
Inteligencia Artificial

Problemas de satisfacción de restricciones.

In whitch we see how treating states as more than just little black boxes leads to the invention of a range of powerful new search methods and a deeper understanding of problem structure and complexity.

Un problema es considerado de **satisfacción de restricciones** cuando se lo representa con un conjunto de variables a las cuales se le tienen que asignar valores de manera de no violar las restricciones.

Definiendo problemas de satisfacción de restricciones.

Poseen 3 componentes, *X*, *D* y *C*:

- *X*: es un conjunto de variables, {x1, x2, ..., xn}
- *D*: es un conjunto de dominios, {d1, d2, ..., dn}, uno por variable.

Cada dominio posee un conjunto de valores permitidos para la variable relacionada.

- *C*: es un conjunto de restricciones que especifica las combinaciones permitidas de valores.

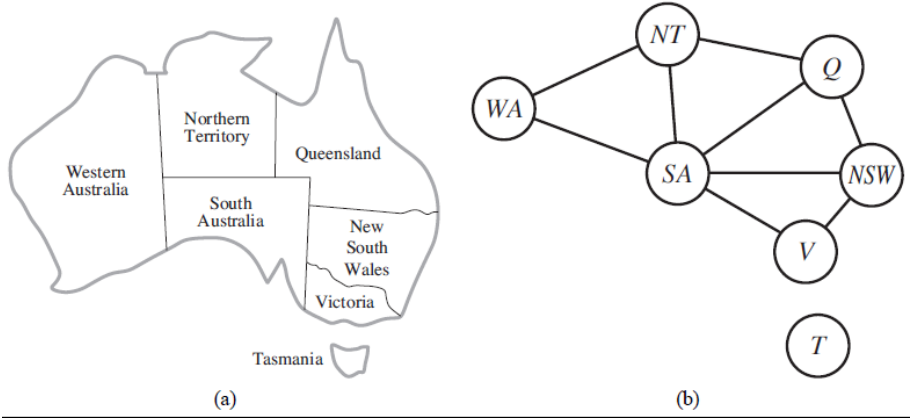
Una restricción consiste en una tupla <*scope, rel*>, donde *scope* es una lista de las variables afectadas y *rel* es la condición que se debe cumplir.

Definiendo problemas de satisfacción de restricciones.

Cada estado en un CSP está definido por una asignación de valores a algunas o todas las variables.

- **Asignación consistente:** no viola ninguna de las restricciones
- **Asignación completa:** todas las variables tienen algún valor asignado
- Una **solución** es una asignación **completa** y **consistente**

Ejemplo: Coloreo de mapas



Hay que encontrar la manera de pintar todos los paises usando 3 colores (**red**, **green**, **blue**) de manera que ningún par de paises adyacentes tengan el mismo color.

Ventajas de CSP sobre búsqueda tradicional

- Posee una representación natural para una gran variedad de problemas
- Son más rápidos ya que pueden eliminar grandes partes del árbol de búsqueda.

Ej: si hacemos la asignación **SA = blue**, CSP puede eliminar de los 5 vecinos el color **blue**.

Esto nos deja solo 2 colores para probar en los 5 vecinos => 2^5 = 32 asignaciones por probar.

Sin la propagación de la restricción nos quedaría 3^5 = 243 asignaciones por probar.

- En las búsquedas tradicionales solo podemos saber si un nodo es solución o no; en CSP podemos saber si una asignación parcial **no** conduce a una solución.
- En CSP podemos ver **porqué** una asignación no es solución.
- En CSP hay heurísticas genéricas para este tipo de problemas.

Variantes del formalismo de CSP

- Dominios discretos y finitos; discretos e infinitos; continuos e infinitos.

Según los tipos de restricciones:

- **Restricciones unarias:** restringen el valor de una variable. **Ej:** *SA* <> *blue*
- **Restricciones binarias:** la restricción relaciona dos variables. Si un CSP solo posee este tipo de restricciones es un **CSP binario**. **Ej:** *SA* <> *NSW* o *x* < *y*
- **Restricciones globales:** involucran un número **arbitrario** de variables. Uno de los ejemplos más conocidos es *AllDiff*.
  - pueden ser representadas mediante el uso de hiper-grafos.
  - cualquier CSP puede ser convertido a un CSP binario, aunque una restricción global puede ser más sencilla de leer y es posible diseñar inferencias de propósito especial que funcionen con las restricciones globales

Ejemplo cripto aritmética

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

Todas las letras deben relacionarse a un dígito distinto y la suma debe ser correcta.

¿Cuales serían las restricciones?

Ejemplo cripto aritmética

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

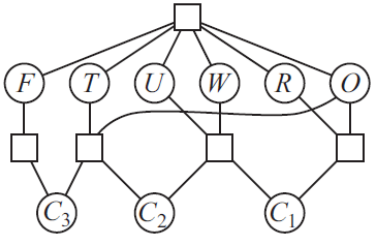
Todas las letras deben relacionarse a un dígito distinto y la suma debe ser correcta.

