Backtracking I

Clase 15

IIC 2133 - Sección 3

Prof. Eduardo Bustos

Duelo Deep Blue -Kasparov

- Deep Blue fue una supercomputadora desarrollada por IBM para jugar al ajedrez a un nivel de competencia mundial. Su serie de partidas contra el campeón mundial Garry Kasparov en 1996 y 1997 fue ampliamente publicitada, y su victoria en la revancha de 1997 fue un hito en el campo de la inteligencia artificial.
- La estrategia de Deep Blue involucró varios componentes clave:

1. **Búsqueda de fuerza bruta**: Deep Blue podía evaluar hasta 200 millones de posiciones por segundo. Utilizó técnicas de búsqueda alfa-beta para examinar tantos movimientos posibles como fuera práctico, mirando muchos movimientos por adelantado.
2. **Libros de apertura y cierre**: Al igual que muchos programas de ajedrez, Deep Blue tenía acceso a bases de datos extensas de movimientos de apertura y finales conocidos. Estos le permitían jugar estas fases del juego con eficiencia y precisión.
3. **Evaluación personalizada**: Aunque la fuerza bruta fue una parte importante de su estrategia, Deep Blue también tenía una función de evaluación altamente detallada y adaptada para valorar las posiciones. Esta función fue diseñada con la ayuda de ajedrecistas de alto nivel y tomaba en cuenta muchos aspectos del juego, desde el valor material de las piezas hasta consideraciones posiciones más sutiles.
4. **Paralelización**: Deep Blue utilizó múltiples procesadores (en su versión de 1997, tenía 30 procesadores principales y 480 procesadores especializados en ajedrez) trabajando en paralelo para aumentar la velocidad y profundidad de su búsqueda.
5. **Aprendizaje entre partidas**: Entre las dos series de matches (1996 y 1997), el equipo de Deep Blue revisó y ajustó el sistema para mejorar su rendimiento basándose en la experiencia de las partidas anteriores contra Kasparov.

ChatGPT

El Encuentro de 1996				
Partida	**Blancas**	**Negras**	**Resultado**	**Encuentro**
1	**Deep Blue**	Kasparov	1–0	Deep Blue 1 - 0 Kasparov
2	**Kasparov**	Deep Blue	1–0	Deep Blue 1 - 1 Kasparov
3	Deep Blue	Kasparov	½–½	Deep Blue 1½ - 1½ Kasparov
4	Kasparov	Deep Blue	½–½	Deep Blue 2 - 2 Kasparov
5	Deep Blue	**Kasparov**	0–1	Deep Blue 2 - 3 Kasparov
6	**Kasparov**	Deep Blue	1–0	Deep Blue 2 - 4 Kasparov
Resultado: **Kasparov – Deep Blue: 4–2**				

El Encuentro de 1997				
Partida	**Blancas**	**Negras**	**Resultado**	**Encuentro**
1	**Kasparov**	Deep Blue	1–0	Deep Blue 0 - 1 Kasparov
2	**Deep Blue**	Kasparov	1–0	Deep Blue 1 - 1 Kasparov
3	Kasparov	Deep Blue	½–½	Deep Blue 1½ - 1½ Kasparov
4	Deep Blue	Kasparov	½–½	Deep Blue 2 - 2 Kasparov
5	Kasparov	Deep Blue	½–½	Deep Blue 2½ - 2½ Kasparov
6	**Deep Blue**	Kasparov	1–0	Deep Blue 3½ - 2½ Kasparov
Resultado: **Deep Blue–Kasparov: 3½–2½**				

Sumario

CSPs

Backtracking

Cierre

Un problema clásico

Un problema clásico

Consideremos el problema de posicionar 8 reinas en un tablero de ajedrez de modo que no se ataquen

Un problema clásico

Consideremos el problema de posicionar 8 reinas en un tablero de ajedrez de modo que no se ataquen

- Las reinas se desplazan por filas, columnas y diagonales

Un problema clásico

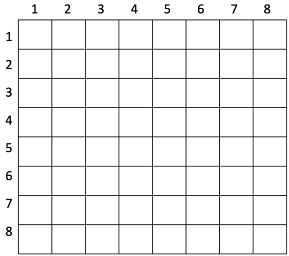
Consideremos el problema de posicionar 8 reinas en un tablero de ajedrez de modo que no se ataquen

- Las reinas se desplazan por filas, columnas y diagonales
- Para lograr el objetivo: deben estar en columnas, filas y diagonales diferentes

Un problema clásico

Consideremos el problema de posicionar 8 reinas en un tablero de ajedrez de modo que no se ataquen

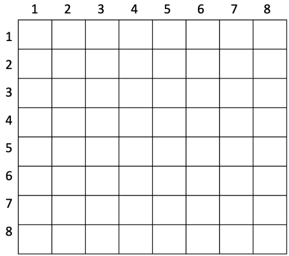
- Las reinas se desplazan por filas, columnas y diagonales
- Para lograr el objetivo: deben estar en columnas, filas y diagonales diferentes



Un problema clásico

Consideremos el problema de posicionar 8 reinas en un tablero de ajedrez de modo que no se ataquen

- Las reinas se desplazan por filas, columnas y diagonales
- Para lograr el objetivo: deben estar en columnas, filas y diagonales diferentes



¿Qué tan fácil es resolverlo?

