1 PROBLEMAS DE SATISFACCION DE RESTRICCIONES (CONSTRAINT SATISFACTION PROBLEMS)

Son problemas donde la asignación de ciertos valores a una variable restringe los posibles valores de otras variables, el ejemplo comun que se utiliza para explicar esta característica es que suponer que se tiene que pintar de un color único los países de un determinado color de un numero de colores limitados, cuando se pinta el primer país, ya en si esta acción restringe que los países aledaños ya no pueden ser pintados del color elegido en primera instancia, es decir no pueden existir dos países aledaños pintados con el mismo color. Si bien tienen características especiales, se dan muchísimo en el mundo real, en el ámbito de la planificación, en sus diversas áreas. Para la explicación practica se utilizarán los modelos y algoritmos vistos anteriormente y como estos pueden ser aplicados en este ámbito. Es importante establecer que no se debe confundir con las restricciones estudiadas en los algoritmos de búsquedas.

1.1 CARACTERÍSTICAS

Este tipo de problemas poseen las siguientes características en relación a:

- Estados: los cuales no tienen un solo valor sino un conjunto de variables. Se usará representación factorizada (listas clave-valor, preferentemente diccionario) en vez de atómica.
- Solución: se considera que se ha llegado a una solución cuando un conjunto de valores de las variables, donde cada una que satisfacen una serie de restricciones. Se puede buscar:
    - Una solución (cualquiera). En este tipo de problemas, en principio se buscará una sola solución. En este tipo de problemas ya no se busca encontrar la solución optima como prioridad, aunque esta no deja de ser importante, sin embargo en principio lo que se busca es una solución.
    - Todas las soluciones, estos algoritmos también son adecuados para encontrar todas las soluciones existentes.
    - La solución óptima, o en su caso buscar la mejor solución si es el caso.

En principio se trabaja con algoritmos que encuentran una solución, para posteriormente trabajar con los algoritmos en ciertas modificaciones para que puedan encontrar todas las soluciones y finalmente adecuarlos para encontrar la solución óptima.

1.2 DEFINICIONES

En este tipo de problemas cambian las definiciones asumidas en relación a la representación del problema.

Es importante establecer algunas definiciones importantes:

- Asignación: dar un valor a una variable, puede ser:
    - Parcial: sólo algunas variables de una tabla tienen valores.
    - Completa: todas las variables de una tabla tienen valores.
- Consistencia: si los valores que están asignados a las variables cumplen las restricciones.

1.3 TIPOS DE RESTRICCIONES

Para el trabajo con problemas de satisfacciones de restricciones se deben considerar los siguientes tipos de restricciones:

- Unarias: reducen el dominio de una variable. Supongamos que tenemos la edad de una persona que como máximo suponemos viven hasta 100 y sobre eso se considera la restricción de introducir la mayoría de edad, eso ya genera una condicionante adicional.
- Binarias: se aplica a pares de variables. El valor asignado a una variable reduce el dominio de la otra variable.
- Múltiples: se aplica a más de dos variables. El valor asignado a una variable reduce el dominio del resto de variables.

Una técnica importante es convertir las restricciones múltiples en restricciones binarias, que se realizan en un proceso denominado preprocesado, al igual que las restricciones unarias.

1.4 RESTRICCIONES SUAVES

Existen otro tipo de restricciones que no se parecen a las anteriores y estas tiene que ver con los valores propios que tiene dentro de una misma variable.

Las restricciones suaves definen preferencias de valores de una misma variable sobre otros:

- Un valor de un dominio es mejor que otro del mismo dominio.
- Para introducirlos en este tipo de problemas se representan y manejan mediante costes de asignación.
- Los problemas con satisfacción de restricciones y que tienen la característica de presentar restricciones suaves se solucionan mejor con redes bayesianas (o programación lineal).

1.5 DEFINICIÓN DEL PROBLEMA

Por lo mencionado anteriormente es importante realizar una redefinición de algunos elementos para abordas este tipo de problemas, para esto se definen 3 conjuntos de datos:

- Variables: $X = \{X_1, X_2, \ldots, X_n\}$.
- Dominios: $D = \{D_1, D_2, \ldots, D_n\}$, donde, $D_i = \{v_1, v_2, \ldots, v_k\}$ valores válidos para X_i.
- Restricciones: $C = \{C_1, C_2, \ldots, C_m\}$, donde C_i = <variables, relación>.
    - Relación Implícita: lista completa.
    - Relación Explícita: función.

Si bien en los problemas de búsqueda se trabajo con una sola variable, en estos problemas se trabajará con varias variables, que tienen dominios y restricciones.

1.6 DOMINIOS

Un dominio es un conjunto de valores igual o inferior al conjunto de todos los estados posibles. Si bien un numero puede tomar un numero infinito de valores, al asignarle un dominio, ya se restringe el numero de posibles valores que puede tomar una variable. Los dominios depende del tipo de variable.

