UN MODELO PARA EL TRATAMIENTO DE SISTEMAS DINÁMICOS BASADO EN LA SATISFACCIÓN DE RESTRICCIONES

R. Forradellas[1], F. Ibañez,[1] R. Berlanga[2]

RESÚMEN

En numerosas aplicaciones industriales complejas de plaificación y scheduling, resulta frecuente encontrar casos donde un problema ya resuelto debe ser reconsiderado a causa de una ligera modificación en la instancia de dicho problema. Estas modificaciones se originan generalmente a partir de sucesos externos que implican un cambio de creencias y en consecuencia el conjunto de soluciones obtenido para el problema resuelto ha de modificarse.

Estos casos son referidos generalmente como *problemas dinámicos*, frente a los problemas *estáticos*. En los primeros, el conjunto de soluciones puede ser ligeramente modificado, mientras que en los segundos, el conjunto de soluciones es fijo e inalterable.

El tipo de problemas que nos preocupa se refieren a problemas modelados a través de restricciones, concretamente, restricciones lineales sobre variables de dominio finito. Estos tipos de problemas son estáticos, cuando las soluciones obtenidas no son reconsideradas ante el cambio de la instancia del problema. Los casos dinámicos antes expuestos son resueltos iniciando de nuevo el proceso de resolución con la instancia modificada como si fuese un problema diferente.

Un resolvedor de problemas que reconsidere las soluciones obtenidas en un problema anterior ante un cambio ligero de su instancia lo denominaremos dinámico, frente a la denominación de estático antes utilizada. Así pues, un Sistema Dinámico de Restricciones (SDR) será aquel que considere las soluciones obtenidas para resolver la instancia modificada. Al contrario de los sistemas estáticos, un SDR plantea las modificaciones de las instancias como un único problema.

En este trabajo definiremos un modelo de SDR e identificaremos el tipo de transiciones permitidas en el mismo, y discutiremos como abordar la resolución dinámica del SDR desde diferentes aproximaciones. Además, se propondrán varios métodos para el manejo dinámico de un sistema de restricciones. Finalmente, discutiremos brevemente que opción de las analizadas es la más adecuada para los problemas que estamos abordando.

1.- MODELO DEL SISTEMA DINÁMICO CON RESTRICCIONES –SDR-

En general el costo de la búsqueda de una solución o todas las soluciones, en un Sistema con Restricciones, requiere de una cantidad de tiempo bastante elevada, más aún, impredecible. En muchos casos el problema se acentúa cuando, además, se imponen restricciones de optimización sobre el Sistema, como sucede en las aplicaciones de planificación industrial.

Por otro lado, es muy frecuente encontrar en aplicaciones de planificación, en los que, una vez ya representado y resuelto el problema basado en restricciones, algunos valores de variables o de restricciones del mismo problema

---

[1]LISI/IdeI – UNSJ {*kike, fibanez*}*iinfo.unsj.edu.ar*

[2]Dpto. Informática – UJI berlanga@inf.uji.es

Parte de las investigaciones presentadas en este trabajo fueron realizadas en el marco del Proyecto CICITCA (UNSJ) "*Resolución de Problemas de Asignación de Recursos aplicando tecnicas de Inteligencia Artificial*"

se modifican en virtud de algún suceso externo. Por ejemplo, en los casos de re-planificación de tareas ya programadas, debido a que algún dominio de alguna variable ha sido modificado a causa de un cambio externo.

Denominamos a esta clase de problemas como Sistemas Dinámicos de Restricciones (SDR), cuyo esquema puede verse en la Figura 1, donde cada $\wp_i$ está compuesto por un conjunto de posibles valores de variables y un conjunto de restricciones.

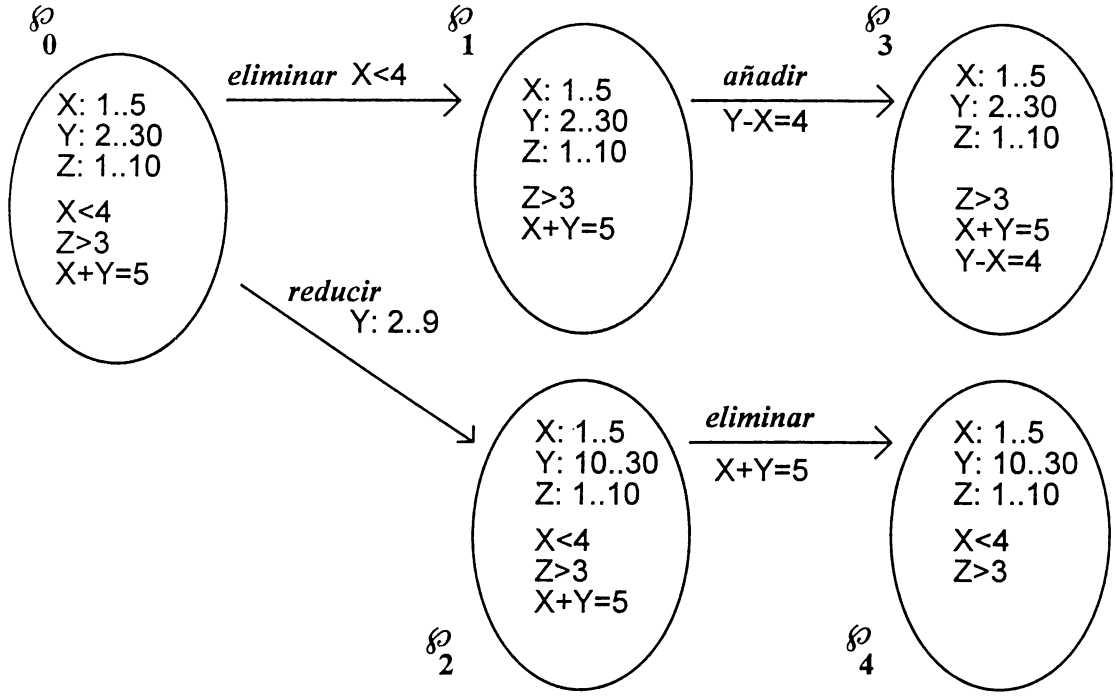


Figura 1: Sistema Dinámico de Restricciones

Un SDR para nuestra aproximación, se plantea como un único problema denominado $\wp$, y no como un conjunto de problemas $\wp_0$, $\wp_1$, $\wp_n$, independientes que deban resolverse separadamente. En este caso un SDR es visto como una sucesión temporal de Sistemas de Restricciones, que denominados *estados del SDR*. De éste modo el paso de un sistema (o problema) a otro se realiza a través de un cambio en el conjunto de dominios o en el conjunto de restricciones. A estos cambios los denominamos *transiciones del SDR*.

