Universidad
Francisco de Vitoria
UFV Madrid

Ingeniería del Conocimiento

Tema 6:
Búsqueda con Restricciones

Roberto Rodriguez Galán
,--

Departamento de Ingeniería Informática
Universidad Francisco de Vitoria

Objetivos del tema

- Ubicación
  - Unidad 2: **BUSQUEDA EN ESPACIO DE ESTADOS**
  - *Tema 6: Búsqueda con Restricciones*

- Objetivos generales
  - Entender los Problemas de Satisfacción de Restricciones (PSR o CSP en inglés) cuyos estados y test objetivo forman una representación simple, estándar y estructurada.
  - Definir la estructura de los estados y utilizar heurísticas de propósito general permitiendo extrapolarse a problemas grandes.
  - Conocer la estructura del problema mediante la representación estándar del test objetivo.
  - Utilizar métodos de descomposición de problemas y entender la conexión entre la estructura de un problema y la dificultad de resolverlo.

Contenido

1. Problemas de Satisfacción de Restricciones.

2. Búsqueda con vuelta atrás
   1. Variables y orden
   2. Implicaciones variables actuales
   3. Evitar repetir fracasos

3. Métodos de Búsqueda local

4. La estructura de los problemas

1. Visión General de los PSR

- Hasta ahora, el algoritmo de búsqueda ve cada estado como una caja negra (estructura de datos arbitraria accesible solo por función sucesor, función heurística y test objetivo).
  - Vamos a estudiar los Problemas de Satisfacción de Restricciones (PSR o CSP en inglés) cuyos estados y test objetivo forman una representación simple, estándar y estructurada.
  - Se definen aprovechándose de la estructura de los estados y utilizan heurísticas de propósito general (no específicas) permitiendo extrapolarse a problemas grandes.
  - La representación estándar del test objetivo revela la estructura del problema.
  - Se utilizan métodos de descomposición de problemas y una conexión entre la estructura de un problema y la dificultad de resolverlo.

Contenido

1. **Problemas de Satisfacción de Restricciones.**

2. Búsqueda con vuelta atrás
   1. Variables y orden
   2. Implicaciones variables actuales
   3. Evitar repetir fracasos

3. Métodos de Búsqueda local

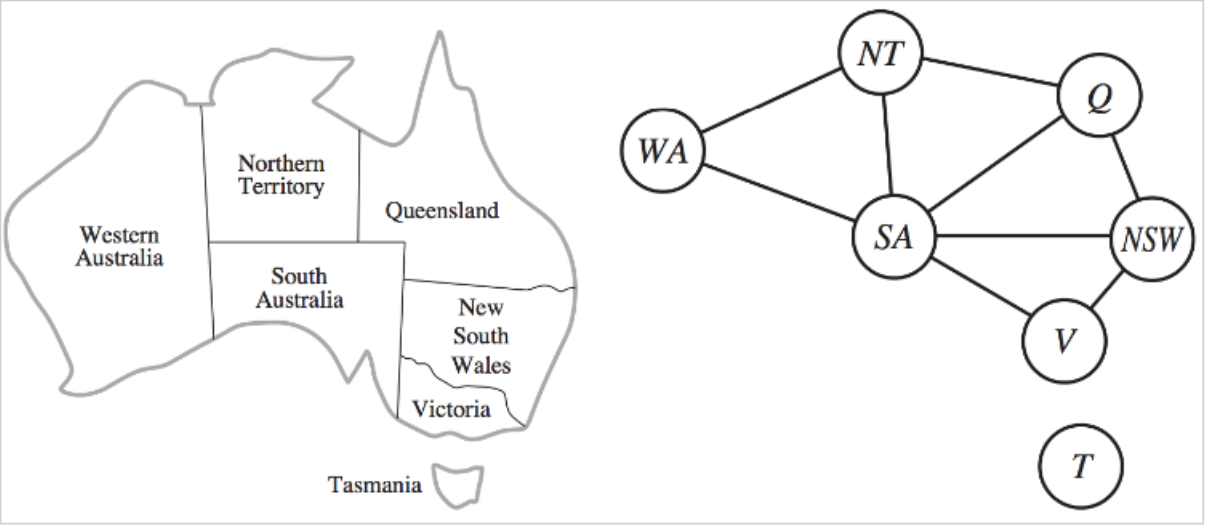
4. La estructura de los problemas

1. Problemas de satisfacción de restricciones

- La idea es **representar problemas mediante la declaración de restricciones** sobre el área del problema (el espacio de posibles soluciones) **y encontrar soluciones que satisfagan todas las restricciones**. A veces, se buscan soluciones que, además, optimicen algunos criterios determinados.
- La resolución de problemas con restricciones puede dividirse en dos ramas diferenciadas, donde se aplican metodologías distintas: aquella que trata con problemas con **dominios finitos**, y la que trata problemas sobre **dominios infinitos** o dominios más complejos.
- Nos centraremos en **problemas con dominios finitos**, que básicamente consisten en un **conjunto finito de variables**, un **dominio de valores finito para cada variable** y un **conjunto de restricciones que acotan** las posibles combinaciones de valores que estas variables pueden tomar en su dominio.

1. Problemas de satisfacción de restricciones

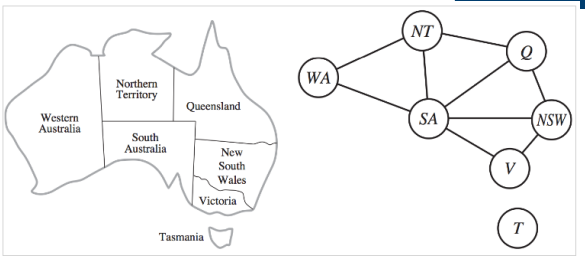
- Antes de seguir, un ejemplo: **problema de coloración de mapas**.



Los estados y territorios principales de Australia. Colorear este mapa puede verse como un problema de satisfacción de restricciones. El objetivo es asignar colores a cada región de modo que ninguna de las regiones vecinas tengan el mismo color.

El problema del coloreo del mapa representado como un grafo de restricciones.

1. Problemas de satisfacción de restricciones



Suponga que, cansados de Rumanía, miramos un **mapa de Australia** que muestra cada uno de sus estados y territorios y que nos encargan la tarea de **colorear cada región de rojo, verde o azul** de modo que **ninguna de las regiones vecinas tenga el mismo color**.

Para formularlo como un PSR, **definimos las variables de las regiones**: *AO, TN, Q, NGS, V, AS* y *T*. El **dominio de cada variable** es el conjunto {*rojo, verde, azul*}. Las **restricciones** requieren que las regiones vecinas tengan colores distintos; por ejemplo, las combinaciones aceptables para *AO* y *TN* son los pares: {(*rojo,verde*}, (*rojo,azul*), (*verde, rojo*), (*verde,azul*), (*azul,rojo*), (*azul,verde*)}

Hay **muchas soluciones posibles**, como {*AO = rojo, TN = verde, Q = rojo, NGS = verde, V = rojo, AS = azul, T = rojo*}

Es bueno visualizar un PSR como un **grafo de restricciones**, Los nodos del grafo corresponden a variables del problema y los arcos corresponden a restricciones.