- Variables Discretas:
    - Dominios Finitos: exponencial (inicialmente).
    - Dominios Infinitos: no decidibles (no se pueden generar un algoritmo para solucionarlo) si las restricciones no son lineales. Si las restricciones corresponden a una relación lineal en ese caso son decidibles.
- Variables Continuas: polinomial (inicialmente) con restricciones lineales (mejor con programación lineal).

Los problemas de satisfacción de restricciones, se utilizan solo con variables discretas y dominios finitos.

1.7 RESTRICCIONES

Tienen como función principal reducir el dominio.

- Unarias: reducen dominio de una variable.
- Binarias: sobre pares de variables.
  notación: $<(A, B) , A \neq B> \rightarrow A \neq B$
- Múltiples: sobre 3 o más variables.
- Suaves: preferencias de los valores de un dominio sobre otros (mejor con redes bayesianas o programación lineal).

1.8 GRAFOS DE RESTRICCIONES

Cuando se definía el problema dentro de grafos, se utilizaban dos grafos, uno que representaba el problema y otro que utilizaba el algoritmo de búsqueda. En los problemas de satisfacción de restricciones, el grafo de representación del problema, el grafo se denomina grafo de restricciones, donde:

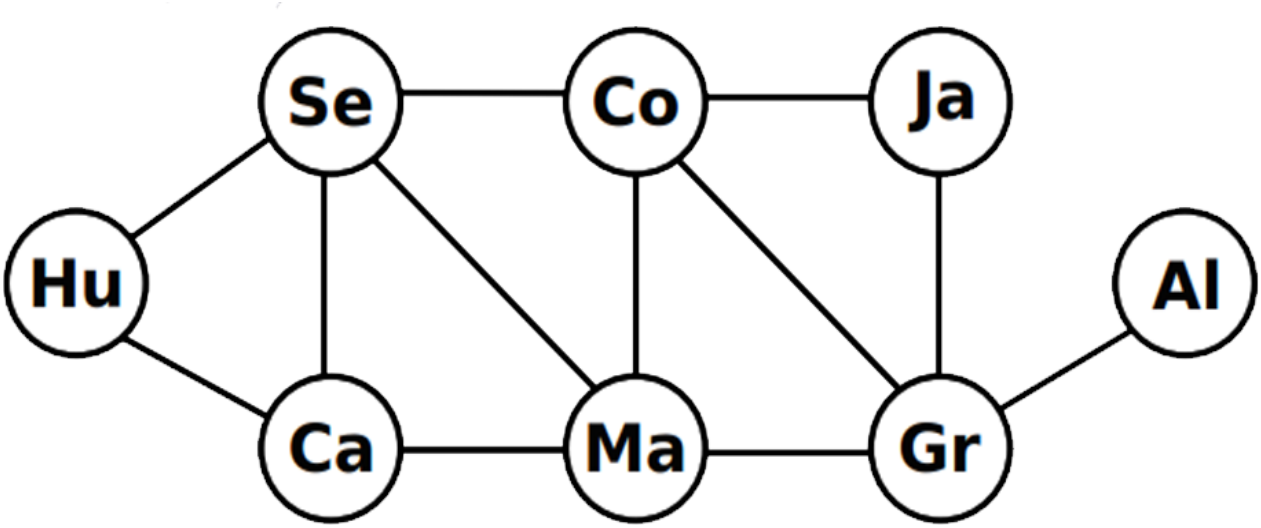
- Los nodos, similar a los anteriores, van a ser variables, cada nodo será una variable y a partir de ese conjunto de variables se generan los nodos. Cada variable es un nodo y representan un estado y va a tener una serie de valores dentro de un dominio probablemente dinámico.
- Los arcos, son restricciones entre variables. Si hay un arco entre dos nodos, entre esos nodos habrá una restricción.
- Solo se consideran tipos binarios, por las limitaciones de la presente técnica algorítmica.
- Una propiedad importante es que permiten dividir el problema en subproblemas de forma más fácil. (Divide y venceras)

1.9 EJEMPLO

El ejemplo de coloreado del mapa de Andalucia (España)**:**

EL problema consiste en buscar la manera de colorear cada provincial con un color diferente utilizando tres colores, buscando que dos provincias que tienen frontera no tengan el mismo color.

cEl grafo de restricciones es el siguiente:



Considerando el grafo se pueden establecer las restricciones de que cada nodo solo puede pintarse de un color de tres, es decir la variable del nodo almacenara por decir rojo, verde o azul. Y También pueden existir restricciones unitarias que podrían exigir que por ejemplo Sevilla debe pintarse si o si azul o los de Malaga no sean nunca rojos. Recordar que los arcos representan restricciones.

