Programación por Restricciones

Gerardo M. Sarria M.

Pontificia Universidad Javeriana

20 de agosto de 2008

Gerardo M.
Sarria M.

RESTRICCIONES DE DOMINIO FINITO

Gerardo M.
Sarria M.

Contenido

Problema de Satisfacción de Restricciones

Gerardo M. Sarria M.

Definión:
Un Problema de Satisfacción de Restricciones, o CSP, consiste en una restricción C sobre variables $x_1, \ldots, x_n$ y un dominio D que asigna cada variable x_i a un conjunto finito de valores, escrito $D(x_i)$. El CSP consiste entonces en resolver la restricción $C \ \wedge \ x_1 \in D(x_1) \ \wedge \ \ldots \ \wedge \ x_n \in D(x_n)$.

Estos problemas son importantes ya que pueden ser usados para modelar problemas combinatorios, tales como programación de horarios e inventarios, inteligencia artificial, procesamiento de imágenes y visión computarizada.

Gerardo M. Sarria M.

Ejemplo 1:
El problema de coloración de mapas consisten en colorear las diferentes regiones de un mapa particular con un número limitado de colores, sujeto a la condición que dos regiones adyacentes no pueden tener el mismo color.

Considere el mapa de Australia en la siguiente figura y los colores azul, rojo y amarillo.

CSPs

Cada región es asociada a una variable, WA, NT, SA, Q, NSW, V y T, y dentro del dominio $\{rojo, amarillo, azul\}$.

La siguiente restricción captura la propiedad de que regiones adyacentes no pueden ser coloreadas con el mismo color:

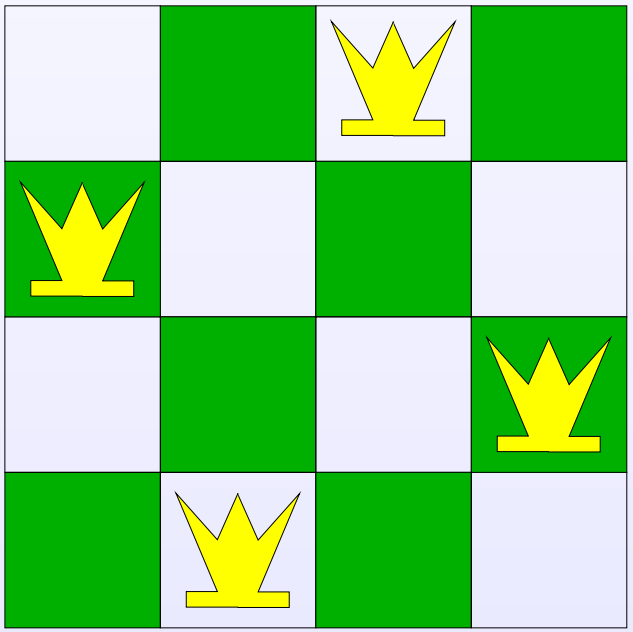
$$WA \neq NT \ \wedge \ WA \neq SA \ \wedge \ NT \neq SA \ \wedge \ NT \neq Q \ \wedge$$
$$SA \neq Q \ \wedge \ SA \neq NSW \ \wedge \ SA \neq V \ \wedge \ Q \neq NSW \ \wedge$$
$$NSW \neq V$$

Ejemplo 2:
El problema de las N-reinas consisten en colocar N reinas en un tablero de ajedrez de $N \times N$, tal que ninguna reina pueda capturar a otra reina.

Considere el problema de las 4-reinas, donde la i-ésima reina se asocia a dos variables, F_i y C_i, las cuales son la fila y la columna donde están acomodadas. El dominio de cada variable es $\{1, 2, 3, 4\}$.

Gerardo M.
Sarria M.

Una posible solución a este problema es mostrada es la siguiente figura:



CSPs

La restricción

$$F_1 \neq F_2 \ \wedge \ F_1 \neq F_3 \ \wedge \ F_1 \neq F_4 \ \wedge$$
$$F_2 \neq F_3 \ \wedge \ F_2 \neq F_4 \ \wedge \ F_3 \neq F_4$$

asegura que ningún par de reinas pueden estar en la misma fila, la restricción

$$C_1 \neq C_2 \ \wedge \ C_1 \neq C_3 \ \wedge \ C_1 \neq C_4 \ \wedge$$
$$C_2 \neq C_3 \ \wedge \ C_2 \neq C_4 \ \wedge \ C_3 \neq C_4$$

asegura que ningún par de reinas pueden estar en la misma columna.

