ESTRATEGIAS DE PROGRAMACIÓN

Algoritmos y Programación

Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

28 de septiembre de 2023

Estrategias

- Fuerza bruta (*brute force*)
- Vuelta atrás (*backtracking*)
- Voráz (*greedy*)

- Divide y vencerás
  - Reduce y vencerás

Técnica

- Programación Dinámica

Introducción

- **Objetivo**: Encontrar todas las combinaciones de 3 variables binarias (x_1, x_2, x_3) en las que no haya valores iguales consecutivos.

  - Ejemplo de solución: 1-0-1

Ejemplo: Solución por fuerza bruta

- Utilizando un iterador generamos y comprobamos todas las combinaciones (*generation & test*)

$x_1\ x_2\ x_3$

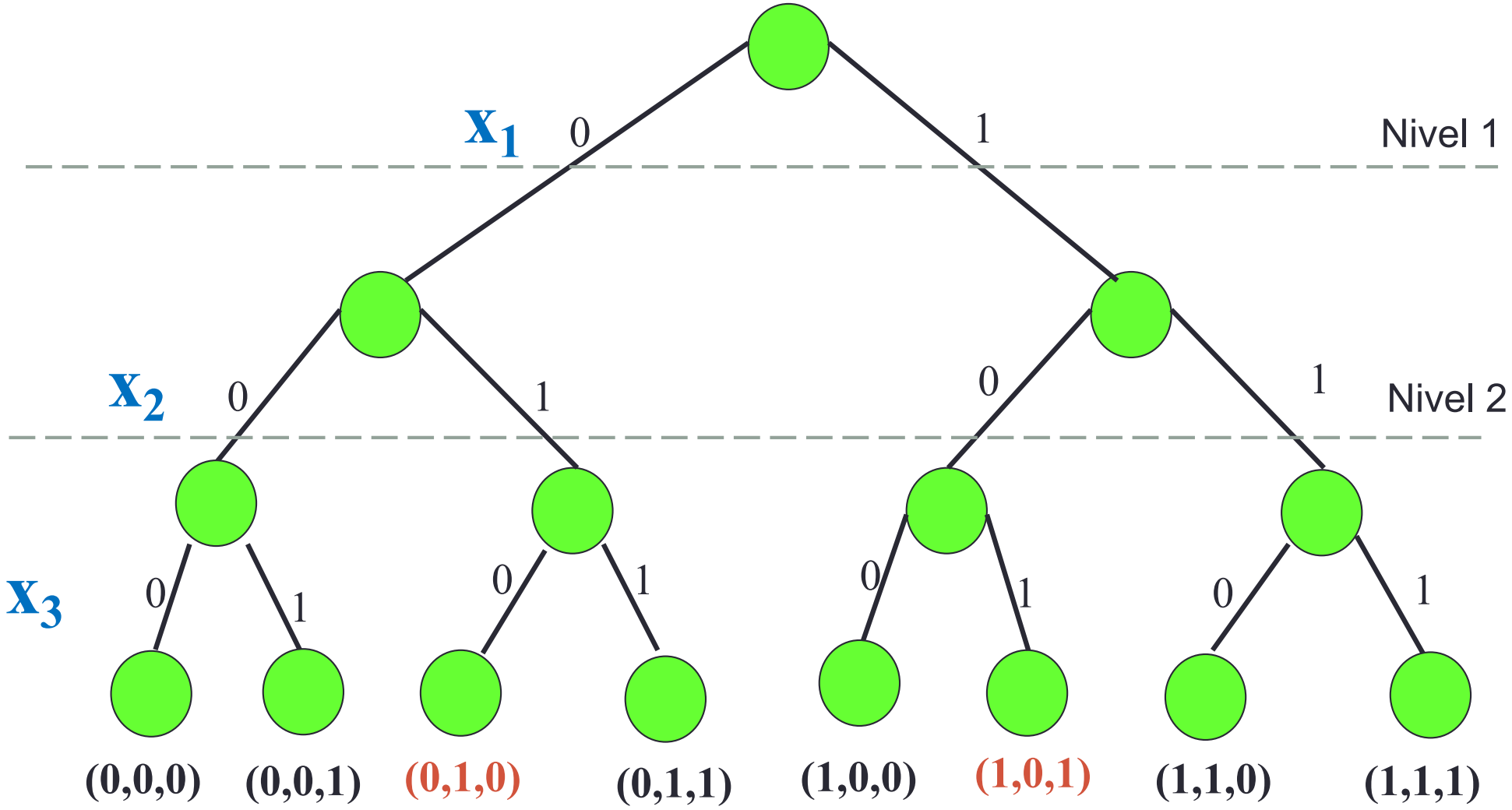
0-0-0
0-0-1
0-1-0
0-1-1
1-0-0
1-0-1
1-1-0
1-1-1

