

Problemas de Satisfacción de Restricciones

José Luis Ruiz Reina

José Antonio Alonso Jiménez

Franciso J. Martín Mateos

María José Hidalgo Doblado

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Índice

Representación

Backtracking

Consistencia de arcos

Otras mejoras

Reparación heurística

Optimización de Restricciones

Problemas de Satisfacción de Restricciones

- Un *problema de satisfacción de restricciones* (PSR) viene definido por los siguientes elementos:
 - Un conjunto finito de *variables* X_1, \dots, X_n
 - Un conjunto finito de *dominios* D_i asociados a cada variable X_i , especificando los posibles valores que puede tomar
 - Un conjunto finito de *restricciones* C_1, \dots, C_m , que definen una serie de propiedades que deben verificar los valores asignados a las variables
- Una solución al problema es una asignación de valores a las variables $\{X_1 = v_1, \dots, X_n = v_n\}$ tal que $v_i \in D_i$ y se verifican las restricciones
- Esta formulación permite una representación simple del problema y el uso de heurísticas de propósito general, *independientes* del problema

Clasificación de los PSR (I)

- Clasificación según el tipo de restricciones:
 - Restricciones de obligación (hard constraints): asocia a cada combinación de valores de los dominios de las variables implicadas un valor lógico. Todos los valores asociados deben ser verdaderos.
 - Restricciones de preferencia (soft constraints): asocia a cada combinación de valores de los dominios de las variables implicadas un valor numérico. La suma de todos los valores asociados debe ser máxima (o mínima).

Clasificación de los PSR (II)

- Clasificación según los dominios:
 - Dominios finitos
 - Dominios infinitos (discretos o continuos)
 - Posibilidad de infinitas soluciones
 - Resoluble si las restricciones de obligación son lineales
 - En el caso de restricciones de obligación no lineales no hay un algoritmo de propósito general
- Clasificación según el número de variables implicadas en las restricciones:
 - Restricciones binarias
 - Restricciones múltiples
 - Todo problema de obligación con restricciones múltiples se puede reducir a uno equivalente con restricciones binarias

Ejemplo: N reinas (I)

- Situar N reinas en un tablero de ajedrez de tamaño $N \times N$ de forma que no se den jaque mutuamente.
- Variables: V_1, \dots, V_N
- Dominios: $D_i = \{(f, c) : 1 \leq f, c \leq N\}$
- Restricciones:
 - Jaque horizontal: $\Pi_1(V_i) \neq \Pi_1(V_j)$ ($1 \leq i < j \leq N$)
 - Jaque vertical: $\Pi_2(V_i) \neq \Pi_2(V_j)$ ($1 \leq i < j \leq N$)
 - Jaque diagonal: $|\Pi_1(V_i) - \Pi_1(V_j)| \neq |\Pi_2(V_i) - \Pi_2(V_j)|$ ($1 \leq i < j \leq N$)
- Problema con dominios finitos y restricciones binarias (de obligación)
- 8-reinas: Con esta representación hay
 $64^8 = 281,474,976,710,656$ posibles asignaciones y
126 restricciones.

Ejemplo: N reinas (II)

- Situar N reinas en un tablero de ajedrez de tamaño $N \times N$ de forma que no se den jaque mutuamente.
- Variables: $V_{1,1}, \dots, V_{N,N}$
- Dominios: $D_{i,j} = \{0, 1\}$
- Restricciones:
 - Jaque horizontal: $\sum_{i=1}^N (V_{i,j}) = 1 (1 \leq j \leq N)$
 - Jaque vertical: $\sum_{j=1}^N (V_{i,j}) = 1 (1 \leq i \leq N)$
 - Jaque diagonal: $V_{i,j} = 1 \rightarrow V_{i',j'} = 0$
 $((1, 1) \leq (i, j) < (i', j') \leq (N, N), |i - j| = |i' - j'|)$
- Problema con dominios finitos y restricciones binarias y N -arias (de obligación)
- 8-reinas: Con esta representación hay $64^2 = 4096$ posibles asignaciones, **16** restricciones **8**-arias y **280** restricciones binarias.

Ejemplo: N reinas

- Situar N reinas en un tablero de ajedrez de tamaño $N \times N$ de forma que no se den jaque mutuamente.
- Variables: V_1, \dots, V_N
- Dominios: $D_i = \{1, \dots, N\}$ ($1 \leq i \leq N$)
- Restricciones:
 - Jaque horizontal: $V_i \neq V_j$ ($1 \leq i < j \leq N$)
 - Jaque diagonal: $|V_i - V_j| \neq |i - j|$ ($1 \leq i < j \leq N$)
- Problema con dominios finitos y restricciones binarias (de obligación)
- 8-reinas: Con esta representación hay $8^8 = 16,777,216$ posibles asignaciones y **84** restricciones.

Ejemplo: coloreado de mapas

- Usando tres colores (p.e. azul,rojo,verde) colorear el mapa de Andalucía:



- Variables: **Almería, Cádiz, Córdoba, Granada, Huelva, Jaén, Málaga, Sevilla**
- Dominios: **{azul, rojo, verde}**
- Restricciones: las provincias vecinas deben tener distinto color. Ejemplo: **Granada ≠ Jaén**
- Problema con dominios finitos y restricciones binarias (de obligación)

Ejemplo: satisfacibilidad proposicional

- Problema SAT (para cláusulas): Dado un conjunto de cláusulas proposicionales $\{C_1, \dots, C_m\}$ formadas con las variables $\{P_1, \dots, P_n\}$ determinar si existe una asignación de valores de verdad de manera que se satisfagan todas las cláusulas
- Variables: P_1, \dots, P_n
- Dominios: $D_i = \{V, F\}$ (booleanos)
- Restricciones: $\{C_1, \dots, C_m\}$
- Problema con dominios finitos y restricciones múltiples (de obligación)
- Problema NP-completo
 - En el peor caso, no es posible resolver PSRs con dominios finitos en tiempo polinomial (si $P \neq NP$)

Ejemplo: criptoaritmética

- Asignar valores del 1 al 8 a cada letra de manera que:

$$\begin{array}{r} (\text{C}_1 \ \text{C}_2 \ \text{C}_3) \\ \text{I} \ \text{D} \ \text{E} \ \text{A} \\ + \\ \text{I} \ \text{D} \ \text{E} \ \text{A} \\ \hline \text{M} \ \text{E} \ \text{N} \ \text{T} \ \text{E} \end{array}$$

- Variables: $I, D, E, A, M, N, T, C_1, C_2, C_3$
- Dominios: $\{1, \dots, 8\}$ para I, D, E, A, M, N, T y $\{0, 1\}$ para C_1, C_2, C_3
- Restricciones:
 - Primera suma: $2 * A = (10 * C_3) + E$
 - Segunda suma: $(2 * E) + C_3 = (10 * C_2) + T$
 - Tercera suma: $(2 * D) + C_2 = (10 * C_1) + N$
 - Cuarta suma: $(2 * I) + C_1 = (10 * M) + E$
- Problema con dominios finitos y restricciones múltiples (de obligación)