CSPs

Las restricciones

$$C_1 - F_1 \neq C_2 - F_2 \ \wedge \ C_1 - F_1 \neq C_3 - F_3 \ \wedge$$
$$C_1 - F_1 \neq C_4 - F_4 \ \wedge \ C_2 - F_2 \neq C_3 - F_3 \ \wedge$$
$$C_2 - F_2 \neq C_4 - F_4 \ \wedge \ C_3 - F3 \neq C_4 - F_4,$$

y

$$C_1 + F_1 \neq C_2 + F_2 \ \wedge \ C_1 + F_1 \neq C_3 + F_3 \ \wedge$$
$$C_1 + F_1 \neq C_4 + F_4 \ \wedge \ C_2 + F_2 \neq C_3 + F_3 \ \wedge$$
$$C_2 + F_2 \neq C_4 + F_4 \ \wedge \ C_3 + F_3 \neq C_4 + F_4$$

forzan a que ningún par de reinas estén en la misma diagonal.

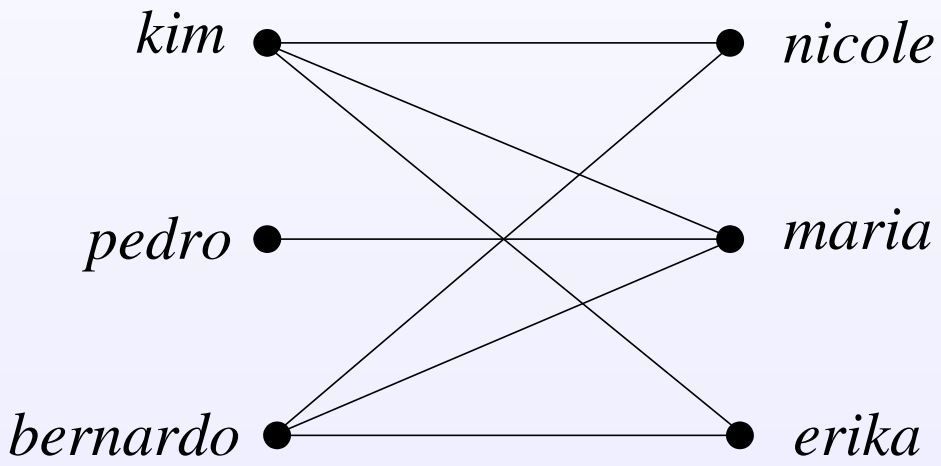
CSPs

Ejemplo 3:
El problema del matrimonio a la antigua consisten en aparear un conjunto de hombres y mujeres, tal que cada pareja se guste.

Considere el problema de un conjunto de hombres $\{kim, pedro, bernardo\}$, un conjunto de mujeres $\{nicole, maria, erika\}$, y una relación de interés definida como el conjunto de parejas

$$\{(kim, nicole), (kim, maria), (kim, erika), (pedro, maria), (bernardo, nicole), (bernardo, maria), (bernardo, erika)\}$$

CSPs



Este problema puede ser descrito usando una restricción que utilice tres variables $X_{nicole}, X_{maria}, X_{erika}$ las cuales representan el hombre elegido para *nicole*, *maria* y *erika*, respectivamente.

CSPs

Cada variable tiene el dominio $\{kim, pedro, bernardo\}$ y la restricción es

$$interes(X_{nicole}, nicole) \ \wedge \ interes(X_{maria}, maria) \ \wedge$$
$$interes(X_{erika}, erika) \ \wedge \ X_{nicole} \neq X_{maria} \ \wedge$$
$$X_{nicole} \neq X_{erika} \ \wedge \ X_{maria} \neq X_{erika}$$

Una solución a este problema es
$\{X_{nicole} \mapsto kim, X_{maria} \mapsto pedro, X_{erika} \mapsto bernardo\}$

CSPs

Las investigaciones han estado orientadas a CSPs binarios, es decir, CSPs donde las restricciones primitivas tienen máximo dos variables. El problema de coloración de mapas y el matrimonio a la antigua son ejemplos de CSPs binarios.

CSPs

Las investigaciones han estado orientadas a CSPs binarios, es decir, CSPs donde las restricciones primitivas tienen máximo dos variables. El problema de coloración de mapas y el matrimonio a la antigua son ejemplos de CSPs binarios.

Una característica agradable de los CSPs binarios es que pueden ser representados como grafos no dirigidos.