Un problema clásico

Para modelar este problema, podemos numerar las filas y columnas

Un problema clásico

Para modelar este problema, podemos numerar las filas y columnas

- Cada fila y columna en el rango $1 \ldots 8$

Un problema clásico

Para modelar este problema, podemos numerar las filas y columnas

- Cada fila y columna en el rango $1 \dots 8$
- Denotamos por x_i a la columna de la reina en la fila i

Un problema clásico

Para modelar este problema, podemos numerar las filas y columnas

- Cada fila y columna en el rango $1 \dots 8$
- Denotamos por x_i a la columna de la reina en la fila i
- Las posiciones de las 8 reinas se describe como un vector

$$(x_1, \dots, x_8)$$

Un problema clásico

Para modelar este problema, podemos numerar las filas y columnas

- Cada fila y columna en el rango $1 \ldots 8$
- Denotamos por x_i a la columna de la reina en la fila i
- Las posiciones de las 8 reinas se describe como un vector

$$(x_1, \ldots, x_8)$$

¿Cómo sabemos si $(4, 6, 8, 2, 7, 1, 3, 5)$ es una solución?

Un problema clásico

El vector $(4, 6, 8, 2, 7, 1, 3, 5)$ representa la siguiente configuración

	1	2	3	4	5	6	7	8
1				Q_1				
2						Q_2		
3								Q_3
4		Q_4						
5							Q_5	
6	Q_6							
7			Q_7					
8					Q_8			

Un problema clásico

El vector $(4, 6, 8, 2, 7, 1, 3, 5)$ representa la siguiente configuración

	1	2	3	4	5	6	7	8
1				Q_1				
2						Q_2		
3								Q_3
4		Q_4						
5							Q_5	
6	Q_6							
7			Q_7					
8					Q_8			

Efectivamente es solución al problema

Un problema clásico

Este problema tiene un conjunto de **restricciones**

Un problema clásico

Este problema tiene un conjunto de **restricciones**

1. Dos reinas no deben estar en la misma fila

Un problema clásico

Este problema tiene un conjunto de **restricciones**

1. Dos reinas no deben estar en la misma fila
2. Dos reinas no deben estar en la misma columna

Un problema clásico

Este problema tiene un conjunto de **restricciones**

1. Dos reinas no deben estar en la misma fila
2. Dos reinas no deben estar en la misma columna
3. Dos reinas no deben estar en un camino diagonal

Un problema clásico

Este problema tiene un conjunto de **restricciones**

1. Dos reinas no deben estar en la misma fila
2. Dos reinas no deben estar en la misma columna
3. Dos reinas no deben estar en un camino diagonal

	1	2	3	4	5	6	7	8
1				Q_1				
2						Q_2		
3								Q_3
4		Q_4						
5							Q_5	
6	Q_6							
7			Q_7					
8					Q_8			

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \ldots, D_n\}$ es un conjunto de **dominios** respectivos

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \ldots, D_n\}$ es un conjunto de **dominios** respectivos
- $C = \{C_1, \ldots, C_m\}$ es un conjunto de **restricciones**

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \ldots, D_n\}$ es un conjunto de **dominios** respectivos
- $C = \{C_1, \ldots, C_m\}$ es un conjunto de **restricciones**

donde cada restricción involucra un subconjunto de variables de X.

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \ldots, D_n\}$ es un conjunto de **dominios** respectivos
- $C = \{C_1, \ldots, C_m\}$ es un conjunto de **restricciones**

donde cada restricción involucra un subconjunto de variables de X. Una **solución** es una asignación de las variables en sus dominios tal que se satisfacen todas las restricciones.

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \ldots, D_n\}$ es un conjunto de **dominios** respectivos
- $C = \{C_1, \ldots, C_m\}$ es un conjunto de **restricciones**

donde cada restricción involucra un subconjunto de variables de X. Una **solución** es una asignación de las variables en sus dominios tal que se satisfacen todas las restricciones.

Observemos que

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \ldots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \ldots, D_n\}$ es un conjunto de **dominios** respectivos
- $C = \{C_1, \ldots, C_m\}$ es un conjunto de **restricciones**

donde cada restricción involucra un subconjunto de variables de X. Una **solución** es una asignación de las variables en sus dominios tal que se satisfacen todas las restricciones.

Observemos que

- No necesariamente las variables son del mismo dominio

Problemas de satisfacción de restricciones

Definición

Un problema de satisfacción de restricciones o *constraint satisfaction problem* **(CSP)** es una tripleta (X, D, C) tal que

- $X = \{x_1, \dots, x_n\}$ es un conjunto de **variables**
- $D = \{D_1, \dots, D_n\}$ es un conjunto de **dominios** respectivos
- $C = \{C_1, \dots, C_m\}$ es un conjunto de **restricciones**

donde cada restricción involucra un subconjunto de variables de X. Una **solución** es una asignación de las variables en sus dominios tal que se satisfacen todas las restricciones.

Observemos que

- No necesariamente las variables son del mismo dominio
- Una restricción C_j puede involucrar 1, 2 o más variables de X

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

- $D = \{B, \ldots, B\}$ con $B = \{1, \ldots, 8\}$

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

- $D = \{B, \ldots, B\}$ con $B = \{1, \ldots, 8\}$
- En este caso el dominio de cada variable en X es el mismo

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

- $D = \{B, \ldots, B\}$ con $B = \{1, \ldots, 8\}$
- En este caso el dominio de cada variable en X es el mismo

Restricciones

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

- $D = \{B, \ldots, B\}$ con $B = \{1, \ldots, 8\}$
- En este caso el dominio de cada variable en X es el mismo

Restricciones

- La restricción sobre las filas está implícita en la elección de las variables

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

- $D = \{B, \ldots, B\}$ con $B = \{1, \ldots, 8\}$
- En este caso el dominio de cada variable en X es el mismo

