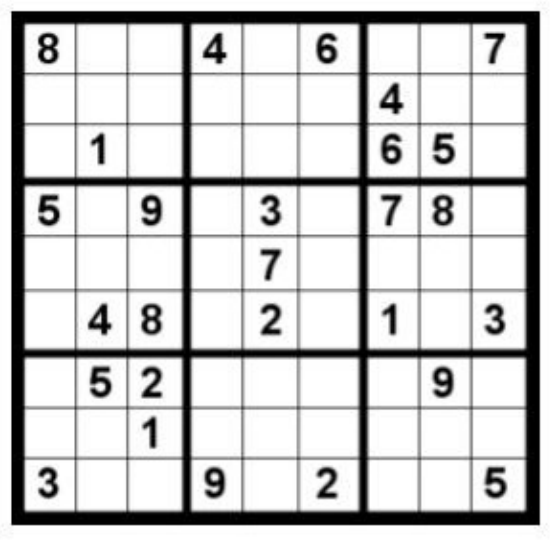
Técnicas de Inteligencia Artificial

Tema 5

Problemas de satisfacción de restricciones

Introducción

- Existen problemas que pueden resolverse de forma más eficiente que con los métodos de búsqueda vistos en temas anteriores.
- En este caso, utilizamos una representación factorizada para cada estado: un conjunto de variables, cada una de las cuales tiene un valor.
- El problema se considera resuelto cuando cada variable tenga un valor que satisfaga todas las restricciones de la variable, así como las restricciones globales del problema.
- Un problema planteado de esta forma se conoce como problema de satisfacción de restricciones o CSP.
- Los algoritmos de búsqueda CSP permiten eliminar una gran parte del espacio de estados identificando las combinaciones de variable-valor que violan las restricciones.



Metodología

- La resolución de un CSP consta de dos fases:

  1) Modelar el problema como un problema de satisfacción de restricciones.
     - El problema se expresa mediante un conjunto de variables, dominios en los que toman los posibles valores y restricciones sobre estas variables.
  2) Procesar las restricciones.
     - **Técnicas de consistencia.**
       - Eliminación de valores inconsistentes de los dominios de las variables que no verifican las restricciones impuestas.
       - En ocasiones se pueden aplicar técnicas derivadas de la lógica proposicional o de primer orden.
     - **Algoritmos de búsqueda.**
       - Exploración sistemática del espacio de soluciones hasta encontrar una solución que verifique todas las restricciones del problema, o probar que no existe tal solución.
     - Lo normal es combinar ambas aproximaciones, ya que las técnicas de consistencia permiten reducir el espacio de soluciones para poder explorarlo usando algoritmos de búsqueda.

Formalización del CSP

- Un **CSP** es una terna (X, D, C) donde:
  - X es un conjunto n de variables $\{x_1,\ldots,x_n\}$
  - $D = <D_1,...D_n>$ es un vector de dominios, donde D_i es el dominio que contiene todos los posibles valores que puede tener la variable x_i.
  - C es un conjunto finito de restricciones. Cada restricción está definida sobre un conjunto k de variables por medio de un predicado que restringe los valores que las variables pueden tomar simultáneamente.
- Una **asignación de variables** (x,a) es un par variable-valor que representa la asignación del valor *a* a la variable *x.*
- Una **asignación de un conjunto de variables** es una tupla de pares ordenados $((x_1,a_1), \ldots,(x_i,a_i))$, donde cada par ordenado (x_i, a_i) asigna el valor a_i a la variable x_i. Una tupla se dice localmente consistente si satisface todas las restricciones formadas por variables de la tupla. Una tupla es localmente consistente si satisface todas las restricciones formadas por las variables de la tupla.

Formalización del CSP

- Una **solución** es una asignación de valores a todas las variables de forma que satisfagan todas las restricciones.
  - Tupla consistente que contiene todas las variables del problema.
- Una **solución parcial** es una tupla consistente que contiene algunas de las variables del problema.
- Un **problema es consistente** si existe, al menos, una solución.
- Los objetivos del CSP pueden ser alguna de las siguientes opciones:
  - Encontrar una solución, sin preferencia alguna.
  - Encontrar todas las soluciones.
  - Encontrar una solución óptima a partir de una función objetivo definida en términos de las variables.

Formalización del CSP

- Las variables se suelen denotar con las últimas letras del alfabeto (x,y,z), así como esas mismas letras acompañadas de un subíndice.
- La **aridad** de una restricción es el número de variables que intervienen en ella.
  - Una restricción unaria es una restricción de una sola variable. Ej: $x<5$.
  - Una restricción binaria es una restricción de dos variables. Ej: $x_1-x_2 = 3$
  - Una restricción n-aria es una restricción de n variables.
- Una **restricción k-aria** entre las variables $\{x_1,...x_k\}$ se suele denotar como $C_{1...k}$.
  - Una restricción binaria entre las variables x_i y x_j se denota como C_{ij}.
  - Cuando los índices de las variables en una restricción no son relevantes, se denota simplemente por C.

Formalización del CSP

- Una tupla *p* de una restricción $C_{i...k}$ es un elemento del producto cartesiano Di x … x Dk.
- Una tupla *p* que satisface la restricción $C_{i...k}$ se llama tupla permitida o válida.
- Una tupla *p* de una restricción $C_{i...k}$ se dice que es soporte para un valor $a \in D_j$ si la variable $x_j \in X_{Ci...k}$, *p* es una tupla permitida y contiene a (x_j,a).
- Verificar si una tupla es permitida o no por una restricción se llama **comprobación de la consistencia**.
- Una restricción puede definirse extensionalmente mediante un conjunto de tuplas válidas o no válidas o intencionalmente mediante un predicado entre las variables.

Ejemplo de representación

- Consideremos una restricción entre 4 variables x1, x2, x3, x4, todas ellas con dominios en el conjunto {1,2}.
- La suma entre las variables x1 y x2 es menor o igual que la suma entre x3 y x4.
  - Esta restricción se puede representar intencionalmente mediante la expresión:

    $x1 + x2 \leq x3 + x4$
  - También puede representarse extensionalmente mediante el conjunto de tuplas permitidas:

    {(1,1,1,1),(1,1,1,2),(1,1,2,1),(1,1,2,2),(2,1,2,2),(1,2,2,2),(1,2,1,2),(1,2,2,1),(2,1,1,2),(2,1,2,1),(2,2,2,2)}
  - O mediante el conjunto de tuplas no permitidas:

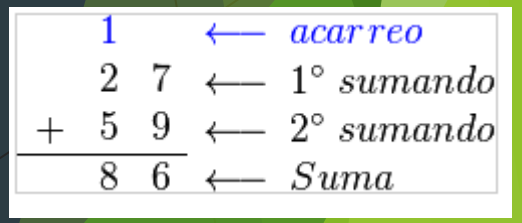
    {(1,2,1,1),(2,1,1,1),(2,2,1,1),(2,2,1,2),(2,2,2,1)}.

Modelización del CSP

- La etapa fundamental para resolver problemas mediante CSP es el modelado del problema en términos de variables, dominios y restricciones.
- Para entender este proceso se va a ver con el ejemplo del conocido problema criptoaritmético **send+more=money**.
- Este problema consiste en asignar a cada letra {s,e,n,d,m,o,r,y} un dígito diferente del conjunto {0,…,9} de forma que satisfaga la expresión **send+more=money.**
- La primera aproximación para modelar el problema sería asignando una variable a cada una de las letras, cada una en el dominio {0,…,9} con las restricciones de:
  - Todas las variables toman valores distintos.
  - Se satisface *send + more = money*.