1.10 INFERENCIA EN PROBLEMAS DE SATISFACCION DE RESTRICCIONES

Si el problema lo veríamos como problemas de búsqueda tradicionales, dos cosas serian importantes: solo importa el estado objetivo, no importa el camino, solo importa que el conjunto de variables cumplan con las restricciones, solo interesa el estado final y el segundo aspecto es que se asume que todos los caminos tienen la misma profundidad (según el número de variables). Estas dos características son las que ayudaran a determinar el tipo de algoritmos que es apropiado utilizar.

Considerando los algoritmos vistos anteriormente para búsquedas:

- Basados en anchura: siempre hay que llegar al último nivel, el resto es intrascendente (exposición exponencial).
- Basados en profundidad: prueba muchísimas asignaciones inútiles, es exponencial, pero se podría mejorar.

En principio de debe partir de uno de ellos, sin embargo, por la cantidad de recursos y complejidad que generan los basados en anchura, este tipo de algoritmos no son útiles para aplicarlos en este tipo de búsquedas. Los algoritmos basados en profundidad si bien generan prueba con muchísimas asignaciones inútiles, siendo exponencial, este puede ser mejorado y adecuado a los propósitos de este tipo de problemas.

Las posibles mejoras que se pueden aplicar al algoritmo de búsqueda elegido son:

- Dominios: la asignación de un valor a una variable puede reducir mucho el dominio del resto.
- Poda: una asignación que no cumpla una restricción es descartada y no se sigue expandiendo esa rama.
- Asignaciones Conmutativas: da igual el orden en que se asignen los valores a las variables, el resultado final es el mismo.

Aparte de elegir la siguiente variable a asignar (fase búsqueda), el algoritmo reducirá los dominios del resto de variables (fase inferencia), existen dos formas para aplicar la propagación de restricciones (momentos):

- Preprocesado (antes de empezar): unarias, múltiples a binarias.
- Intercalado: se aplican alternativamente las fases. (Revisar el efecto de la asignación de un valor a una variable en relación a los demás nodos)

1.11 CONSISTENCIA LOCAL

Técnicas de propagación de restricciones (inferencia), pueden ser:

- de Nodo.
- de Arco.

- de Camino.
- k-Consistencia.

1.11.1 CONSISTENCIA LOCAL DE NODO

Si todos los valores del dominio de una variable satisfacen sus restricciones unarias. Reduce los dominios de los valores de cada variable. Una vez reducidos, se pueden eliminar las restricciones unarias.

1.11.2 CONSISTENCIA LOCAL DE ARCO

Si todos los valores del dominio de una variable satisfacen sus restricciones binarias. Para comprobarla, se usa el algoritmo AC-3.

1.11.3 CONSISTENCIA LOCAL DE CAMINO

Si infieren restricciones implícitas comprobando tripletas de variables con las restricciones binarias. Si para cada asignación de valores consistentes para 2 variables, existe una asignación para una 3ª que sea consistente con las anteriores. Se utiliza el algoritmo: PC-2 (variante de AC-3).

1.11.4 K-CONSISTENCIA

Es una ampliación de k variables, es decir n restricciones son consideradas.

- Cualquier nº de variables:
  - de Nodo : 1-Consistencia.
  - de Arco : 2-Consistencia.
  - de Camino: 3-Consistencia.
- Coste Tiempo: polinomial.
- Coste Espacio: exponencial.
- Uso Normal: máx. 3-Consistencia.

Aplicar este proceso reduce el árbol de búsqueda.

Gracias a la consistencia se pueden estableces tres enfoques para resolver problemas de busquedas:

- Aplicar algoritmos de búsquedas tradicionales
- Aplicar consistencias locales leves y algoritmos de búsqueda tradicionales.
- Utilizar solamente consistencias locales avanzadas.

Si bien la k-consistencia puede suplir a los algoritmos de búsqueda, esto para problemas grandes no es aplicable por el espacio que requieren. Lo recomendado es utilizar máximo 3-consistencia.