Ejemplo: Reglas Golumb

- Dar el menor tamaño, L , para una regla en la que se puedan colocar M marcas en distintas unidades tal que todas las distancias entre las marcas son distintas.
 - Para 5 marcas la regla más pequeña es de tamaño 11:
 - Para 23 marcas la mejor regla es de tamaño 372. No se conoce el mejor tamaño para 24 marcas.
- Comprobar si existe solución para una regla de tamaño L .
- Variables: m_1, \dots, m_M
- Dominios: $D_i = \{0, \dots, L\}$
- Restricciones: $|m_i - m_j| \neq |m_k - m_n|$

Ejemplo: asignación de tareas

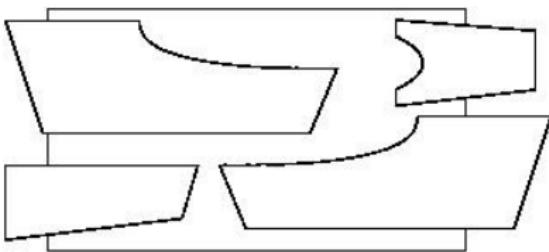
- Asignar tareas a empleados de acuerdo con su capacidad para desarrollarlas
- Tabla de capacidades (C_i):

| | T_1 | T_2 | T_3 |
|-------|-------|-------|-------|
| E_1 | 1 | 3 | 2 |
| E_2 | 3 | 2 | 1 |
| E_3 | 2 | 3 | 1 |

- Variables: E_i
- Dominios: $D_i = \{T_1, \dots, T_n\}$
- Restricciones:
 - Obtener la mejor $\sum_{i=1}^n C_i$
- Problema con dominios finitos y restricciones múltiples (de preferencia)

Ejemplo: planificación de corte

- Encontrar la manera de situar patrones de corte en una pieza de cartón
- Variables: P_1, P_2, P_3, P_4 (piezas)
- Dominios: Coordenadas en el plano
- Restricciones:
 - Las piezas no deben superponerse



- Problema con dominios continuos y restricciones binarias (de obligación)

Ejemplos de aplicación real de PSR

- Asignación de tareas: qué profesor imparte qué asignatura, qué trabajador realiza qué tarea,...
- Elaboración de horarios: qué días, a qué hora y en qué aulas se imparten las clases,...
- Configuración de hardware
- Verificación de hardware y software (a través de su codificación como problema de satisfacibilidad proposicional)
- Diagnóstico de errores en hojas de cálculo
- Diseño de planos: viviendas,fábricas,...
- Planificación de transportes: aeropuertos, estaciones de tren, empresas de transportes

Muchos implican variables continuas

Tipos de Problemas de Satisfacción de Restricciones

Tipos de variables

- Variables discretas
 - Con dominios finitos.

Con n variables y cada una de ellas con dominio de tamaño d , hay d^n "soluciones" posibles.
 - Con dominios infinitos.
- Variables continuas

Tipos de restricciones

- Binarias o múltiples

Todo problema con restricciones múltiples se puede reducir a uno equivalente con restricciones binarias
- De obligación o de preferencia

Variantes de PSR

- Determinar si existe o no una asignación que verifique las restricciones (un modelo)
- Encontrar un modelo
- Encontrar todos los modelos
- Determinar el número de modelos
- Encontrar el mejor modelo (dada alguna medida sobre la calidad de un modelo)
- Determinar si se verifica alguna propiedad en todos los modelos
- ...

Problemas de Satisfacción de Restricciones

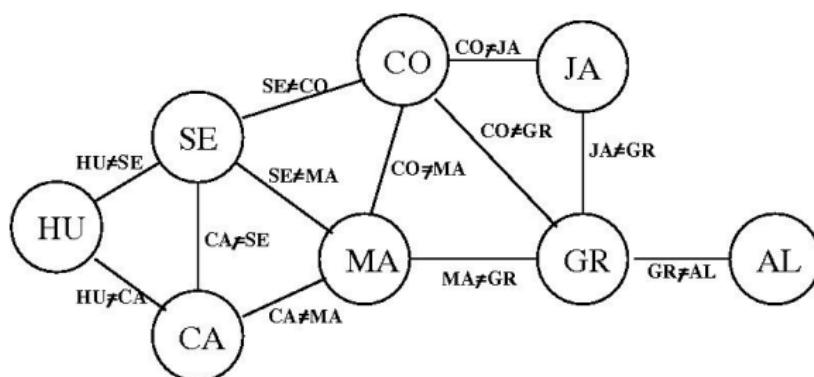
- En este tema, trataremos PSRs:
 - Con dominios finitos
 - Con restricciones binarias de obligación
- Solución de un PSR: Asignación de un elemento del dominio de cada variable que satisfaga todas las restricciones



Representación en forma de grafo

Representación en forma de grafo de un PSR binario

- Un nodo por cada variable
- Un arco por cada restricción



Implementación de la representación de un PSR

Un problema PSR estará compuesto por:

- **dominios**: Dada una variable x , `dominios[x]` será el conjunto de posibles valores para la variable x .
- **restricciones**: Dado un conjunto de variables $\{x_1, \dots, x_k\}$, `restricciones[\{x_1, \dots, x_k\}]` será la función que determine, dada un conjunto de valores para esas variables, si satisface la restricción entre los mismos.

A partir de estos elementos, asociados a un PSR se definen

- **variables**: el conjunto de variables del problema.
- **vecinos**: Dada una variable x , `vecinos[x]` será el conjunto de variables que aparecen con ella en alguna restricción.

PSR como problema de espacio de estados (I)

Estados:

- Asignaciones parciales (y *consistentes* con las restricciones) de valores a variables:
 $\{X_{i_1} = v_{i_1}, \dots, X_{i_k} = v_{i_k}\}$
- Estado inicial: {}
- Estados finales: asignaciones completas



Un estado



Estado inicial



Un estado final

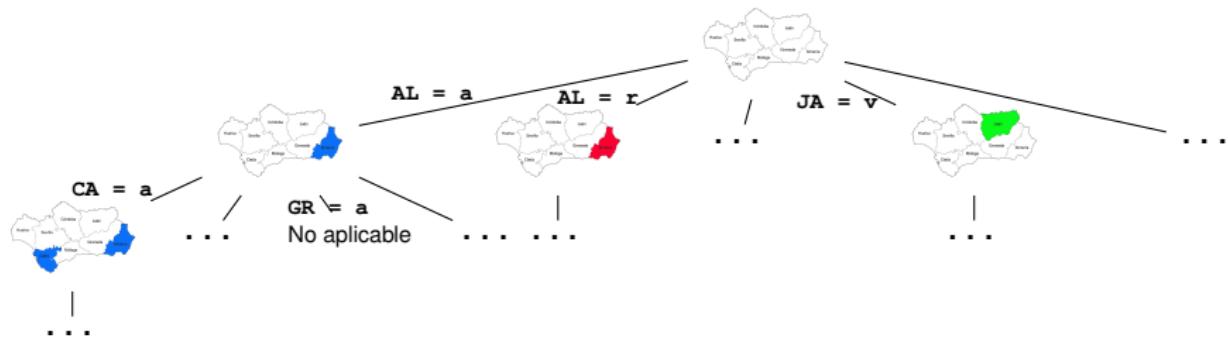
PSR como problema de espacio de estados (II)