1. Problemas de satisfacción de restricciones

Tratar un problema como un PSR confiere varias ventajas importantes:

1. Como la **representación del estado** en un PSR se ajusta a un **modelo estándar** (es decir, un conjunto de variables con valores asignados) la **función sucesor y el test objetivo** pueden escribirse de un **modo genérico** para que se aplique a todo PSR.
2. Además, podemos desarrollar **heurísticas eficaces y genéricas** que no requieran ninguna información adicional ni experta del dominio específico.
3. Finalmente, la **estructura del grafo de las restricciones** puede usarse para **simplificar el proceso de solución**, en algunos casos produciendo una reducción exponencial de la complejidad.

1. Problemas de satisfacción de restricciones

A un PSR se le puede dar una **formulación incremental** como en un problema de búsqueda estándar:

- **Estado inicial:** la asignación vacía {}, en la que todas las variables no están asignadas.
- **Función de sucesor:** un valor se puede asignar a cualquier variable no asignada, a condición de que no suponga ningún conflicto con variables antes asignadas.
- **Test objetivo:** la asignación actual es completa.
- **Costo del camino:** un coste constante (por ejemplo, 1) para cada paso.

Cada solución debe ser una asignación completa y por lo tanto aparecen a profundidad *n* si hay *n* variables (el árbol de búsqueda se extiende sólo a profundidad *n)*. Por estos motivos, los algoritmos de búsqueda **primero en profundidad** son los más utilizados para solucionar PSRs.

También ***el camino que alcanza una solución es irrelevante***. De ahí, que podemos usar también una **formulación completa de estados**, en la cual cada estado es una asignación completa que podría o no satisfacer las restricciones; por ello, los métodos de búsqueda local trabajan bien para esta formulación.

Hay $n!d^n$ hojas!

1. Problemas de satisfacción de restricciones

La clase más simple de PSR implica variables **discretas** y **dominios finitos** (colorear mapas). Otros ejemplos son las 8 reinas.

Los PSR con dominio finito incluyen a los **PSRs booleanos**, cuyas variables pueden ser verdaderas o falsas. Los PSRs booleanos incluyen como casos especiales algunos problemas NP-completos, como 3SAT.

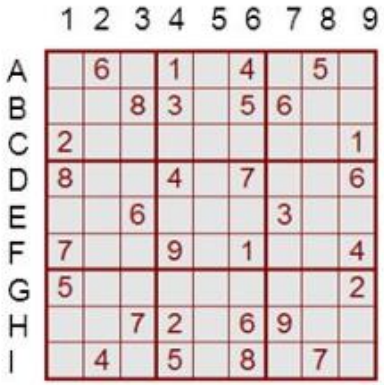
Las **variables discretas pueden tener también dominios infinitos** (por ejemplo, el conjunto de números enteros o de cadenas). Por ejemplo, cuando programamos trabajos de la construcción en un calendario, la fecha de comienzo de cada trabajo es una variable y los valores posibles son números enteros de días desde la fecha actual. Con dominios infinitos, no es posible describir restricciones enumerando todas las combinaciones permitidas de valores. En cambio, se debe utilizar un lenguaje de restricción.

Los problemas de **satisfacción de restricciones con dominios continuos** son muy comunes en el mundo real y son ampliamente estudiados en el campo de la investigación operativa. (la programación de experimentos sobre el Telescopio Hubble requiere el cronometraje muy preciso de las observaciones; al comienzo y al final de cada observación y la maniobra son variables continuas que deben obedecer a una variedad de restricciones)

1. Problemas de satisfacción de restricciones

Ejemplo 1:

Sudoku as a Constraint Satisfaction Problem (CSP)



- Variables: 81 variables
  - A1, A2, A3, …, I7, I8, I9
  - Letters index rows, top to bottom
  - Digits index columns, left to right
- Domains: The nine positive digits
  - A1 ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Etc.; all domains of all variables are {1,2,3,4,5,6,7,8,9}
- Constraints: 27 *Alldiff* constraints
  - *Alldiff*(A1, A2, A3, A4, A5, A6, A7, A8, A9)
  - Etc.; all rows, columns, and blocks contain all different digits

1. Problemas de satisfacción de restricciones

Ejemplo 2:

Consideremos una restricción entre 4 variables x_1,x_2,x_3,x_4, todas ellas con dominios el conjunto {1,2}, donde la suma entre las variables x_1 y x_2 es menor o igual que la suma entre x_3 y x_4.

Esta restricción puede representarse intencionalmente mediante la expresión

$$x_1+x_2 \leq x_3+x_4.$$

Además, esta restricción 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:

$$\{(1,2,1,1),(2,1,1,1),(2,2,1,1),(2,2,1,2),(2,2,2,1)\}$$

1. Problemas de satisfacción de restricciones

Ejemplo 3:

Vamos a ver como ejemplo diversas representaciones de un mismo problema usando la representación PSR: el **problema de la N reinas**, que consiste en disponer en un tablero de ajedrez de tamaño N×N, N reinas de forma que no haya amenazas entre ningún par de ellas.

Observa las **diferencias que se producen en la complejidad de las representaciones,** tanto por el **espacio de combinaciones** en las soluciones como por **las restricciones**.

1. Problemas de satisfacción de restricciones

Ejemplo 3: Primera Representación

1. $X=\{R_1,\ldots,R_N\}=\{R_i: 1\leq i\leq N\}$
2. $D=\{(x,y): 1\leq x,y\leq N\}$, donde $R_i=(x_i,y_i)$
3. Las restricciones:
   - No hay amenaza horizontal: $x_i\neq x_j$, para todo $i\neq j$.
   - No hay amenaza vertical: $y_i\neq y_j$, para todo $i\neq j$.
   - No hay amenaza en la diagonal principal: $x_i-x_j\neq y_i-y_j$, para todo $i\neq j$.
   - No hay amenaza en la diagonal secundaria: $x_i-x_j\neq y_j-y_i$, para todo $i\neq j$.

(Las dos diagonales se pueden tratar conjuntamente con: $|x_i-x_j|\neq|y_i-y_j|$, para todo $i\neq j$)

En esta representación el dominio es finito y hace uso de restricciones binarias de obligación. Para N=8 hay 64^8 posibles asignaciones y 154 restricciones (28 + 28 + 49 + 49)

1. Problemas de satisfacción de restricciones

Ejemplo 3: Segunda Representación

1. $X=\{R_{i,j}: 1\leq x,y\leq N\}\}$ (una por cada casilla).
2. $D=\{0,1\}$
3. Las restricciones:
   - $\sum_i R_{i,j}\leq 1$, para todo $1\leq j\leq N$.
   - $\sum_j R_{i,j}\leq 1$, para todo $1\leq i\leq N$.
   - $\sum_{(i,j)\in D} R_{i,j}\leq 1$, para toda D diagonal.

En esta representación el dominio es finito y hace uso de restricciones de varias aridades. Para N=8 hay 64^2 posibles asignaciones y unas 300 restricciones.

1. Problemas de satisfacción de restricciones

Ejemplo 3: Tercera Representación (igual a la anterior pero con predicados lógicos).