Coste de ejecución = $2^3 = 8$

Veamos gráficamente estas combinaciones

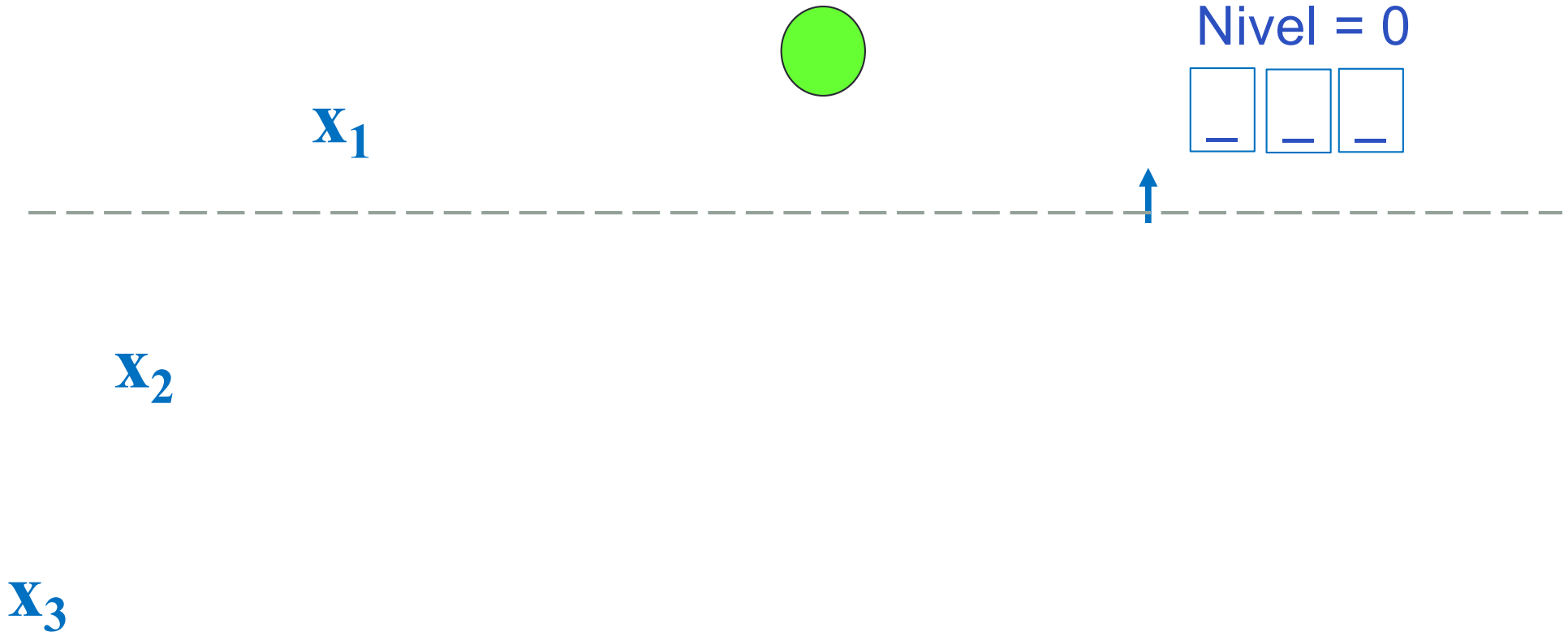
Ejemplo: Solución por fuerza bruta

Asociamos cada variable a un nivel de profundidad (x_1, x_2, x_3) para ver el árbol de combinaciones



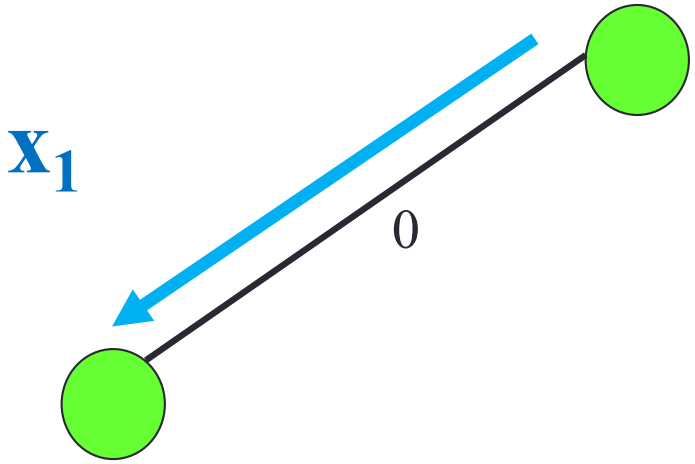
Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