Operadores:

- Asignar un valor (de su dominio) a una variable no asignada en el estado
- Aplicabilidad: la asignación tiene que ser consistente

Soluciones mediante búsqueda en profundidad

- Las soluciones (si las hay) están todas a la misma profundidad (igual al número de variables): $n!d^n$ hojas



Simplificaciones en la búsqueda

- El orden en el que se aplican los operadores es irrelevante para la solución final (*comutatividad*)
 - Los operadores pueden reducirse a considerar las posibles asignaciones a *una única variable* no asignada: d^n hojas
- El camino hacia la solución no es importante
 - No necesitamos la estructura de nodo
 - No necesitamos una representación explícita para los operadores
- No es posible repetir un estado durante la búsqueda:
 - No es necesario realizar comprobaciones para evitar repeticiones
 - Es decir, no es necesario tener una lista con las asignaciones parciales ya analizadas

Búsqueda en profundidad para PSR

- Estados: Asignaciones parciales
- Usaremos una función que elige la nueva variable a asignar (de entre las no asignadas) y otra función que ordena los valores del dominio de la variable seleccionada:
 - **SELECCIONA-VARIABLE (ESTADO)**
 - **ORDENA-VALORES (DOMINIO)**
- Función que calcula los sucesores:

```
FUNCTION PSR-SUCESORES (ESTADO)
1 Hacer VARIABLE igual a SELECCIONA-VARIABLE (ESTADO)
2 Hacer DOMINIO-ORD igual a
    ORDENA-VALORES (dominios[VARIABLE])
3 Hacer SUCESORES igual a vacío
4 Para cada VALOR en DOMINIO-ORD,
    4.1 Hacer NUEVO-ESTADO añadir VARIABLE=VALOR a ESTADO
    4.2 Si NUEVO-ESTADO es consistente con restricciones,
        añadir NUEVO-ESTADO a SUCESORES
5 Devolver SUCESORES
```

Búsqueda en profundidad para PSR

- Función que implementa el proceso de búsqueda:

FUNCTION PSR-BÚSQUEDA-EN-PROFUNDIDAD ()

- 1 Hacer **ABIERTOS** igual a una lista con un único elemento, la asignación vacía
- 2 Mientras que **ABIERTOS** no esté vacía,
 - 2.1 Hacer **ACTUAL** igual al primer estado de **ABIERTOS**
 - 2.2 Hacer **ABIERTOS** igual al resto de **ABIERTOS**
 - 2.3 Si **ACTUAL** es una asignación completa:
 - 2.4.1 devolver **ACTUAL** y terminar.
 - 2.4.2 en caso contrario,
 - 2.4.2.1 Hacer **NUEVOS-SUCESORES** igual a **PSR-SUCESORES (ACTUAL)**
 - 2.4.2.2 Hacer **ABIERTOS** igual al resultado de incluir **NUEVOS-SUCESORES** al principio de **ABIERTOS**
 - 3 Devolver **FALLO**.

Algoritmo de *backtracking*

- El algoritmo de búsqueda en profundidad anterior se suele llamar algoritmo de *backtracking*, aunque usualmente, se presenta de manera recursiva:

FUNCIÓN PSR-BACKTRACKING ()

1 Devolver PSR-BACKTRACKING-REC ({})

FUNCIÓN PSR-BACKTRACKING-REC (ESTADO)

1 Si ESTADO es una asignación completa, devolver ESTADO
y terminar.

2 Hacer VARIABLE igual a SELECCIONA-VARIABLE (ESTADO)

3 Hacer DOMINIO-ORD igual a
ORDENA-VALORES (**dominios[VARIABLE]**)

4 Para cada VALOR en DOMINIO-ORD,

 4.1 Hacer NUEVO-ESTADO igual a la asignación
 obtenida ampliando ESTADO con VARIABLE=VALOR

 4.2 Si NUEVO-ESTADO es consistente con **restricciones**:

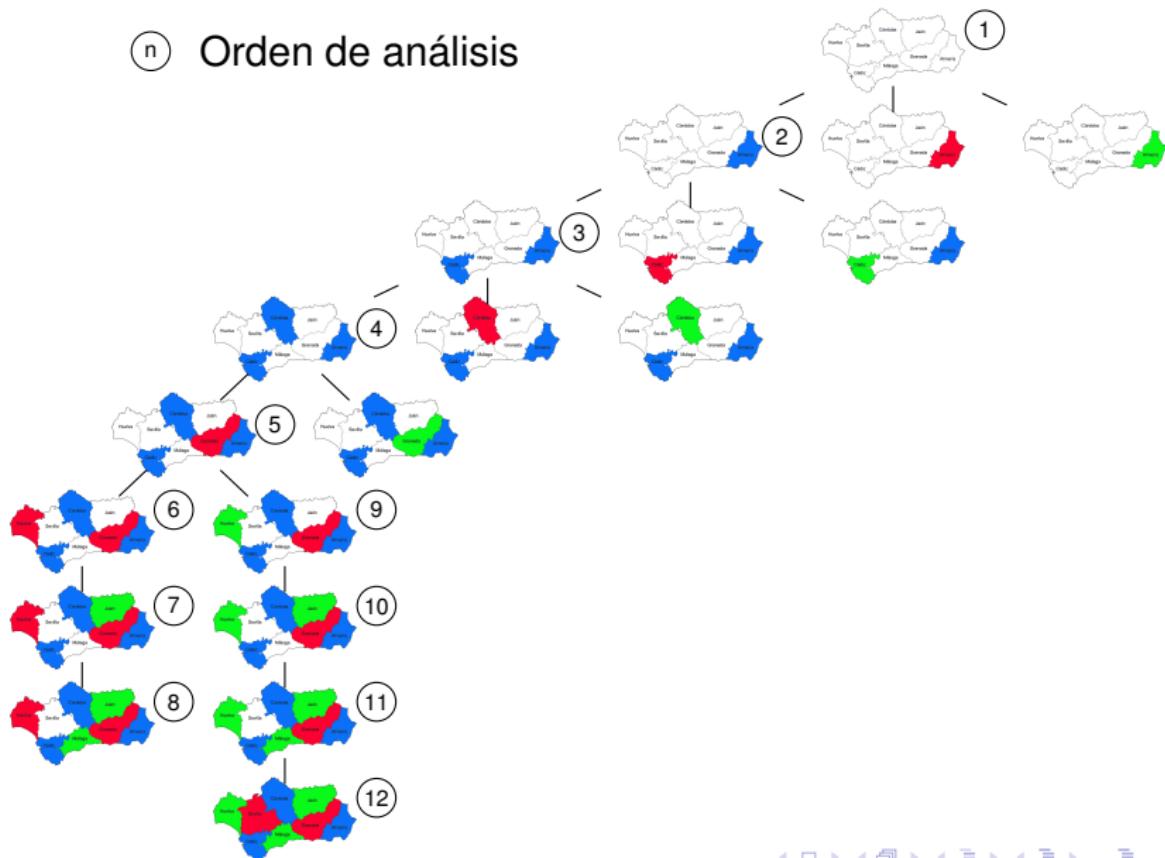
 4.2.1 Hacer RESULTADO igual a
 PSR-BACKTRACKING-REC (NUEVO-ESTADO)