Estas transiciones pueden ser de eliminación o agregado de un dominio o una restricción en el SDR, como se muestra en la Figura 1. En estos casos, todo SDR parte de una instancia del problema de restricciones inicial $\wp_0$. Esta instancia deriva en otras a partir de las anteriormente mencionadas transiciones.

A partir de ahora supondremos que el tipo de SDR que estamos tratando, es resuelto a través de la combinación de dos procesos:

I) un algoritmo de propagación de restricciones, y

II) un esquema de backtracking.

En este caso, el tratamiento se hace en base a los CSP's estáticos, por lo tanto las soluciones obtenidas no son reconsideradas ante el cambio de la instancia

del problema. La necesidad del algoritmo de backtracking surge de la incompletitud de los algoritmos de propagación utilizados (basados en *arco-consistencia*, *path-consistencia*, etc.) que por contra ofrecen una buena performance [COOPE92] [McWOR92] [VANHE92] [LHOMM93].

Los casos dinámicos antes expuestos son resueltos iniciando de nuevo el proceso de resolución con la instancia modificada como si fuese un problema diferente. De esta forma, un SDR puede ser visto como una sucesión cronológica de Sistemas Estáticos de Restricciones, que denominamos *estados*, de modo que el paso de un estado a otro se realiza a través de un cambio en el conjunto de los dominios de las variables o de las restricciones. Estos cambios los denominamos *transiciones*, cuyas características presentaremos a continuación

1.2.- Transiciones en el SDR

En el Sistema general, cada instancia del SDR es un problema $\wp_i$. que está compuesto por el conjunto de posibles valores de las variables V_i, V_n, y por el conjunto de restricciones σ.

Como antes indicamos, un SDR puede caracterizarse por un conjunto de estados obtenidos por medio de transiciones. Dichas transiciones se identifican con los posibles cambios que pueden producirse en la instancia del problema manejado. Los estados y las transiciones forman un grafo, cuyo origen es la instancia inicial $\wp_0$, a partir de: $\wp_0 = <V_0, \sigma_0>$, es posible pasar a los otros estados, como muestra la Figura 1.

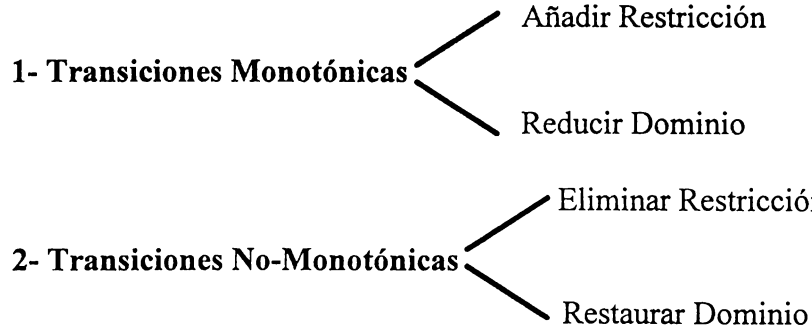
La instancia de un sistema de restricciones sólo puede ser modificada a través de los dominios de las variables o del conjunto de restricciones definidas sobre éstas. De este modo distinguiremos varios modos de transición de una instancia de problema:

- añadir nueva restricción al conjunto de restricciones de la instancia actual
- eliminar valores del dominio de una variable del sistema de restricciones
- eliminar una restricción del conjunto de restricciones de la instancia actual
- restaurar valores del dominio de una variable del sistema de restricciones

Cada una de estas restricciones implica una alteración en el conjunto de soluciones de la instancia modificada. Podemos distinguir dos clases de transiciones de acuerdo a si el conjunto de soluciones aumenta o disminuye:

- *Transiciones Monotónicas*, aquellas que reducen el conjunto de soluciones de la instancia modificada
- *Transiciones No-Montónicas*, aquellas que amplían el conjunto de soluciones de la instancia modificada

La distinción entre éstos dos tipos de transiciones es importante desde el punto de vista de la incrementalidad de la resolución del Sistema de Restricciones. En el caso de las transiciones monotónicas, las nuevas soluciones se obtienen directamente de la instancia del problema anterior a la modificada, eliminando aquellas que no cumplen las modificaciones introducidas. Sin embargo, las transiciones no-monotónicas necesitan conocer la "historia" del proceso de resolución para decidir que partes del espacio de soluciones necesitan ser ampliadas. Siguiendo esta clasificación, las transiciones mencionadas se ordenarían de acuerdo al Esquema 1.



Esquema 1: Tipos de Transiciones en SDR's

Otro aspecto importante que distingue estos dos tipos de transiciones es el concepto de integridad. En este caso, un proceso de resolución se dice que es *íntegro* si las opciones descartadas, no forman parte del conjunto de soluciones del sistema de restricciones.

Para las monotónicas es importante imponer la condición de integridad. Los diferentes algoritmos de propagación que permiten eliminar opciones no válidas han de asegurar que dichas opciones realmente no forman parte del espacio de soluciones [NADEL89] [KUMAR92]. En cambio, en las no-monotónicas este concepto no tiene el mismo sentido.