1. $X=\{R_{i,j}: 1\leq x,y\leq N\}\}$ (una por cada casilla).
2. $D=\{0,1\}$
3. Las restricciones:
   - $(R_{i,j}\rightarrow\neg R_{i,j'})$, para todo $j'\neq j$.
   - $(R_{i,j}\rightarrow\neg R_{i',j})$, para todo $i'\neq i$.
   - $(R_{i,j}\rightarrow\neg R_{i+k,j+k})$, para todo $1\leq i+k,j+k\leq N$.
   - $(R_{i,j}\rightarrow\neg R_{i+k,j-k})$, para todo $1\leq i+k,j-k\leq N$.

En esta representación el dominio es finito y hace uso de restricciones de varias aridades. Para N=8 hay 64^2 posibles asignaciones y unas 300 restricciones.

1. Problemas de satisfacción de restricciones

Ejemplo 3: Cuarta Representación.

Haciendo uso de conocimiento implícito del problema (que va a haber una, y solo una, reina en cada columna):

1. $X=\{R_i: 1 \leq i \leq N\}$
2. $D=\{1,\ldots,N\}$
3. Las restricciones:
   - $R_i \neq R_j$, para todo $i \neq j$.
   - $|R_i - R_j| \neq |i-j|$, para todo $i \neq j$.

En esta representación el dominio es finito y hace uso de restricciones binarias. Para N=8 hay 8^8 posibles asignaciones y unas 80 restricciones.

LA REPRESENTACIÓN IMPORTA EN UN PSR!!!!

Contenido

1. Problemas de Satisfacción de Restricciones.

2. **Búsqueda con vuelta atrás**
   1. Variables y orden
   2. Implicaciones variables actuales
   3. Evitar repetir fracasos

3. Métodos de Búsqueda local

4. La estructura de los problemas

2. Búsqueda con vuelta atrás

Hemos formulado un PSR como un problema de búsqueda…

Supongamos que aplicamos la **búsqueda primero en anchura**… el factor de ramificación en la raíz es nd (cualquiera de los d valores se puede asignar a cualquiera de las n variables); en el siguiente nivel el factor de ramificación es (n-1)d y así para los siguiente niveles. Es decir, generamos un árbol con **$n!d^n$ hojas** aunque haya solo d^n asignaciones posibles…

Propiedad (crucial) a todos los PSRs: **Conmutatividad**. El orden de aplicación de cualquier conjunto de acciones no tiene ningún efecto sobre el resultado.

AS = rojo y AO = verde es lo mismo que AS = verde y AS = rojo.

Con esta restricción el número de hojas es d^n

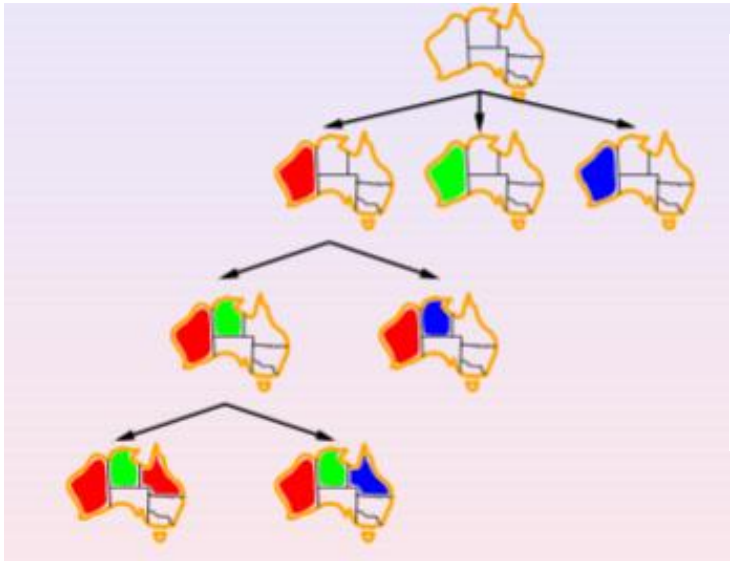
2. Búsqueda con vuelta atrás

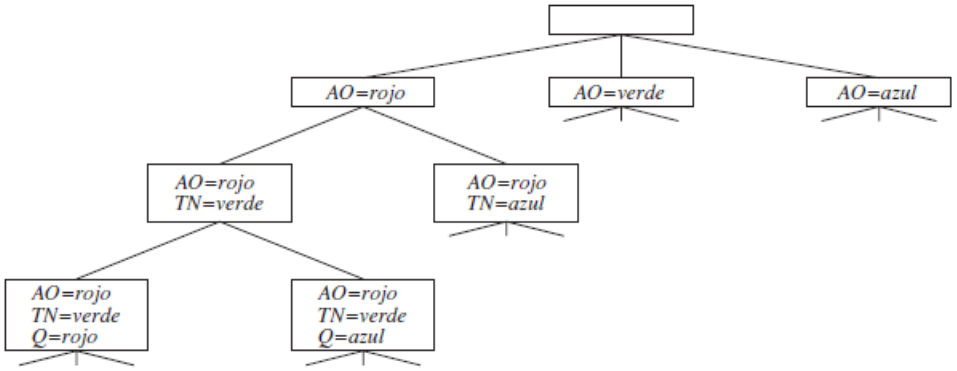
El algoritmo que utilizaremos será una **búsqueda con vuelta atrás**; este término se utiliza para una búsqueda primero en profundidad que elige valores para una variable a la vez y vuelve atrás cuando una variable no tiene ningún valor legal para asignarle.

Se llama a la función recursiva VUELTAATRAS({},PSR)

```
Función VUELTAATRAS(Asignacion,PSR)
1: if Asignacion es completa then return Asignacion
2: Seleccionar una variable no asignada X  //Heurística
3: for Valor posible de X do  //Orden de valores. Heurística
4:    if X=Valor es consistente con las restricciones then
5:          Añadir X=Valor a Asignacion
6:          Sol ← VUELTAATRAS(Asignacion,PSR)
7:          if Sol no es No-Solucion then
8:                return Resultado
9: return No-solucion
```

2. Búsqueda con vuelta atrás





La vuelta atrás es un algoritmo sin información (según la terminología que hemos utilizado) por lo que no esperemos que sea muy eficaz para problemas grandes.

2. Búsqueda con vuelta atrás

Problema	Vuelta atrás	VA + MVR	Comprobación hacia delante	CD + MVR	Mínimo conflicto
EE.UU.	(> 1.000K)	(> 1.000K)	2K	60	64
n-reinas	(> 40.000K)	13.500K	(> 40.000K)	817K	4K
Zebra	3.859K	1K	35K	0,5K	2K
Aleatorio 1	415K	3K	26K	2K	
Aleatorio 2	942K	27K	77K	15K	

En cada celda está el número medio de comprobaciones consistentes (sobre cinco ejecuciones) requerido para resolver el problema; notemos que todas las entradas excepto las dos de la parte superior derecha están en miles (K). Los números en paréntesis significan que no se ha encontrado ninguna respuesta en el número asignado de comprobaciones.

El primer problema es colorear, con cuatro colores, los 50 estados de los Estados Unidos de América. El segundo problema cuenta el número total de comprobaciones requeridas para resolver todos los problemas de n-reinas para n de dos a 50. El tercero es el «puzle Zebra». Los dos últimos son problemas artificiales aleatorios.