 4.2.2 Si RESULTADO no es FALLO,
 devolver RESULTADO y terminar

5 Devolver FALLO

Backtracking en el coloreado de mapas

n Orden de análisis



Heurísticas en el algoritmo de *backtracking*

- En principio, *Backtracking* realiza una búsqueda ciega
 - Búsqueda no informada, ineficiente en la práctica
- Sin embargo, es posible dotar al algoritmo de cierta heurística que mejora considerablemente su rendimiento
 - Estas heurísticas son de propósito general
 - Independientes del problema
 - Sacan partido de la estructura especial de los PSR
- Posibles mejoras heurísticas:
 - Selección de nueva variable a asignar
 - Orden de asignación de valores a la variable elegida
 - Propagación de información a través de las restricciones
 - Vuelta atrás “inteligente” en caso de fallo

Selección de la siguiente variable a asignar

- Incorporar heurística en la definición de **SELECCIONA-VARIABLE (ESTADO)**
- Heurística MRV:
 - Seleccionar la variable con el menor número de valores en su dominio consistentes con la asignación parcial actual
 - Ejemplo: dado el estado (la asignación parcial)
{Huelva=Rojo, Sevilla=Azul, Málaga=Rojo}
la siguiente variable a asignar sería **Cádiz** o **Córdoba** (sólo queda un valor consistente, para **Granada** quedan dos y para **Almería** quedan los tres)
- Otra heurística: variable que aparece en el mayor número de restricciones (*grado*)
 - Usada para desempatar MRV
 - Ejemplo: en el caso anterior elegiríamos **Córdoba**

Ordenación de valores de la variable seleccionada

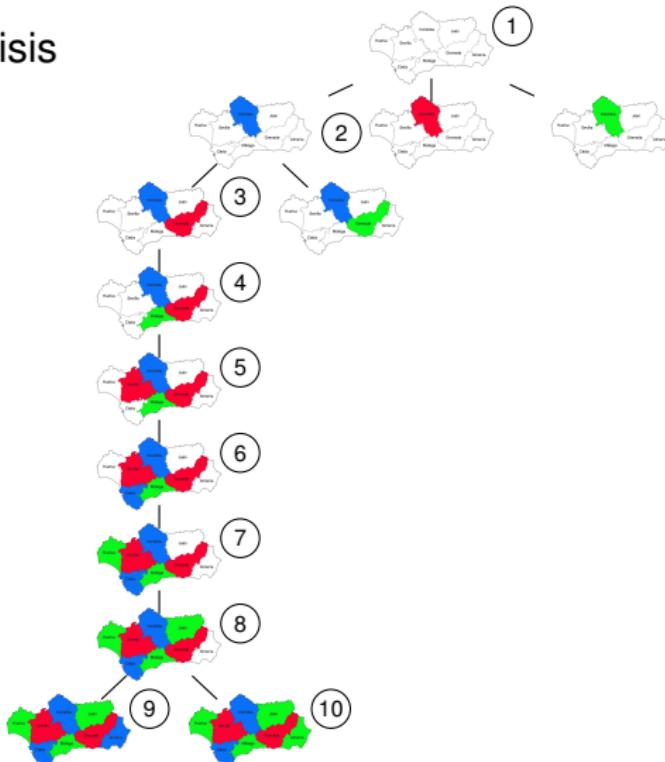
- Definición adecuada para la función

ORDENA-VALORES (DOMINIO)

- Considerar antes los que eliminan menos posibles valores de las variables por asignar

Backtracking con heurísticas en el coloreado de mapas

① Orden de análisis



Comprobación hacia adelante (*forward checking*)

- Para cada estado, mantener información sobre los posibles valores para las variables que quedan por asignar:
 - Cada vez que se asigna un nuevo valor a una variable, quitar del dominio de las variables por asignar, aquellos valores que no sean consistentes con el nuevo valor asignado
- Cuestiones técnicas:
 - Cada nodo del árbol debe contener el estado junto la lista de valores posibles en las variables por asignar
 - Muy fácil de usar en conjunción con la heurística MRV

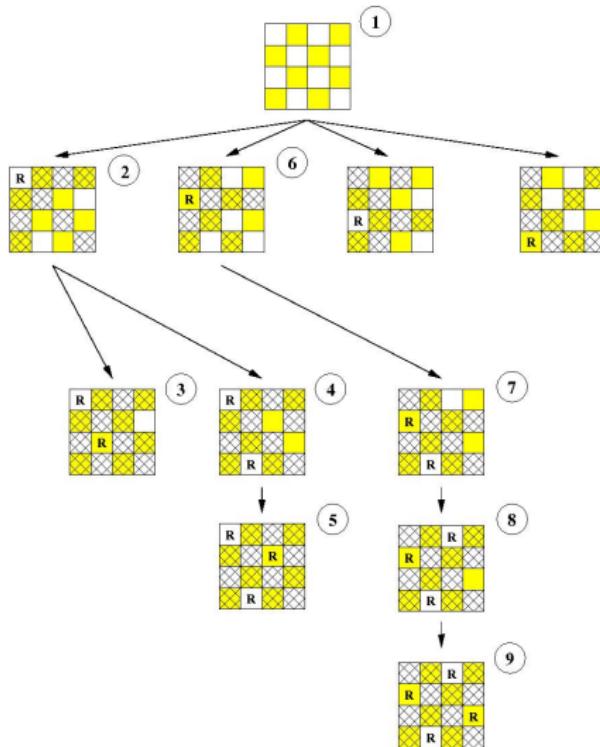
Forward checking en el coloreado de mapas