CSPs

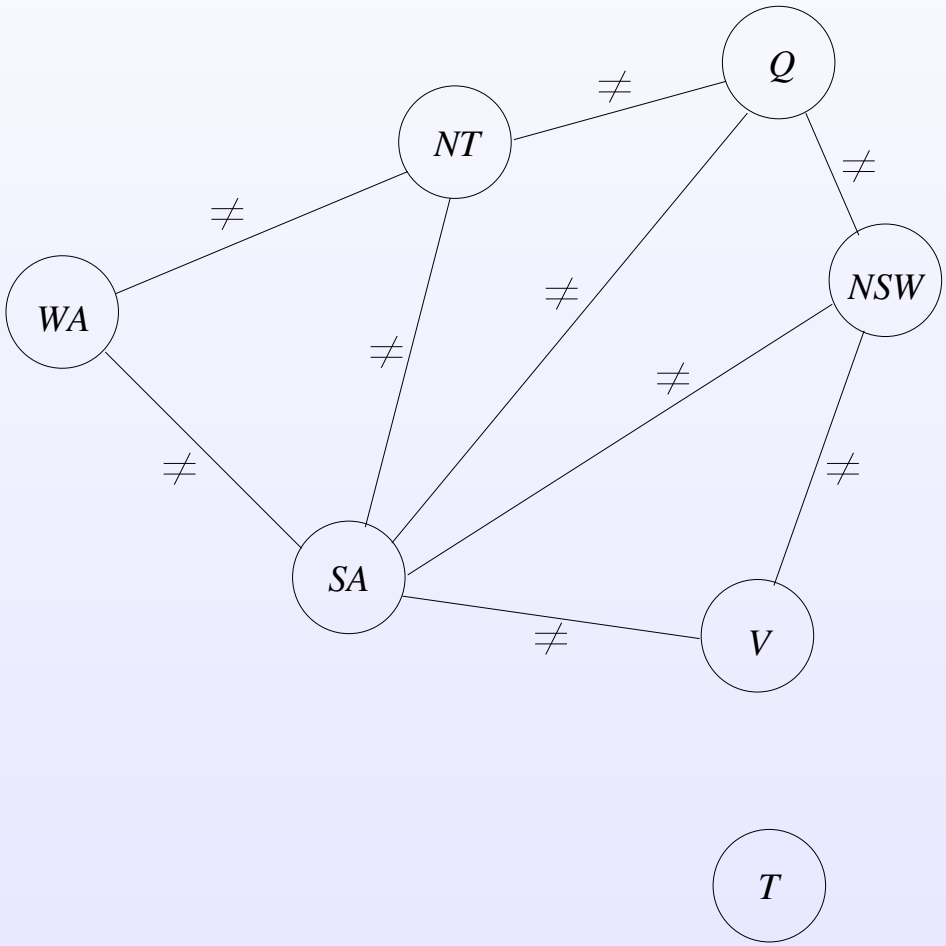
Las investigaciones han estado orientadas a CSPs binarios, es decir, CSPs donde las restricciones primitivas tienen máximo dos variables. El problema de coloración de mapas y el matrimonio a la antigua son ejemplos de CSPs binarios.

Una característica agradable de los CSPs binarios es que pueden ser representados como grafos no dirigidos.

Cada variable es representada por un nodo; una restricción unaria sobre una variable es representada por un lazo, etiquetada con el nombre de la restricción; y una restricción binaria sobre dos variables es representada por una arista entre los nodos correspondientes a las variables y etiquetada también con el nombre de la restricción.

Gerardo M.
Sarria M.

El grafo que representa el CSP del ejemplo 1 es:



¿Cómo Resolver un CSP?

Los CSP son problemas NP-completos, por lo que NO pueden existir *solvers* completos y eficientes para CSP arbitrarios.

Sin embargo, encontrar una solución por fuerza bruta (con la cual siempre es posible determinar la satisfacibilidad de un CSP) es la más ineficiente de todas las soluciones.

Backtracking Solver

La idea de esta técnica es determinar la satisfacibilidad de un CSP escogiendo una variable, luego una valor en su dominio, determinando la satisfacibilidad de la restricción que resulta de reemplazar dicha variable con dicho valor.

Ejemplo:
Considere el CSP con la restricción

$$X < Y \ \wedge \ Y < Z$$

y las variables X, Y, Z con el dominio $\{1, 2\}$.

Backtracking Solver

Primero se escoge una variable en la restricción: digamos X. El algoritmo itera entonces en el dominio de X: $\{1, 2\}$. Se toma el valor 1 y se reemplaza X dando la restricción

$$1 < Y \ \wedge \ Y < Z$$

Luego un llamado recursivo es hecho al algoritmo tomando otra variable: digamos Y (esto es posible ya que la restricción dada es parcialmente satisfacible). El algoritmo itera ahora sobre el dominio de Y: $\{1, 2\}$. Se toma el valor 1 y se reemplaza Y dando la restricción

$$1 < 1 \ \wedge \ 1 < Z$$

Backtracking Solver

La restricción resultante no es parcialmente satisfacible ya que $1 < 1$ es insatisfacible.