Restricciones

- La restricción sobre las filas está implícita en la elección de las variables
- Para las columnas: $i \neq j \rightarrow x_i \neq x_j$

Problemas de satisfacción de restricciones

Ejemplo

El **problema de las 8 reinas** efectivamente es un CSP

Variables

- $X = \{x_1, \ldots, x_8\}$
- Cada variable x_i se interpreta como columna de la reina en la fila i

Dominios

- $D = \{B, \ldots, B\}$ con $B = \{1, \ldots, 8\}$
- En este caso el dominio de cada variable en X es el mismo

Restricciones

- La restricción sobre las filas está implícita en la elección de las variables
- Para las columnas: $i \neq j \to x_i \neq x_j$
- Para las diagonales: $i \neq j \to |(x_j - x_i)/(j - i)| = 1$

Otro problema clásico

Consideremos un tablero de sudoku parcialmente completado

								9
	7					6		8
						1		4
				3				2
		1			5	3		7
	5							3
					9			5

Otro problema clásico

Consideremos un tablero de sudoku parcialmente completado

								9
	7					6		8
						1		4
				3				2
		1			5	3		7
	5							3
					9			5

¿Podemos ver el sudoku como un CSP? ¿X, D, C?

¿Es fácil resolver los CSP?

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Qué tan rápido pueden resolverse los problemas de esta clase?

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Qué tan rápido pueden resolverse los problemas de esta clase?

Existe un problema central en computación que puede ayudarnos

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Qué tan rápido pueden resolverse los problemas de esta clase?

Existe un problema central en computación que puede ayudarnos

Definición

El problema de decisión **SAT** toma como input una fórmula en lógica proposicional $\varphi \in \mathcal{L}(P)$ y responde si φ es satisfacible

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Qué tan rápido pueden resolverse los problemas de esta clase?

Existe un problema central en computación que puede ayudarnos

Definición

El problema de decisión **SAT** toma como input una fórmula en lógica proposicional $\varphi \in \mathcal{L}(P)$ y responde si φ es satisfacible

Ejemplo

Para el conjunto $P = \{p\}$

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Qué tan rápido pueden resolverse los problemas de esta clase?

Existe un problema central en computación que puede ayudarnos

Definición

El problema de decisión **SAT** toma como input una fórmula en lógica proposicional $\varphi \in \mathcal{L}(P)$ y responde si φ es satisfacible

Ejemplo

Para el conjunto $P = \{p\}$

- $\varphi_1 = p \rightarrow \neg p$ es satifacible, pues $\sigma(\varphi_1) = 1$ para la valuación $\sigma(p) = 0$

¿Es fácil resolver los CSP?

Las 8 reinas y el sudoku son ejemplos de la clase de problemas CSP

¿Qué tan rápido pueden resolverse los problemas de esta clase?

Existe un problema central en computación que puede ayudarnos

Definición

El problema de decisión **SAT** toma como input una fórmula en lógica proposicional $\varphi \in \mathcal{L}(P)$ y responde si φ es satisfacible

Ejemplo

Para el conjunto $P = \{p\}$

- $\varphi_1 = p \rightarrow \neg p$ es satifacible, pues $\sigma(\varphi_1) = 1$ para la valuación $\sigma(p) = 0$
- $\varphi_2 = p \wedge \neg p$ no es satifacible, pues no existe valuación que la haga verdadera

¿Es fácil resolver los CSP?

Ahora, para $\varphi \in \mathcal{L}(P)$, podemos interpretar la pregunta

$$¿\varphi \text{ es satisfacible?}$$

como un CSP donde

¿Es fácil resolver los CSP?

Ahora, para $\varphi \in \mathcal{L}(P)$, podemos interpretar la pregunta

$$¿\varphi \text{ es satisfacible?}$$

como un CSP donde

- $X = P$, conjunto de variables proposicionales

¿Es fácil resolver los CSP?

Ahora, para $\varphi \in \mathcal{L}(P)$, podemos interpretar la pregunta

$$¿\varphi \text{ es satisfacible?}$$

como un CSP donde

- $X = P$, conjunto de variables proposicionales
- $D = \{B \ldots, B\}$ con $B = \{0, 1\}$

¿Es fácil resolver los CSP?

Ahora, para $\varphi \in \mathcal{L}(P)$, podemos interpretar la pregunta

$$¿\varphi \text{ es satisfacible?}$$

como un CSP donde

- $X = P$, conjunto de variables proposicionales
- $D = \{B \ldots, B\}$ con $B = \{0, 1\}$
- Restricción de que el valor de verdad de φ sea 1 al evaluar los valores asignados a cada variable

¿Es fácil resolver los CSP?

Ahora, para $\varphi \in \mathcal{L}(P)$, podemos interpretar la pregunta

¿φ es satisfacible?

como un CSP donde

- $X = P$, conjunto de variables proposicionales
- $D = \{B \ldots, B\}$ con $B = \{0, 1\}$
- Restricción de que el valor de verdad de φ sea 1 al evaluar los valores asignados a cada variable

Si tuviéramos una forma eficiente de resolver un **CSP**,
podríamos usarla para resolver **SAT**

¿Es fácil resolver los CSP?

Teorema

El problema de decisión **SAT** es **NP-completo**

¿Es fácil resolver los CSP?

Teorema

El problema de decisión **SAT** es **NP-completo**

Los problemas NP-completos son considerados difíciles

¿Es fácil resolver los CSP?