Modelización del CSP

- De esta forma, las restricciones podrían escribirse como:
  - $10^3(s+m) + 10^2(e+o) + 10(n+r) + d + e = 10^4m + 10^3o + 10^2n + 10e + y$
  - $s \neq e, s \neq n, ..., r \neq y$
- No obstante, esta representación no es útil ya que la primera restricción exige manipular todas las variables simultáneamente, lo que no permite recorrer el espacio de estados de manera óptima.
  - Al no tener restricciones locales entre las variables, no se puede podar el espacio de búsqueda para agilizar la búsqueda.
- Se puede descomponer la restricción global en otras más pequeñas haciendo uso de las relaciones entre los dígitos que ocupan la misma posición en la suma.
  - Se introducen los dígitos de acarreo para descomponer la ecuación en un conjunto de pequeñas restricciones.



Modelización del CSP

- Según el planteamiento del problema:
  - M debe valer 1 (el acarreo de 2 cifras que como mucho pueden sumar 18, más un posible acarreo de 1, que limita el resultado a 19).
  - Por tanto, s solamente puede tomar los valores {1,…,9} ya que si s=0 no habría acarreo.
  - Además de las variables del modelo anterior, se introducen nuevas variables c1, c2 y c3 para representar el acarreo.
  - Aunque el espacio de búsqueda se amplía con nuevas variables, permiten simplificar las restricciones y restringir el espacio de búsqueda.
  - El dominio de s es {1,...9}, el dominio de m es {1} y el dominio de los dígitos de acarreo es {0,1}. El dominio del resto de variables es {0,…,9}.
  - Las restricciones más pequeñas pueden comprobarse de forma más sencilla y local, permitiendo podar inconsistencias y reduciendo el tamaño del espacio de búsqueda efectivo.

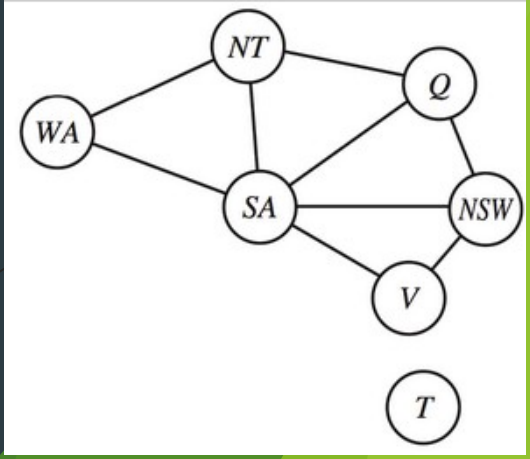
$$\begin{array}{r} \text{S E N D} \\ +\,\text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

$$\begin{array}{rcl} & 1 & \longleftarrow \textit{acarreo} \\ & 2\ 7 & \longleftarrow 1^{\circ}\ \textit{sumando} \\ +\ & 5\ 9 & \longleftarrow 2^{\circ}\ \textit{sumando} \\ \hline & 8\ 6 & \longleftarrow \textit{Suma} \end{array}$$

$$\begin{aligned} e + d &= y + 10c_1 \\ c_1 + n + r &= e + 10c_2 \\ c_2 + e + o &= n + 10c_3 \\ c_3 + s + m &= 10m + o \end{aligned}$$

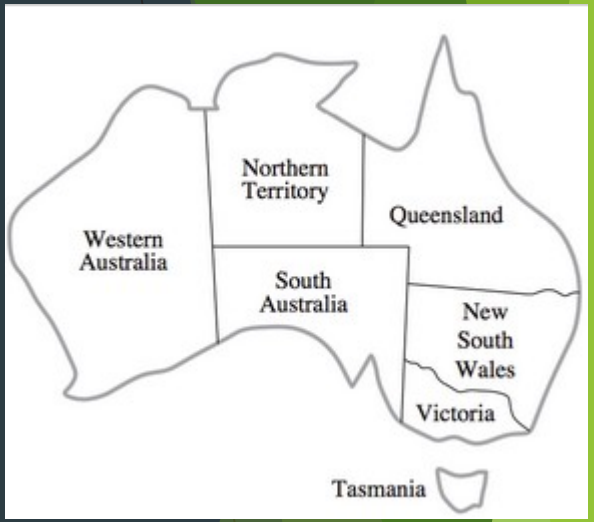
Representación del problema de coloreado de mapas

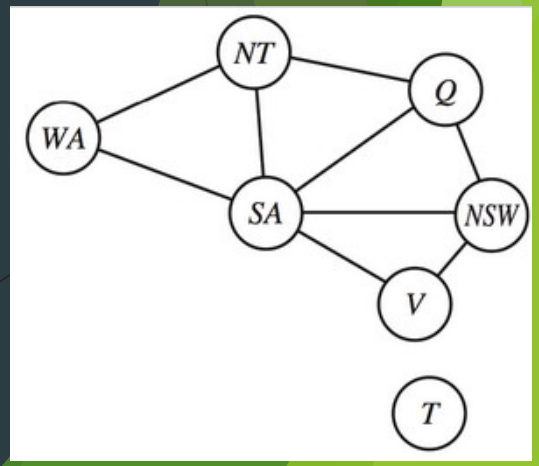
- Problema clásico formulable como un CSP.
- Tenemos un conjunto de colores y un mapa dividido en regiones.
- El objetivo es colorear cada región del mapa de forma que regiones adyacentes tengan distintos colores.
- Modelización del problema:
  - Se asocia una variable por cada región del mapa
  - El dominio de cada variable es el conjunto de posibles colores disponibles
  - Cada par de regiones contiguas tiene una restricción sobre los valores de las variables impidiendo que tengan el mismo color.
- Este mapa puede ser representado como un grafo donde los nodos son las regiones y las aristas son las uniones entre regiones adyacentes.
- La representación en forma de grafo es natural en muchos problemas CSP.



Representación del problema de coloreado de mapas

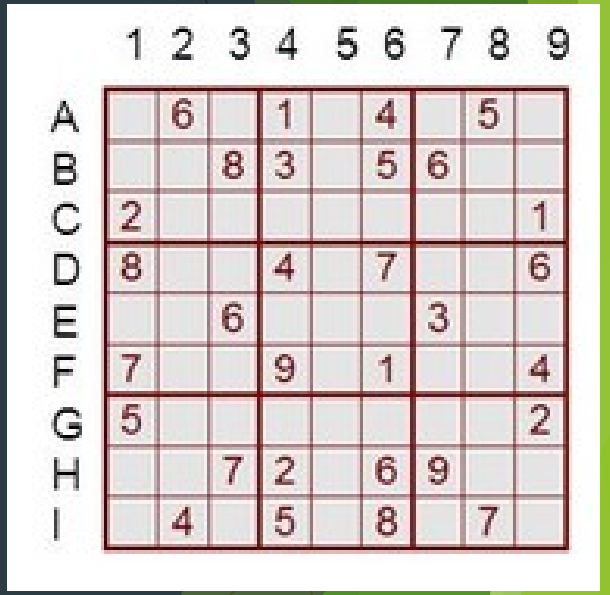
- Variables.
  - X = {WA,NT,Q,NSW, V,SA, T}
- Dominios.
  - El dominio de cada variables es el conjunto D_i = {rojo, verde, azul}
- Restricciones.
  - Cada región vecina debe tener colores distintos.
    - C = {SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, WA = NT, NT ≠ Q, Q ≠ NSW, NSW ≠ V}.