- Ejemplo (seleccionando las variables en orden alfabético):

| Estado | | Dominios |
|--------------------------|---|--|
| {} | → | <u>AL</u> <u>CA</u> <u>CO</u> <u>GR</u> <u>HU</u> <u>JA</u> <u>MA</u> <u>SE</u> █ █ █ █ █ █ █ █ █ |
| {AL=r} | → | <u>CA</u> <u>CO</u> <u>GR</u> <u>HU</u> <u>JA</u> <u>MA</u> <u>SE</u> █ █ █ █ █ █ █ █ █ |
| {AL=r, CA=r} | → | <u>CO</u> <u>GR</u> <u>HU</u> <u>JA</u> <u>MA</u> <u>SE</u> █ █ █ █ █ █ █ █ █ |
| {AL=r, CA=r, CO=a} | → | <u>GR</u> <u>HU</u> <u>JA</u> <u>MA</u> <u>SE</u> █ █ █ █ █ █ █ █ █ |
| {AL=r, CA=r, CO=a, GR=v} | → | <u>HU</u> <u>JA</u> <u>MA</u> <u>SE</u> █ █ █ █ █ █ █ █ █ |

- Málaga** no tiene ningún valor posible, no es necesario seguir explorando esa rama del árbol
- Se ha detectado inconsistencia sin asignar valores a **Huelva** y **Jaén**, para cualquier extensión del estado
- No detecta inconsistencias anteriores. **Málaga ≠ Sevilla**

Backtracking y forward checking en 4-reinas



Propagación de restricciones

- La comprobación hacia adelante es un caso particular de *propagación de restricciones*:
 - Propagar las implicaciones de las restricciones sobre una variable sobre otras variables
 - Existen técnicas más completas para propagar restricciones que la comprobación hacia adelante
 - La propagación debe ser rápida: completitud vs rapidez
- La *consistencia de arcos* proporciona un método con un buen compromiso entre eficiencia y completitud

Consistencia de arcos

- En un PSR, por *arco* entendemos un *arco dirigido* en el grafo que lo representa
 - Equivalentemente, un arco es una restricción en la que hay una variable distinguida
 - Notación: $B > E$, $\underline{CO} \neq SE$, $|V_i - V_j| \neq |i - j|$
 - Arco *consistente* respecto a un conjunto de dominios asociados a un conjunto de variables:
 - Para cualquier valor del dominio asociado a la variable distinguida del arco, existen valores en los dominios de las restantes variables que satisfacen la restricción del arco
 - Ejemplo: si SE y CO tienen ambas asignadas como dominio $\{\text{rojo}, \text{verde}\}$, el arco $\underline{CO} \neq SE$ es consistente
 - Ejemplo: si el dominio de SE es $\{\text{rojo}\}$ y el de CO es $\{\text{rojo}, \text{verde}\}$, el arco $\underline{CO} \neq SE$ no es consistente.
- Nota:** el arco puede hacerse consistente eliminando **rojo** del dominio de **CO**

Idea intuitiva de AC-3 (consistencia de arcos)

- Objetivo:
 - Devolver un conjunto de dominios actualizado tal que todos los arcos del problema sean consistentes
- Actualización de dominios:
 - Si un arco es inconsistente, podemos hacerlo consistente eliminando del dominio de la variable distinguida aquellos valores para los que no existen valores en los dominios de las restantes variables que satisfagan la restricción

Idea intuitiva de AC-3 (consistencia de arcos)

- Revisión de arcos:
 - Si el dominio de una variable se actualiza, es necesario revisar la consistencia de los arcos en los que aparece la variable como variable no distinguida
- Criterio de parada:
 - Todos los arcos son consistentes respecto a los dominios de las variables
 - O algún dominio queda vacío (inconsistencia)

AC-3 en el coloreado de mapas

Estado: $\{AL=r, CA=r, CO=a\}$, Dominios: $\begin{array}{ccccc} GR & HU & JA & MA & SE \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$

Arcos

$\underline{GR} \neq \underline{JA}, \underline{GR} \neq \underline{MA}, \underline{GR} \neq \underline{MA}$

$\underline{GR} \neq \underline{MA}, \underline{HU} \neq \underline{SE}$

$\underline{HU} \neq \underline{SE}, \underline{MA} \neq \underline{SE}$

Dominios

$\begin{array}{ccccc} GR & HU & JA & MA & SE \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$

$\begin{array}{ccccc} GR & HU & JA & MA & SE \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$

$\begin{array}{ccccc} GR & HU & JA & MA & SE \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$

$\begin{array}{ccccc} GR & HU & JA & MA & SE \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$

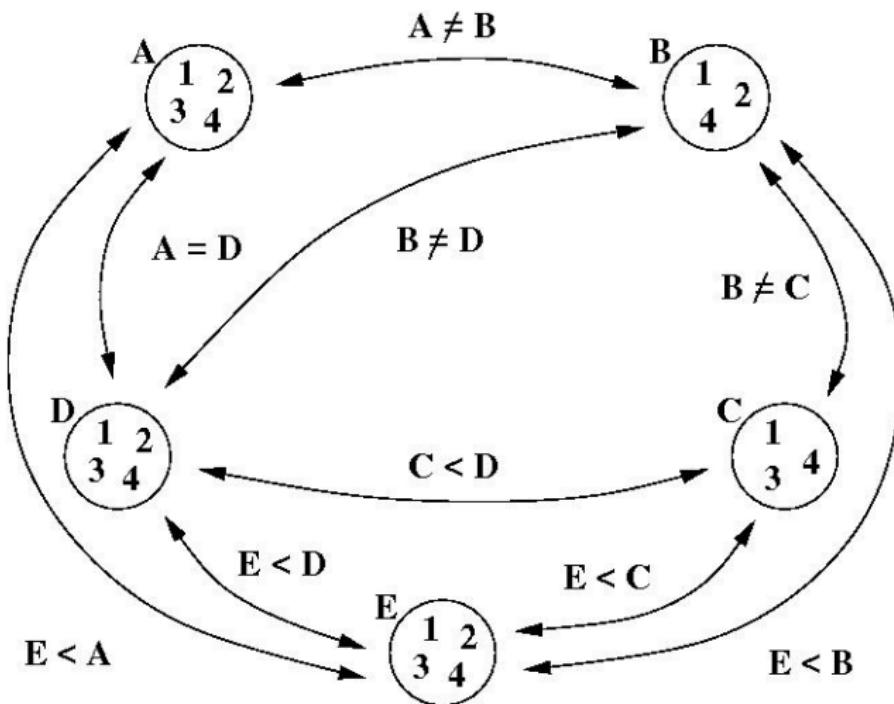
- **Málaga** no tiene ningún valor posible, no es necesario seguir explorando esa rama del árbol
- Se ha detectado inconsistencia para cualquier extensión del estado

Otro ejemplo de AC-3 (I)

- Planificación de las acciones de un robot (Poole, pag. 149): Un robot necesita planificar cinco actividades (A, B, C, D y E), donde cada actividad ha de comenzar en un momento en el tiempo (1, 2, 3, ó 4) y dura exactamente una unidad de tiempo. Restricciones:
 - La actividad B no puede realizarse en el momento 3.
 - La actividad C no puede realizarse en el momento 2.
 - Las actividades A y B no pueden realizarse simultáneamente.
 - Las actividades B y C no pueden realizarse simultáneamente.
 - La actividad C ha de realizarse antes de la D.
 - Las actividades B y D no pueden realizarse simultáneamente.
 - Las actividades A y D han de realizarse simultáneamente.
 - La actividad E ha de ser la primera.