2. Búsqueda con vuelta atrás

Empezamos con búsquedas sin información; las mejoramos incluyéndoles una heurística especifica del dominio obtenida de nuestro conocimiento del problema (búsqueda con información).

Pero los PSRs los podemos resolver sin un conocimiento específico del dominio (métodos de propósito general) si nos respondemos a las siguientes preguntas:

1. Qué **variable** debe asignarse después, y en qué **orden** deberían intentarse sus valores?
2. ¿Cuáles son las implicaciones de las **variables actuales para el resto de variables** no asignadas?
3. Cuando un camino falla, ¿puede la búsqueda **evitar repetir este fracaso** en caminos siguientes?

Contenido

1. Problemas de Satisfacción de Restricciones.

2. **Búsqueda con vuelta atrás**
   1. **Variables y orden**
   2. Implicaciones variables actuales
   3. Evitar repetir fracasos

3. Métodos de Búsqueda local

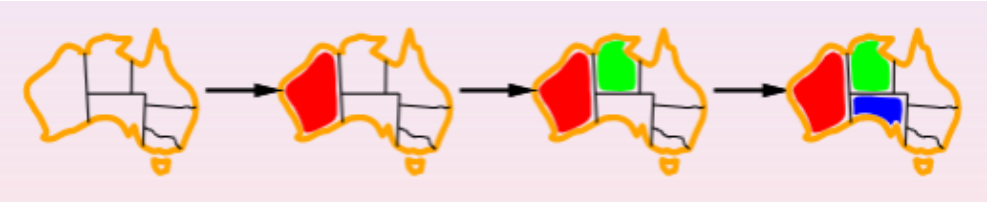
4. La estructura de los problemas

2. Búsqueda con vuelta atrás

1. Qué **variable** debe asignarse después, y en qué **orden** deberían intentarse sus valores?

Tenemos varias respuestas:

- Heurística Mínimos Valores Restantes (MVR)
- Grado heurístico
- Valor menos restringido



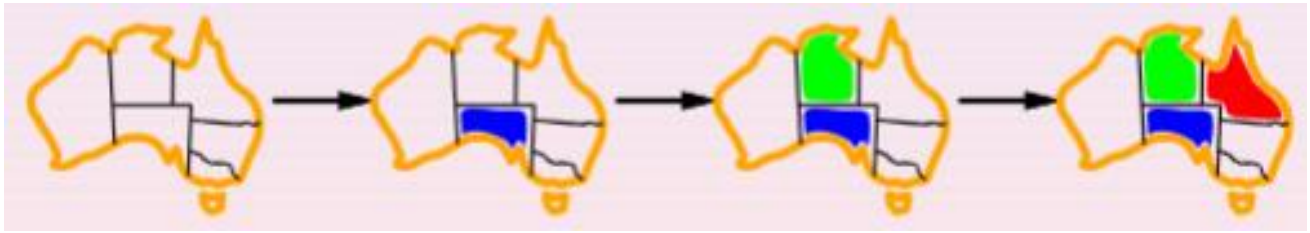
Heurística MVR (o variable más restringida): Elegir la variable a la que le quedan menos valores posibles (está más restringida).

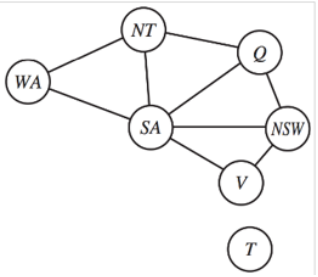
Problema	Vuelta atrás	VA + MVR	Comprobación hacia delante	CD + MVR	Mínimo conflicto
EE.UU.	(> 1.000K)	(> 1.000K)	2K	60	64
n-reinas	(> 40.000K)	13.500K	(> 40.000K)	817K	4K
Zebra	3.859K	1K	35K	0,5K	2K
Aleatorio 1	415K	3K	26K	2K	
Aleatorio 2	942K	27K	77K	15K	

2. Búsqueda con vuelta atrás

Grado Heurístico: la heurística MVR no ayuda en la elección de la primera región a colorear…

El Grado Heurístico intenta reducir el factor de ramificación sobre futuras opciones seleccionando la variable, entre las no asignadas, que esté implicada en el mayor número de restricciones. La heurística MVR es, en general, una guía más poderosa, pero el Grado Heurístico puede ser útil en casos de empate.

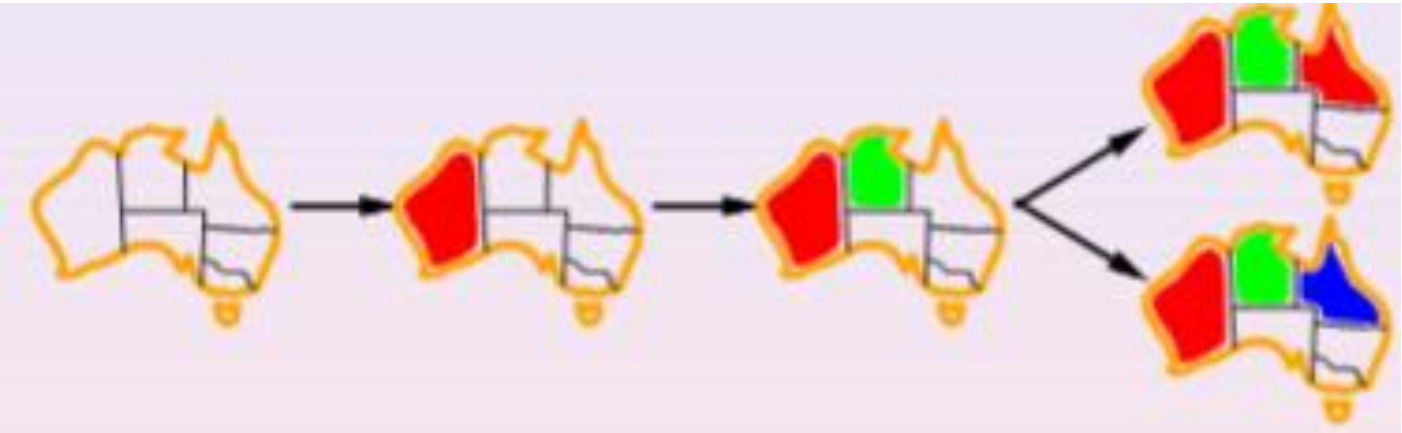




Una vez elegida *SA*, aplicando el grado heurístico que resuelve el problema sin pasos en falso puede elegir cualquier color consistente en cada punto seleccionado y todavía llegar a una solución sin vuelta atrás.

2. Búsqueda con vuelta atrás

Valor menos restringido: Dada una variable, elegir el valor que restringe menos los valores de las variables no asignadas.



Se elige Q = Roja.

Trata de dejar la flexibilidad máxima de las asignaciones de las variables siguientes (no válido si queremos encontrar todas las soluciones o el problema no tiene solución).