Teorema

El problema de decisión **SAT** es **NP-completo**

Los problemas NP-completos son considerados difíciles

- Es un problema abierto saber si se pueden resolver de manera eficiente

¿Es fácil resolver los CSP?

Teorema

El problema de decisión **SAT** es **NP-completo**

Los problemas NP-completos son considerados difíciles

- Es un problema abierto saber si se pueden resolver de manera eficiente
- Además, todo problema NP-completo sirve para resolver otro problema NP-completo

¿Es fácil resolver los CSP?

Teorema

El problema de decisión **SAT** es **NP-completo**

Los problemas NP-completos son considerados difíciles

- Es un problema abierto saber si se pueden resolver de manera eficiente
- Además, todo problema NP-completo sirve para resolver otro problema NP-completo

Con esto, los CSP servirían para resolver cualquier problema NP-completo

¿Es fácil resolver los CSP?

Teorema

El problema de decisión **SAT** es **NP-completo**

Los problemas NP-completos son considerados difíciles

- Es un problema abierto saber si se pueden resolver de manera eficiente
- Además, todo problema NP-completo sirve para resolver otro problema NP-completo

Con esto, los CSP servirían para resolver cualquier problema NP-completo

Conclusión: los CSP son difíciles

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables
- Verificar cada asignación para ver si cumple **todas** las restricciones

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables
- Verificar cada asignación para ver si cumple **todas** las restricciones
- Si se encuentra una asignación que cumple, se retorna como solución

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables
- Verificar cada asignación para ver si cumple **todas** las restricciones
- Si se encuentra una asignación que cumple, se retorna como solución

Para un CSP (X, D, C), esto requiere revisar en general las tuplas de

$$D_1 \times D_2 \times \cdots \times D_n$$

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables
- Verificar cada asignación para ver si cumple **todas** las restricciones
- Si se encuentra una asignación que cumple, se retorna como solución

Para un CSP (X, D, C), esto requiere revisar en general las tuplas de

$$D_1 \times D_2 \times \cdots \times D_n$$

Ejemplo

Para el problema de las 8 reinas, hay

$$8^8 = 16.777.216$$

tuplas posibles de la forma $(x_1, \ldots, x_8)$.

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables
- Verificar cada asignación para ver si cumple **todas** las restricciones
- Si se encuentra una asignación que cumple, se retorna como solución

Para un CSP (X, D, C), esto requiere revisar en general las tuplas de

$$D_1 \times D_2 \times \cdots \times D_n$$

Ejemplo

Para el problema de las 8 reinas, hay

$$8^8 = 16.777.216$$

tuplas posibles de la forma $(x_1, \ldots, x_8)$. ¿Cuántas hay en el sudoku?

Resolviendo CSPs

Para resolver un CSP, podemos partir con **fuerza bruta**

- Generar todas las asignaciones de variables
- Verificar cada asignación para ver si cumple **todas** las restricciones
- Si se encuentra una asignación que cumple, se retorna como solución

Para un CSP (X, D, C), esto requiere revisar en general las tuplas de

$$D_1 \times D_2 \times \cdots \times D_n$$

Ejemplo

Para el problema de las 8 reinas, hay

$$8^8 = 16.777.216$$

tuplas posibles de la forma $(x_1, \ldots, x_8)$. ¿Cuántas hay en el sudoku?

¿Cómo mejoramos esto?

Sumario

Resolviendo CSPs

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles
- Esa búsqueda puede *arrepentirse* si se rompe una restricción

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles
- Esa búsqueda puede *arrepentirse* si se rompe una restricción

Utilizaremos la estrategia algorítmica de **backtracking**, que incluye

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles
- Esa búsqueda puede *arrepentirse* si se rompe una restricción

Utilizaremos la estrategia algorítmica de **backtracking**, que incluye

- un conjunto de variables $X = \{x_1, \ldots, x_n\}$

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles
- Esa búsqueda puede *arrepentirse* si se rompe una restricción

Utilizaremos la estrategia algorítmica de **backtracking**, que incluye

- un conjunto de variables $X = \{x_1, \ldots, x_n\}$
- un conjunto de dominios **finitos** $D = \{D_1, \ldots, D_n\}$

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles
- Esa búsqueda puede *arrepentirse* si se rompe una restricción

Utilizaremos la estrategia algorítmica de **backtracking**, que incluye

- un conjunto de variables $X = \{x_1, \ldots, x_n\}$
- un conjunto de dominios **finitos** $D = \{D_1, \ldots, D_n\}$
- un conjunto de restricciones sobre variables

Resolviendo CSPs

Quizás no es necesario generar todas las tuplas

- Podemos *informar* la búsqueda en el espacio de tuplas posibles
- Esa búsqueda puede *arrepentirse* si se rompe una restricción

Utilizaremos la estrategia algorítmica de **backtracking**, que incluye

- un conjunto de variables $X = \{x_1, \ldots, x_n\}$
- un conjunto de dominios **finitos** $D = \{D_1, \ldots, D_n\}$
- un conjunto de restricciones sobre variables

Backtracking es la forma central para resolver CSPs
(también se usa para otros problemas)

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema
3. Si no es así, nos **retractamos** y deshacemos la asignación de x_k

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema
3. Si no es así, nos **retractamos** y deshacemos la asignación de x_k

El paso de retractarse se conoce como **backtrack**

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema
3. Si no es así, nos **retractamos** y deshacemos la asignación de x_k

El paso de retractarse se conoce como **backtrack**

- Permite descartar tuplas que violan alguna restricción

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema
3. Si no es así, nos **retractamos** y deshacemos la asignación de x_k