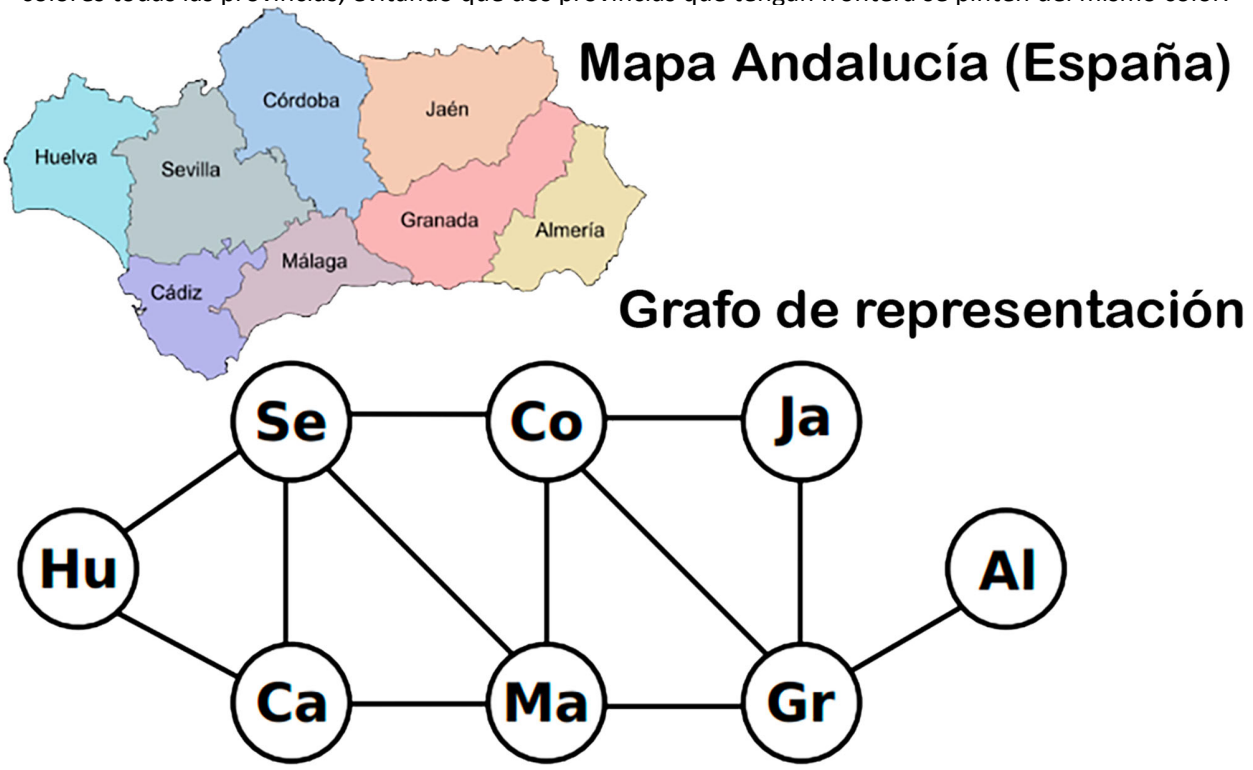
1.11.5 PSEUDOCODIGO DEL ALGORITMO AC-3

```
función AC-3(psr) devuelve booleano
  conjunto ← CREAR-CONJUNTO(psr.ARCOS)
  mientras no conjunto.ESTÁ-VACIO() hacer
    arco ← conjunto.POP()
    si REDUCIDO(psr, arco) entonces
      si arco.X.DOMINIO.ESTÁ-VACIO() entonces
        devolver false
      vecinos ← arco.X.VECINOS.QUITAR(arco.Y)
      por cada Z en vecinos hacer
        arco-vecino ← CREAR-ARCO(Z,X)
        conjunto.AGREGAR(arco-vecino)
  devolver true
```

```
función REDUCIDO(psr, arco) devuelve booleano
  reducido ← false
  por cada valor-X en arco.X.DOMINIO hacer
    consistente ← false
    por cada valor-Y en arco.Y.DOMINIO hacer
      si arco.PERMITE(valor-X, valor-Y) entonces
        consistente ← true
        break
    si no consistente entonces
      arco.X.DOMINIO.QUITAR(valor-X)
      reducido ← true
  devolver reducido
```

1.11.6 EJEMPLO CONSISTENCIA LOCAL

SE tomara en cuenta el problema del mapa de Andalucía (España) anteriormente presentado para revisar la aplicación de las distintas consistencias, al problema planteado de pintar con un determinado número de colores todas las provincias, evitando que dos provincias que tengan frontera se pinten del mismo color.



Para la consistencia local de nodo (1-Consistencia), se considera la restricción de dominio que las provincias que tienen salida al mar no pueden ser pintadas de azul. La representación de las variables y sus respectivos dominios, las restricciones unarias se pueden representar de la siguiente manera:

PROVINCIA	DOMINIO INICIAL	APLICANDO RESTRICCIONES UNARIAS (DOMINIO REDUCIDO)
HU	{R,V,A}	{R,V}
SE	{R,V,A}	{R,V,A}
CA	{R,V,A}	{R,V}
CO	{R,V,A}	{R,V,A}
MA	{R,V,A}	{R,V}
JA	{R,V,A}	{R,V,A}
GR	{R,V,A}	{R,V}
AL	{R,V,A}	{R,V}

Las provincias que tienen acceso marítimo son: Huelva(HU), Cádiz(CA), Málaga(MA), Granada(GR) y Armería(AL). El dominio inicial son los posibles valores que puede asumir cada una de las variables.