Contenido

1. Problemas de Satisfacción de Restricciones.

2. **Búsqueda con vuelta atrás**
   1. Variables y orden
   2. **Implicaciones variables actuales**
   3. Evitar repetir fracasos

3. Métodos de Búsqueda local

4. La estructura de los problemas

2. Búsqueda con vuelta atrás

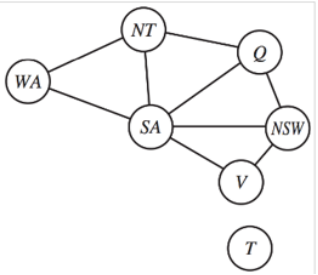
2. ¿Cuáles son las implicaciones de las **variables actuales para el resto de variables** no asignadas?

Tenemos varias respuestas:

- Comprobación hacia delante
- Propagación de restricciones
- Manejo de restricciones especiales

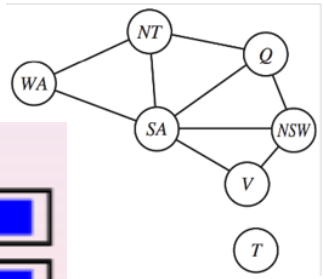
2. Búsqueda con vuelta atrás

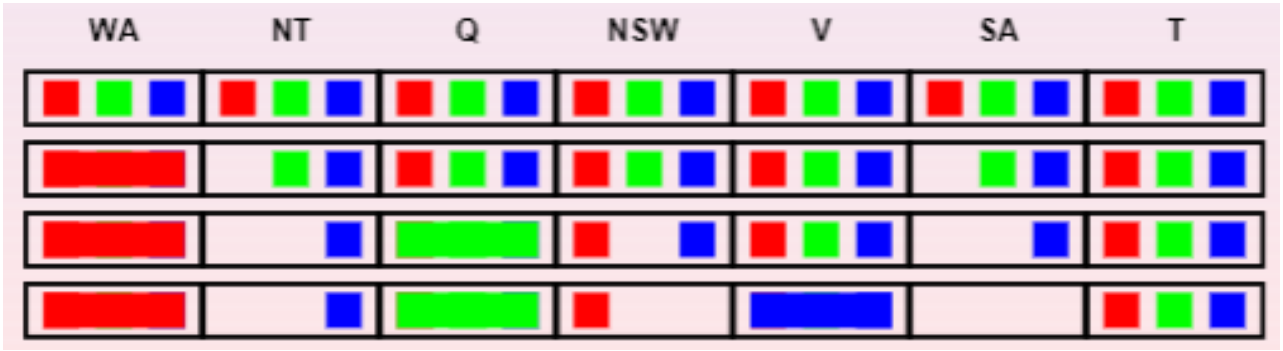
- **Comprobación hacia delante:** Siempre que se asigne una variable X, el proceso de búsqueda hacia delante mira cada variable no asignada Y que esté relacionada con X por una restricción y suprime del dominio de Y cualquier valor que sea inconsistente con el valor elegido para X. Si hay una variable sin valores posible vuelve atrás.



2. Búsqueda con vuelta atrás

- **Comprobación hacia delante:**

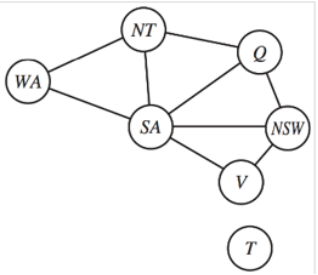


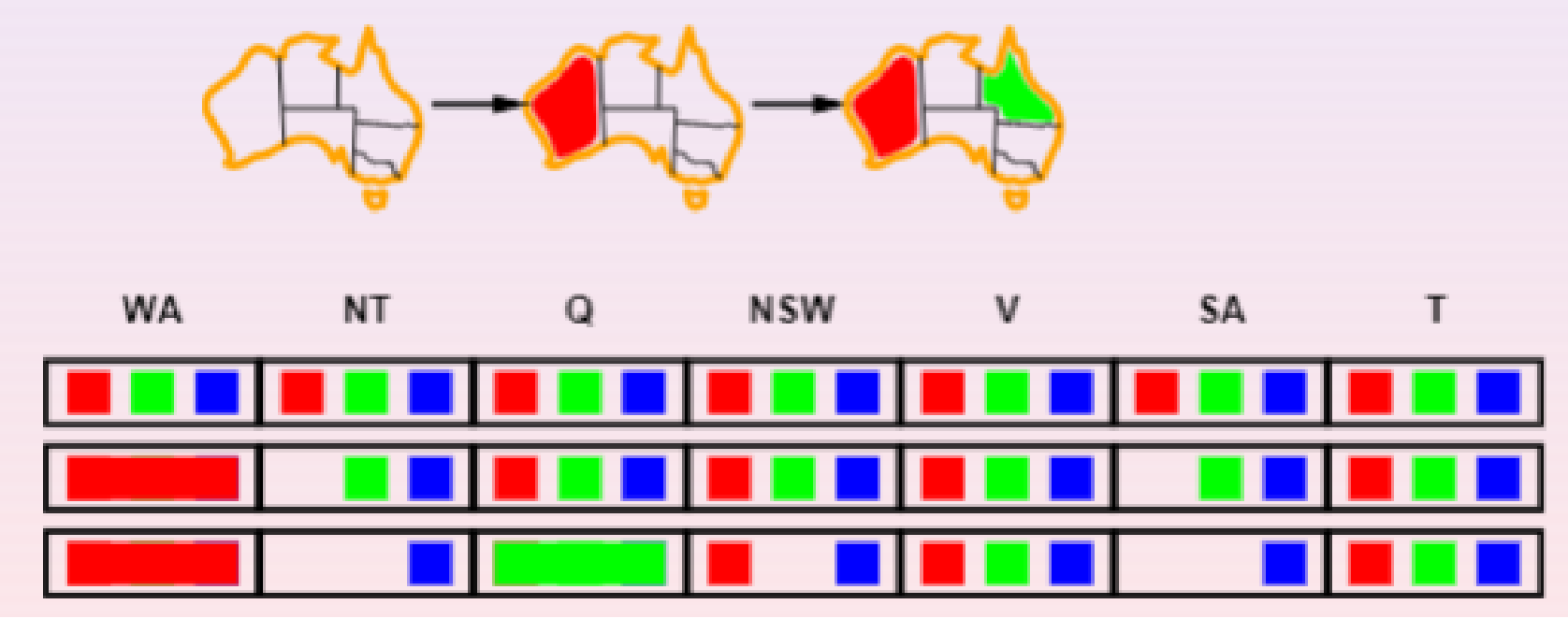


Problema	Vuelta atrás	VA + MVR	Comprobación hacia delante	CD + MVR	Mínimo conflicto
EE.UU.	(> 1.000K)	(> 1.000K)	2K	60	64
n-reinas	(> 40.000K)	13.500K	(> 40.000K)	817K	4K
Zebra	3.859K	1K	35K	0,5K	2K
Aleatorio 1	415K	3K	26K	2K	
Aleatorio 2	942K	27K	77K	15K	

2. Búsqueda con vuelta atrás

- **Comprobación hacia delante:** La comprobación hacia delante no descubre todas las inconsistencias, por ejemplo, después de asignar WA=R y Q=V, tanto NT como SA están obligadas a tomar el valor A, pero como son adyacentes no pueden tomar el mismo valor.





