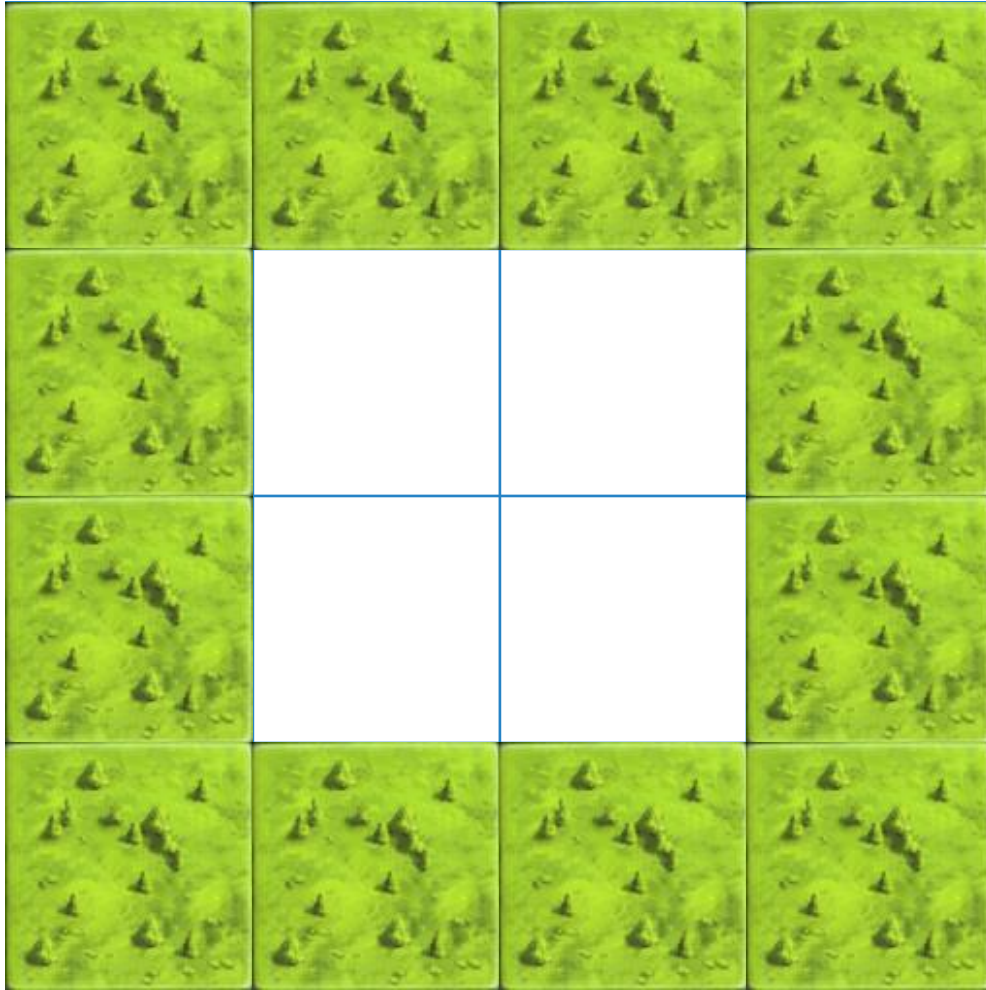


Carcassonne

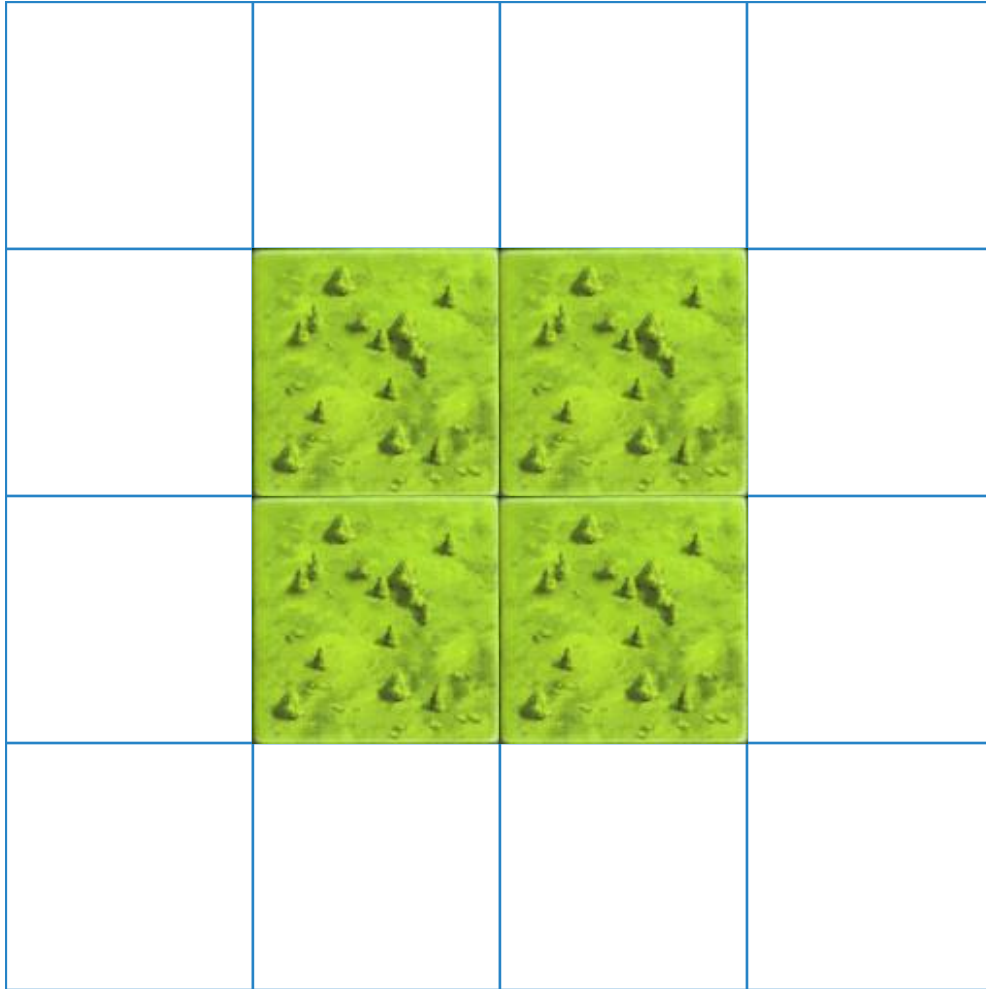


x 2



x 2

Carcassonne



x 2

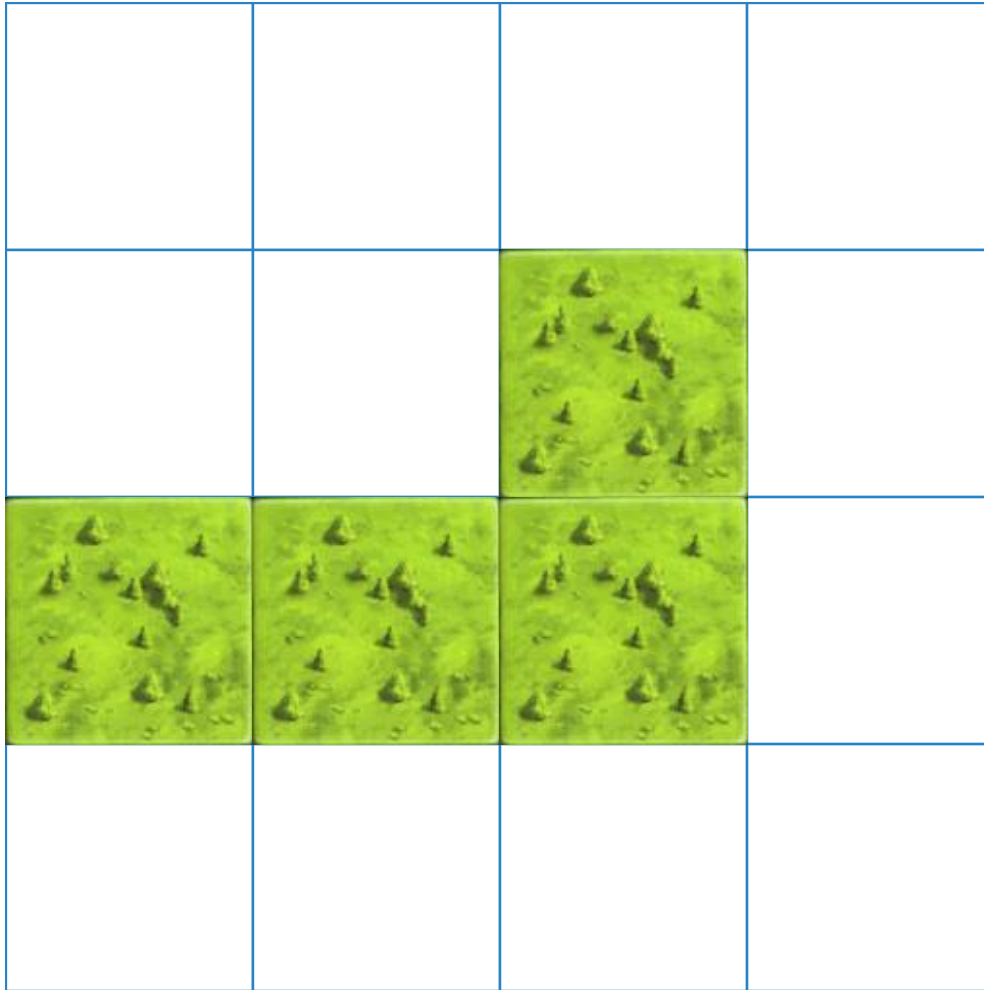


x 6



x 4

Carcassonne



x 1



x 5



x 5



x 1

Backtracking

La estrategia para resolver esto se conoce como **backtracking**

La idea es **descartar** permutaciones que violan alguna restricción

Eso significa que **siempre** es igual o más rápido que fuerza bruta

Descarte



La idea es **descartar** permutaciones que no llevan a una solución

Una forma de hacer esto es revisar las restricciones

¿Hay alguna otra manera?

is solvable(X, D, R, i):

if $i > |X|$, *return true*

$x \leftarrow x_i$

for $v \in d_i$:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X, D, R, i + 1$):

return true

$x \leftarrow \emptyset$

return false

is solvable(X, D, i):

if $i > |X|$, *return true*

$x \leftarrow x_i$

for $v \in d_i$:

if $x = v$ *no es válida*, *continue*

$x \leftarrow v$

if is solvable($X, D, i + 1$):

return true

$x \leftarrow \emptyset$

return false

Podas

Son restricciones **adicionales** que le ponemos al problema

Se **deducen** de las restricciones originales

Pueden ser más costosas de revisar, pero suelen valerlo

Sudoku

				7				
	6					4	9	2
						7	5	3
		4						
					3			
								6
2								
			7					
						9		

Sudoku

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

Sudoku

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