Restricciones:

- *AllDiff* (F, T, U, W, R, O)
- O + O = R + 10 * C1
- C1 + W + W = U + 10 * C2
- C2 + T + T = O + C3 * 10
- C3 = F

Representación mediante hipergrafo

- *AllDiff* (F, T, U, W, R, O)
- O + O = R + 10 * C1
- C1 + W + W = U + 10 * C2
- C2 + T + T = O + C3 * 10
- C3 = F



Los nodos (círculos) representan las variables y los hipernodos (cuadrados) representan las relaciones *n-arias*.

Propagación de restricciones.

- La **propagación de restricciones** es un tipo de **inferencia** en la que se reduce el número de valores legales que puede tomar una variable.
- La propagación de restricciones puede realizarse como una tarea de preprocesamiento o intercalada con la búsqueda.

Distintos tipos de **consistencia local**:

- Consistencia de nodo.
- Consistencia de arco.
- Consistencia de camino.
- K-Consistencia.

Consistencia de nodo

Una variable es nodo-consistente si **todos** los valores de su dominio satisfacen las restricciones unarias.

Una red es nodo-consistente si todas sus variables son nodo-consistentes.

Ej: *SA* posee el dominio *{ red, blue, green }* pero hay una restricción de que a *SA* no se le puede asignar *green*. La inferencia hace que *green* sea eliminado del dominio de *SA*.

Consistencia de arco

Xi es arco-consistente respecto a *Xj* si, para cada valor de *Di* existe algún valor en *Dj* que satisfaga la restricción binaria en el arco (*Xi*, *Xj*).

Una red es arco-consistente si cada variable es arco-consistente con las demás variables.

El algoritmo más popular para asegurar consistencia de arco se denomina **AC-3**. La complejidad es *O(cd^3)*

Ej: se tienen dos variables, *X* e *Y*, donde los dominios de cada una son los dígitos.

Se posee además esta restricción binaria: *Y=X^2*

Para hacer que X sea arco-consistente con Y hay que reducir el dominio de X a {0,1,2,3}; notese que 4^2 = 16, valor que no pertenece al dominio de Y.

Para hacer Y arco-consistente con X hay que reducir el dominio de Y a {0,1,4,9}

Consistencia de camino

Un conjunto de 2 variables *{Xi, Xj}* es camino-consistente con respecto a una tercer variable *Xm* si, para cada asignación {Xi = a, Xj = b} consistente con las restricciones de {Xi, Xj}, existe una asignación a Xm que satisface las restricciones en {Xi, Xm} y {Xm, Xj}

K-Consistencia

Un CSP es k-consistente si, para cualquier conjunto *k - 1* de variables y para cualquier asignación consistente de esas variables, un valor consistente siempre puede ser asignado a cualquier *k-esima* variable.

A mayor valor de k, menor el tiempo de búsqueda de la solución; pero, la cantidad de tiempo y memoria utilizado para propagar la consistencia crece de manera exponencial a k

Frecuentemente se utiliza consistencia de arco; menos frecuentemente, consistencia de camino.

Restricciones globales

- Son comunes en problemas reales y pueden ser manejadas por algoritmos especiales
- **Alldiff**: comprueba que todos los valores sean distintos. Si len(set(variables)) > len(set(dominios)) => Inconsistencia
- **Atmost**: comprueba que como mucho se usen X recursos.

Ej: Atmost(10, P1, P2, P3, P4) => como mucho se tienen que asignar 10 recursos entre las 4 variables. Se puede comprobar la consistencia sumando los valores mínimos de cada dominio. Se puede propagar eliminando los valores máximos de los dominios que sean inconsistentes con los mínimos del resto de dominios.

Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Backtraking search for CSPs

- Trabaja con asignaciones incompletas
- **Conmutatividad** -> hay que asignar solo una variable por nodo
- No se especifica estado inicial, action, result o goal test (son siempre los mismos)
- La performance se puede mejorar mediante heurísticas genéricas

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment then
          add {var = value} to assignment
          inferences ← INFERENCE(csp, var, value)
          if inferences ≠ failure then
            add inferences to assignment
            result ← BACKTRACK(assignment, csp)
            if result ≠ failure then
              return result
      remove {var = value} and inferences from assignment
  return failure
```

Selección de variable

- **la siguiente...**
- **Minimun remaining values (MRV):** elige la variable más restringida primero
- **Grado heurístico:** variable con mayor apariciones en restricciones primero. Se usa generalmente para desempatar luego de aplicar MRV

Orden de valores del dominio

- **Least constraining value (LCV)**: elige primero el valor que menos opciones quita al resto de las variables

Intercalando búsqueda e inferencia

- **Forward checking:** cuando una variable *X* es asignada chequea toda variable conectada eliminando de sus dominios valores inconsistentes con la asignación de *X*.

Se complementa bien con MRV ya que elimina valores y permite dicernir las variables más restringidas.

- **MAC (Maintaining Arc Consistency)**: llama a AC3 luego de cada asignación para aplicar consistencia de arco recursivamente

Local search for CSPs

- Usa asignaciones completas.
- En cada paso se elige una variable y se cambia un valor
- Para elegir el valor se usa la heurística **minimos conflictos**
- Se puede utilizar para mejorar soluciones cuando se producen cambios en el problema. Ej. problemas de planeación

function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS($var, v, current, csp$)
    set $var = value$ in $current$
  return $failure$

Bibliografía y enlaces útiles.

- Russell S., Norvig P.: Artificial Intelligence: A modern Approach. Third Edition