Dado que una transición no-monotónica amplia el espacio de soluciones, el sistema sigue siendo íntegro ya que este proceso no implica en ningún momento desconsiderar opciones, si no todo lo contrario. Ampliemos como queramos el espacio de soluciones que la propiedad de integridad seguirá cumpliéndose. Sin embargo, lo que buscamos es ampliar el espacio de soluciones lo justo, es decir, no queremos introducir nuevas opciones que siguen siendo no válidas incluso con la instancia actual. Este sería un concepto paralelo al de integridad, pero aplicado a las transiciones no-monotónicas. A esta propiedad la denominaremos *expansión mínima* del espacio de soluciones.

Los conceptos de *hipótesis, suposiciones* y *consistencia*, especializados para el problema que nos preocupa, ofrecen una taxonomía de SDR's de acuerdo al criterio de qué se considera como hipótesis *-partes invariantes del problema-* y qué se consideran como suposiciones *-parte dinámica del problema-*. A tal fin se consideran los siguientes casos:

a.- *Dominios* como *Suposiciones* y *Restricciones* como *Hipótesis*.

b.- *Dominios* como *Hipótesis* y *Restricciones* como *Suposiciones*.

c.- *Dominios* y *Suposiciones* pueden ser tanto *Suposiciones* como *Hipótesis*.

Para tratar estos problemas se desarrollarán algoritmos que permitirán realizar operaciones adicionales de los Sistemas de Restricciones actuales tales como: añadir, eliminar o modificar un dominio o una restricción. De esta manera el sistema propuesto permitirá expresar en forma explícita las transiciones posibles en un SDR.

2.- Tratamiento de Restricciones en SDR's

Se propone un tratamiento a realizar en el proceso de satisfacción de restricciones para el modelo del SDR propuesto. Un SDR quedará constituido por:

- un grafo de transiciones
- una *Estructura de Mantenimiento*
- un conjunto de algoritmos para cada transición

En principio nos centraremos en los dos primeros componentes. Los algoritmos del tercer componente serán evaluados más adelante, de acuerdo a su complejidad e integridad

2.1.- Estructuras de Mantenimiento –EM-

Para un SDR indicamos que el rasgo más importante de éste era tratar la secuencia de modificaciones de las instancias de los problemas como un único problema que podía ser representado mediante un grafo de transiciones. Dicha consideración implica que los algoritmos de resolución deben ser incrementales tanto para las transiciones monotónicas como para las no-monotónicas.

Existe una serie de estrategias para la resolución de sistemas de restricciones, las cuales pueden aplicarse incrementalmente cada vez que se introduce una nueva restricción; incluyendo entre éstas las transiciones referentes a reducciones de dominio (restricciones unarias). La incrementalidad de dichos algoritmos permite tratar las Transiciones Monótonicas ya descriptas.

Estos algoritmos no podrían aplicarse directamente para el tratamiento incremental de las Transiciones No-Monótonicas. Debido a que los algoritmos de propagación de restricciones no recuerdan porqué un determinado valor fue descartado del dominio de una variable, o porqué el sistema de restricciones falló bajo determinadas condiciones. Esta falta de recuerdo es, por otro lado, el responsable de la eficiencia de estos algoritmos [VANHE89] [KUMAR92]. Un problema similar lo encontramos con el esquema de backtracking, que no es capaz de reconocer "fallos tipos" que suceden de forma regular en el árbol de búsqueda.

Ya que estos métodos de resolución no pueden recordar la historia del proceso de búsqueda, tampoco podrán dar pistas acerca de qué decisiones habría que tomar para que un Sistema de Restricciones detectado inconsistente pueda tornarse consistente. Este tipo de inferencias se denominan resolución de conflictos dentro de la terminología de los Sistemas de Mantinimiento de la Razón (SMR/*TMS*) [DeKLE89].

En nuestro caso, determinar el tipo de información que debe ser almacenada es una tarea comprometedora y, en definitiva, la eficiencia en la resolución del SDR dependerá de esta elección. Exiten varias aproximaciones de métodos de resolución que hacen uso del recuerdo tanto de los pasos durante la propagación de las restricciones, como los fallos tipo producidos en el proceso de backtracking (*nogoods*) [DeKLE89] [MARUY92]. Estos son métodos basados en SMR's, y métodos dinámicos sobre CSP's [BESSI92], [DECHT88].

En los métodos basados en SMR, concretamente aquellos basados en Suposiciones, se utiliza el propio algoritmo de propagación de etiquetas para las creencias del sistema para reproducir la propagación de restricciones. Con ello, el proceso de propagación queda enteramente registrado en forma de suposiciones y dependencias entre suposiciones. Las definiciones de los SMR's permiten:

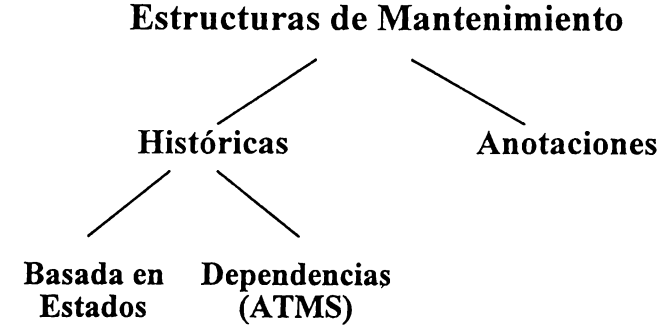
- manejar situaciones dinámicas, mediante el cambio de creencias y,
- la resolución de conflictos mediante cambio de contextos.

Sin embargo, estos sistemas se muestran muy ineficientes debido principalmente a dos razones. La primera es el gran número de suposiciones que se manejan, al menos una por cada paso de propagación, el cual es generalmente

intratable. El segundo es el tratamiento automático en anchura, [McDER91] para la determinación de los contextos de un sistema de restricciones.

Los métodos dinámicos tratan de superar estas deficiencias limitando el tipo de información que ha de almacenarse. Por ejemplo, en la propagación de restricciones sólo debiera almacenarse los casos en los que se ha producido una reducción de un dominio a causa de una restricción. Sin embargo, existen pocos formalismos de este estilo en la literatura, y además, están enteramente orientados a CSP's y no a sistemas de restricciones como los que venimos tratando.