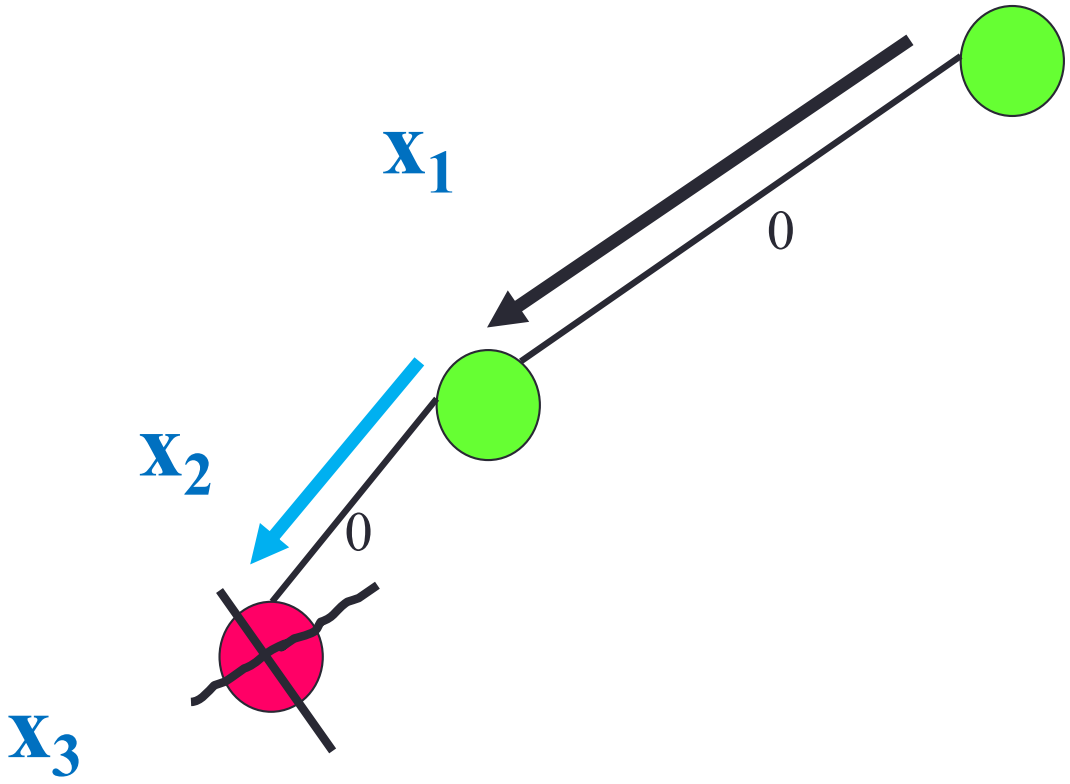
Nivel = 1

0	_	_

x_3

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas

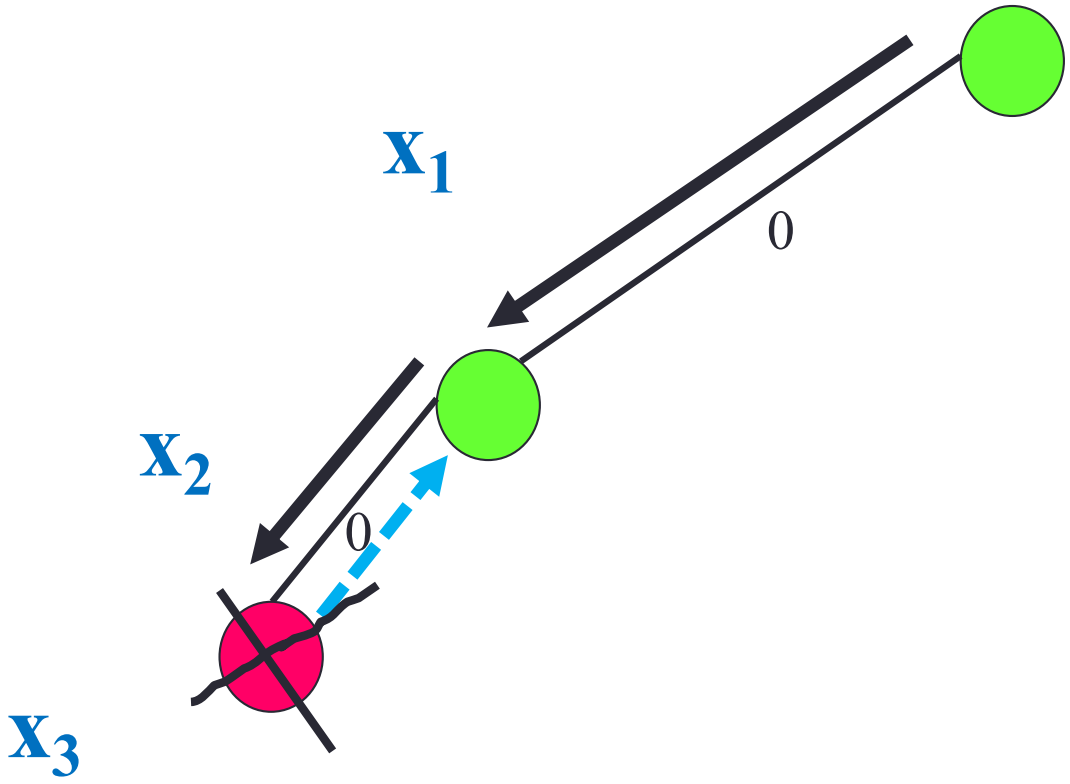