El paso de retractarse se conoce como **backtrack**

- Permite descartar tuplas que violan alguna restricción
- Lo hacemos sin necesidad de conocer la tupla completa

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema
3. Si no es así, nos **retractamos** y deshacemos la asignación de x_k

El paso de retractarse se conoce como **backtrack**

- Permite descartar tuplas que violan alguna restricción
- Lo hacemos sin necesidad de conocer la tupla completa
- Nos ahorramos revisar $|D_{k+1}| \times \cdots \times |D_n|$ tuplas

Backtracking

La estrategia de *backtracking* se basa en el siguiente principio

1. Realizar una asignación de la variable x_k cuando ya se han asignado $x_1, \ldots, x_{k-1}$
2. Se verifica si la nueva asignación **parcial** $x_1, \ldots, x_{k-1}, x_k$ puede terminar en una solución al problema
3. Si no es así, nos **retractamos** y deshacemos la asignación de x_k

El paso de retractarse se conoce como **backtrack**

- Permite descartar tuplas que violan alguna restricción
- Lo hacemos sin necesidad de conocer la tupla completa
- Nos ahorramos revisar $|D_{k+1}| \times \cdots \times |D_n|$ tuplas

Backtracking es igual o más rápido que la fuerza bruta

Backtracking

¿Tiene solución el siguiente tablero?

								9
	7					6		8
						1		4
				3				2
		1			5	3		7
	5							3
					9			5

Backtracking

Queremos garantías sobre la existencia de soluciones

Backtracking

Queremos garantías sobre la existencia de soluciones

- Si el problema tiene solución, queremos saberlo

Backtracking

Queremos garantías sobre la existencia de soluciones

- Si el problema tiene solución, queremos saberlo
- Si no tiene, también queremos saberlo

Backtracking

Queremos garantías sobre la existencia de soluciones

- Si el problema tiene solución, queremos saberlo
- Si no tiene, también queremos saberlo

Podemos responder recursivamente la pregunta

Dado un problema, ¿es posible resolverlo?

Backtracking

Queremos garantías sobre la existencia de soluciones

- Si el problema tiene solución, queremos saberlo
- Si no tiene, también queremos saberlo

Podemos responder recursivamente la pregunta

Dado un problema, ¿es posible resolverlo?

aprovechando que extender una asignación parcial

$$(x_1, \ldots, x_{k-1}) \to (x_1, \ldots, x_{k-1}, x_k)$$

genera una nueva instancia del problema

Backtracking

Queremos garantías sobre la existencia de soluciones

- Si el problema tiene solución, queremos saberlo
- Si no tiene, también queremos saberlo

Podemos responder recursivamente la pregunta

Dado un problema, ¿es posible resolverlo?

aprovechando que extender una asignación parcial

$$(x_1, \ldots, x_{k-1}) \to (x_1, \ldots, x_{k-1}, x_k)$$

genera una nueva instancia del problema

Hacemos *Backtracking* para la nueva instancia

Backtracking: idea de pseudocódigo

Backtracking: idea de pseudocódigo

input : Conjunto de variables sin asignar X, dominios D, restricciones R

```
isSolvable(X, D, R):
    if X = ∅ : return true
    x ← alguna variable de X
    for v ∈ D_x :
        if x = v no rompe R :
            x ← v
            if isSolvable(X − {x}, D, R) :
                return true
            x ← ∅
    return false
```

Backtracking: idea de pseudocódigo

input : Conjunto de variables sin asignar X, dominios D, restricciones R

$\texttt{isSolvable}(X, D, R)$**:**

    if $X = \varnothing$ **: return true**

    $x \leftarrow$ alguna variable de X

    for $v \in D_x$ **:**

        if $x = v$ *no rompe* R **:**

            $x \leftarrow v$

            if $\texttt{isSolvable}(X - \{x\}, D, R)$ **:**

                return true

            $x \leftarrow \varnothing$

    return false

Esto es solo una orientación: las variables, argumentos y estructura dependerá del problema particular

Problema de las 8 reinas

Problema de las 8 reinas

A continuación, un algoritmo para determinar si una asignación parcial de las 8 reinas puede dar lugar a una solución válida

Problema de las 8 reinas

A continuación, un algoritmo para determinar si una asignación parcial de las 8 reinas puede dar lugar a una solución válida

input **:** Arreglo $T[0 \dots 7]$,
índice $0 \leq i \leq 8$

output: **true** ssi hay solución

```
Queens(T, i):
    if i = 8 : return true
    for v = 0 ... 7 :
        if Check(T, i, v) :
            T[i] ← v
            if Queens(T, i + 1) :
                return true
    return false
```

Problema de las 8 reinas

A continuación, un algoritmo para determinar si una asignación parcial de las 8 reinas puede dar lugar a una solución válida

input : Arreglo $T[0 \dots 7]$,
índice $0 \leq i \leq 8$
output: **true** ssi hay solución

```
Queens(T, i):
    if i = 8 : return true
    for v = 0...7 :
        if Check(T, i, v) :
            T[i] ← v
            if Queens(T, i + 1) :
                return true
    return false
```

input : Arreglo $T[0 \dots 7]$,
índices $0 \leq i, j \leq 7$
output: **false** ssi es ilegal

```
Check(T, i, v):
    for j = 0...i − 1 :
        if v = T[i] :
            return false
        if |(v−T[j])/(i − j)| = 1 :
            return false
    return true
```

Problema de las 8 reinas

A continuación, un algoritmo para determinar si una asignación parcial de las 8 reinas puede dar lugar a una solución válida

input : Arreglo $T[0 \dots 7]$,
índice $0 \leq i \leq 8$
output: true ssi hay solución

```
Queens(T, i):
    if i = 8 : return true
    for v = 0...7 :
        if Check(T, i, v) :
            T[i] ← v
            if Queens(T, i + 1) :
                return true
    return false
```

input : Arreglo $T[0 \dots 7]$,
índices $0 \leq i, j \leq 7$
output: false ssi es ilegal

```
Check(T, i, v):
    for j = 0...i − 1 :
        if v = T[i] :
            return false
        if |(v−T[j])/(i − j)| = 1 :
            return false
    return true
```

¿Cómo podemos modificar el algoritmo para obtener una solución?