Para la consistencia local de arco (2-Consistencia), considerando los elementos que emplea el algoritmo pc-3, se tiene:

Todos los posibles arcos (restricciones) se cargan en un conjunto

Conjunto: SE-HU, SE-CA, SE-MA, SE-CO, CA-HU, CO-MA, … (Aristas del grafo de provincias)

Supongamos que se realiza la asignación: HU=R → D_{HU} = {R} (Cuando un dominio se reduce a un solo valor se asume que es una asignación).

Procedemos con otro arco o arista:

SE-HU D_{SE} = {R,V,A}

Si se asigna a SE:

SE=R → D_{HU} = {} → D_{SE} = {V,A}

SE=V → D_{HU} = {R}

SE=A → $D_{HU\ =}$ ={R}

Si se reduce el dominio se vuelven a cargar los arcos que parten de SE.

Conjunto.AGREGAR(SE-CA)

Conjunto.AGREGAR(SE-MA)

Conjunto.AGREGAR(SE-CO)

Para la consistencia local de camino (3-Consistencia), considerando que se desea colorear el mapa utilizando solo dos colores(Blanco y Negro) y solo se consideran las provincias Hu, Se y Ca, se tendría:

Al no existir consistencias unarias, no se reduce nada.

Al revisar la consistencia de arco:

HU={B,N} HU-SE C.Arco

SE={B,N} SE-CA C.Arco

CA={B,N} CA-HU C.Arco

¿CA C.Camino con (HU,SE)?

Es consistente en arco por que se toma dos a dos, cuando se consideran ya la consistencia de los tres nodos simultáneamente, existe el problema que al asigna a dos provincia un color la tercera ya no puede ser asignada con ningún color por lo cual se reduce su dominio y con eso se deduce que no hay solución.

(HU=B,SE=N) → CA={}

(HU=N,SE=B) → CA={}

Respuesta: No (No tiene solución)

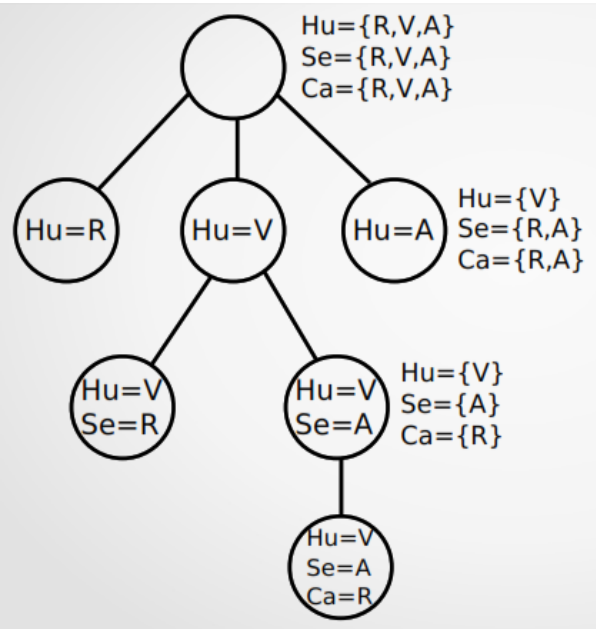
La consistencia de arco no es suficiente para identificar si existe o no solución.

1.12 BÚSQUEDA CON VUELTA ATRÁS

Se basa en el algoritmo Primero en Profundidad, lo primero que se debe realizar es el preprocesado que restrinja los dominios, luego se debe considerar un bucle que aplique los siguientes tres pasos:

1. Se escoge una variable sin valor, sin considerar un orden especifico.
2. Se expande los valores que cumplan con las restricciones. Se expanden hijos según los valores que tenga, con dominio de valores que cumplan restricciones y se toma uno de esos valores y se asigna a esa variable, se genera un hijo con todas las variables cambiantes más la nueva variable, luego se aplica la consistencia local en arco.
3. Vuelta atrás si el dominio de una variable se queda vacío. Se continua con el paso uno, pero con otra variable, hasta encontra una asignación que permita que todas las variables tengan un valor.

Para aclarar la explicación emplearemos un árbol generado a partir del mapa de Andalucía (España), a partir del grafo generado, solo se consideran tres provincias (Huelva (Hu), Sevilla(Se) y Cádiz(Ca)).



Inicialmente el árbol de búsqueda empieza con un nodo vacío que no tiene ninguna variable asignada, a lado de este nodo se puede apreciar los dominio de color que puede tomar cada una de las tres provincias, es decir en principio cualquiera de las tres provincias puede ser asignada con cualquiera de los tres colores (rojo (R), verde (V) o azul (A)).