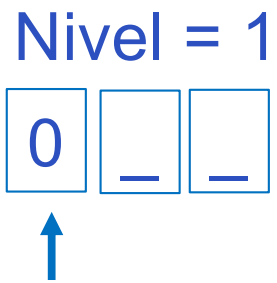
Nivel = 2

0	0	_

Ejemplo: Solución por Backtracking

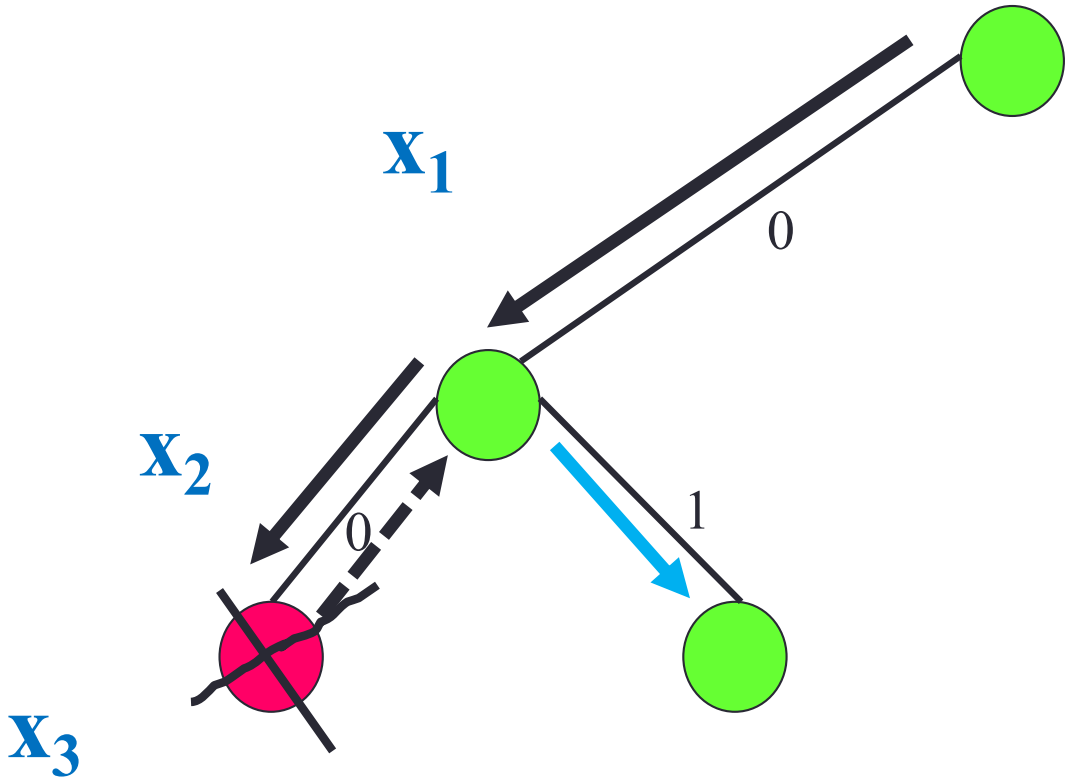
- Con backtracking evitamos combinaciones no válidas





Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas

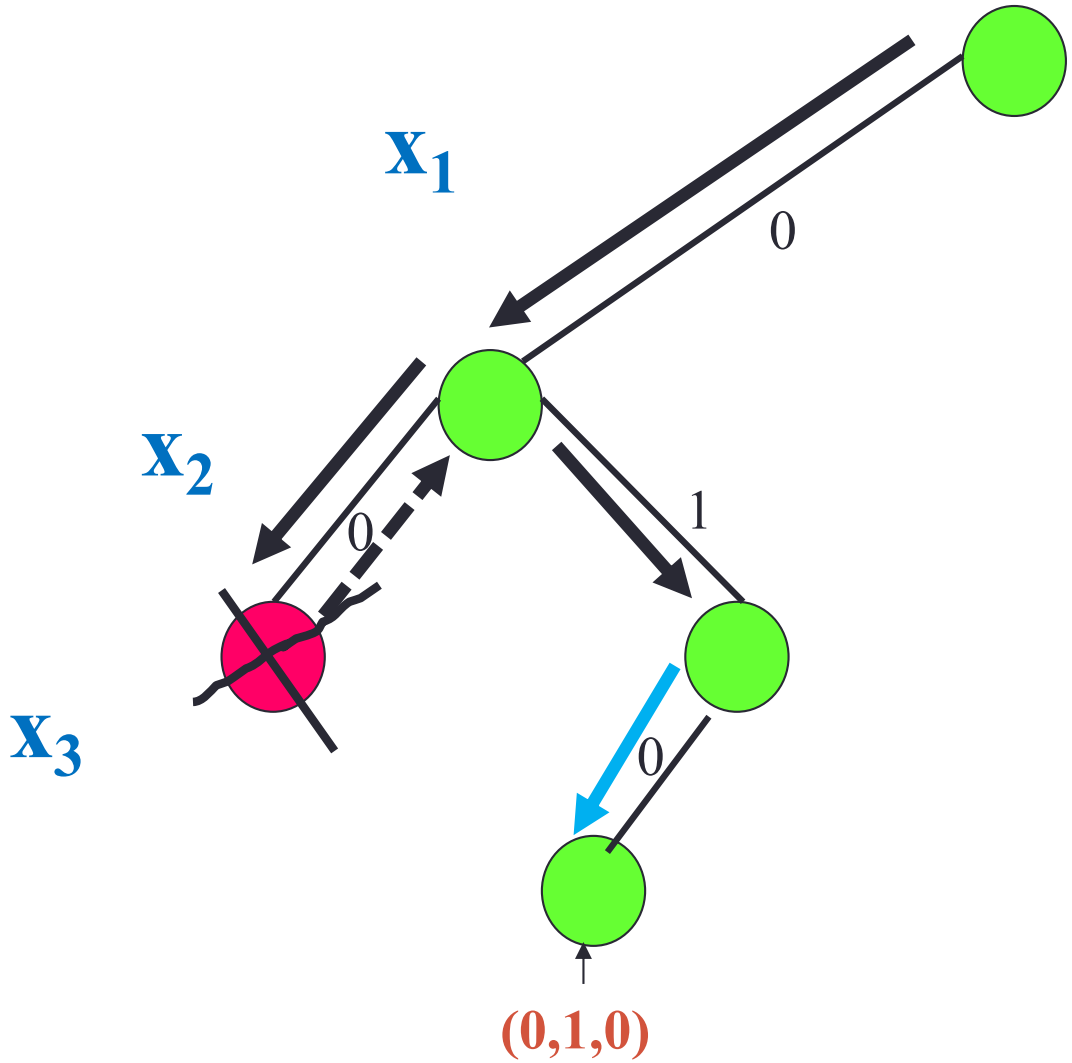


Nivel = 2

0	1	_

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas

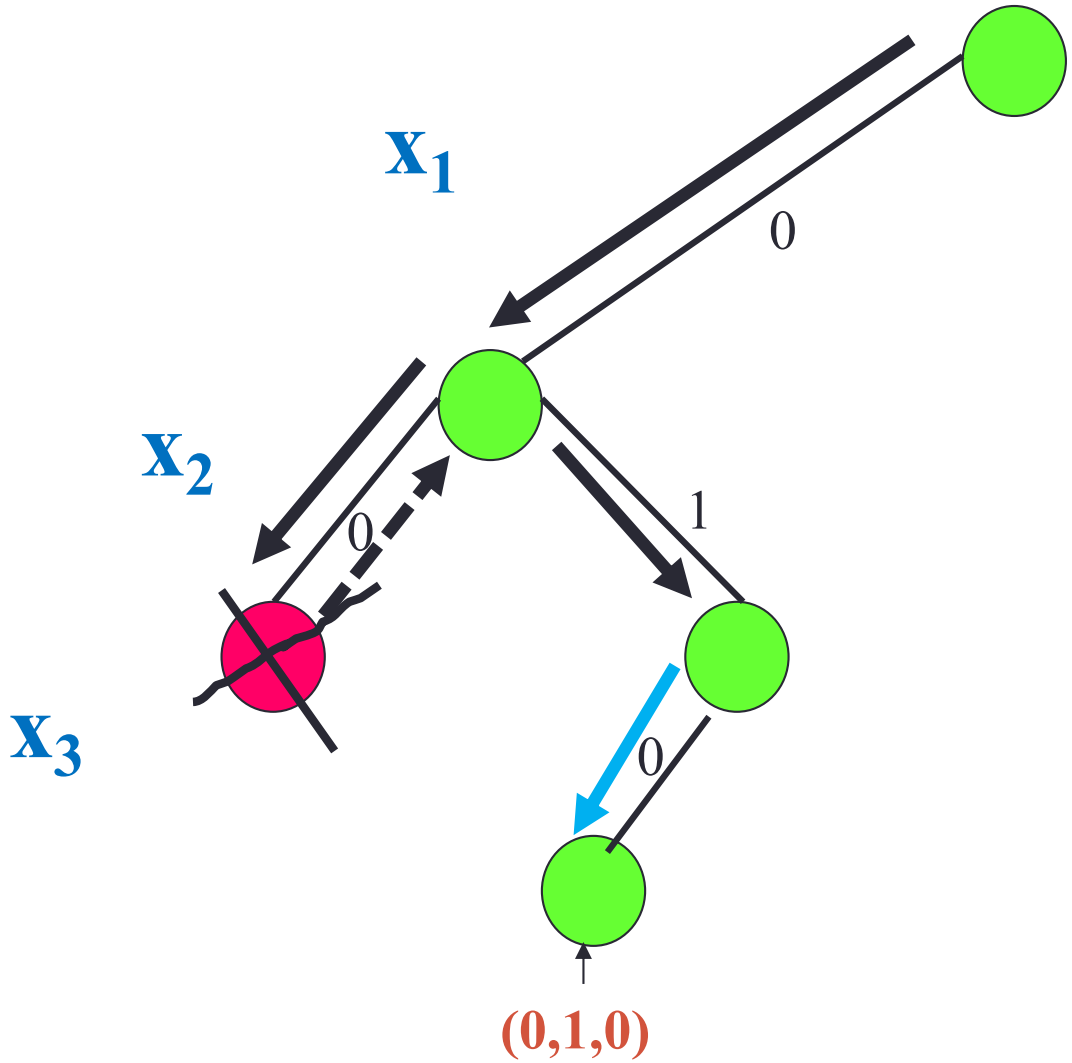


Nivel = 3

0	1	0