2. Búsqueda con vuelta atrás

Propagación de restricciones: Una forma más sofisticada de comprobar que los valores son los **arcos consistentes**.

Se basa en comprobar que toda arista ordenada (X,Y) del grafo de restricciones es consistente.

Si las restricciones no son binarias, se puede aplicar pero es más complejo. Habría que considerar las aristas consistentes para toda pareja ordenada (X,Y) tales que haya una restricción que involucre a ambas variables.

La consistencia de los arcos se puede hacer al principio y, de forma recursiva, habría que comprobar la consistencia de todo arco (X,Z) después de que Z haya perdido alguno de sus valores posibles

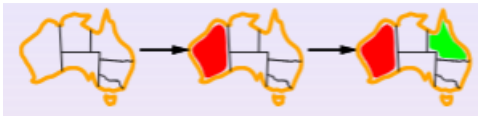
Arco consistente: **Un arco (X,Y) es consistente si para todo valor posible x de X existe un valor posible y de la variable Y.**

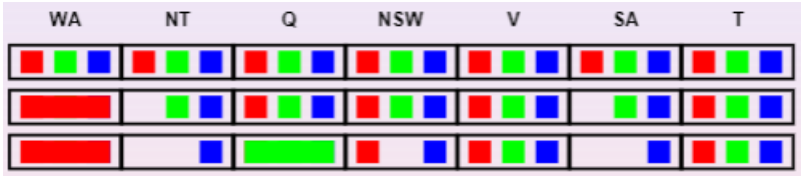
Si una arco no es consistente, se eliminan todos los valores posibles de X para los que no exista un valor consistente de Y.

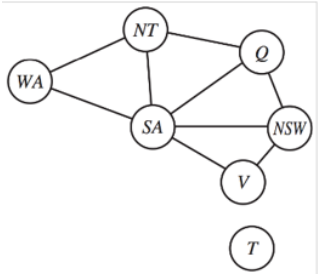
2. Búsqueda con vuelta atrás

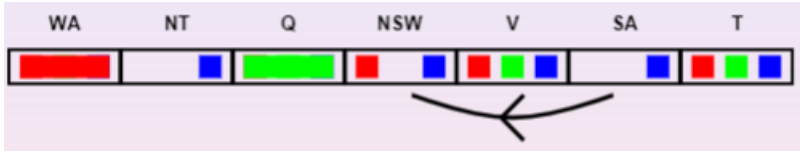
Propagación de restricciones :

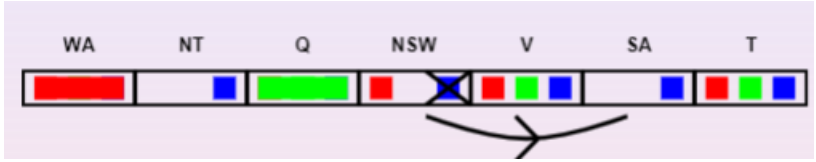
- **Ejemplo de arco consistente**











Se comprueba que no existe ningún valor posible para SA con lo cual podemos realizar una vuelta atrás.

2. Búsqueda con vuelta atrás

Propagación de restricciones :

- **Arco consistente – Algoritmo AC-3**

  El siguiente algoritmo se aplica al principio a todos los arcos (X,Y) del grafo de restricciones y, se aplica a todo arco (Z,X) cada vez que X restringe el conjunto de valores posibles.

```
1:  for Cada valor posible x de X do
2:       if No existe un valor y de Y que sea compatible con x then
3:                 Eliminar x del conjunto de valores posibles de X
```

  La propagación de arcos consistentes (AC-3) realiza una inferencia para eliminar posibles valores de las variables y determinar si hay inconsistencia en una asignación parcial. Sin embargo, **este método no es completo**: es posible que haya inconsistencia y no se detecte. Por ejemplo, si queremos determinar al principio si se puede colorear a Australia con 2 colores. No es posible, pero AC-3 no lo detecta. Existen otros métodos más potentes. Siempre hay que buscar el equilibrio entre el coste de la inferencia y lo que nos ahorramos en la búsqueda.

2. Búsqueda con vuelta atrás

- **Manejo de Restricciones Especiales:**

1. **Restricciones todas distintas**: TodasDistintas (X1, . . . ,Xm)
   Si hay *m* variables implicadas en la restricción, y si tienen *n* valores posibles distintos, y *m* > *n*, entonces la restricción no puede satisfacerse.
   Mejora el criterio de arcos consistentes.

2. **Restricciones de recursos:**
   Dos vuelos, 271 y 272 con capacidades 156, 385 respectivamente: Vuelo 271 pertenece [0,165] y Vuelo 272 pertenece [0,385]
   Restricción adicional: los dos vuelos juntos deben llevar a 420 personas.
   Entonces se deduce: Vuelo 271 pertenece [35, 165] y
   Vuelo 272 pertenece [255, 385]
   Esto se conoce como **propagación de límites.**

.

Contenido

1. Problemas de Satisfacción de Restricciones.

2. **Búsqueda con vuelta atrás**
   1. Variables y orden
   2. Implicaciones variables actuales
   3. **Evitar repetir fracasos**

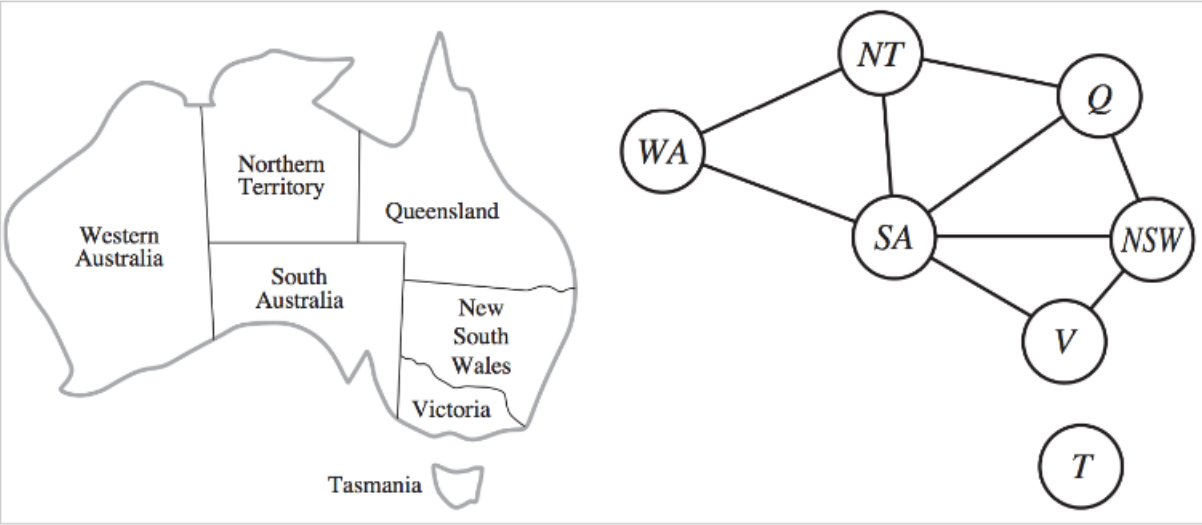
3. Métodos de Búsqueda local

4. La estructura de los problemas

2. Búsqueda con vuelta atrás

3. Cuando un camino falla, ¿puede la búsqueda **evitar repetir este fracaso** en caminos siguientes?

Vuelta Atrás Inteligente: Se visita el punto de decisión más reciente.