Sudoku

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

Dominios



No todos los valores de un dominio son siempre válidos

Depende de las restricciones que afectan a la variable

¿Cómo va cambiando un dominio a medida que resolvemos?

Múltiples asignaciones



¿Será posible hacer más de una asignación por paso?

¿En que circunstancias tiene sentido?

is solvable(X, D, R, i):

if $i > |X|$, *return true*

$x \leftarrow x_i$

for $v \in d_i$:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X, D, R, i + 1$):

return true

$x \leftarrow \emptyset$

return false

is solvable(X, D, R, i):

if $i > |X|$, *return true*

$x \leftarrow x_i$

for $v \in d_i$:

if $x = v$ viola R , *continue*

$x \leftarrow v$, *propagar*

if is solvable($X, D, R, i + 1$):

return true

$x \leftarrow \emptyset$, *propagar*

return false

Propagación

Al asignar, es posible invalidar valores del dominio de otra variable

Es útil **propagar** esta información luego de una asignación

Si $|d_i| = 1$, entonces podemos asignar x_i y volver a **propagar**

Hay que tener más cuidado al deshacer las asignaciones

Sudoku

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

Sudoku

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

Sudoku

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

Orden de asignación



A la hora de resolver un problema de asignación,

¿Afecta el orden en que asignamos las variables?

¿Afecta el orden en que probamos sus posibles valores?