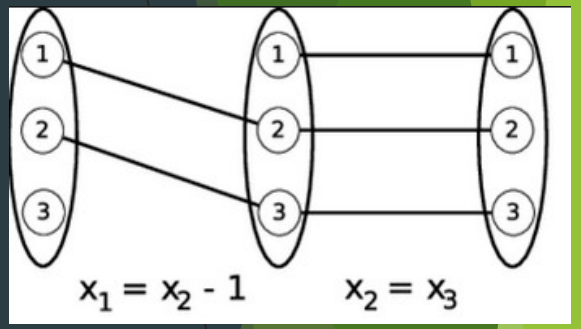
Representación del Sudoku

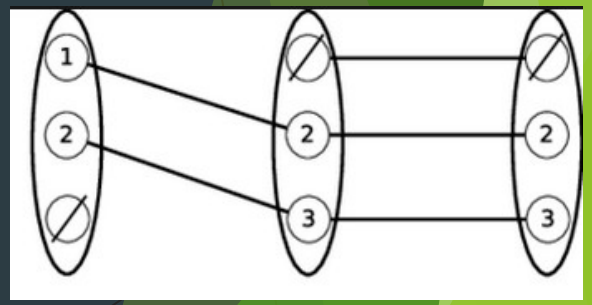
- Variables: 81 variables.
  - A1, A2, A3, …, I7, I8, I9.
  - Las letras son los índices de las filas, de arriba a abajo.
  - Los dígitos son los índices de las columnas, de izquierda a derecha.
- Dominios: Los nueve dígitos positivos.
  - A1, …, I9 ∈ {1,2,3,4,5,6,7,8,9}
  - El dominio de todas las variables es el mismo.
- Restricciones: 27 restricciones de diferencia.
  - AllDiff(A1,A2,A3,A4,A5,A6,A7,A8,A9)...
  - Todas las elementos de las filas, columnas y bloques contienen dígitos diferentes.



Consistencia de un CSP

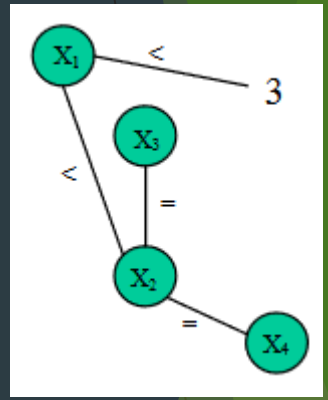
- El backtracking es una forma común de crear algoritmos de búsqueda sistemática para CSP.
  - Esta búsqueda sufre una explosión combinatoria en el espacio de búsqueda, lo que lo hace poco eficiente por sí solo.
- Uno de los principales problemas es la aparición de inconsistencias locales que van apareciendo continuamente.
  - Las inconsistencias locales son valores individuales, o combinación de valores de las variables, que no pueden ser parte de una solución porque no satisfacen alguna propiedad de consistencia.
- Se han propuesto varias técnicas de consistencia local para mejorar la eficiencia de los algoritmos de búsqueda.
  - Borrar valores inconsistentes de las variables o inducen restricciones implícitas que podan el espacio de búsqueda.





Consistencia de un CSP

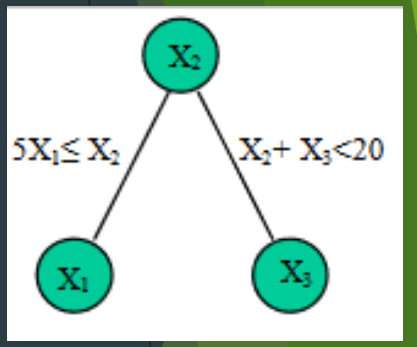
- **Consistencia de nodo (1-consistencia)**
  - Consistencia local más simple.
  - Esta consistencia asegura que todos los valores en el dominio de una variable satisfacen todas las restricciones unarias sobre esa variable.
  - Un problema es nodo-consistente si y sólo si todas sus variables son nodo-consistentes:
  - $\forall x_i \in X, \forall C_i, \exists a \in D_i$: a satisface C_i
  - Si el algoritmo elimina todos los valores del dominio de una variable, entonces el CSP es inconsistente.



Variable	Domain Size	Domain Size after Node-Consistency
X_1	{1,2,3,4,5,6,7,8,9,10}	{1,2}
X_2	{1,2,3,4,5,6,7,8,9,10}	{1,2,3,4,5,6,7,8,9,10}
X_3	{1,2,3,4,5,6,7,8,9,10}	{1,2,3,4,5,6,7,8,9,10}
X_4	{1,2,3,4,5,6,7,8,9,10}	{1,2,3,4,5,6,7,8,9,10}

Consistencia de un CSP

- **Consistencia de arco (2-consistencia)**
  - Un problema binario es arco-consistente si para cualquier par de variables x_i y x_j, para cada valor a en D_i, hay al menos un valor b en D_j tal que las asignaciones (x_i, a) y (x_j,b) satisfacen la restricción entre x_i y x_j.
  - Cualquier valor en el dominio D_i de la variable x_i que no es arco-consistente puede ser eliminado de D_i, ya que no puede formar parte de ninguna solución.
  - El dominio de una variable es arco-consistente si todos sus valores son arco-consistentes.
  - Un problema es arco-consistente si y sólo si todos sus arcos son arco-consistentes:
  - $\forall C_{ij} \in C, \forall a \in D_i, \exists b \in D_j$: b es un soporte para a en C_{ij}



Variable	Domain Size	Domain Size after AC
X_1	{1,2,3,4,5,6,7,8,9,10}	{1,2}
X_2	{1,2,3,4,5,6,7,8,9,10}	{5,6,7,8,9,10}
X_3	{1,2,3,4,5,6,7,8,9,10}	{1,2,3,4,5,6,7,8,9,10}

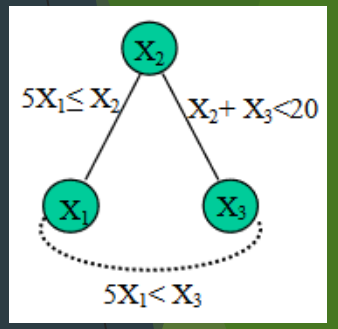
Consistencia de un CSP

```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

Consistencia de un CSP

- **Consistencia de caminos (3-consistencia)**
  - Nivel más alto de consistencia local que la arco-consistencia.
  - Para cada par de valores a y b de dos variables xi y xj, se requiere que la asignación de ((xi,a),(xj,b)) satisfaga la restricción entre xi y xj, y que además exista un valor para cada variable a lo largo del camino entre xi y xj, de forma que todas las restricciones a lo largo del camino se satisfagan.
  - Un problema satisface la consistencia de caminos si y sólo si todo par de variables (xi,xj) verifica la consistencia de caminos.
  - Cuando un problema satisface la consistencia de caminos y además es nodo-consistente y arco-consistente se dice que satisface fuertemente la consistencia de caminos.



Variable	Domain Values	Domain Size After AC	Domain Size after PC
X_1	{1,2,3,4,5,6,7,8,9,10}	{1,2}	{1}
X_2	{1,2,3,4,5,6,7,8,9,10}	{5,6,7,8,9,10}	{5,6,7,8,9,10}
X_3	{1,2,3,4,5,6,7,8,9,10}	{1,2,3,4,5,6,7,8,9,10}	{6,7,8,9,10}

Consistencia de un CSP

- **Consistencia global**
  - Una restricción global implica un número arbitrario de variables (no necesariamente todas).
  - Ocurren frecuentemente en problemas reales y se pueden manejar con algoritmos específicos más eficientes que los de propósito general.
  - Por ejemplo, la restricción Alldiff implica que todas las variables relacionadas deben tener valores distintos.
    - Una forma simple de inconsistencia detectada por Alldiff es que si m variables están relacionadas y tienen n posibles valores distintos, si m > n entonces la restricción no puede ser satisfecha.
  - Otra restricción importante es la restricción de recursos.
    - La restricción Atmost(10, P1, P2, P3, P4) indica que como máximo se pueden asignar 10 empleados a las 4 tareas en total, siendo Pi el nº de empleados asignados a la tarea i.
      - Se pueden detectar inconsistencias simplemente comprobando la suma de los valores mínimos de los dominios de cada variable.
    - En grandes problemas de limitación de recursos lo habitual es representar los dominios con límites superior e inferior, y se realizan operaciones de propagación de límites con sus respectivas comprobaciones de consistencia globales.

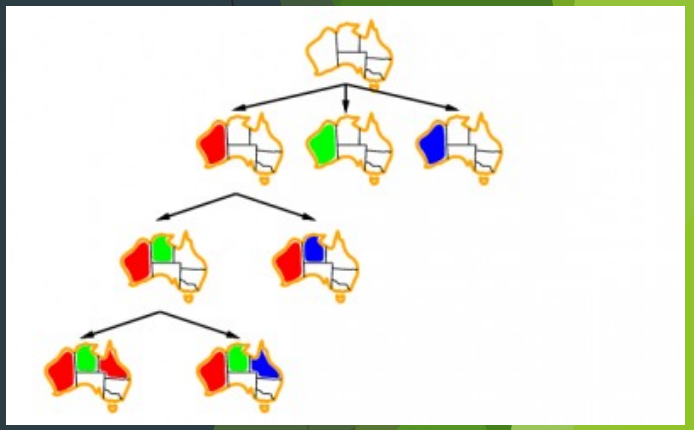
Resolución del CSP

- Los CSP son típicamente resueltos mediante un algoritmo de búsqueda en árbol.
- Cada nodo en el árbol de búsqueda corresponde con un conjunto de dominios $D'_1 \times D'_2 \times \cdot \cdot \cdot \times D'_n$ de forma que $D'_j \subset D_j$.
  - Un nodo no es más que una contracción de los dominios originales que todavía no se ha demostrado como no válido.
- El árbol de búsqueda puede ramificar desde un nodo a otro asignando un valor a una variable del problema.