Otro ejemplo de AC-3 (II)

- Representación como grafo:



Otro ejemplo de AC-3 (III)

| Variables y dominios | Arcos pendientes de revisar |
|--|--|
| (A 1 2 3 4) (B 1 2 4) (C 1 3 4) (D 1 2 3 4) (E 1 2 3 4) | $\underline{A} \neq \underline{B}, \underline{A} \neq \underline{B}, \underline{B} \neq \underline{C}, \underline{B} \neq \underline{C}, \underline{C} < \underline{D}, \underline{C} < \underline{D}, \underline{B} \neq \underline{D}, \underline{B} \neq \underline{D},$ $\underline{A} = \underline{D}, \underline{A} = \underline{D}, \underline{A} > \underline{E}, \underline{A} > \underline{E}, \underline{B} > \underline{E}, \underline{B} > \underline{E}, \underline{C} > \underline{E}, \underline{C} > \underline{E},$ $\underline{D} > \underline{E}, \underline{D} > \underline{E}$ |
| (A 1 2 3 4) (B 1 2 4) (C 1 3) (D 1 2 3 4) (E 1 2 3 4) | $\underline{C} < \underline{D}, \underline{B} \neq \underline{D}, \underline{B} \neq \underline{D}, \underline{A} = \underline{D}, \underline{A} = \underline{D}, \underline{A} > \underline{E}, \underline{A} > \underline{E}, \underline{B} > \underline{E},$ $\underline{B} > \underline{E}, \underline{C} > \underline{E}, \underline{D} > \underline{E}, \underline{D} > \underline{E}, \underline{B} \neq \underline{C}$ |
| (A 1 2 3 4) (B 1 2 4) (C 1 3) (D 2 3 4) (E 1 2 3 4) | $\underline{B} \neq \underline{D}, \underline{B} \neq \underline{D}, \underline{A} = \underline{D}, \underline{A} = \underline{D}, \underline{A} > \underline{E}, \underline{A} > \underline{E}, \underline{B} > \underline{E}, \underline{B} > \underline{E},$ $\underline{C} > \underline{E}, \underline{C} > \underline{E}, \underline{D} > \underline{E}, \underline{D} > \underline{E}, \underline{B} \neq \underline{C}$ |
| (A 2 3 4) (B 1 2 4) (C 1 3) (D 2 3 4) (E 1 2 3 4) | $\underline{A} = \underline{D}, \underline{A} > \underline{E}, \underline{A} > \underline{E}, \underline{B} > \underline{E}, \underline{B} > \underline{E}, \underline{C} > \underline{E}, \underline{C} > \underline{E}, \underline{D} > \underline{E},$ $\underline{D} > \underline{E}, \underline{B} \neq \underline{C}, \underline{A} \neq \underline{B}$ |
| (A 2 3 4) (B 1 2 4) (C 1 3) (D 2 3 4) (E 1 2 3) | $\underline{B} > \underline{E}, \underline{B} > \underline{E}, \underline{C} > \underline{E}, \underline{C} > \underline{E}, \underline{D} > \underline{E}, \underline{D} > \underline{E}, \underline{B} \neq \underline{C}, \underline{A} \neq \underline{B}$ |
| (A 2 3 4) (B 2 4) (C 1 3) (D 2 3 4) (E 1 2 3) | $\underline{B} > \underline{E}, \underline{C} > \underline{E}, \underline{C} > \underline{E}, \underline{D} > \underline{E}, \underline{D} > \underline{E}, \underline{B} \neq \underline{C}, \underline{A} \neq \underline{B}, \underline{A} \neq \underline{B},$ $\underline{B} \neq \underline{C}, \underline{B} \neq \underline{D}$ |
| (A 2 3 4) (B 2 4) (C 3) (D 2 3 4) (E 1 2 3) | $\underline{C} > \underline{E}, \underline{D} > \underline{E}, \underline{D} > \underline{E}, \underline{B} \neq \underline{C}, \underline{A} \neq \underline{B}, \underline{A} \neq \underline{B}, \underline{B} \neq \underline{C}, \underline{B} \neq \underline{D}$ $\underline{C} < \underline{D}$ |

Otro ejemplo de AC-3 (IV)

| Variables y dominios | Arcos pendientes de revisar |
|--|---|
| (A 2 3 4) (B 2 4) (C 3) (D 2 3 4) (E 1 2) | $D > E, D > \underline{E}, \underline{B} \neq C, A \neq \underline{B}, \underline{A} \neq B, B \neq \underline{C}, B \neq \underline{D}, \textcolor{red}{C} < \underline{D},$ $\underline{A} > E, \underline{B} > \underline{E}$ |
| (A 2 3 4) (B 2 4) (C 3) (D 4) (E 1 2) | $\underline{A} > E, \underline{B} > E, \textcolor{red}{B} \neq D, \underline{A} = D, D > \underline{E}$ |
| (A 2 3 4) (B 2) (C 3) (D 4) (E 1 2) | $\textcolor{red}{A} = D, D > \underline{E}, \underline{A} \neq B, B \neq \underline{C}, B > \underline{E}$ |
| (A 4) (B 2) (C 3) (D 4) (E 1 2) | $D > \underline{E}, \underline{A} \neq B, B \neq \underline{C}, \textcolor{red}{B} > \underline{E}, A \neq \underline{B}, A > \underline{E}$ |
| (A 4) (B 2) (C 3) (D 4) (E 1) | $\textcolor{red}{A} \neq \underline{B}, A > \underline{E}, \underline{A} > E, \underline{C} > \underline{E}, \underline{D} > E$ |
| (A 4) (B 2) (C 3) (D 4) (E 1) | |

Implementación del algoritmo AC-3

- Arco: Pareja de variables que aparece en alguna restricción. La primera variable del par es la variable distinguida.
- Generación de todos los arcos de un PSR:
 $\{ (X, Y) : X \in \text{variables}, Y \in \text{vecinos}[X] \}$
- Cálculo de dominio actualizado:

```
FUNCTION ACTUALIZA-DOMINIO(X, Y, DOMINIOS)
1 Hacer MODIFICADO igual a FALSO
2 Para cada VALOR en DOMINIOS[X] hacer
    2.1 Si no existe un valor en DOMINIOS[Y]
        que satisfaga la restricción entre X e Y
        2.1.1 eliminar VALOR de DOMINIOS[X]
        2.1.2 Hacer MODIFICADO igual a VERDADERO
3 Devolver MODIFICADO
```

Implementación del algoritmo AC-3

```
FUNCTION AC-3(DOMINIOS)
1 Hacer COLA igual a {(X, Y): X ∈ variables, Y ∈ vecinos[X]}
2 Mientras COLA no sea vacío
    2.1 Hacer (X, Y) igual el primer elemento de COLA y
        COLA el resto de COLA
    2.2 Si ACTUALIZA-DOMINIO(X, Y, DOMINIOS) entonces
        incluir en COLA todos los arcos (Z, X) para cada Z de
            vecinos[X] - {Y}
```

- Entrada: una asociación de las variables del problema a conjuntos de posibles valores para cada una de ellas.
- Al finalizar, los conjuntos de valores actualizados (destructivamente) son consistentes para el PSR.