Supongamos que iniciamos con Hu y le asignamos inicialmente V, por lo cual solo se expandiría el nodo del medio, otros dos no se tomarían en cuenta a no ser que volveríamos atrás, es decir si por el nodo de Hu = V, no se encontraría solución se volvería a elegir otro color para Hu y se expedirían recién uno de los otros nodos. Continuando con Hu = V, una vez que se ha asignado el valor se debe aplicar el algoritmo de consistencia local en arco, lo cual ocasionara que se reduzcan los dominios de valores para Se y Ca, una vez que se ha realizado este proceso se continua eligiendo uno de los otros nodos, ya sea el que asigna a Se = A o Se = R. Suponiendo que se eligió Se = A, se debe revisar las consistencias de nodos que reducirá el dominio para Ca = R. En este proceso se eliminarán muchos nodos que no son necesarios expandir en el algoritmo de búsqueda normal, optimizando de manera importante la complejidad temporal. Es importante resaltar si en cada expansión se encontraría que el dominio para el nodo expandido es cero, en ese caso se debe volver atrás y elegir otro nodo y realizar el mismo proceso.

Es importante asumir las siguientes características:

- El nodo inicial no tiene variables,
- El nivel indica el numero de variables que poseen los nodos.
- No hace falta función objetivo, heurística, acciones o modelo de transición: van implícitos en PSR.

Algunas mejoras que se pueden considerar en este proceso son las siguientes:

Considerando que en el inicio se de este algoritmo se debe elegir una variable, la variable que se elija influye de manera importante en el resultado, por lo cual se debe considerar un procedimiento de ordenación tanto a las variables como a los valores. Se consideran las siguientes técnicas:

- MVR (variables), mínimos valores restantes, se escoge la variable con el menor dominio, en caso de empate, la que participe en el mayor número de restricciones.
- VMR (valores)(valor menos restringido), una vez elegida una variable, se elige el valor que dé mayores opciones a los vecinos.

Este tipo de directrices funcionan cuando se está buscando una sola solución, si se buscan todas las soluciones la ordenación da igual porque hay que probar todas y no es necesario realizar esta mejora.

Otra mejora que se puede considerar es la estrategia de filtrado, que consiste en detectar los fallos lo antes posible. Si nos encontramos expandiendo el árbol y de acuerdo al camino que se tomo se esta rumbo a un callejón sin salida, es de alta importancia identificar esta situación lo más antes posible. Existen dos técnicas:

- Comprobación Hacia Delante: cada vez que se asigna un valor se comprueba la consistencia de arco de esa variable con las que esté conectada. (Es aplicar el algoritmo AC-3, consistencia de camino, 4-consistencia), es lo que ya se reviso anteriormente.
- MAC: mejora de la anterior. Se usa AC-3 pero, en vez de meter todos los arcos al conjunto, se meten sólo los de los vecinos de la variable aun no asignados.

Otras mejoras que se puede aplicar son:

- Salto Atrás: cuando se hace vuelta atrás, en vez de escoger la siguiente variable según el orden, se localizar la variable que provocó el conflicto (conjuntos conflicto). Se almacenan las variables que pudieron ser causantes de un conflicto, para cuando se retorne atrás y encontrar la variable que fue la que inicio el conflicto.
- Aprendizaje de Restricciones: se guarda un conjunto con las asignaciones que llevaron a una vuelta atrás para que, si vuelven a aparecer, se vuelva atrás lo antes posible.

1.13 PSEUDOCODIGO ALGORITMO VUELTA ATRÁS

Es un algoritmo recursivo, su pseudocódigo es el siguiente:

```
función BÚSQUEDA-VUELTA-ATRÁS(psr)
devuelve solución o fallo
  nodo-inicial ← CREAR-NODO-RAIZ()
  devolver VUELTA-ATRÁS(nodo-inicial, psr)
```

```
función VUELTA-ATRÁS(nodo, psr)
devuelve solución o fallo
  si nodo.ES-COMPLETO() entonces devolver nodo
  vacias ← psr.VARIABLES.VACIAS()
  variable ← SELECCIONAR-VARIABLE(vacias)
  ordenacion ← ORDENAR-VALORES(variable, nodo, psr)
  por cada valor en ordenacion hacer
    si nodo.CONSISTENTE(valor) entonces
      nodo.VARIABLES[variable] ← valor
      inferencias ← INFERENCIAS(psr, variable, valor)
      si inferencias ≠ fallo entonces
        nodo.INTEGRAR(inferencias)
        resultado ← VUELTA-ATRÁS(nodo, psr)
        si resultado ≠ fallo entonces
          devolver resultado
    nodo.VARIABLES[variable] ← nulo
    nodo.DESINTEGRAR(inferencias)
  devolver fallo
```

1.14 MEJORAS EN LOS PROBLEMAS DE SATISFACCION DE RESTRICCIONES

Se considera la estructura de los Problemas, utilizando la técnica: divide y vencerás. Divide el problema en subproblemas. Se encuentra soluciones más rápido, algunos problemas pueden pasar de una complejidad de orden exponencial a orden lineal. Las técnicas mas utilizadas son:

1.14.1 INDEPENDENCIA

Si el grafo se puede dividir en zonas independientes entre sí, entonces la unión de soluciones para cada zona es una solución global.