```
WHILE not solved AND not infeasible DO
    consistency checking (domain reduction)
    IF a dead-end is detected THEN
        try to escape from dead-end (backtrack)
    ELSE
        select variable
        assign value to variable
    ENDIF
ENDWHILE
```

Resolución del CSP

- La búsqueda mediante backtracking es la base sobre la que se soportan la mayoría de algoritmos de CSP, correspondiente a la búsqueda en profundidad DFS.
- La forma más habitual de estructurar el árbol es dotar de un orden estático a las variables.
  - Un nodo en el nivel k del árbol representará un estado donde las variables $x_1,\ldots,x_k$ estarán asignadas a valores concretos de sus dominios, mientras que el resto de variables $x_{k+1},\ldots,x_n$ no lo están.
  - Cada nodo se puede asignar con la tupla formada por las variables asignadas en ese momento.
    - Los nodos en el primer nivel son 1-tuplas con valor para x_1.
    - Los nodos del segundo nivel son 2-tuplas con valores en x_1 y x_2.
    - Un nodo del nivel k es hijo de un nodo del nivel k-1 si la tupla del hijo es una extensión de la del padre añadiendo una asignación para x_k.
    - Los nodos en el nivel n, las n-tuplas, son las soluciones al problema si son consistentes.



Backtracking cronológico (BT)

- El backtracking cronológico (BT) es el algoritmo más utilizado.
- Asumiendo un orden estático de las variables y sus valores:

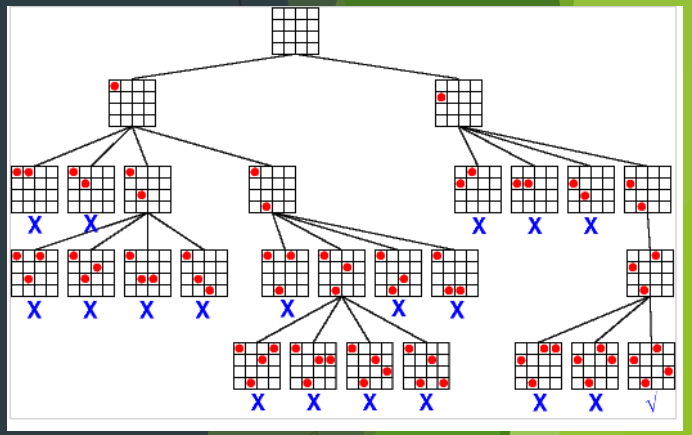
1) Selecciona la siguiente variable de acuerdo al orden de las variables y les asigna su próximo valor.

2) Esta asignación se comprueba en todas las restricciones de la variable actual y las anteriores.

   - Si todas las restricciones se han satisfecho, vuelta al punto 1.
   - Si alguna restricción no se satisface, la asignación actual se deshace y se prueba con el próximo valor de la variable.

3) Si no se encuentra ningún valor consistente, se encuentra una situación sin salida (dead-end) y se debe retroceder a la anterior variable asignada y probar con un nuevo valor.

4) Si asumimos que estamos buscando una sola solución, BT termina cuando a todas las variables se les ha asignado un valor (devolviendo la solución), o cuando todas las combinaciones variable-valor se han probado sin éxito (sin solución).



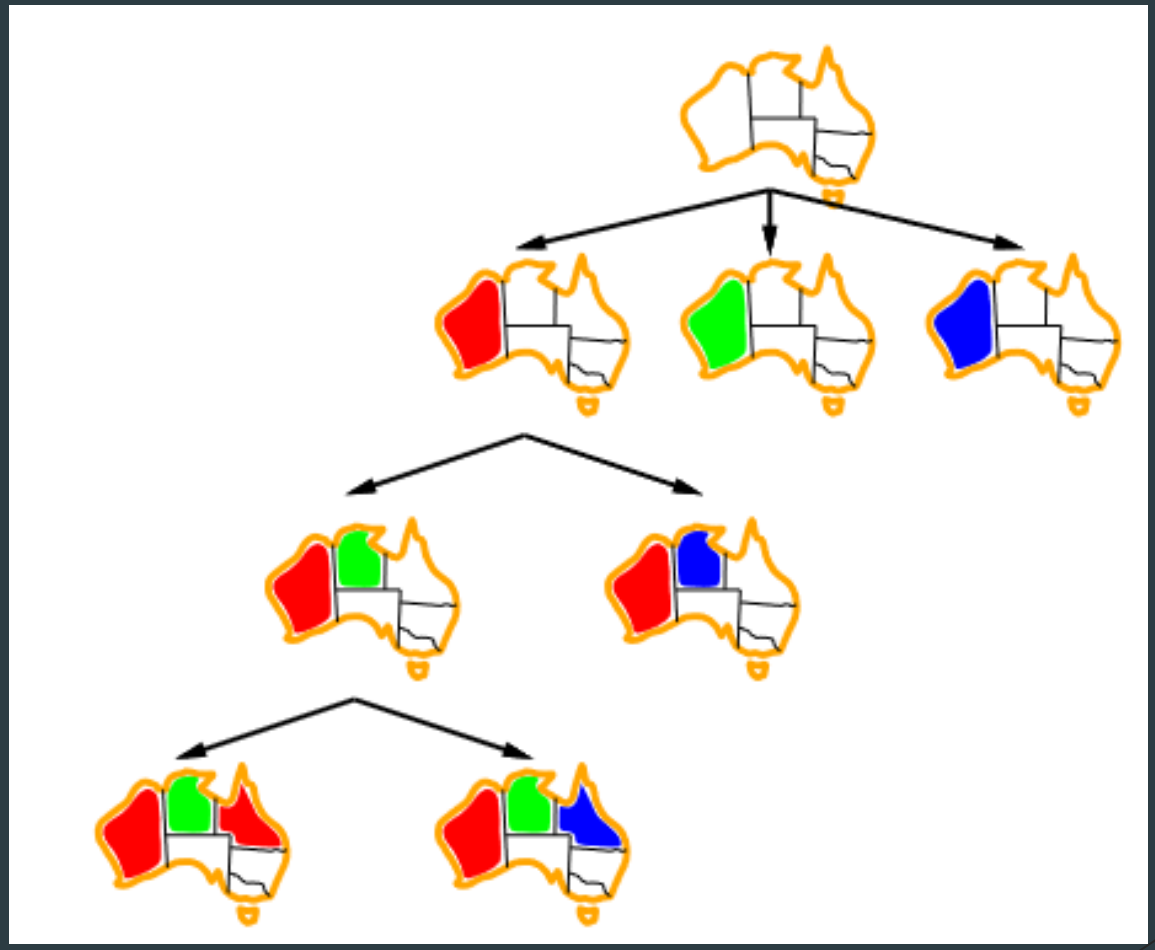
Backtracking cronológico (BT)

- BT es fácilmente generalizable a restricciones no binarias.
  - Cuando se prueba un valor de la variable actual, se comprueban todas las restricciones de la variable actual con las anteriores.
  - Si una restricción involucra a la variable actual y a algunas variables futuras, esta restricción no se comprobará hasta que se asigne valores a las variables futuras.
- Se trata de un algoritmo simple pero ineficiente.
  - Tiene visión local del problema.
  - Ignora las relaciones entre la variable actual y las futuras.
  - No recuerda las acciones previas, por lo que puede repetir la misma acción varias veces.
- Se han desarrollado algunos algoritmos de búsqueda más robustos para paliar estos defectos.

Backtracking cronológico (BT)

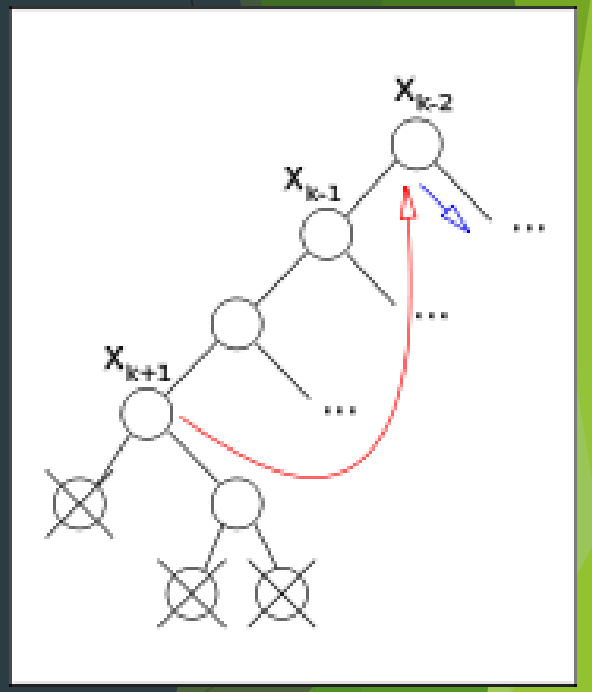
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Backtracking cronológico (BT)



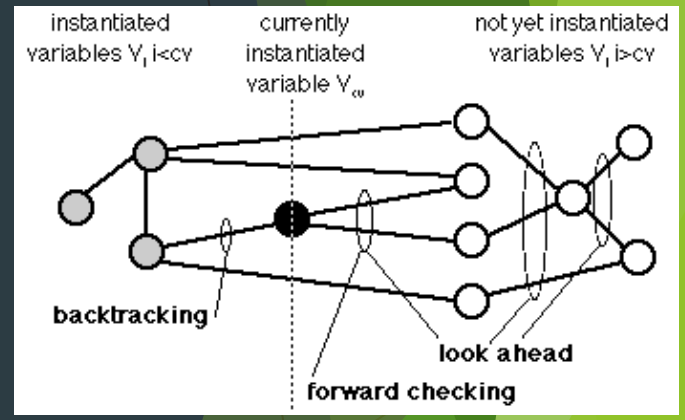
Backjumping (BJ)

- BJ pertenece a la familia de algoritmos **look-back**, que llevan a cabo la comprobación de la consistencia hacia atrás de una forma más eficiente.
- BJ se comporta de una forma más inteligente cuando encuentra un dead-end. En vez de retroceder a la variable instanciada anterior, salta a la variable más profunda (cercana) x_j que está en conflicto con la variable actual x_i, donde $j < i$.