Vamos a proponer diferentes aproximaciones para el tratamiento de SDR, para ello, haremos uso de la clasificación mostrada en el Esquema 2, que se basa en el tipo de Estructura utilizada.



Esquema 2: Clasificación de las Estructuras de Mantenimiento

En general una Estructura de Mantenimiento (*EM*) debe reaccionar adecuadamente ante la nueva información aportada por el *Solucionador de Problemas*. En nuestra caso este *Proceso de Actualización* se realizará mediante un CSP. Para lograr este objetivo se incorporarán al CSP nuevos algoritmos para tratar las denominadas *Anotaciones*. El proceso de actualización dependerá en cada caso del tipo de transición que se produzca en el Sistema de Restricciones.

El estudio de cada EM se hará en función de su forma de reacción ante dichas transiciones. Por otro lado es necesario conocer *cómo* el CSP puede hacer uso de la EM, es decir, su utilidad y efectividad. A continuación se estudiarán las Transiciones Monotónicas y No-Monotónicas que se producen en las *EM Históricas*, y en las *EM por Anotaciones*. Sobre éstas últimas además, se desarrollarán nuevos algoritmos para la Satisfacción de Restricciones en SDR.

2.1.1.- EM Históricas Basadas en Estados

Hemos englobado bajo este nombre aquellas aproximaciones que almacenan la "historia" de cómo se llegó a una conclusión dada. Una *EM* basada en estados es aquella que simplemente almacenaría el "estado" del Sistema de Restricciones después de cada transición.

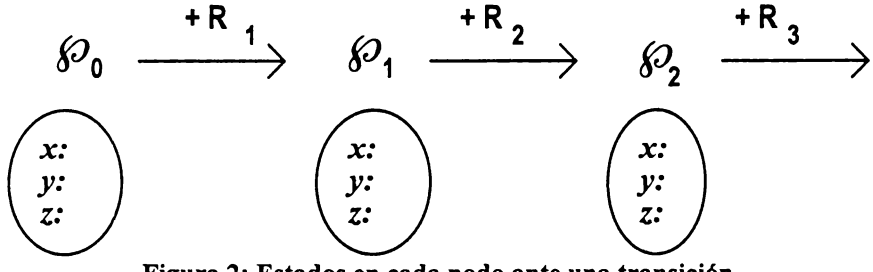


Figura 2: Estados en cada nodo ante una transición

Así, el grafo de transiciones (Figura 2) contendría en sus nodos la información referente a los *dominios* de las *variables* después de aplicar determinada transición (en caso de que la transición tuviese éxito). Con ello la *EM Histórica*, sería el grafo de transiciones conteniendo los estados en cada nodo.

2.1.1.1.- Transiciones Monotónicas

Estas transiciones sólo requieren el estado inmediatamente anterior a la transición a realizar. Supongamos que estamos en el estado $\wp_k$ y queremos obtener el nuevo estado $\wp_{k+1}$ tras añadir la restricción **R**.

El algoritmo para esta transición se basaría en propagar los efectos introducidos por la restricción **R** sobre las variables de $\wp_k$.

```
function add(<Vk,σk>,R,<Vk+1,σk+1>)    boolean;
  begin
    PILA ← NIL;
    if propagar(σk ∪ R)   then
                            σk+1 ← σk ∪ R
                            Vk+1 ← Vk ∪ {v:v∈Var(R) ∧ v∉Vk}
                            add ← TRUE
                          else add ← FALSE
  end
end.
```

En éste algoritmo, `Var(R)` devuelve el conjunto de variables de la restricción **R**, cómo por ejemplo: `Var(x+y #= z)` devuelve `{x,y,z}`

```
function propagar(σ):boolean;
  begin
    if PILA ← NIL   then
                      desapilar(PILA,v);
                      ∀(R∈σ ∧ v∈Var(R)) do
                      if revise(R)  then
                                propagar ← propagar(σ)
                                     else
                                propagar ← FALSE
                    else propagar ← TRUE;
  end.
```

La función `revise` consiste en aplicar la regla de inferencia *Partial Looking Ahead Inference Rule* (*PLAIR*) sobre las variables involucradas, *apilando* aquellas variables que han sido modificadas [FREUDE92]. Además ésta función comprueba los límites de los dominios de las variables, para hacer que la restricción **R** sea *localmente* consistente.

2.1.1.2.- Transiciones No-Monotónicas

En este caso es necesario recorrer el grafo de transiciones para restablecer el nuevo estado $\wp_{k+1}$. Necesitaremos conocer la historia de cómo el nuevo estado $\wp_k$ ha sido obtenido a través de las distintas transiciones $\wp_i$ con $0 \leq i \leq k$.

Analizaremos en primer lugar las transiciones que involucran *eliminar una restricción*. La aproximación más sencilla consistiría en buscar en el grafo de

transiciones aquel estado al cual se le aplicó la restricción R que es la que deseamos eliminar para encontrar el nuevo estado $\wp_{k+1}$.

Por ejemplo, en la Figura 3 puede verse que para la obtención de $\wp_{k+1}$ se realizará aplicando de nuevo todas las transiciones exceptuando aquellas que deseamos eliminar.