Por lo anterior, en la siguiente iteración se toma el valor 2 y se reemplaza Y obteniendo la restricción

$$1 < 2 \ \wedge \ 2 < Z$$

Como esta restricción *SI* es parcialmente satisfacible, se llama recursivamente el algoritmo con dicha restricción.

Backtracking Solver

En el segundo llamado recursivo la variable Z es escogida. Se toma el valor 1 de su dominio y se reemplaza Z dando

$$1 < 2 \ \wedge \ 2 < 1$$

que no es parcialmente satisfacible, por lo que en la siguiente iteración se toma el valor 2 y se tiene

$$1 < 2 \ \wedge \ 2 < 2$$

De nuevo, esta no es parcialmente satisfacible, luego el segundo llamado recursivo retorna *false*.

Backtracking Solver

Como los dos valores de Y fueron escogidos, el primer llamado recursivo también retorna *false*.

Entonces el llamado original al algoritmo toma el segundo valor del dominio de X : 2, reemplazándolo y dando como resultado

$$2 < Y \ \wedge \ Y < Z$$

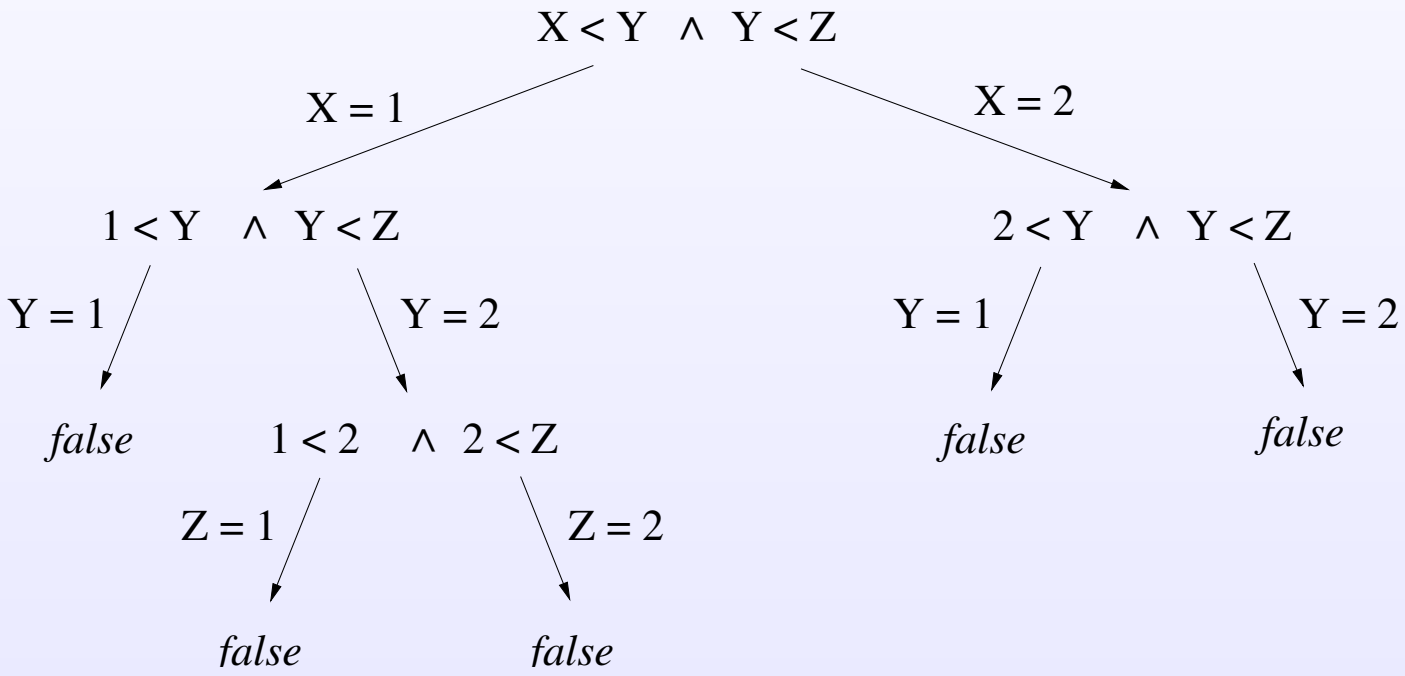
Luego se llama recursivamente el algoritmo, cuyo resultado será *false* ya que ninguno de los dos valores de Y producen una restricción parcialmente satisfacible.