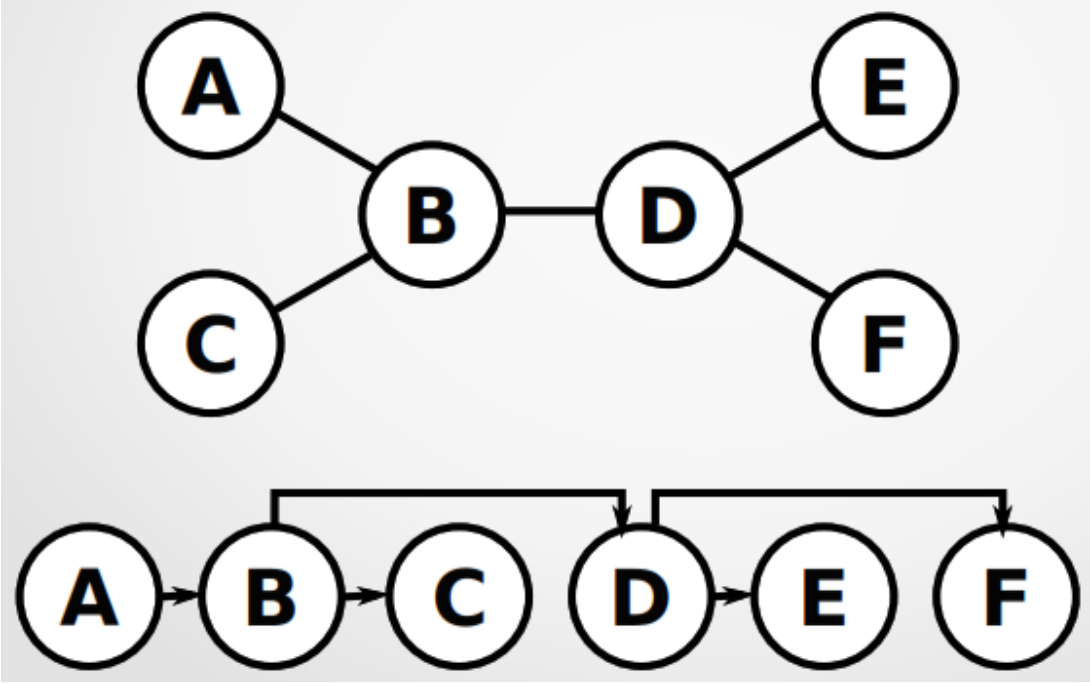
1.14.2 ÁRBOLES DE RESTRICCIONES

Si grafo de restricciones es árbol el coste lineal según variables.

Linealizar Árbol: se ordena haciendo un recorrido en preorden (ordenar variables) a partir de cualquier nodo.

Consistencia de Arco Dirigido: si cada variable es consistente en arco con cada una de sus sucesoras. Si fuera el caso no es necesario utilizar el algoritmo de Vuelta Atrás.

Ejemplo de linealizar árbol.