Orden de actuación: Q, NSW, V, T, SA, WA, NT

{Q = rojo, NSW = verde, V = azul, T = rojo}

SA ?

Vuelta atrás a T inútil

2. Búsqueda con vuelta atrás

- **Vuelta Atrás Inteligente:**
  Conjunto conflicto para la variable X es el conjunto de variables previamente asignadas Y, tales que el valor asignado a Y evita uno de los valores de X, debido a la restricción entre ambas.
  El método de salto-atrás retrocede a la variable más reciente en el conjunto conflicto.

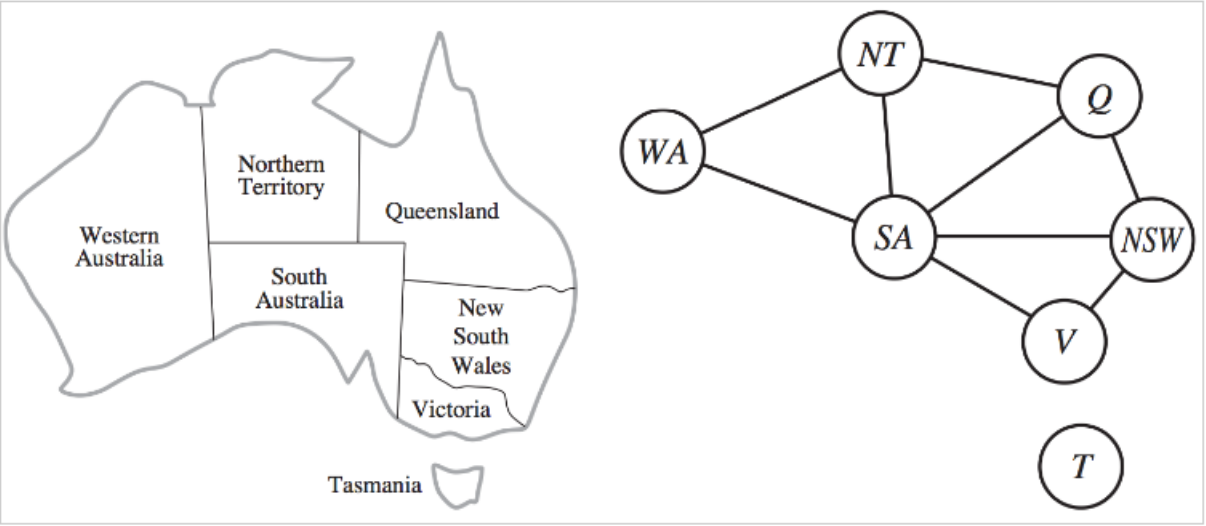
  Ejemplo:
  Conjunto conflicto para AS:{Q,NSW,V}
  El método salto-atrás retrocede a V sin considerar T

  La propagación hacia delante puede proporcionar el conjunto de conflicto inicial.

  Si desde X_i volvemos hasta X_j, porque X_i se ha quedado sin valores, entonces debemos de actualizar el conflicto de X_j con:
  $Conf(X_j) \leftarrow Conf(X_j) \cup Conf(X_i) - \{X_j\}$.

2. Búsqueda con vuelta atrás

- **Vuelta Atrás Inteligente:**



1. Orden de actuación: WA, NSW, T, NT, Q, V, SA
2. Asignación parcial: WA=Rojo, NSW=Rojo, T=Rojo
3. Cuando NT se queda sin valores, entonces su conjunto de conflicto es: {WA,NSW}
   - Cuando SA falla, su conjunto de conflicto es {WA,NT,Q}.
   - Volvemos a Q, con conjunto de conflicto {NT,NSW}, que queda actualizado a {WA,NT,NSW}.
   - Volvemos a NT con conjunto de conflicto {WA} que se actualiza a {WA,NSW}

Contenido

1. Problemas de Satisfacción de Restricciones.

2. Búsqueda con vuelta atrás
   1. Variables y orden
   2. Implicaciones variables actuales
   3. Evitar repetir fracasos

3. **Métodos de Búsqueda local**

4. La estructura de los problemas

3. Métodos de Búsqueda Local

Volvemos a la formulación completa de estados.

Aplicamos **algoritmos de búsqueda local**: ascensión de colinas, enfriamiento simulado, algoritmos genéticos, etc.

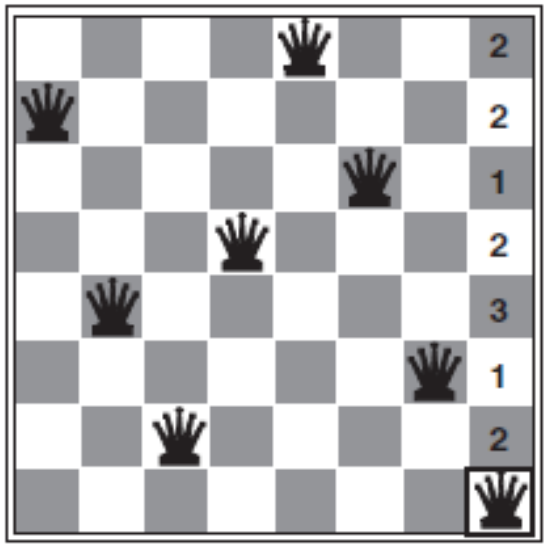
Heurística: **Mínimos conflictos**. Seleccionar el **valor que cause el número mínimo de conflictos** con otras variables.

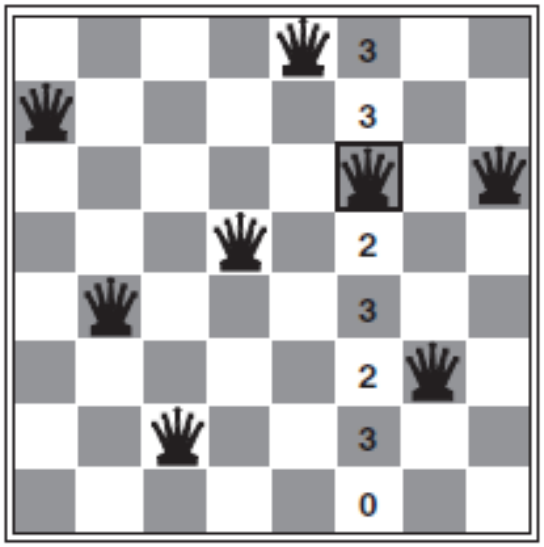
Problema	Vuelta atrás	VA + MVR	Comprobación hacia delante	CD + MVR	Mínimo conflicto
EE.UU.	(> 1.000K)	(> 1.000K)	2K	60	64
n-reinas	(> 40.000K)	13.500K	(> 40.000K)	817K	4K
Zebra	3.859K	1K	35K	0,5K	2K
Aleatorio 1	415K	3K	26K	2K	
Aleatorio 2	942K	27K	77K	15K	

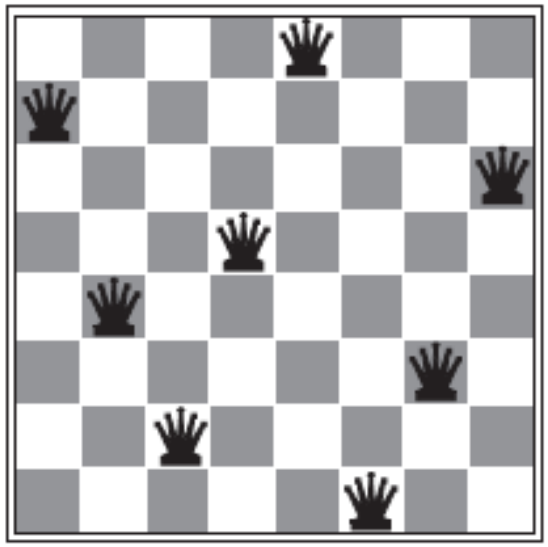
3. Métodos de Búsqueda Local

Ejemplo: 8 reinas.