Propiedades del algoritmo AC-3

- Complejidad en tiempo $O(n^2d^3)$ (en el caso de restricciones binarias), donde:
 - n : número de variables
 - d : máximo número de valores de un dominio
- La complejidad polinomial nos indica que sólo con AC-3 no es posible resolver un PSR
 - Pero puede combinarse con algún tipo de búsqueda
- Dos aproximaciones:
 - Incorporar en el algoritmo de *backtracking*, como un paso de *propagación de restricciones* después de cada asignación
 - Intercalar AC-3 con búsqueda

Búsqueda de soluciones mediante consistencia de arcos

- Posibles resultados tras aplicar AC-3:
 - Existe un dominio vacío: no hay solución.
 - Todos los dominios son unitarios: hay una solución.
 - No hay dominios vacíos y al menos uno no es unitario: ¿soluciones?
- Idea: romper un dominio no vacío en subdominios y aplicar AC-3 a los estados resultantes
 - Esto se puede hacer de manera sistemática (búsqueda)
 - Hasta que se obtiene un estado con todos los dominios unitarios (solución)
 - O bien hasta que se detecta inconsistencia en todas las alternativas (no hay solución)

Búsqueda de soluciones mediante consistencia de arcos

FUNCTION BÚSQUEDA-AC3()

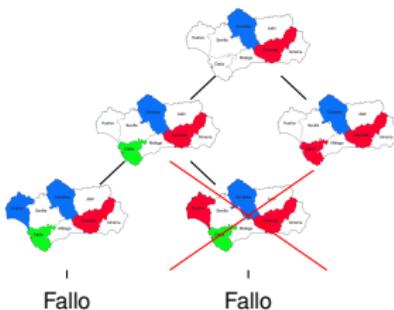
- 1 Hacer ABIERTOS igual a una lista cuyo único elemento, una copia de **dominios**
- 2 Mientras que ABIERTOS no esté vacía,
 - 2.1 Hacer ACTUAL igual al primer elemento de ABIERTOS
 - 2.2 Hacer ABIERTOS igual al resto de ABIERTOS
 - 2.3 AC-3(ACTUAL)
 - 2.3 Si no existen dominios vacíos en ACTUAL,
 - 2.3.1 Si ACTUAL tiene todos los dominios unitarios, devolver ACTUAL y terminar.
 - 2.3.2 en caso contrario, calcular los sucesores de ACTUAL y añadirlos a la lista de ABIERTOS
- 3 Devolver FALLO.

- Detalles a concretar de la implementación:
 - Sucesores: escoger un dominio no unitario y dividirlo de alguna manera (se puede incorporar heurística para la división y para el orden de sucesores)
 - Gestión de la cola de abiertos: anchura o profundidad

Backjumping vs Backtracking

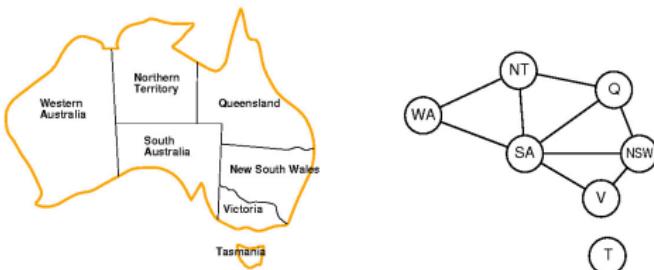
- En caso de fallo, usualmente se prueba un valor distinto para la variable anterior
- Saltar a una vecina de la variable que produce el fallo
 - la última que se ha modificado

Orden de las variables: co, GR, CA, HU, MA, SE, JA, AL



Estudio de la estructura del grafo

Consideremos el problema del coloreado de mapas en el caso de Australia.



Tiene dos subproblemas: **Tasmania** y el **contiente**:

- Cada componente conexa es un problema independiente
Si cada componente tiene c variables: $\frac{n}{c} d^c$ hojas

$$n = 80, d = 2, c = 20$$

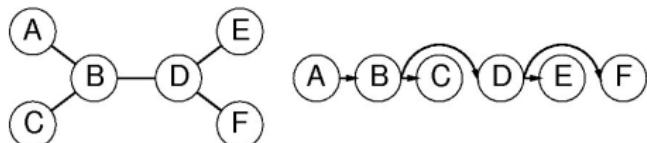
$2^{80} = 4$ billones (am.) de años a 10 millones de nodos/seg

$4 \cdot 2^{20} = 0.4$ segundos a 10 millones de nodos/seg

Árboles (I)

Teorema: si el grafo no tiene ciclos la complejidad del PSR en tiempo es $O(nd^2)$

- Elegir una variable como raíz, ordenar las variables desde la raíz a las hojas: cada padre de cada nodo aparece antes en el orden



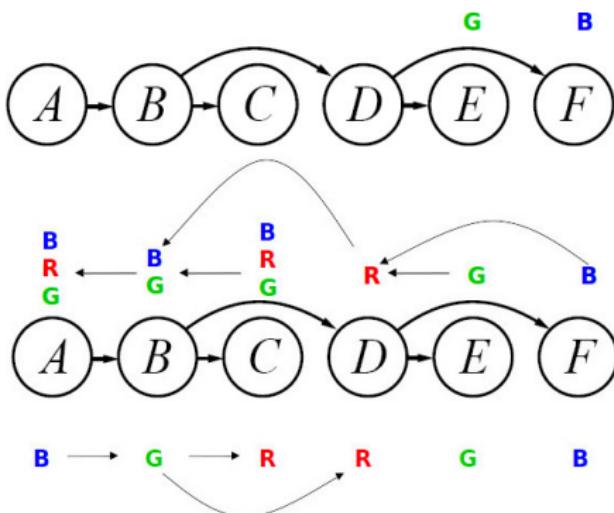
- Para j desde n hasta 2, aplicar
ACTUALIZA-DOMINIO (Padre (X_j), x_j)
- Para j desde 1 hasta n , asignar un valor a X_j consistente con $\text{Padre} (X_j)$

Árboles (II)