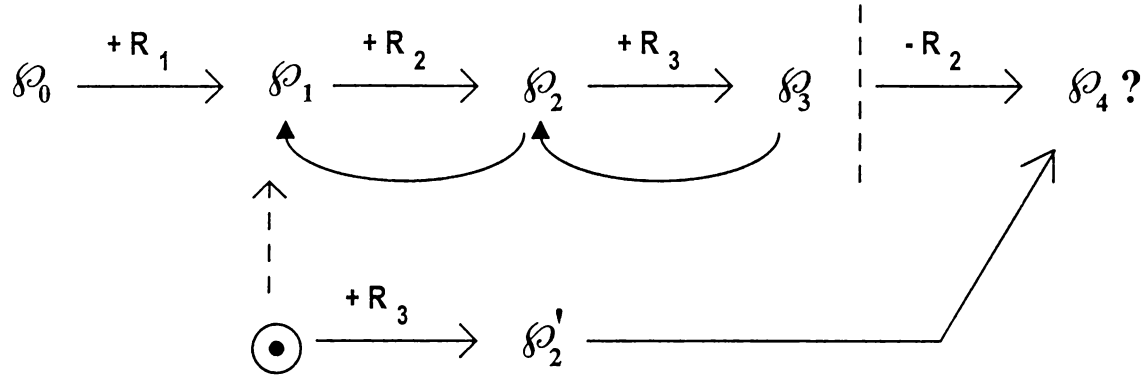


Figura 3: Grafo de Estados de una Transición No-Monotónica

```
function delete(EM, R, <V_{k+1}, σ_{k+1}>):boolean;
  begin
    if buscar (EM, R, <V_i, σ_i>) then
              actual ← <V_i, σ_i>
              Ractual ← arco(EM, i+1);  j ← i+1
                    repeat
                       add(actual, Ractual, <V_act, σ_act>)
                       actual ← <V_act, σ_act>;  j ← i+1
                       Ractual ← arco(EM, j)
                    until i=j
  end.
```

El principal inconveniente de ésta aproximación es que depende gradualmente de la posición en el grafo de transiciones de la restricción que deseamos eliminar. Para estados cercanos al actual el proceso será muy rápido, en cambio, para estados cercanos al inicial el algoritmo será prácticamente equivalente a empezar desde cero. En estos casos resulta innecesaria la *Estructura de Mantenimiento.*

2.1.2.-EM Históricas Basadas en Dependencias

Estas estructuras corresponden a aproximaciones que han utilizado los Sistemas de Mantenimiento de la Razón, en la implementación de los modelos de ATMS y JTMS, para el proceso de resolución de Sistemas de Restricciones [SARAS91].

2.1.2.1.- Críticas a los SMR's

Sobre los distintos modelos y sus extensiones, desarrollados hasta ahora, para tratar la problemática del Mantenimiento de la Razón y su relación con lo referente a la Resolución de Problemas en Sistemas Basados en el Conocimiento, se concluye con los siguientes requerimientos de características ventajosas a incorporar en un nuestro paradigma de especialización:

a) formalización independiente de cualquier implementación en concreto, para permitir la utilización en el mayor ámbito posible de aplicaciones de resolución de problemas.

b) tratamiento del problema de la inferencia de modo que sea el sistema el encargado de calcular las dependencias, y de esta forma determinar automáticamente la relación de dependencias entre las creencias anteriores cuando sea generada una nueva creencia.

c) tratamiento de la no-monotonicidad, de forma que permita un modo de razonamiento no-monotónico al momento de realizar actualizaciones en la Base de Conocimiento.

d) tratamiento simultáneo de múltiples contextos, lo que permitirá al sistema determinar los contextos en los que un conocimiento específico es aplicable y adoptar soluciones de inferencia simultánea en contexto múltiple.

e) adoptar un modelo de relación de dependencia basado en suposiciones, lo que permitirá utilizar algoritmos de búsqueda dirigidos por dependencias más eficientes frente a los tradicionales de búsqueda por backtracking.

A partir de estos requerimientos se adoptará el modelo del SMR/*TMS* basado en suposiciones (ATMS) propuesto por DeKleer [DeKLE89] que es el que más se aproxima para soportar los mencionados requerimientos, cuyas características se resumen en:

- estar basado en suposiciones
- ser de contexto múltiple
- contemplar la problemática de la no-monotonicidad
- evitar la búsqueda en partes del árbol determinadas como inconsistentes.
- evitar la búsqueda de soluciones innecesarias.

De esta forma se pretende que a medida que el solucionador de problemas proporciona justificaciones y suposiciones al sistema, se va construyendo incrementalmente el contexto. Cuando surgen conflictos son tratados por los mecanismos del sistema para evitar la contradicción (o fallo), mediante un cambio de creencias o cambio de ambiente de las variables (elección de otro contexto). De esta forma evitan la búsqueda en aquella parte del árbol que ha sido previamente determinada como inconsistente.

En las implementaciones existentes de estos sistemas, en el marco de la Satisfacción de Restricciones, se observa una gran ineficiencia y alto coste espacial en el almacenamiento de todo el proceso de propagación [VANHE89], [FORRA93], [FORRA94]. Debido a las características de Razonamiento No-Monotónico de estos sistemas, se utilizarán los conceptos de los mismos, pero todos especializados para el tratamiento del problema de transiciones dinámicas.

2.1.3.- EM por Anotaciones

En este caso, en lugar de almacenar la historia del Sistema de Restricciones guardando los sucesivos estados, anotaremos *cada valor eliminado* por una transición junto con la *causa* de su eliminación.

Esta *EM* sustituirá al grafo de transiciones y sus estados. En cierto modo, ésta estructura almacena los cambios producidos por las transiciones tomando éstas como las *causas* de dichos cambios.

En este caso cada variable V_j tiene asociada una estructura de la forma:

$$< V_j, [(v_1^+, R_1^+), ..., (v_n^+, R_n^+)], [(v_1^-, R_1^-), ..., (v_m^-, R_m^-)] >$$

donde R_i^+ es la restricción que ha reducido el límite superior de la variable V_j al valor v_i^+ y R_i^- es la restricción que ha actualizado el límite inferior al valor v_i^-.

El par (v_n^+, R_n^+) corresponde a la última restricción que modificó el extremo superior de la variable V_j y el par (v_m^-, R_m^-), a la última restricción que modificó el extremo inferior. A partir de estas estructuras se define la función $Dom(V_k)^+$ la cual, a partir de la estructura asociada a la variable V_k, devuelve el extremo superior actual de la variable V_k, esto es, v_n^+. De modo similar, la función $Dom(V)^-$, devuelve el extremo inferior de V_k esto es, v_m.