En cada etapa, se elige una reina para la reasignación en su columna. El número de conflictos (en este caso, el número de reinas atacadas) se muestra en cada cuadrado. El algoritmo mueve a la reina a un cuadrado de mínimo conflicto, deshaciendo los empates de manera aleatoria.







3. Métodos de Búsqueda Local

Con un estado inicial aleatorio, con esta heurística se puede resolver el **problema de las n-reinas en tiempo constante** y con una alta probabilidad (por ejemplo con n = 10,000,000).

La técnica de los mínimos conflictos es sorprendentemente eficaz para muchos PSRs, en particular, cuando se parte de un estado inicial razonable.

Por ejemplo, se ha utilizado para programar las observaciones del telescopio Hubble, reduciendo el tiempo utilizado para programar **1 semana** de observaciones, de **3 semanas (!!??)** a alrededor de **10 minutos**.

Otra ventaja es que se puede utilizar en ajuste online cuando las condiciones del problema cambian.

Contenido

1. Problemas de Satisfacción de Restricciones.

2. Búsqueda con vuelta atrás
   1. Variables y orden
   2. Implicaciones variables actuales
   3. Evitar repetir fracasos

3. Métodos de Búsqueda local

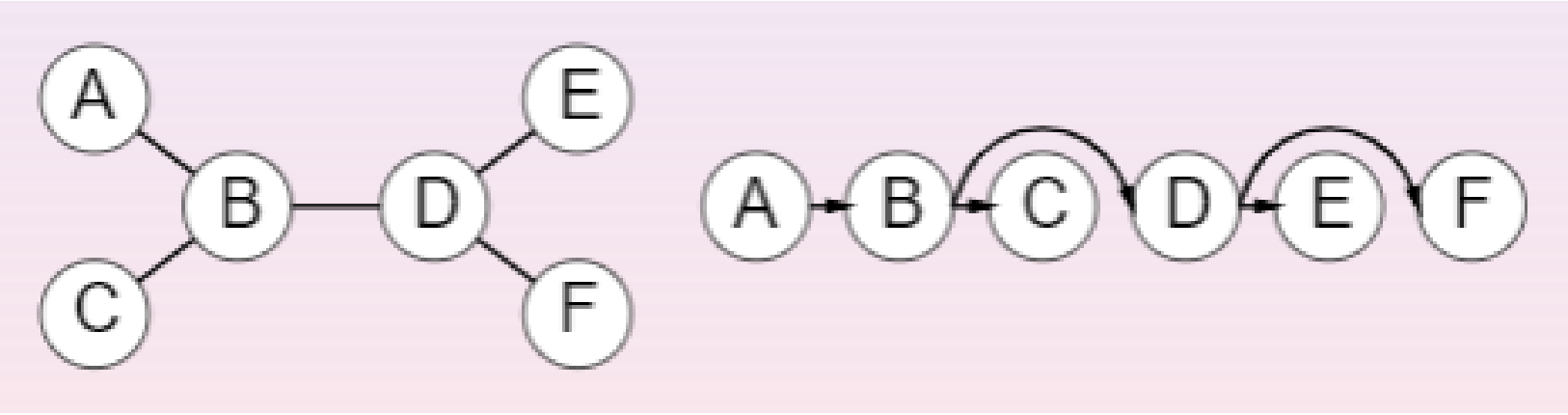
4. **La estructura de los problemas**

4. La estructura de los problemas

Idea: aprovecharse de la existencia de problemas independientes.

Técnicas:

1. Algoritmos en árboles.
2. Método del condicionamiento.
3. Construcción de árbol agrupando variables.



4. La estructura de los problemas

Algoritmos en árboles.

Algoritmo**:**

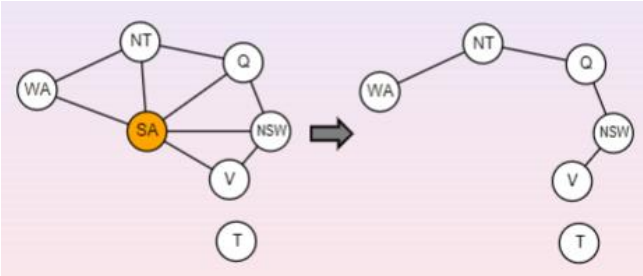
1. Elegir un nodo como raíz
2. Ordenar los nodos desde esta raíz a las hojas, de forma que el padre de cada nodo lo preceda en este orden.
3. Elegir todos los nodos X_j en orden inverso y aplicar la consistencia de arco entre X_i y X_j, donde X_i es el padre de X_j. Eliminar los valores de X_i que sea necesario.
4. Recorrer los nodos en orden directo empezando por la raíz. Para cada nodo X_j elegir un valor que sea consistente con el elegido para su padre X_i

El algoritmo tiene complejidad $O(nd^2)$ donde
n es el número de variables y
d el número de casos por variable.

4. La estructura de los problemas

Método del condicionamiento

Si el grafo no es un árbol, se puede tratar de conseguir un conjunto pequeño de variables S, tal que si las quitamos del grafo de restricciones, nos queda un árbol.



1. Elegir un subconjunto S de variables que al quitarlas nos quede una estructura de árbol
2. **for** Cada posible asignación consistente de valores s de S **do**
   1. Quitar del resto de las variables, los valores inconsistentes con estos valores
   2. Resolver el problema de satisfacción de restricciones para el resto de las variables (árbol)
   3. Si se encuentra la solución, adjuntarla a S=s para obtener una solución al problema original y parar

4. La estructura de los problemas

Construcción de un árbol agrupando variables.

¿Los grafos de restricción pueden reducirse a árboles?

Si. Mediante un proceso de agrupación de variables:

- Cada variable tiene que estar, al menos, en un grupo, pero puede estar en varios.
- Si dos variables están unidas en el grafo, debe de existir un grupo que las contenga.
- Si una variable está en dos grupos, debe de estar en todo grupo que se encuentre en el camino entre ambas.

Si un grafo tiene la anchura de árbol w, y **nos dan la descomposición en árb correspondiente**, entonces el problema puede resolverse en $O(nd^{w+1})$ veces.
Los PSRs con grafos de restricciones de anchura de árbol acotada son resolubles en tiempo polinomial.