Complejidad

El análisis de complejidad del *backtracking* involucra el conteo de tuplas posibles

Complejidad

El análisis de complejidad del *backtracking* involucra el conteo de tuplas posibles

- En un conjunto de n variables $X = \{x_1, \ldots, x_n\}$

Complejidad

El análisis de complejidad del *backtracking* involucra el conteo de tuplas posibles

- En un conjunto de n variables $X = \{x_1, \ldots, x_n\}$
- con valores posibles en dominios $D = \{D_1, \ldots, D_n\}$

Complejidad

El análisis de complejidad del *backtracking* involucra el conteo de tuplas posibles

- En un conjunto de n variables $X = \{x_1, \ldots, x_n\}$
- con valores posibles en dominios $D = \{D_1, \ldots, D_n\}$
- tenemos $|D_1| \times |D_2| \times \cdots \times |D_n|$ tuplas posibles

Complejidad

El análisis de complejidad del *backtracking* involucra el conteo de tuplas posibles

- En un conjunto de n variables $X = \{x_1, \dots, x_n\}$
- con valores posibles en dominios $D = \{D_1, \dots, D_n\}$
- tenemos $|D_1| \times |D_2| \times \cdots \times |D_n|$ tuplas posibles

Luego, en el caso particular de que $|D_i| = K$ para todo i,

Complejidad

El análisis de complejidad del *backtracking* involucra el conteo de tuplas posibles

- En un conjunto de n variables $X = \{x_1, \ldots, x_n\}$
- con valores posibles en dominios $D = \{D_1, \ldots, D_n\}$
- tenemos $|D_1| \times |D_2| \times \cdots \times |D_n|$ tuplas posibles

Luego, en el caso particular de que $|D_i| = K$ para todo i,

- revisar todas las tuplas es $\mathcal{O}(K^n)$

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas**

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas** $\mathcal{O}(K^n)$

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas** $\mathcal{O}(K^n)$
- el backtracking puede revisar menos tuplas, pero sigue siendo proporcional

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas** $\mathcal{O}(K^n)$
- el backtracking puede revisar menos tuplas, pero sigue siendo proporcional $\mathcal{O}(K^n)$

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas** $\mathcal{O}(K^n)$
- el backtracking puede revisar menos tuplas, pero sigue siendo proporcional $\mathcal{O}(K^n)$

Es decir, asintóticamente estas estrategias tienen la misma complejidad

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas** $\mathcal{O}(K^n)$
- el backtracking puede revisar menos tuplas, pero sigue siendo proporcional $\mathcal{O}(K^n)$

Es decir, asintóticamente estas estrategias tienen la misma complejidad

¿Cuál es más rápido en la práctica?

Complejidad

La complejidad de las posibles soluciones para CSP cumplen,

- la estrategia de fuerza bruta revisa **todas las tuplas** $\mathcal{O}(K^n)$
- el backtracking puede revisar menos tuplas, pero sigue siendo proporcional $\mathcal{O}(K^n)$

Es decir, asintóticamente estas estrategias tienen la misma complejidad

¿Cuál es más rápido en la práctica?

No olvidar: *Backtracking* es igual o más rápido que la fuerza bruta

Otra interpretación del backtracking

Otra interpretación del backtracking

Podemos pensar en la estrategia de backtracking como **búsqueda en un grafo implícito**

Otra interpretación del backtracking

Podemos pensar en la estrategia de backtracking como **búsqueda en un grafo implícito**

Los CSP generan muchas tuplas posibles como asignaciones para las variables de X

Otra interpretación del backtracking

Podemos pensar en la estrategia de backtracking como **búsqueda en un grafo implícito**

Los CSP generan muchas tuplas posibles como asignaciones para las variables de X

- Cada posible asignación genera un camino

Otra interpretación del backtracking

Podemos pensar en la estrategia de backtracking como **búsqueda en un grafo implícito**

Los CSP generan muchas tuplas posibles como asignaciones para las variables de X

- Cada posible asignación genera un camino
- Las nuevas asignaciones abren nuevos caminos

Otra interpretación del backtracking

Podemos pensar en la estrategia de backtracking como **búsqueda en un grafo implícito**

Los CSP generan muchas tuplas posibles como asignaciones para las variables de X

- Cada posible asignación genera un camino
- Las nuevas asignaciones abren nuevos caminos
- A la colección de todas estas alternativas le llamamos **grafo implícito**

Otra interpretación del backtracking

Podemos pensar en la estrategia de backtracking como **búsqueda en un grafo implícito**

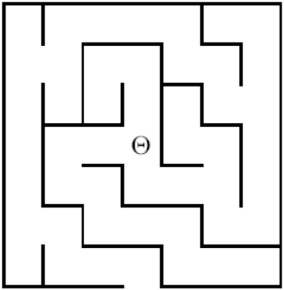
Los CSP generan muchas tuplas posibles como asignaciones para las variables de X