De allí, el llamado original retorna *false*, indicando que la restricción original es insatisfacible.

Gerardo M. Sarria M.

Backtracking Solver

El árbol de búsqueda con los llamados recursivos es el siguiente:



Backtracking Solver

El ejemplo anterior muestra que el *backtracking solver* es muy ineficiente (en el peor de los casos será exponencial).

Cada variable escogida determina un árbol de búsqueda diferente.

Un buen heurístico es escoger la variable más restringida de primera, lo que daría un árbol más pequeño.

Solvers Basados en Consistencia

Otras técnicas para resolver CSPs son basadas en observación de dominios: si el dominio de alguna variable es vacío entonces la restricción es insatisfacible.

Estos solvers son incompletos pero tienen una complejidad polinomial.

Solvers Basados en Consistencia

La idea es transformar el CSP en otro equivalente (donde las restricciones tienen el mismo conjunto de soluciones) pero en el cual los dominios de las variables sean más pequeños.

Si alguno de los dominios llega a ser vacío entonces este nuevo CSP, y por lo tanto el original, serán insatisfacibles.

Estos *solvers* trabajan tomando cada restricción primitiva y usándola para eliminar valores del dominio de las variables involucradas que no la satisfagan.

Solvers Basados en Consistencia

Los *solvers* se dicen que son basados en consistencia cuando propagan la información de eliminación de los valores de los dominios de una variable a otra hasta que los dominios sean consistentes con la restricción.

Estos *solvers* pueden ser combinados con backtracking para bajar la complejidad y aumentar la completitud.

Dominios

Definición:
Un dominio es un dominio falso si alguna variable en el dominio tiene dominio vacío:

$$\exists x \mid D(x) = \emptyset$$

Un dominio es un dominio valuación si todas las variables tienen asociado un dominio de tamaño 1:

$$\forall x. |D(x) = 1|$$

La función $satisfiable(C, D)$ toma una restricción C y un dominio valuación D y retorna *true* o *false* indicando si C es satisfacible o no bajo esa valuación.

Dominios

Ejemplo:
Considere las variables X, Y, Z.

$D_1(X) = \{1, 2\}, D_1(Y) = \{1, 2\}, D_1(Z) = \emptyset.$
$D_2(X) = \{1\}, D_2(Y) = \{2\}, D_2(Z) = \{1\}.$

$C : X < Y \ \wedge \ Y < Z$

D_1 es un dominio falso.
D_2 es un dominio valuación.
$satisfiable(C, D)$ es *false*.

Nodo Consistencia

Definición:
Una restricción primitiva c es nodo consistente con el dominio D si

- $|vars(c)| \neq 1$, ó
- $vars(c) = \{x\}$ y para cada $d \in D(x)$, $\{x \mapsto d\}$ es una solución de c.

Un CSP con restricción $c_1 \wedge \ldots \wedge c_n$ y dominio D es nodo consistente si cada restricción primitiva c_i es nodo consistente con D para $1 \leq i \leq n$.

Arco Consistencia

Definición:
Una restricción primitiva c es arco consistente con el dominio D si

- $|vars(c)| \neq 2$, ó
- $vars(c) = \{x, y\}$,
  $\forall d_x \in D(x).\exists d_y \in D(y) : \{x \mapsto d_x, y \mapsto d_y\}$ es una solución de c, y
  $\forall d_y \in D(y).\exists d_x \in D(x) : \{x \mapsto d_x, y \mapsto d_y\}$ es una solución de c.

Un CSP con restricción $c_1 \wedge \ldots \wedge c_n$ y dominio D es arco consistente si cada restricción primitiva c_i es arco consistente con D para $1 \leq i \leq n$.

Ejemplos

Los CSP en los ejemplos coloración de mapas y N-reinas son nodo consistentes, debido a que no tienen restricciones que envuelvan solo a una variable.

Ejemplos

Los CSP en los ejemplos coloración de mapas y N-reinas son nodo consistentes, debido a que no tienen restricciones que envuelvan solo a una variable.

El CSP del ejemplo coloración de mapas es arco consistente debido a que para cada color en el dominio de la primera variable en la desigualdad hay un color diferente al color en el dominio de la segunda variable, y viceversa.

Ejemplos

Los CSP en los ejemplos coloración de mapas y N-reinas son nodo consistentes, debido a que no tienen restricciones que envuelvan solo a una variable.