Además de estas tuplas, cada variable tendrá asociada los siguientes datos:

<Dominio-Actual, Ultima-Restricción, Reducción-tipo>

donde:

- *Dominio-Actual,* contiene el domino actual de la variable en cuestión,
- *Ultima-Restricción,* contiene el identificador de la última restricción que redujo *V*,
- *Reducción-tipo*, expresa el tipo de reducción producido por la última restricción:

'+': si fue el extremo superior

si fue el extremo inferior

'±': si fue en ambos extremos

A partir de estas tuplas podemos describir las sucesivas reducciones de cada variable y las causas de éstas reducciones, estas son, las restricciones.A modo de ejemplo, la Figura 4 muestra el dominio actual de una variable junto con los segmentos que han sido eliminados indicando la causa de cada eliminación.

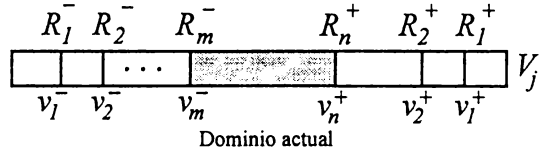


Figura 4

Las siguientes definiciones serán de interés en el desarrollo de los algoritmos de Propagación y Mantenimiento:

Def. 1 Si dos valores de extremos superiores de una variable genérica *V*, v_i^+ y v_j^+, (véase Figura 5), pertenecen a restricciones distintas y $v_i^+ > v_j^+$, entonces diremos que v_j^+ subsume a v_i^+.

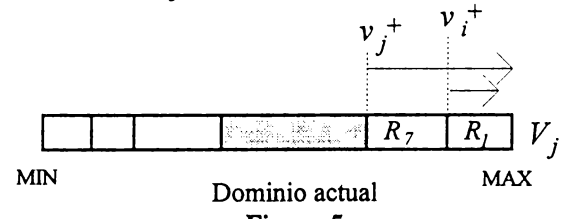


Figura 5

Def 2 Si dos valores de extremos inferiores de una variable genérica, v_i^- y v_j^- pertenecen a restricciones distintas y $v_i^- < v_j^-$, entonces diremos que v_i^- subsume a v_j^-.

2.1.3.1.- Regla de Simplificación de la EM

Se propone la siguiente Regla:

Regla 1 Si dos valores de extremos superiores (contiguos), de una variable *V*, tienen asociada a una misma restricción, entonces eliminaremos el menor de ellos.

El mismo mecanismo se aplica para el caso de los extremos inferiores de la variable *V*.

2.1.3.2.- Regla de Inferencia *PLAIR* para CSP Dinámicos

Se propone la siguiente Regla:

Regla 2 Dada una Restricción Lineal, la regla de inferencia *Partial Looking Ahead Inference Rule - PLAIR*, infiere cuales son los extremos de las variables que de forma local hacen válida la restricción.

La regla se formula sobre la forma normal de una restricción, a partir de las formulaciones existente [VANHE89]. Para el caso de la desigualdad estricta, la aplicación de la Regla de Inferencia *PLAIR* a una variable, devuelve el valor FALSE, si la restricción falla, para todo valor de la variable, y retorna el valor TRUE, en este caso, modificando, si es necesario, las listas de justificaciones asociadas a la variable. El Algoritmo *PLAIR_CSPDn*, produce tal efecto.

En el algoritmo PLAIR_CSPDn las denominadas Ecuación A y Ecuación B (no se escriben para lograr una mejor presentación del algoritmo) se corresponden con las siguientes ecuaciones:

$$(-\sum_{a_i<0} a_i \;\; Dom(V_i)^+ - \sum_{a_j>0} a_j \;\; Dom(V_j)^- - a_0)/a_k \quad \textbf{[Ecuación A]}$$

$$(\sum_{a_i<0} a_i \;\; Dom(V_i)^- + \sum_{a_j>0} a_j \;\; Dom(V_j)^+ - a_0)/a_k \quad \textbf{[Ecuación B]}$$

El Algoritmo PLAIR_CSPDn funciona de la siguiente manera:

```
function PLAIR_var(<r,Vk>):boolean;
    begin
        PLAIR_var ← TRUE;
        coef(r,[a0, a1,      an]);
        if ak > 0
            then begin
                    Right ← Ecuación A;
                    if ¬∃vk∈Vk:vk<Right
                      then PLAIR_var ←### FALSE
                      else    begin
                                New+ ←### max{vk∈Vk:vk<Right};
                                add_just+(vk,<New+,r>)
                              end
                 end
            else  begin
                    Left ← Ecuación B;
                    if ¬∃vk∈Vk:vk>Left
                        then PLAIR_var ← FALSE
                        else begin
                                New- ← min{vk:Vk:vk>Left};
                                add_just-(vk,<New+,r>)
                             end
                  end
    end
```

- Si el coeficiente asociado a la variable V_k es positivo, sólo puede modificarse el extremo superior de la variable.
- La variable New^+ contiene el máximo valor perteneciente a la variable V_k que verifica la restricción.
- Si tal valor no existe, la función devuelve FALSE

caso contrario, agrega a la lista de justificaciones asociadas al extremo superior de la variable V_k, el valor calculado en la variable New^+, junto con la restricción $r(add\_just^+)$.

- Si el coeficiente asociado a V_k es negativo, sólo puede modificarse el extremo inferior de la variable. Este valor, en caso de existir, es el mínimo valor perteneciente a la variable V_k que verifica la restricción.

2.1.3.3.- Actualizaciones de la EM

En el SDR se tratarán las siguientes actualizaciones de la Estructura de Mantenimiento, motivadas por algún suceso externo:

a) ***Reducción de Dominios***: Supongamos que la variable *V* ha modificado su dominio, reduciendo su extremo superior a v^+ debido a la restricción *r*.