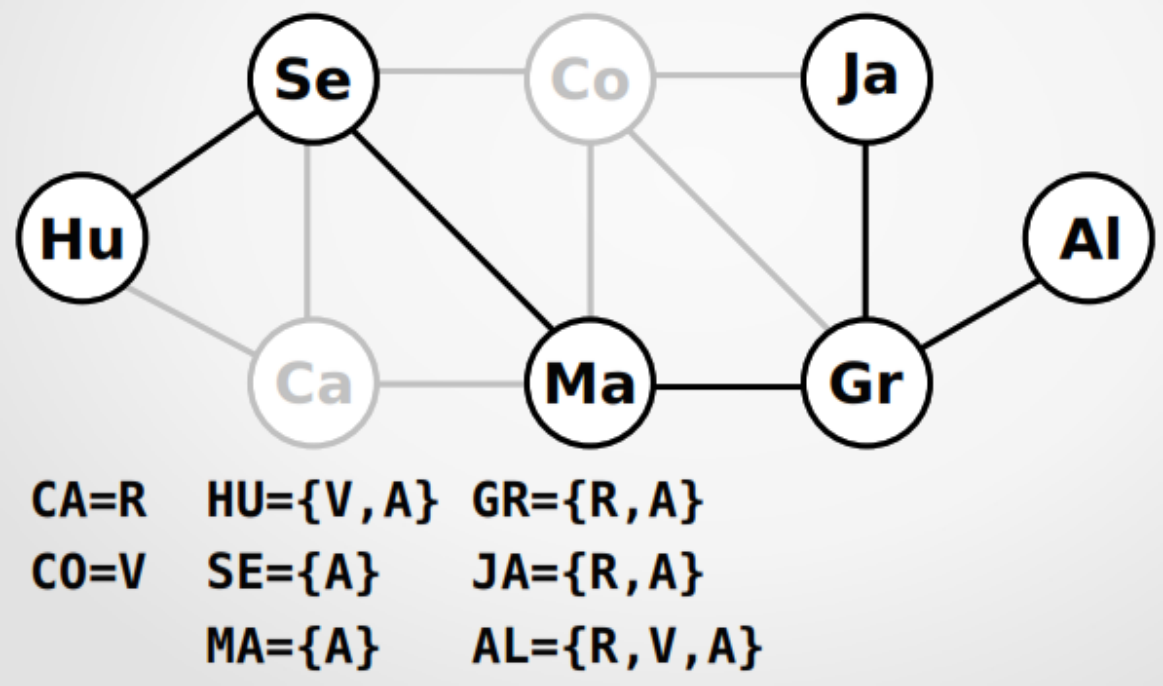
1.15 PSEUDOCODIGO PSR-ARBOL

```
función PSR-ÁRBOL(psr) devuelve solución o fallo
  cantidad ← psr.VARIABLES.CANTIDAD
  asignaciones ← CREAR-DICCIONARIO(psr.VARIABLES)
  raiz ← CREAR-ESTADO-RAIZ(ALEATORIO(psr.VARIABLES))
  ORDENAR-VARIABLES(psr.VARIABLES)
  desde i = cantidad hasta 1 hacer
    X ← psr.VARIABLES[i]
    desde j = 1 hasta cantidad - 1 hacer
      Y ← psr.VARIABLES[j]
      si no CONSISTENTE-ARCO(X, Y) entonces
        devolver fallo
  desde 1 hasta cantidad hacer
    X ← psr.VARIABLES[i]
    valor ← VALOR-CONSISTENTE(X.DOMINIO)
    si no valor entonces devolver fallo
    asignaciones[X] ← valor
  devolver asignaciones
```

1.16 REDUCIR GRAFOS A ÁRBOLES

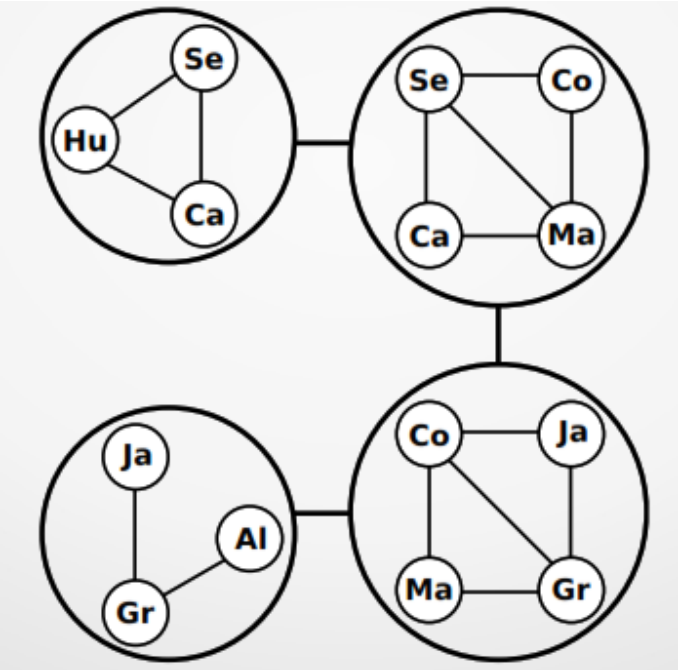
Existen dos técnicas importantes para convertir un grafo en nodo:

Eliminar Nodos: se asignan valores a algunas variables para que el resto formen un árbol. De los dominios de las variables que forman el árbol hay que quitar los valores inconsistentes con las asignaciones iniciales.



La segunda alternativa para convertir un grafo en árbol:

Juntar Nodos: se crea árbol de hipernodos (varios nodos adyacentes del grafo de restricciones). Si dentro de un hipernodo no se encuentra una solución, no existe solución global. Se tiene que obtener todas las soluciones de cada hipernodo, se mira cuáles son consistentes entre sí.



Para aplicar esta técnica deben cumplirse las siguientes reglas:

- Cada nodo debe estar en, al menos, un hipernodo.
- Si 2 variables están conectadas, deben aparecer en, al menos, un hipernodo.
- Si una variable aparece en 2 hipernodos, debe aparecer en cada hipernodo intermedio.