El CSP del ejemplo coloración de mapas es arco consistente debido a que para cada color en el dominio de la primera variable en la desigualdad hay un color diferente al color en el dominio de la segunda variable, y viceversa.

Si modificamos el problema de coloración de mapas de tal manera que existan solo dos colores, entonces se vuelve insatisfacible. Sin embargo, se mantiene la arco consistencia, demostrándose que un CSP binario puede ser insatisfacible incluso cuando es nodo y arco consistente.

Algoritmo de Nodo Consistencia

Para trasformar un CSP en uno equivalente el cual es nodo consistente se sigue el siguiente algoritmo:

x es una variable;
C es una restricción;
D es un dominio;
$c_1, \ldots, c_n$ son restricciones primitivas;
d es un valor de dominio.

```
node_consistent(C, D)
    sea C de la forma c_1, ..., c_n
    for i := 1 to n do
        D := node_consistent_primitive(c_i, D)
    endfor
    return D

node_consistent_primitive(c, D)
    if |vars(c)| = 1 then
        sea {x} = vars(c)
        D(x) := {d ∈ D(x) | {x ↦ d} es una solución de c}
    endif
    return D
```

Algoritmo de Arco Consistencia

Para trasformar un CSP en uno equivalente el cual es arco consistente se sigue el siguiente algoritmo:

```
arc_consistent(C, D)
    sea C de la forma c_1, ..., c_n
    repeat
        W := D
        for i := 1 to n do
            D := arc_consistent_primitive(c_i, D)
        endfor
    until W ≡ D
    return D

arc_consistent_primitive(c, D)
    if |vars(c)| = 2 then
        sea {x, y} = vars(c)
        D(x) := {d_x ∈ D(x)| para algún d_y ∈ D(y),
                                {x ↦ d_x, y ↦ d_y} es una solución de c}
        D(y) := {d_y ∈ D(y)| para algún d_x ∈ D(x),
                                {x ↦ d_x, y ↦ d_y} es una solución de c}
    endif
    return D
```

Algoritmo *Solver* Incompleto

Se pueden emplear los algoritmos de arco y nodo consistencia para crear un solver incompleto que determina si un CSP es satisfacible:

```
C es una restricción;
D es un dominio.


arc_solver(C, D)
    D := node_arc_consistent(C, D)
    if D es un dominio falso then
        return false
    elseif D es un dominio valuación then
        return satisfiable(C, D)
    else
        return unknown
    endif


node_arc_consistent(C, D)
    D := node_consistent(C, D)
    D := arc_consistent(C, D)
    return D
```

Algoritmo *Solver* Completo

Gerardo M. Sarria M.

Sin embargo, los algoritmos de nodo y arco consistencia también pueden ser combinados con el algoritmo de backtracking para obtener un solver completo:

```
back_arc_solver(C, D)
    D := node_arc_consistent(C, D)
    if D es un dominio falso then
        return false
    elseif D es un dominio valuación then
        if satisfiable(C, D) then
            return D
        else
            return false
        endif
    endif
    escoja una variable x tal que |D(x)| ≥ 2
    for each d ∈ D(x)
        D_1 := back_arc_solv(C ∧ x = d, D)
        if D_1 ≠ false then
            return D_1
        endif
    endfor
    return false
```

Gerardo M.
Sarria M.

¿Cómo Modelar y Resolver CSPs Eficientemente?

El modelamiento es el corazón de la programación con restricciones ya que mediante este proceso el problema es especificado en términos de restricciones que pueden ser manejadas por el *solver*.

Para sacar partido del modelado con restricciones de dominio finito normalmente se restringen y se distribuyen las variables combinando el solver con backtracking.

La clave está en la estrategia de propagación y exploración que reduce el espacio de búsqueda.

Reglas de Propagación

Dado un rango para cada variable en una restricción primitiva, se pueden idear métodos eficientes para calcular un nuevo rango para cada variable en la restricción el cual es consistente con la restricción.

Ejemplo:
Considere la restricción $X = Y + Z$.

Escribiendo la restricción en tres formas:

$$Y = X + Z, \qquad Z = X + Y, \qquad X = Y + Z$$

Reglas de Propagación

Razonando acerca del mínimo y máximo valor de los lados derechos, se puede ver que

$$X \geq min_D(Y) + min_D(Z), \quad X \leq max_D(Y) + max_D(Z),$$
$$Y \geq min_D(X) - max_D(Z), \quad Y \leq max_D(X) - min_D(Z),$$
$$Z \geq min_D(X) - max_D(Y), \quad Z \leq max_D(X) - min_D(Y).$$

De las desigualdades anteriores se pueden derivar reglas simples para asegurar la consistencia que usa las restricciones actuales de cada variable de modo que se puedan calcular los valores de los lados derechos de las expresiones, y puedan usarse para actualizar los máximos y mínimos del dominio de cada variable.