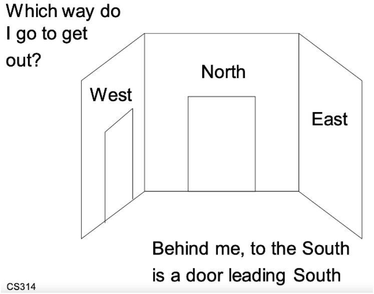
- Cada posible asignación genera un camino
- Las nuevas asignaciones abren nuevos caminos
- A la colección de todas estas alternativas le llamamos **grafo implícito**

El ejemplo por excelencia para visualizar el grafo implícito
es el **problema de recorrer un laberinto**

Recorrido del laberinto

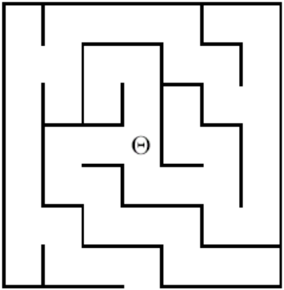
Supongamos que nos interesa salir de un laberinto dado que estamos en Θ

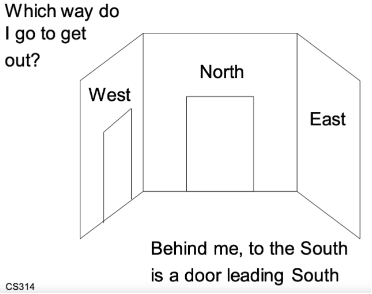




Recorrido del laberinto

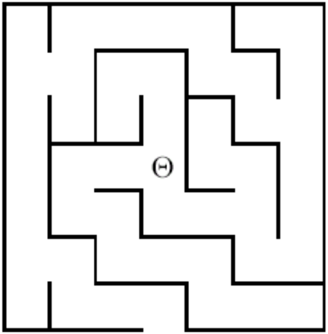
Supongamos que nos interesa salir de un laberinto dado que estamos en Θ





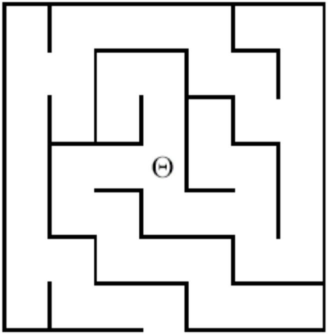
Podemos resolver este problema con *backtracking*

Recorrido del laberinto



Planteamos el problema como un CSP

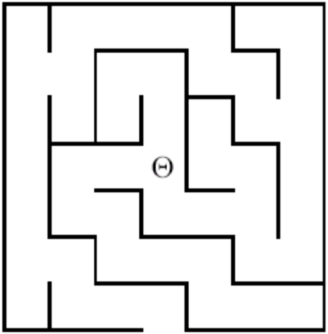
Recorrido del laberinto



Planteamos el problema como un CSP

- Variables?

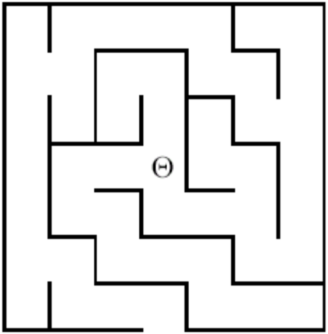
Recorrido del laberinto



Planteamos el problema como un CSP

- Variables?
- Dominios?

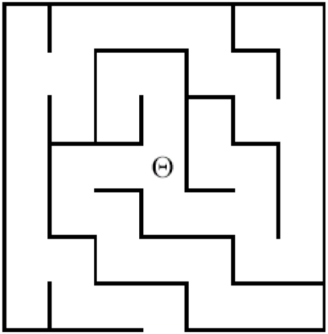
Recorrido del laberinto



Planteamos el problema como un CSP

- Variables?
- Dominios?
- Restricciones?

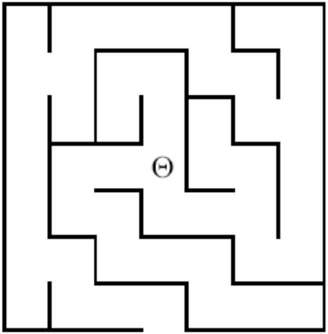
Recorrido del laberinto



Planteamos el problema como un CSP

- Variables?
- Dominios?
- Restricciones?
- Qué define el *éxito*?

Recorrido del laberinto



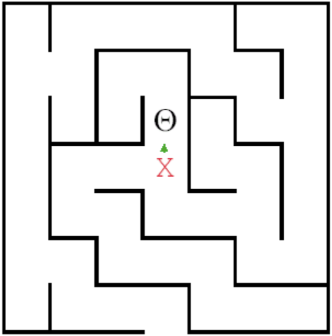
Planteamos el problema como un CSP

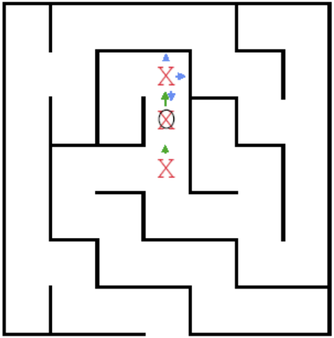
- Variables?
- Dominios?
- Restricciones?
- Qué define el *éxito*?