Surgen dos casos a considerar para el proceso de reducción de dominios:

1) Si la restricción *r* coincide con la última restricción que redujo *r*, entonces simplemente cambiaremos el último elemento de la lista de extremos superiores referentes a la restricción *r* por el nuevo extremo v^+ Ello en virtud a la regla de simplificación antes enunciada. Por último, modificaremos los atributos de *Dominio-Actual* con el valor de v^+.
2) En caso contrario, añadiremos el valor v^+ a la lista de extremos superiores y modificaremos los atributos de *Dominio-Actual* y *Ultima-Restricción* con los valores de v^+ y *r* respectivamente.

b) ***Eliminación de una Restricción***:

Los valores almacenados en la *Estructura de Mantenimiento* representan las distintas modificaciones que va sufriendo la variable_dominio junto con sus causas. Cuando se elimina una restricción, las reducciones producidas por ésta deben ser restituidas, siempre y cuando esa restricción sea la única causa de dichas reducciones. Es aquí donde juega un papel importante el concepto de 'subsume'. Si un extremo es subsumido por otro, no podrá restituirse. Si se restituyera dicho extremo, la restricción perteneciente al extremo que subsume volvería a eliminar los valores recién establecidos. Además de la restauración de valores, la eliminación de una restricción implica suprimir todas aquellas anotaciones que le hagan referencia. Esta operación deberá ir acompañada de la Regla de Simplificación para eliminar las redundancias de la estructura modificada.

```
function Restore_bounds(<V,r>):boolean;
  begin
     if last_constraint(V)=r
        then begin
               case Reduction_type(V) of
                  '+'   drop_last_just+(V)
                  '-'   drop_last_just-(V)
                        begin
                             drop_last_just+(V);
                             drop_last_just-(V)
                        end
               endcase
               Restore_bounds ← TRUE
             end
        else Restore_bounds ← FALSE
  end
```

La función Restore_bounds, toma como argumentos una variable (V), y una restricción (r). Si r es la última restricción que afectó la variable V, esta función restaura el dominio asociado a V, eliminando las justificaciones en las que la restricción r está involucrada.

La función last_constraint toma como argumento una variable y devuelve la última restricción que la ha afectado.

La función drop_last_just$^+$ (de modo similar drop_last_just$^-$) elimina el último par de la lista de justificaciones correspondiente al extremo superior (inferior) de la variable.

2.1.3.4.- Algoritmos de Propagación en CSP's Dinámicos

Los algoritmos que van a presentarse a continuación son una variación del algoritmo de Waltz [WALTZ72] extendido a variables de dominios finitos, y soportando la EM necesaria para realizar las Transiciones No-Monotónicas. Del mencionado algoritmo se adopta la pila de propagación global, denominada Stack en aquellos algoritmos, cuya misión es almacenar las variables modificadas para la posterior revisión de las restricciones donde se involucran.

2.1.3.4.1.- Algoritmos para Transiciones Monotónicas

La función Refine usada para CSP's estáticos, se puede modificar, de modo tal que pueda usarse para resolver CSP's dinámicos.

La llamada: add_just$^+$(Var,<Dom(Var)$^+$,r>), inserta <Dom(Var)$^+$,r> en la lista de justificaciones del extremo superior de la variable Var.

El procedimiento add_just$^-$, corresponde a la lista de justificaciones del extremo inferior.

Algoritmo-1.1

```
function Refine(r):boolean;
  begin
     Refine ← TRUE;
     ∀Var∈r do
       begin
          Aux1 ← Dom(Var)+;
          Aux2 ← Dom(Var)-;
          P ← PLAIR_var(r,Var)
          if PLAIR_var(r,Var)
             then begin
                    if Aux1≠Dom(Var)+ then
                      begin
                        add_just+(Var,<Dom(Var)+,r>);
                        push(Stack,Var)
                      end
                    if Aux2≠Dom(Var)- then
                      begin
                        add_just-(Var,<Dom(Var)-,r>);
                        push(Stack,Var)
                      end
                  end
                  else Refine ← FALSE
       end
  end.
```

La función Refine modifica los dominios de las variables involucradas en r aplicando la regla de inferencia *PLAIR*. La parte que difiere con el procedimiento visto para las otras aproximaciones, consiste en realizar la actualización de la EM, en este caso basada en anotaciones sobre variables. La pila de propagación Stack se actualiza agregando las variables modificadas para revisar posteriormente las restricciones donde participan.

Algoritmo-1.2 **-Añade Restricción-**

```
function Add(<Vk,σk>,r,var<Vk+1,σk+1>):boolean;
  begin
     Stack ← NIL;
     if Refine(r)
        then begin
               failure ← FALSE;
               σk+1 ← σk∪{r}
               while not empty(Stack) and not failure do
                  failure ← propagate(<Vk+1,σk+1>);
             end
        else failure ← FALSE;
     Add ← failure;
  end.
```

La función Add, toma como argumentos, el par $<V_k, \sigma_k>$ formado por el conjunto de variables V_k y por el conjunto de restricciones σ_k correspondientes al CSP del estado actual y la restricción r que debe añadirse y devuelve, el par $<V_{k+1}, \sigma_{k+1}>$, constituido por el conjunto de variables y el conjunto de restricciones de nuevo CSP.

Algoritmo-1.3 **-Propagación de Reducciones-**

```
function propagate(<V,σ>):boolean;
  begin
     propagate ← TRUE;
     pop(Stack,Var);
     ∀ (r∈σ/Var∈r) while propagate do
              propagate ← propagate and Refine(r);
  end
```

2.1.3.4.2.- Algoritmos para Transiciones No-Monotónicas

Algoritmo-2.1 **-Eliminación de una Restricción-**

La función Remove elimina de la situación actual del Sistema de Restricciones $<V_k, \sigma_k>$ la restricción r obteniendo un nuevo Sistema de Restricciones. El nuevo sistema contendrá las restricciones existentes en σ_k menos la restricción r. Si ésta no se encuentra en dicho conjunto, la función devolverá el valor FALSE. Los dominios de las variables involucradas han de ser revisados con el objeto de restaurar los valores que fueron eliminados de forma directa o indirecta.

Para el caso de la eliminación de restricciones, los algoritmos son más complejos, debido a que la restauración de las variables involucradas en la restricción que se elimina, origina una posible restauración de otras variables, proceso que denominaremos "*Propagación Inversa*".