Reglas de Propagación

Dado el dominio

$$D(X) = [4.,8], \qquad D(Y) = [0.,3], \qquad D(Z) = [2.,2]$$

se puede determinar que

$$2 \leq X \leq 5,$$
$$2 \leq Y \leq 6,$$
$$1 \leq Z \leq 8.$$

Por lo tanto, se pueden actualizar los dominios a

$$D(X) = [4.,5], \qquad D(Y) = [2.,3], \qquad D(Z) = [2.,2]$$

sin eliminar alguna solución de la restricción.

Reglas de Propagación

Ejemplo:
Considere la restricción

$$4W + 3P + 2C \leq 9$$

Se puede reescribir en tres formas:

$$W \leq \frac{9}{4} - \frac{3}{4}P - \frac{2}{4}C, \qquad P \leq \frac{9}{3} - \frac{4}{3}W - \frac{2}{3}C,$$

$$C \leq \frac{9}{2} - 2W - \frac{3}{2}P$$

Reglas de Propagación

Así se obtienen las desigualdades

$$W \leq \frac{9}{4} - \frac{3}{4} min_D(P) - \frac{2}{4} min_D(C),$$
$$P \leq \frac{9}{3} - \frac{4}{3} min_D(W) - \frac{2}{3} min_D(C),$$
$$C \leq \frac{9}{2} - 2 min_D(W) - \frac{3}{2} min_D(P)$$

Reglas de Propagación

Dado el dominio inicial

$$D(W) = [0.,9], \qquad D(P) = [0.,9], \qquad D(C) = [0.,9]$$

se puede determinar que $W \leq \frac{9}{4}$, $P \leq \frac{9}{4}$, y $C \leq \frac{9}{2}$. Usando las reglas de propagación, se actualiza el dominio a

$$D(W) = [0.,2], \qquad D(P) = [0.,3], \qquad D(C) = [0.,4]$$

Gerardo M. Sarria M.

Dominios y Exploración

Considere el problema de resolver la ecuación cripto-aritmética

$$\begin{array}{cccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

donde cada letra representa un dígito diferente.

Dominios y Exploración

Este problema es modelado con el siguiente programa

```
proc {Money Root}
   S E N D M O R Y
in
   Root = sol(s:S e:E n:N d:D m:M o:O r:R y:Y)
   Root ::: 0#9
   {FD.distinct Root}
   S \=: 0
   M \=: 0
                  1000*S + 100*E + 10*N + D
   +              1000*M + 100*O + 10*R + E
   =: 10000*M + 1000*O + 100*N + 10*E + Y
   {FD.distribute ff Root}
end
```

Dominios y Exploración

El programa se ejecuta de la siguiente manera. Después de la declaración el dominio es

$$D(S) = [0.,9], D(E) = [0.,9], D(N) = [0.,9], D(D) = [0.,9],$$
$$D(M) = [0.,9], D(O) = [0.,9], D(R) = [0.,9], D(Y) = [0.,9]$$

Agregando las restricciones $S = 0$ y $M = 0$ se reduce el dominio quedando

$$D(S) = [1.,9], \; D(M) = [1.,9]$$

Dominios y Exploración

La restricción `FD.distinct` agrega las desigualdades $S \neq E$, $S \neq N$, $S \neq D$, $\ldots$, $R \neq Y$ al *store* de restricciones.

Simplificando la ecuación original, el *solver* resolverá la ecuación

$$1000S + 91E + D + 10R = 9000M + 900O + 90N + Y$$

Dominios y Exploración

Una de las reglas de propagación para la restricción anterior es basada en la desigualdad

$$\begin{aligned} M \leq & \frac{1}{9} max_D(S) + \frac{91}{9000} max_D(E) + \ldots \\ & + \frac{1}{9000} max_D(D) + \frac{1}{900} max_D(R) - \ldots \\ & - \frac{1}{10} min_D(O) - \frac{1}{100} min_D(N) - \frac{1}{9000} min_D(Y) \end{aligned}$$

Dominios y Exploración

Dados los dominios actuales, esto implica que

$$M \leq \frac{9}{9} + \frac{91 \times 9}{9000} + \frac{9}{9000} + \frac{9}{900} - \frac{0}{10} - \frac{0}{100} - \frac{0}{9000} = 1{,}102$$

Luego, $D(M)$ es actualizado a $[1.,1]$.

De la propagación (usando la restricción $S \neq M$) se obtiene $D(S) = [2.,9]$.