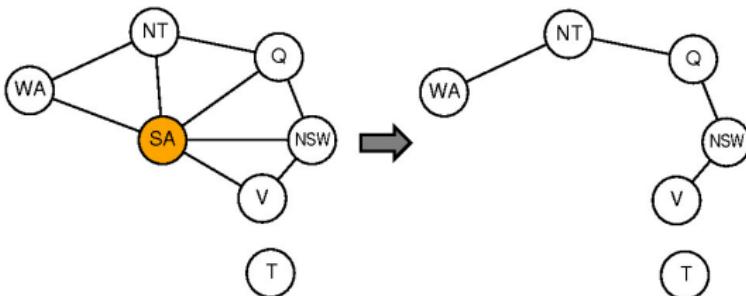
Estado inicial

2. Propagación de restricciones

3. Asignación



Casi árboles



- Instanciar todas las variables (de todas las formas posibles) tales que los grafos resultantes (al eliminarlas) son árboles
- Si el número de variables eliminadas es c la complejidad en tiempo es $O(d^c \cdot (n - c)d^2)$ (bastante rápido si c es pequeño)

Algoritmos de mejora iterativa para PSRs

- Planteamiento como un problema de búsqueda local:
 - Estados: asignaciones completas (consistentes o inconsistentes)
 - Estado inicial escogido aleatoriamente
 - Estados finales: soluciones al PSR
 - Generación de sucesor: elegir una variable y cambiar el valor que tiene asignado (usando cierta heurística y aleatoriedad)
- Escalada, Escalada con reinicio aleatorio, Enfriamiento simulado

Generación de sucesor mediante la heurística de *mínimos conflictos*

- Variable elegida para cambiar su valor:
 - Aquella (distinta de la última modificada) que participa en más restricciones NO satisfechas en el estado (los empates se deshacen aleatoriamente)
- Nuevo valor elegido:
 - El valor (distinto del que tenía) que produce el menor número de restricciones incumplidas
 - Los empates se deshacen aleatoriamente

Implementación de reparación heurística con mínimos conflictos

FUNCTION MIN-CONFLICTOS (MAX-ITERACIONES)

- 1 Hacer ACTUAL igual a una asignación generada aleatoriamente y ÚLTIMA igual a vacío
- 2 Para I desde 1 hasta MAX-ITERACIONES hacer
 - 2.1 Si ACTUAL es una solución, devolverla y terminar.
 - 2.2 Hacer VARIABLE igual a una variable (distinta de ÚLTIMA) escogida aleatoriamente de entre aquellas que participan en el mayor número de restricciones incumplidas.
 - 2.3 Hacer SUCESORES igual a la lista de asignaciones obtenidas cambiando en ACTUAL el valor de VARIABLE por cada uno de los posibles valores de dominios[VARIABLE] (excepto el correspondiente al valor que tenía en ACTUAL)
 - 2.4 Hacer ACTUAL la asignación de SUCESORES escogida aleatoriamente de entre aquellas que incumplen el menor número de restricciones
- 3 Devolver FALLO

Reparación heurística: ejemplo con N-reinas

| V1 | | V2 | | V3 | | V4 | |
|-------|------------|-------|------------|-------|------------|-------|------------|
| valor | conflictos | valor | conflictos | valor | conflictos | valor | conflictos |
| 1 | 3 | 1 | 3 | 1 | 3 | 1 | 3 |
| 1 | 2 | 1 | 2 | 4 | 0 | 1 | 2 |
| 1 | 2 | 2 | 1 | 4 | 0 | 1 | 1 |
| 3 | 1 | 2 | 1 | 4 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 4 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 4 | 0 | 2 | 0 |

Propiedades del algoritmo de reparación heurística

- No es completo
- Complejidad en tiempo: lineal en el número de iteraciones
- Aleatoriedad: en la asignación inicial y al deshacer empates
- Reparación heurística tiene muy buenos resultados en algunos tipos de PSRs:
 - PSRs cuyas soluciones están distribuidas uniformemente a lo largo del espacio de estados
 - PSRs en los que las restricciones pueden cambiar dinámicamente

Comparativa de los distintos algoritmos

| Problem | Backtracking | BT+MRV | Forward Checking | FC+MRV | Min-Conflicts |
|------------------|--------------|------------|------------------|--------|---------------|
| USA | (> 1,000K) | (> 1,000K) | 2K | 60 | 64 |
| <i>n</i> -Queens | (> 40,000K) | 13,500K | (> 40,000K) | 817K | 4K |
| Zebra | 3,859K | 1K | 35K | 0.5K | 2K |
| Random 1 | 415K | 3K | 26K | 2K | |
| Random 2 | 942K | 27K | 77K | 15K | |

Número medio de comprobaciones de consistencia en 5 ejecuciones

- Entre paréntesis: detenido sin encontrar solución
- Mapa de USA con cuatro colores
- *n*-Queens con $n = 50$

Problemas de Optimización de Restricciones

- Se añade una función coste u objetivo basada en la satisfacción de las restricciones
- Además de encontrar una asignación que cumpla las restricciones se intenta optimizar el valor de la función objetivo (maximizar o minimizar)
 - Se añaden costes a distintas asignaciones parciales. La función objetivo es la suma de las funciones coste
 - Los problemas de satisfacción de restricciones pueden abordarse como problemas de optimización de restricciones: encontrar una asignación que satisfaga el mayor número de restricciones

Ejemplo: Problema de la mochila

- Dados n objetos, cada uno con un peso w_i y un beneficio p_i asociado; y una mochila de capacidad C . Colocar dentro de la mochila aquellos elementos que satisfacen que la suma de sus beneficios es máxima y el peso total de los elementos no sobrepasa la capacidad de la mochila.
- Variables: x_1, \dots, x_n
- Dominios: $D_i = \{0, 1\}$
- Restricción: $\sum_{i=1}^n x_i * w_i \leq C$
- Objetivo: $\sum_{i=1}^n x_i * p_i$ (maximizar)

Primera aproximación

- Busca distintas soluciones al PSR contenido en el POR (por ejemplo usando backtracking + forward checking)
- Devolver la mejor solución encontrada
 - Si en la búsqueda de una nueva solución se detecta que su coste no mejorará la mejor solución encontrada hasta el momento: iniciar la búsqueda de una nueva solución al PSR
 - Durante la búsqueda de una solución al PSR: Calcular el coste de la asignación parcial construida hasta el momento y estimar el mejor coste de cualquier extensión de la misma.

Algoritmos de mejora iterativa

- Además de intentar satisfacer las restricciones se busca mejorar el coste.
 - Escalada
 - Escalada con reinicio aleatorio
 - Enfriamiento simulado
 - Algoritmos genéticos
 - ...

Bibliografía

- Russell, S. y Norvig, P. *Inteligencia Artificial (un enfoque moderno)* (Pearson Educación, 2004). Segunda edición
 - Cap. 5: “Problemas de Satisfacción de Restricciones”
- Russell, S. y Norvig, P. *Artificial Intelligence (A Modern Approach)* (Prentice–Hall, 2010). Third Edition
 - Cap. 6: “Constraint Satisfaction Problems”
- Nilsson, N.J. *Inteligencia artificial (una nueva síntesis)* (McGraw–Hill, 2000)
 - Cap. 11 “Métodos alternativos de búsqueda y otras aplicaciones”
- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach)* (Oxford University Press, 1998)
 - Cap. 4.7: “Constraint Satisfaction Problems”