  - Cambiar la instanciación de x_j puede hacer posible encontrar una instanciación consistente de la variable actual.
- La variante conflict-directed Backjumping (CBJ) almacena para cada variable un conjunto de conflictos mutuos que permite no repetir conflictos existentes a la vez que saltar a variables anteriores como BJ.



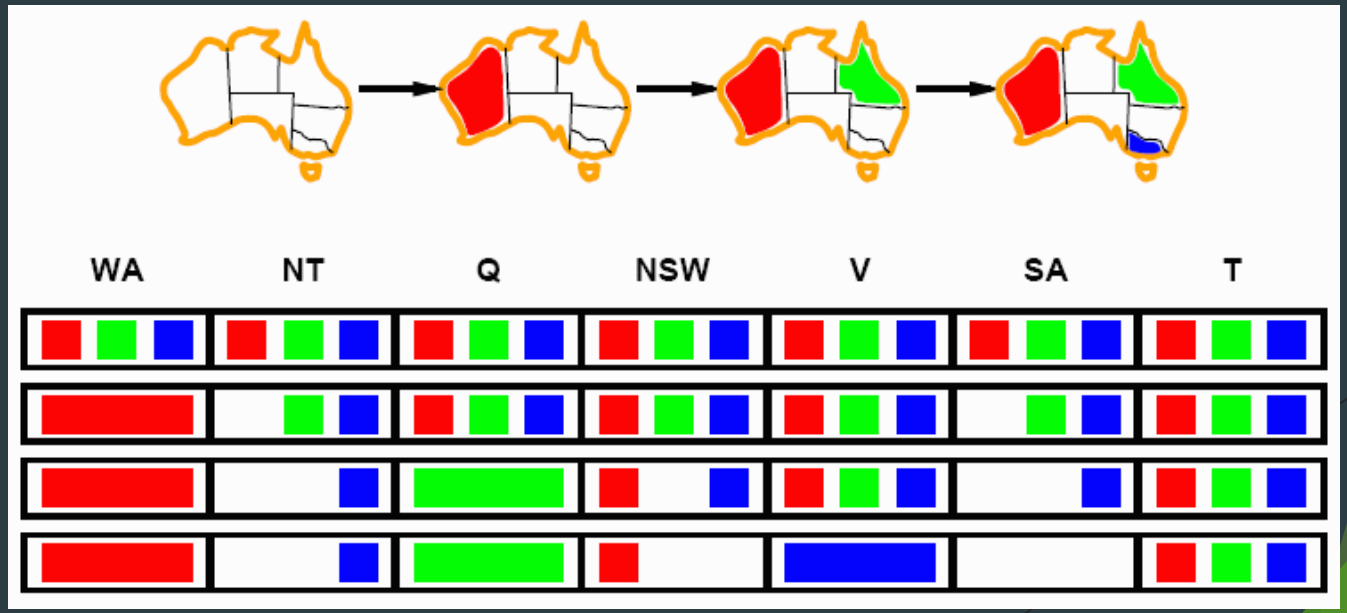
Forward checking (FC)

- FC pertenece a la familia de algoritmos **look-ahead**, que hacen una comprobación hacia delante de la asignación actual para buscar inconsistencias con variables futuras, además de las variables actual y pasadas.
  - Permiten identificar los dead-end con anterioridad y realizar podas más efectivas.
- **Forward checking** (FC) es el algoritmo look-ahead más común.
  - En cada etapa, comprueba la asignación actual con todos los valores de las futuras variables que tienen restricciones con la variable actual.
  - Los valores de las futuras variables que son inconsistentes son temporalmente eliminados de sus dominios para la rama actual de búsqueda.
  - Si el dominio de una variable futura se queda vacío, la instanciación de la variable actual se deshace y se prueba otro valor.
  - Su ningún valor es consistente, se realiza BT estándar.



Forward checking (FC)

- FC garantiza que, en cada etapa, la solución parcial actual es consistente con cada valor de cada variable futura.
- Cuando se asigna un valor a una variable, solo se comprueba hacia delante con las variables con las que tiene restricciones.
- Permite identificar antes situaciones sin salida y podar el espacio de búsqueda.



Forward checking (FC)

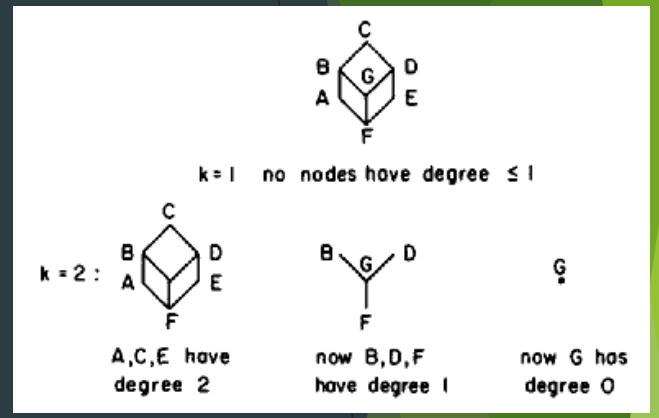
1. Seleccionar x_i.
2. Instanciar $(x_i, a_i) : a_i \in D_i$.
3. Razonar hacia adelante (check-forward): Eliminar de los dominios de las variables aún no instanciadas con un valor aquellos valores inconsistentes con respecto a la instanciación (x_i, a_i), de acuerdo al conjunto de restricciones.
4. Si quedan valores posibles en los dominios de todas las variables por instanciar, entonces:
    - Si $i < n$, incrementar i, e ir al paso 1.
    - Si $i = n$, parar devolviendo la solución.
5. Si existe una variable por instanciar sin valores posibles en su dominio entonces retractar los efectos de la asignación (x_i, a_i):
    - Si quedan valores por intentar en D_i, ir al paso 2.
    - Si no quedan valores:
        - Si $i > 1$, decrementar i y volver al paso 2.
        - Si $i = 1$, salir sin solución.

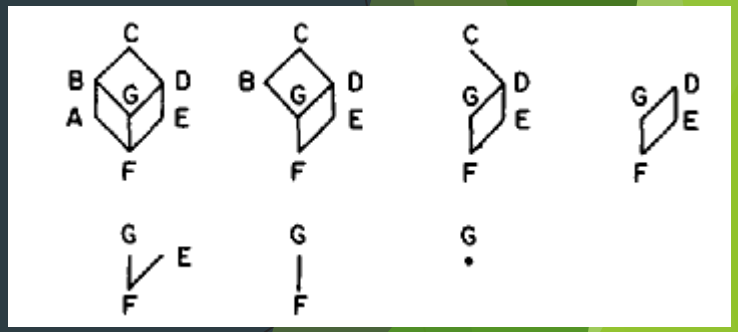
Heurísticas en CSP

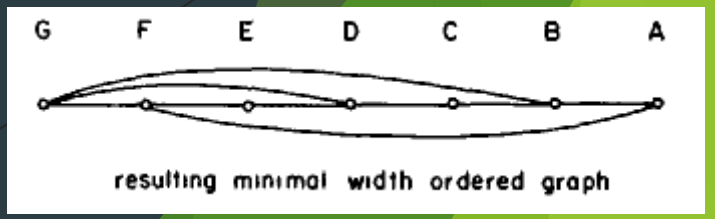
- Los algoritmos de búsqueda vistos requieren el orden en el que se van a asignar las variable, así como el orden de los valores de instanciación para cada variable.
- Seleccionar el orden correcto de las variables y valores puede mejorar notablemente la eficiencia.
- También puede resultar importante ordenar las restricciones del problema.
- **La ordenación de variables** tiene un gran impacto en el tamaño del espacio de búsqueda explorado.
  - Generalmente, tratan de seleccionar lo antes posible las variables más restringidas para detectar lo antes posible los dead-ends y reducir las vueltas atrás.
  - Pueden ser estáticas o dinámicas.
    - Las **heurísticas de ordenación de variables estáticas** generan un orden fijo de las variables antes de iniciar la búsqueda, basado en información global de las restricciones iniciales.
    - Las **heurísticas de ordenación de variables dinámicas** pueden cambiar el orden de las variables dinámicamente basándose en información local durante la búsqueda.

Heurísticas de ordenación de variables estáticas

- Se basan en la información global que se deriva de la topología del grafo de restricciones original del CSP.