Dominios y Exploración

La siguiente regla de propagación se basa en la desigualdad

$$\begin{aligned} S \geq\ & 9min_D(M) + \frac{9}{10}min_D(O) + \ldots \\ & + \frac{9}{100}min_D(N) + \frac{1}{1000}min_D(Y) - \ldots \\ & - \frac{91}{1000}max_D(E) - \frac{1}{1000}max_D(D) - \frac{1}{100}max_D(R) \end{aligned}$$

y es usada con el dominio actual para inferir que $S \geq 8{,}082$.

Dominios y Exploración

Luego la propagación fija $D(S)$ a [9.,9]. También se obtiene (usando la restricción $S \neq E$) que $D(E)$ es [0.,8]. Y de igual manera los dominios de N, D, O, R, Y son [0.,8].

La propagación continúa hasta obtener los siguientes dominios:

$$D(S) = [9.,9], D(E) = [4.,7], D(N) = [5.,8], D(D) = [2.,8],$$
$$D(M) = [1.,1], D(O) = [0.,0], D(R) = [2.,8], D(Y) = [2.,8]$$

Dominios y Exploración

Dado que solo tres de las variables tienen valores fijos (aunque las otras variables tienen un dominio disminuído) el solver no puede determinar si el *store* de restricciones es satisfacible o no.

Para garantizar que se encuentre una solución válida se usa la exploración

Básicamente, la función de exploración es buscar en el dominio inicial de cada variable fijando la variable a cada uno de los valores de 0 a 9 (backtracking).

Las variables son probadas en el orden en que fueron pasadas a la exploración.

Dominios y Exploración

La primera variable a ser asignada es S.

Se prueba con el primer valor del dominio inicial: 0. Este causará una falla ya que la restricción $S = 0$ es inconsistente con el dominio actual de S.

El siguiente valor es probado: 1. Este también causa una falla.

De la misma manera, los otros valores llevarán a falla hasta llegar a $S = 9$, el cual es consistente.

Dominios y Exploración

Ahora la exploración trata de encontrar un valor para E.

Primero se agrega $E = 0$, el cual falla. Se continúa probando valores hasta $E = 4$.

Por propagación se reduce $D(N)$ a [5.,5] y $D(R)$ a [8.,8], y se encuentra una falla ya que la variable Dno tiene un valor posible.

Entonces se elimina la restricción $E = 4$ y se toma el siguiente valor, agregando la restricción $E = 5$.

El proceso de propagación determina entonces que $N = 6$, $R = 8$, $D = 7$ y $Y = 2$.

Distribución

Para muchos problemas, la mayor parte del tiempo de ejecución ocurre durante la ejecución de la exploración. Además, para muchos programas, la eficiencia es sinónimo de eficiencia en la exploración.

La estrategia de distribución para reducir el espacio de búsqueda y llegar a una solución rápido radica en:

- el orden en que las variables son exploradas, y
- el orden en que los valores de una variable son explorados.

Distribución

Orden de las Variables

Alterar el orden en las cuales las variables son exploradas puede tener un efecto dramático en el tamaño del árbol de búsqueda.

Un heurístico es pasar a la exploración primero aquellas variables que tengan el dominio más pequeño (variables más restrigidas).

Distribución

Ejemplo:
En el programa, cuando se va a llamar a la exploración, el rango de los dominios es

$$D(S) = [9.,9], D(E) = [4.,7], D(N) = [5.,8], D(D) = [2.,8],$$
$$D(M) = [1.,1], D(O) = [0.,0], D(R) = [2.,8], D(Y) = [2.,8].$$

Como las variables S, M y O tiene dominios de tamaño 1, éstas variables deben ser exploradas de primero.

Principio de la primera falla:

"Para tener éxito, pruebe primero donde es más posible fallar"

Distribución

Orden de los Valores de los Dominios

Las diferentes opciones de los valores de los dominios cambian el orden en el cual son encontradas las soluciones y el orden en el cual son exploradas las ramas del árbol de búsqueda.

Un heurístico es escoger el valor de la mitad del dominio de cada variable.

Combinando el principio de la primera falla con el heurístico anterior, reduce el espacio de búsqueda mucho más.

Distribución

Dividiendo los Dominios

Las estrategias de distribución, sin embargo, solo necesitan asegurar que cada variable este restringida a un dominio de un solo valor (de manera que un *solver* incompleto pueda retornar *true* o *false*).

Otra estrategia es reducir el dominio de cada variable, partiendo el dominio actual en dos: por ejemplo los menores y los mayores.

Fin de la Presentación