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



Nivel = 3

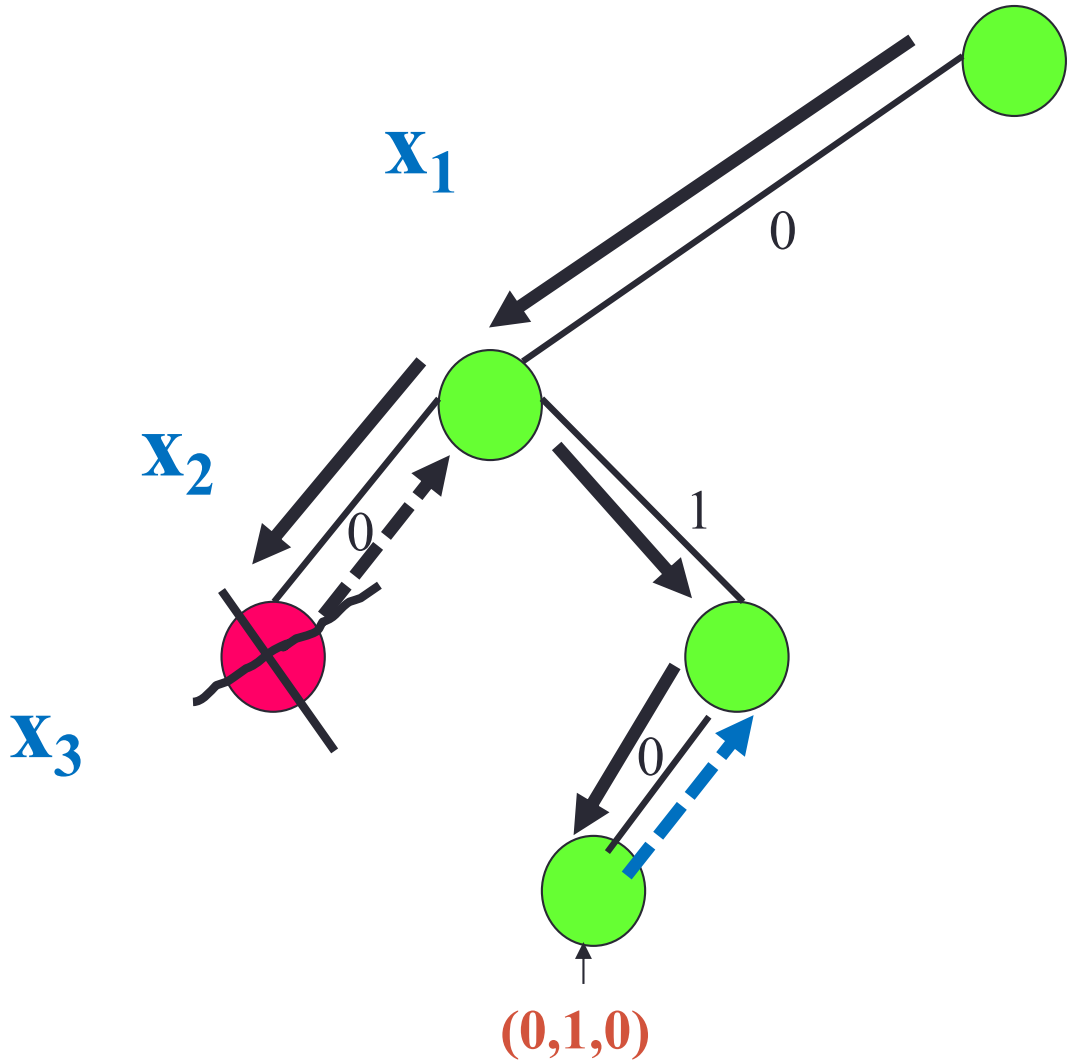
0	1	0

Solución
0 1 0

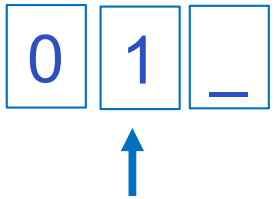
Guardamos o procesamos esta solución

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