is solvable(X, D, R, i):

if $i > |X|$, *return true*

$x \leftarrow x_i$

for $v \in d_i$:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X, D, R, i + 1$):

return true

$x \leftarrow \emptyset$

return false

is solvable(X, D, R):

if $X = \emptyset$, *return true*

$x \leftarrow$ la mejor variable de X

for $v \in d_i$, de mejor a peor:

if $x = v$ viola R , *continue*

$x \leftarrow v$

if is solvable($X - \{x\}, D, R$):

return true

$x \leftarrow \emptyset$

return false

Heurísticas

Cuando un problema es muy difícil, usamos **heurísticas**

Las **heurísticas** tratan de aproximar la realidad

Son una idea de que tan buena es una opción

Sudoku

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

Sudoku

								1
								9
	7					6		8
						1		4
				3				2
		1			5	3		7
	5							3
								16
					9			5

Sudoku

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

Sudoku

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

is solvable(X, D):

if $X = \emptyset$, *return true*

$x \leftarrow$ la mejor variable de X

for $v \in d_i$, de mejor a peor:

if $x = v$ no es válida, *continue*

$x \leftarrow v$, propagar

if is solvable($X - \{x\}, D$):

return true

$x \leftarrow \emptyset$, propagar

return false

Usos de backtracking

Esta estrategia no solo sirve para problemas de **asignación**

Sirve siempre cuando es necesario probar todo

Por ejemplo, problemas de **planificación**, u **optimización**