- Algoritmos:
  - **Minimum Width (MW)**.
    - La anchura de la variable x es el número de variables que están antes de x, según un orden dado, y que son adyacentes a x.
    - La anchura de un orden es la máxima anchura de todas las variables bajo ese orden.
    - La anchura de un grafo de restricciones es la anchura mínima de todos los posibles órdenes.
    - Las variables se ordenan desde la última hasta la primera en anchura decreciente.
    - Las variables al principio de la ordenación son las más restringidas.
    - Asignando las variables restringidas al principio, los dead-ends se pueden identificar antes.







Heurísticas de ordenación de variables estáticas

For a graph with width k, to find an ordered graph with that width:
Repeat for i from n to 1 by -1.
    Find a node connected to $\leq k + 1$ others
    (Its existence is implied by a maximal subgraph linkage of k. If there is more than one, any one will do.)
    Remove the node from the graph, along with any edges connected to it. Make the node the ith node of the ordered graph.

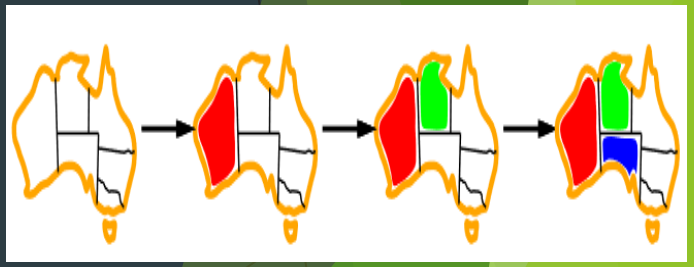
To determine maximal subgraph linkage
Remove from the graph all nodes not connected to any others. Set $k = 0$
Do while there are nodes left in the graph
    Set k to $k + 1$
    Do while there are nodes not connected to more than k others:
        Remove such nodes from the graph, along with any edges connected to them

Heurísticas de ordenación de variables estáticas

- Varios tipos:
  - **Maximum Degree (MD)**.
    - Ordena las variables en un orden decreciente de su grado en el grafo de restricciones.
    - El grado de un nodo se define como el número de nodos que son adyacentes a él.
    - Esta heurística tiene también como objetivo encontrar un orden de anchura mínima, aunque no lo garantiza.
  - **Maximum Cardinality (MC).**
    - Selecciona la primera variable arbitrariamente.
    - En cada fase añade a las variables seleccionadas aquella conectada con el grupo más grande de entre todas las variables ya seleccionadas.
    - La siguiente variable a ser elegida es aquella con restricciones con el número más grande de variables ya instanciada.

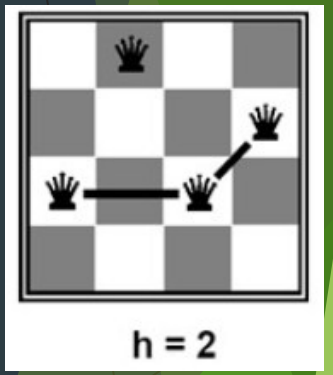
Heurísticas de ordenación de variables dinámicas

- El problema de los algoritmos de ordenación estáticos es que no tienen en cuenta los cambios en los dominios de las variables por la propagación de restricciones durante la búsqueda.
- Algoritmo para abordar este problema:
  - **Heurística del primer fallo (FF) o Minimum remaining Values (MRV)**
    - Para tener éxito, lo mejor es intentar primero donde sea más probable que falle.
    - En cada paso, seleccionar la variable más restringida con el dominio más pequeño.
    - Cuando se usa con algoritmos look-back, equivale a heurística estática que ordena las variables según tamaño de dominio ascendente.
    - Cuando se usa con algoritmos look-ahead, la ordenación es dinámica ya que los valores de las futuras variables se podan tras cada asignación de variables.



Heurísticas de ordenación de valores

- La idea básica consiste en seleccionar el valor de la variable actual que más probabilidad tenga de alcanzar una solución.
- La mayoría de heurísticas tratan de seleccionar el valor menos restringido de la variable actual, el que menos reduce el número de valores útiles para las futuras variables.
- Algoritmos:
  - **Heurística min-conflicts o Least Contraining Value (LCV)**.
    - Ordena los valores de acuerdo a los conflictos en los que están involucrados con las variables no instanciadas.
    - Asocia a cada valor *a* de la variable actual, el número total de valores en los dominios de las futuras variables adyacentes que son incompatibles con *a*.
