-I)}$$
$$End(B) \#= L2 + Start(A) \quad \text{(R-II)}$$

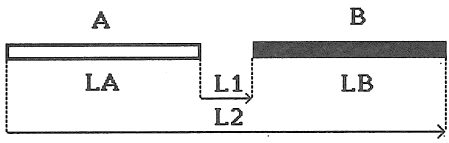

Fig 1. Mapa de distancias entre dos intervalos

A modo de ejemplo obtendremos la transformación de la restricción simbólica *A {before} B* (figura 1) en sus correspondientes métricas.

En este caso tanto *L1* como *L2* son variables de dominio positivo. La restricción de distancias que define esta relación es:

$$LA + L1 + LB \#= L2$$

donde LA y LB son las longitudes de los intervalos A y B respectivamente.

Para todas las relaciones simbólicas existe un conjunto de restricciones sobre las distancias indicadas, que conforman su transformación métrica.

Para cada relación simbólica *i*-ésima de la disyunción obtendremos un juego de variables dominio L1*i*, L2*i*, LA*i* y LB*i* y un conjunto de restricciones métricas sobre éstas. Las retricciones (R-I) y (R-II) son comunes a toda la disyunción, definiendo L1 como la unión de los casos particulares L1*i* y L2 como la unión de las variables L2*i*. Dado que las longitudes de los intervalos A y B están condicionadas a los casos particulares de la disyunción, deberemos de crear dos variables unión, LA y LB que contengan los casos particulares LA*i* y LB*i*. Después deberemos igualar las longitudes de los intervalos con las correspondientes uniones:

$$Lenght(A) \#= LA,$$
$$Lenght(B) \#= LB$$

A la hora de implementar este método en CHIP nos encontramos con el problema de que sólo admite dominios naturales, pero en varias relaciones simbólicas L1 y L2 son dominios de valores negativos. Este problema se soluciona desplazando todas las variables dominio de la unión cierta cantidad para que el dominio pase a ser positivo. Esa cantidad desplazada deberá estar expresada en las restricciones (R-I) y (R-II) de modo que no se altere el resultado.

BIBLIOGRAFÍA

[1] P. Van Hentenryck. "Constraints Satisfaction in Logic Programming" Logic Programming Series. MIT PRESS, Cambridge, 1989.

[2] COSYTEC S.A. "CHIP USER´s Guide". (COSY/REF/001,V 1.1), October 1991.

[3] M.Dincbas, H.Simonis, P. Van Hentenryck. "Solving Large Scheduling Problems in Logic Programming" EURO-TIMS Joint International Conference on Operations Research and Management Science, Paris, France, Julio 1988.

[4] S.French. "Sequencing and Scheduling" Ellis Horwood Ltd, 1982.

[5] R.Bellman, A.O. Esogbue, I. Nabeshima "Mathematical aspects of scheduling and applications" Pergamon Press, 1982.

[6] R.W Conway, W.L Maxwell, L W Miller "Theory of scheduling" Addison Wesley, 1967.

[7] J.Allen "Towards a general theory of action and time" Artificial Intelligence, 23:123-153, 1984.

[8] M.Vilain and H.Kautz "Constraint propagation algorithms for temporal reasoning" Proceedings of AAAI-86, 377-382, 1986.

[9] R.Forradellas, F.Ibañez, R.Berlanga y G.Martín "Manejo de constraints en CHIP", PRODE-91, 234-244, 1991.

[10] I.Meiri "Combining qualitative and quantitaive constraints in temporal reasoning" Proceedings AAAI-91, 260-267, 1991.

[11] H.Kautz and Ladkin "Integrating metric and qualitative temporal reasoning" Proceedings AAAI-91, 241-246, 1991.

[12] R.Dechter, I.Meiri and J.Pearl "Temporal constraint networks". Artificial Intelligence, 23:123-153, 1984.