Pero además, en este proceso de propagación, pueden restaurarse variables, que deben revisarse posteriormente, debido a que, restricciones que no produjeron efecto alguno sobre los dominios originales, pueden producir reducciones sobre los dominios restaurados. El proceso consiste entonces en dos etapas.

```
function Remove(<V_k,σ_k>,r,var<V_k+1,σ_k+1>):boolean;
  begin
     Stack1 ← NIL;
     Stack ← NIL
     if r∉σ_k then Remove ← FALSE
              else
                 begin
                    ∀Var∈r do
                    if Restore_bounds(<Var,r>)
                          then begin
                                    push(Stack1,Var);
                                    push(Stack,Var)
                               end;
                    delete(<Var,r>);
                    σ_k+1 ← σ_k-{r};
                    while not empty(Stack1) do
                               propagate-i(<V_k+1,σ_k+1>);
                    Remove ← TRUE;
                 end
     propagate(<V_k+1,σ_k+1>)
  end
```

En este algoritmo, se utilizan dos pilas `Stack` y `Stack1`. La pila `Stack1` se utiliza sólo en la primer etapa para controlar la *Propagación Inversa* que producirá como efecto la restauración de las variables correspondientes.

La pila `Stack` se usa inicialmente en la primera etapa para almacenar las variables que serán revisadas en la segunda etapa, mediante el proceso de propagación usado para el caso de reducción.

La llamada `delete(Var,r)`, elimina de las listas de justificaciones asociadas a los extremos superior e inferior de la variable `Var`, todos los pares que involucran la restricción `r`. Después de realizar este proceso, elimina los pares necesarios para que las listas no incluyan dos pares consecutivos que involucran la misma restricción.

Algoritmo-2.2 **-Propagación de Restauraciones-**

```
Procedure propagate-i(var<V_i,σ_i>);
  begin
     pop(Stack1,Var);
     ∀(r∈σ_i/Var∈r) do
        ∀V∈r and V≠Var do
              if Restore_bounds(<V,r>)
                    then begin
                              push(Stack1,V);
                              push(Stack,V);
                         end
  end
end.
```

3.- CONCLUSIONES

En este trabajo se han analizado distintas EM para resolver, tanto, Transiciones Monotónicas, como No-Monotónicas. De estas estructuras sólo las de Anotaciones permiten incrementalidad. Además, en este trabajo se ha diseñado un Modelo para representar y resolver problemas dinámicos con restricciones lineales sobre variables que toman sus valores sobre un dominio finito y discreto.

Los algoritmos diseñados están basados en la combinación de una variación de *PLAIR* [VANHE89], con una extensión del algoritmo de propagación de restricciones [WALTZ72]. En comparación con otros métodos dinámicos el sistema propuesto proporciona ventajas respecto a las aproximaciones de basado en Soportes [DECHT88] y basado en Arco-Consistencia [BESSI92]

El formalismo desarrollado en [DECHT88] parte de hipótesis muy restrictivas. En cuanto a la forma de restricciones sólo se consideran CSP's binarios, y por otro lado, los algoritmos que se presentan sólo tratan problemas de resolución de restricciones que involucran grafos de restricciones acíclicos. Para el caso del tipo de problema a los que está orientado este trabajo, normalmente es necesario incluir tanto grafos de restricciones cíclicos como también restricciones no-binarias. El enfoque basado en Arco-Consistencia, desarrollado en [BESSI92], tiene como principal limitación que el sistema no resuelve contradicciones. En este caso, si el Sistema de Restricciones falla, no existe ningún criterio para encontrar las restricciones que originaron el fallo, y por lo tanto no permite resolver una contradicción.

4.- REFERENCIAS

[BESSI92] C. Bessière, "Arc-Consistency for Non-Binary Dynamic CSPs", ECAI'92, 1992.

[COOPE92] P. Cooper and M. Swain "Arc Consistency: parallelism and domain dependence", Artif, Intell. v58, 207-235, 1992

[DECHT88] R. Dechter and A Dechter, "Belief Maintenance in Dynamic Constraint Networks", UCLA Tech. Rep., Los Angeles, 1988

[DeKLE89] J. De Kleer, " A comparasation of ATMS and CSP techniques", 11IJCAI, 1989.

[FORRA93] R.Forradellas, "An ATMS based on CLP", KBG-WS, U. of Birmingham 1993

[FORRA94] R.Forradellas, "An approach to solve Re-Scheduling Problems", STDB'94

[FREUDE92] E. Freude, R. J. Wallace "Partial Constraint Satisfaction", Art. Intell. v58,'92.

[KUMAR92] V. Kumar "Algorithms for constraint-satisfaction problems: A survey" AI Magazine, Spring 1992

[LHOMM93] O. Lhomme, "Consistency Techniques for Numeric CSPs", 13th IJCAI 1993.

[MARUY92] F. Maruyama, Y. Minoda, Sawada S and Y. Takizawa, "Constraint satisfaction and optimization using nogood justifications", 2nd. PRCAI, 1992

[McDER91] D.McDermott "A general framework for reason maintenance", AI,Vol.50/91

[McWOR92] A. K. Mackworth "Constraint Satisfaction" Encyclopedia of AI, Vol. 1, 1992.

[NADEL89] B. Nadel "Constraints Satisfaction algorithms", Comp. Intellig., 5, 1989.

[SARAS91] V. Saraswat, J. DeKleer and B. Williams, "ATMS-based constraint programming", Xeroc PARC, 1991.

[VANHE89] P. Van Hentenryck, "Constraints Satisfaction in Logic Programming", MIT Press, 1989

[VANHE92] P. Van Hentenryck, Y. Deville and C. Teng, "A generic arc-consistency algorithm and its epecializations" Artif. Intellig., Vol. 57, 1992.

[WALTZ72] D. Waltz, "Generating semantic descriptions from drawings of scenes with Shadows", MAC AI-TR-271, MIT, 1972.