Caracterizamos por Θ la posición actual

Recorrido del laberinto

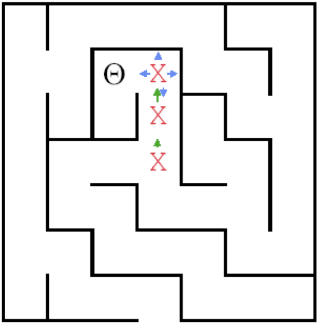
En cada nueva posición Θ solo podemos elegir dar un paso en las direcciones libres y distintas de aquella de la cual venimos

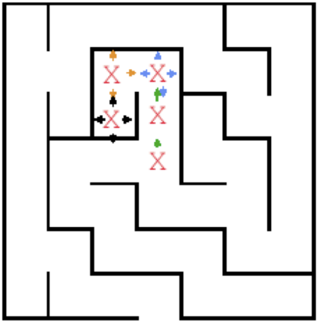




Recorrido del laberinto

Debemos hacer backtrack cuando llegamos a un camino sin salida: solo muros y celdas ya visitadas

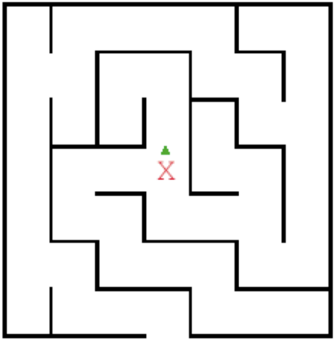


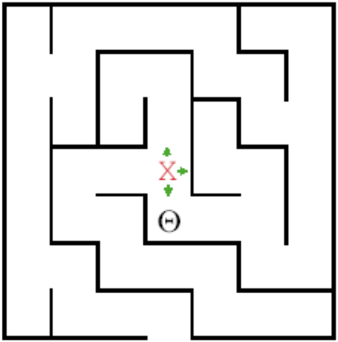


No hay más opciones: ¿hasta dónde nos *arrepentimos* con el backtrack?

Recorrido del laberinto

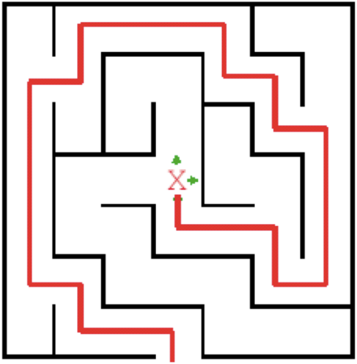
Sabemos que ir al norte no funcionó. Probamos otra opción yendo al sur.





Recorrido del laberinto

En este caso, logramos llegar a una solución que encuentra la salida



Recorrido del laberinto

Le agregamos etiquetas a las posiciones, de modo que sabemos cuáles hemos visitado (visited). Todas comienzan como nonvisited y la salida se marca como exit

input : Conjunto de variables sin asignar X, posición x, dominios D, restricciones R

`isSolvable`(X, x, D, R)**:**

    if $x = \mathit{exit}$ **: return true**
    if *visited* **: return false**
    $x \leftarrow \mathit{visited}$
    for $v \in \{N, E, S, W\}$ **:**
        if $x + v \neq \mathit{wall}$ **:**
            $x \leftarrow x + v$
            if `isSolvable`(X, x, D, R) **:**
                return true
            $x \leftarrow \mathit{nonvisited}$
    return false

Otros problemas habituales

Otros problemas habituales

Hay varios problemas clásicos que se resuelven mediante backtracking

Otros problemas habituales

Hay varios problemas clásicos que se resuelven mediante backtracking

- Recorrido del caballo de ajedrez (*Knight's tour problem*)

Otros problemas habituales

Hay varios problemas clásicos que se resuelven mediante backtracking

- Recorrido del caballo de ajedrez (*Knight's tour problem*)
- Problema de la mochila (capacidad versus número de items)

Otros problemas habituales

Hay varios problemas clásicos que se resuelven mediante backtracking

- Recorrido del caballo de ajedrez (*Knight's tour problem*)
- Problema de la mochila (capacidad versus número de items)
- Balance de carga

Otros problemas habituales

Hay varios problemas clásicos que se resuelven mediante backtracking

- Recorrido del caballo de ajedrez (*Knight's tour problem*)
- Problema de la mochila (capacidad versus número de items)
- Balance de carga
- Coloreo de mapas (Sudoku es un caso particular)

Otros problemas habituales

Hay varios problemas clásicos que se resuelven mediante backtracking

- Recorrido del caballo de ajedrez (*Knight's tour problem*)
- Problema de la mochila (capacidad versus número de items)
- Balance de carga
- Coloreo de mapas (Sudoku es un caso particular)

En general, puzzles NP-completos podemos atacarlos
con alguna idea de backtracking

Sumario

CSPs

Backtracking

Cierre

Objetivos de la clase

Objetivos de la clase

- ☐ Definir la clase de problemas de satisfacción de restricciones

Objetivos de la clase

- ☐ Definir la clase de problemas de satisfacción de restricciones
- ☐ Comprender la dificultan inherente a los CSP

Objetivos de la clase

- ☐ Definir la clase de problemas de satisfacción de restricciones
- ☐ Comprender la dificultan inherente a los CSP
- ☐ Comprender la estrategia de backtracking

Objetivos de la clase

- ☐ Definir la clase de problemas de satisfacción de restricciones
- ☐ Comprender la dificultan inherente a los CSP
- ☐ Comprender la estrategia de backtracking
- ☐ Identificar pseudocódigo base para backtracking y sus partes

Objetivos de la clase

- ☐ Definir la clase de problemas de satisfacción de restricciones
- ☐ Comprender la dificultan inherente a los CSP
- ☐ Comprender la estrategia de backtracking
- ☐ Identificar pseudocódigo base para backtracking y sus partes
- ☐ Aplicar las ideas de backtracking para resolver algunos problemas