    - El valor seleccionado es el asociado a la suma más baja.

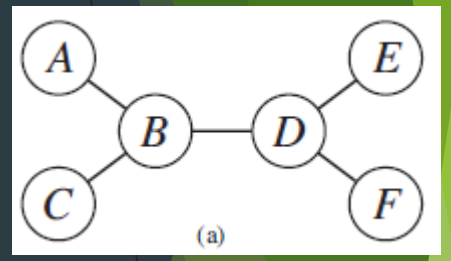


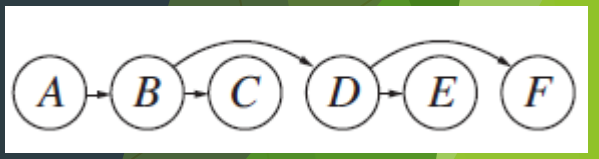
Heurísticas de ordenación de valores

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

Estructura de los problemas CSP

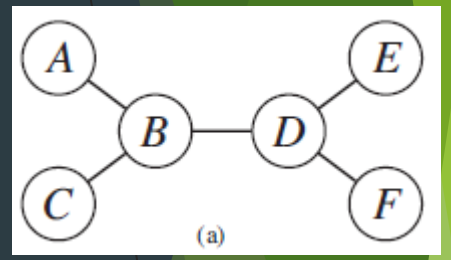
- En algunos casos, la estructura de los problemas, representada por su grafo de restricciones, se puede usar para encontrar soluciones de forma más rápida.
- Una idea consiste en encontrar subproblemas independientes y solucionarlos como CSP independientes, reduciendo las combinaciones posibles drásticamente. Después la solución se puede combinar de manera sencilla.
- Los subproblemas completamente independientes son difíciles de encontrar, pero existen otras estructuras de grafo que son fáciles de resolver.
- Los árboles son grafos donde dos variables solo están conectadas por un único camino.
- Cualquier CSP estructurado como un árbol se puede resolver en tiempo lineal con respecto a su número de variables.





Estructura de los problemas CSP

- La clave de los árboles CSP es la noción de **consistencia directa de arco**.
  - Un CSP es arco-consistente directo bajo una ordenación de variables X_1, X_2, …, X_n si y solo si cada X_i es arco-consistente con cada X_j para j > i.
- Para resolver un CSP en forma de árbol, se debe seleccionar una variable para ser la raíz del árbol, y seleccionar una ordenación de variables de forma que cada variable aparezca después de su padre en el árbol. Esta ordenación se llama **ordenación topológica**.
- Cualquier árbol con n nodos tiene n-1 arcos, por lo que se puede convertir el grafo en arco-consistente directo en tiempo lineal, comparando en cada paso d posibles valores de dominio para dos variables.
  - Como cada unión de padre a hijo es arco-consistente, se puede elegir cualquier valor para el padre sabiendo que el hijo tendrá un valor válido para elegir sin necesidad de backtracking.



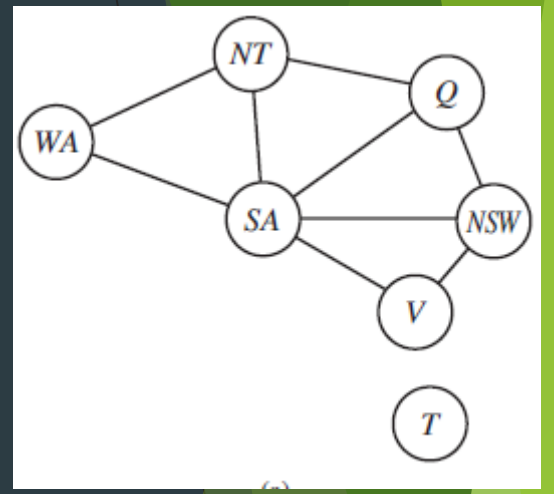
Estructura de los problemas CSP

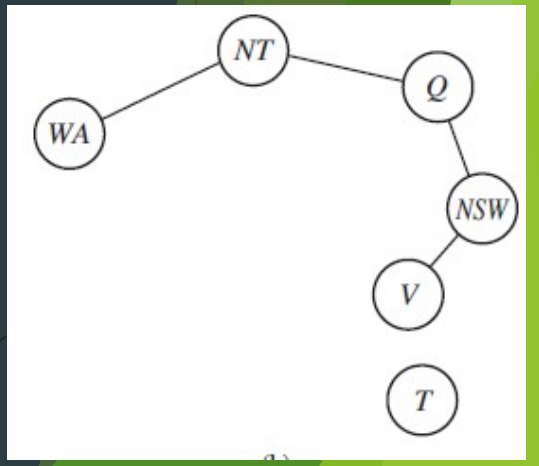
function TREE-CSP-SOLVER(*csp*) **returns** a solution, or failure
  inputs: *csp*, a CSP with components X, D, C

  $n \leftarrow$ number of variables in X
  assignment $\leftarrow$ an empty assignment
  root $\leftarrow$ any variable in X
  $X \leftarrow$ TOPOLOGICALSORT(X, *root*)
  for j = n **down to** 2 **do**
    MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
    if it cannot be made consistent **then return** *failure*
  for i = 1 **to** n **do**
    assignment[X_i] $\leftarrow$ any consistent value from D_i
    if there is no consistent value **then return** *failure*
  return *assignment*

Estructura de los problemas CSP

- Una vez se tiene un algoritmo eficiente para árboles, la idea es reducir los grafos de restricciones a árboles de alguna manera.
- La primera idea consiste en asignar valores a algunas variables de forma que el resto de variables formen un árbol.
  - Considerando el ejemplo de coloreado de mapas, si se eliminase el nodo SA, el grafo se convertiría en un árbol.
  - Esto se puede conseguir asignando un color a SA y borrando de los dominios de las otras variables los valores inconsistentes con el valor asignado a SA.
  - De esta forma, cualquier solución en el CSP después de eliminar el nodo SA y sus restricciones será consistente con el valor asignado a SA.
  - Finalmente, el problema se resuelve de manera sencilla con el algoritmo de resolución de árboles.
  - Como el color elegido para SA puede ser el incorrecto, habría que probar cada posible valor.





Estructura de los problemas CSP