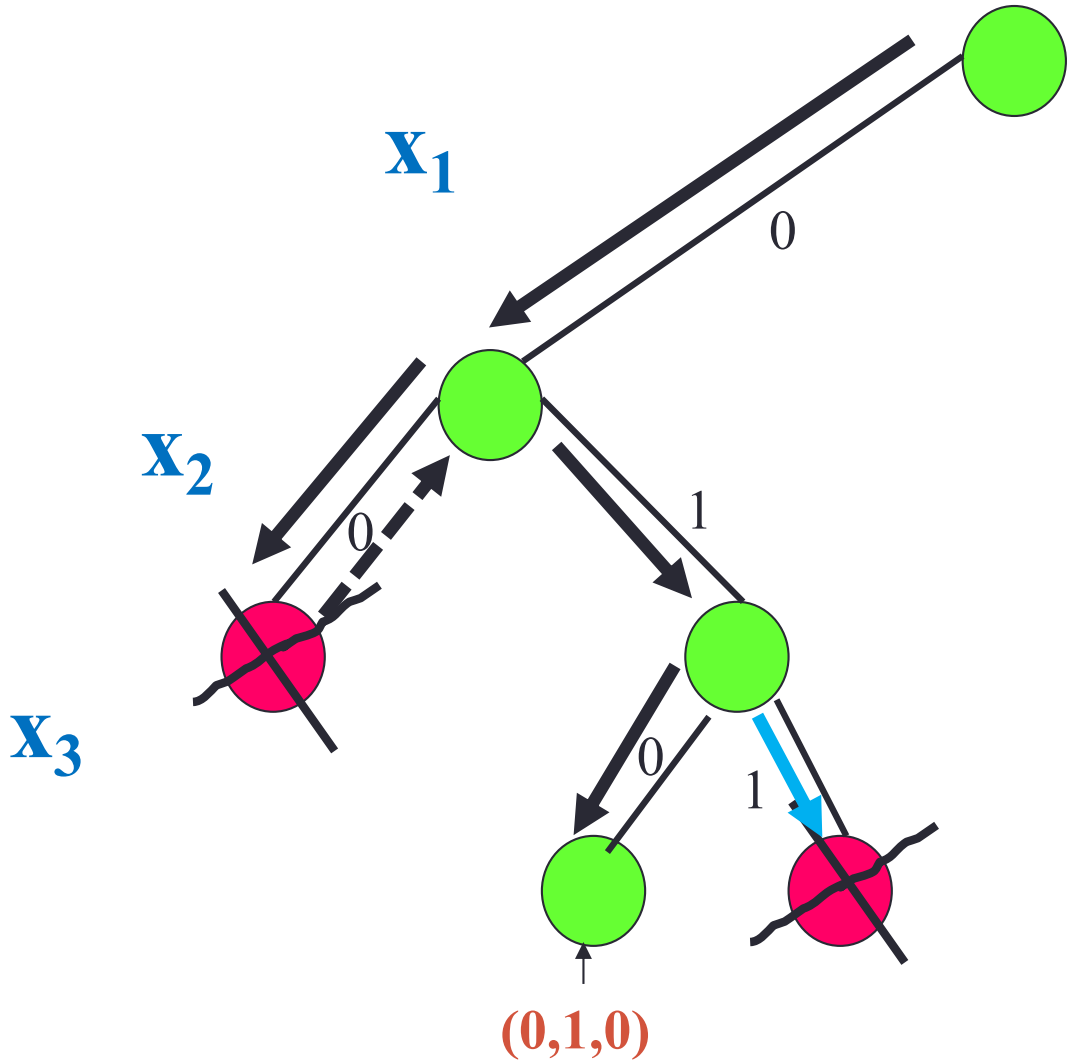
Nivel = 2



Soluciones
0 1 0

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



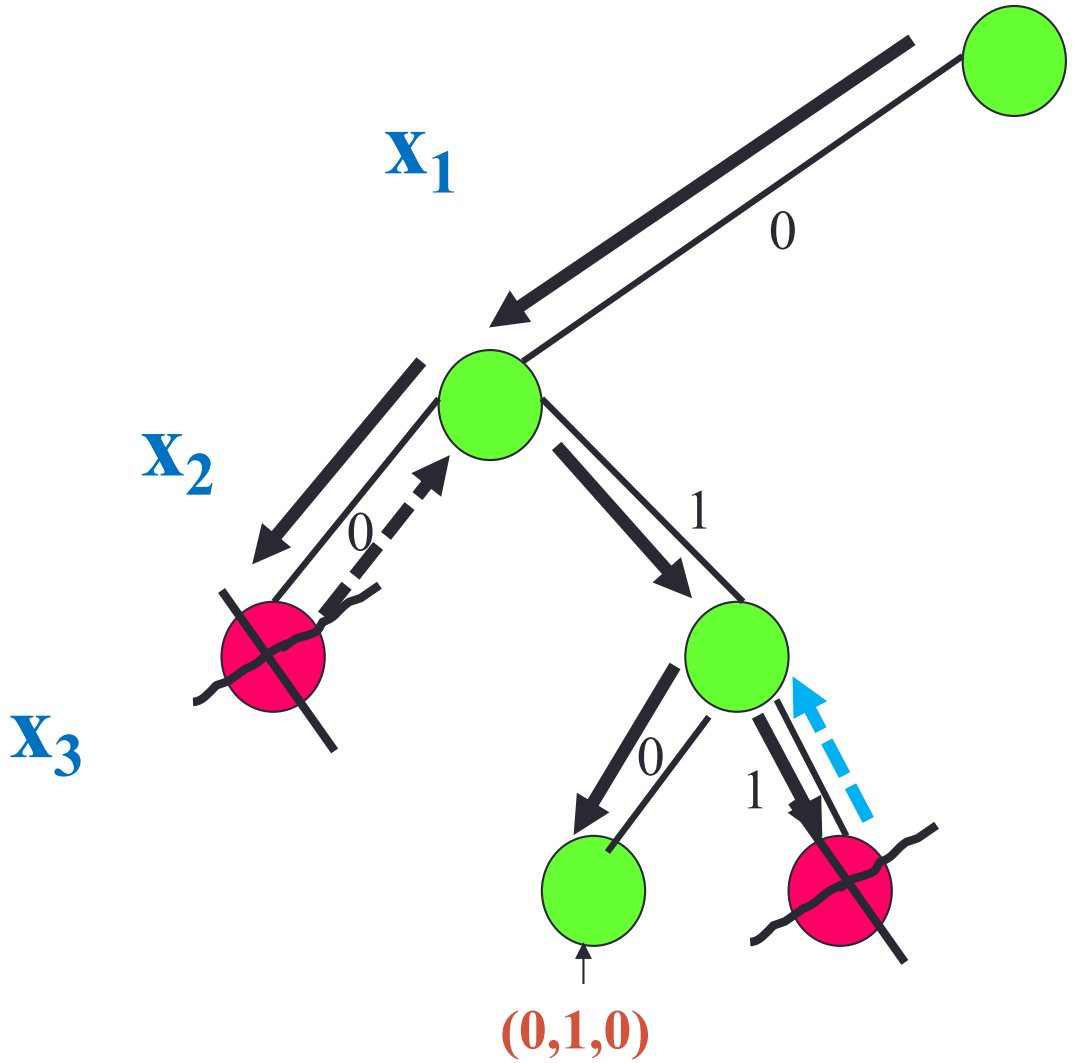
Nivel = 3

0	1	1

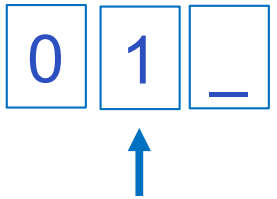