- El algoritmo, conocido **cutset conditioning**:

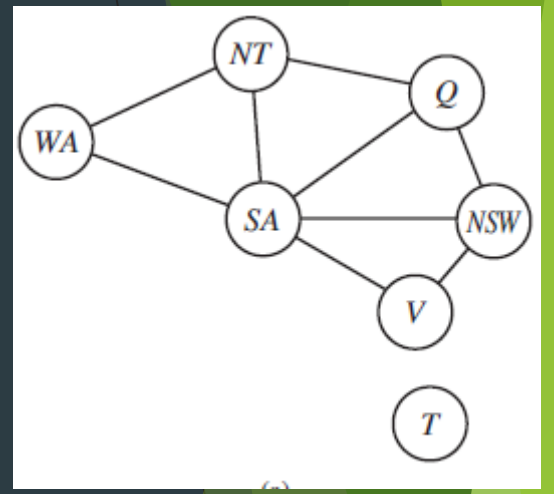
  1) Elegir un subconjunto S de las variables del CSP de forma que el grafo se convierta en un árbol. S se conoce como **conjunto de corte de ciclo (cycle cubset)**.

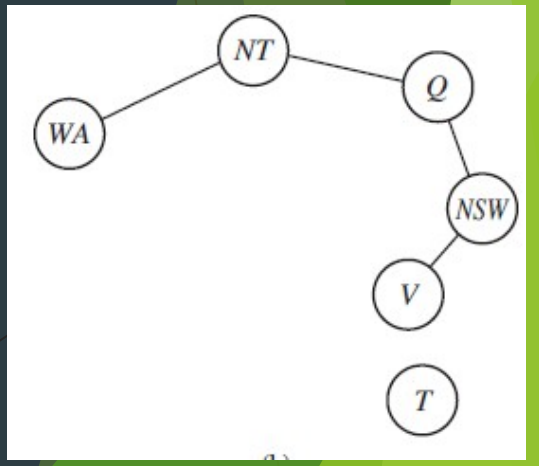
  2) Para cada posible asignación de las variables en S que satisfagan todas las restricciones.

     a) Eliminar de los dominios de las variables restantes los valores inconsistentes con los valores de S elegidos.

     b) Si el CSP restante tiene solución, devolver junto con las asignaciones para S.

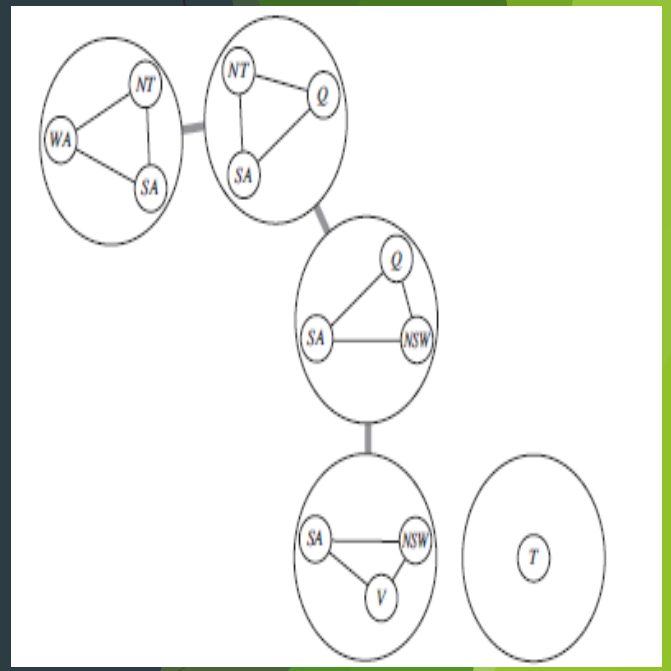
- Cuando más pequeño sea el subconjunto de corte de ciclo, y por tanto más se parezca el grafo original a un árbol, más rápido será el algoritmo frente al backtracking tradicional.
- Encontrar el ciclo de corte más pequeño es un problema complicado, aunque existen diferentes aproximaciones eficientes conocidas.





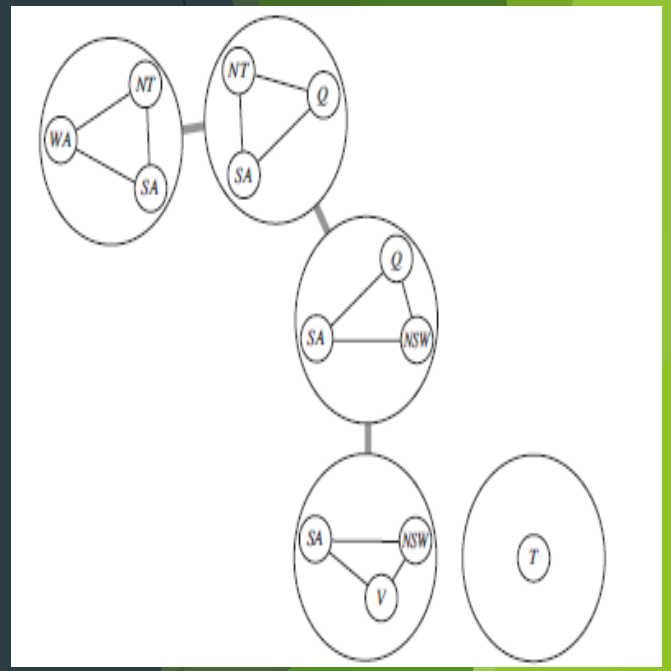
Estructura de los problemas CSP

- La segunda aproximación consiste en construir una **descomposición en árbol** de las restricciones en un **conjunto de subproblemas conectados**.
- Cada problema es resuelto de forma independiente y las soluciones restantes son combinadas.
- La descomposición en árboles debe seguir las siguientes normas:
  - Cada variable del problema original debe aparecer en al menos uno de los subproblemas.
  - Si dos variables están conectadas por una restricción en el problema original, deben aparecer juntas (con la restricción) en al menos uno de los subproblemas.
  - Si una variable aparece en dos subproblemas del árbol, debe aparecer en cada subproblema del camino que conecta los dos subproblemas.



Estructura de los problemas CSP

- Las dos primeras condiciones asegurar que todas las variables y restricciones se representan en la descomposición.
- La tercera condición refleja la restricción de que cada variable debe tener el mismo valor en cada subproblema que aparece.
- Se resuelve cada subproblema de manera independiente.
  - Si un subproblema no tiene solución, el problema entero no tiene solución.
  - Si se pueden resolver todos los subproblemas, se debe intentar construir una solución global.
    - Cada subproblema se ve como una mega variable donde el dominio es el conjunto de todas las soluciones del subproblema.
    - Se resuelven las restricciones conectando los subproblemas usando el algoritmo eficiente de árboles estudiado.
    - Las restricciones entre subproblemas simplemente insisten en que las soluciones a los subproblemas coinciden en los valores para las mismas variables.



Estructura de los problemas CSP

- Un grafo de restricciones admite muchas descomposiciones en árbol. La idea es encontrar aquella que haga los subproblemas lo más pequeños posible.
  - Encontrar la descomposición óptima es complicado, pero existen heurísticas que ayudan a aproximarse a la solución.
- A parte de la estructura del grafo de restricciones, también es importante la **estructura de los valores de las variables**.
  - En muchos problemas, para cada solución consistente, existen un conjunto de soluciones basadas en la permutación de los valores asignados, lo que se conoce como **simetría de valor**.
  - Se puede reducir enormemente el espacio de búsqueda si se consigue eliminar la simetría de las soluciones, introduciendo **restricciones de rotura de simetría**.
  - En muchas ocasiones, esta rotura de simetría se consigue imponiendo órdenes arbitrarios de valor sobre ciertas variables, pero puede resultar un problema costoso de resolver. No obstante, la mejora a obtener es muy importante, por lo que debe tenerse muy en cuenta.