Soluciones
0 1 0

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



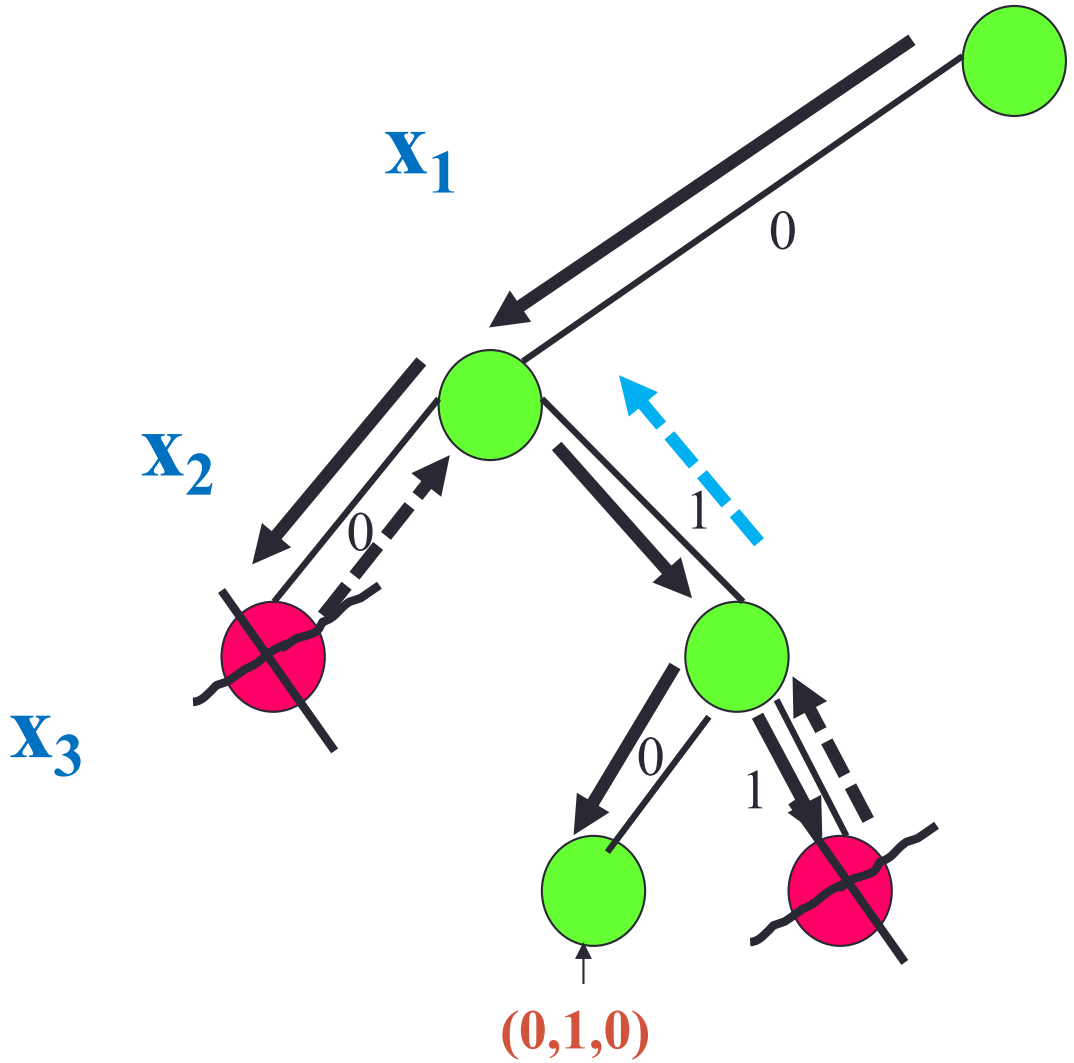
Nivel = 2



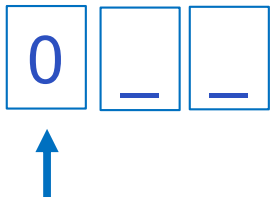
Soluciones
0 1 0

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



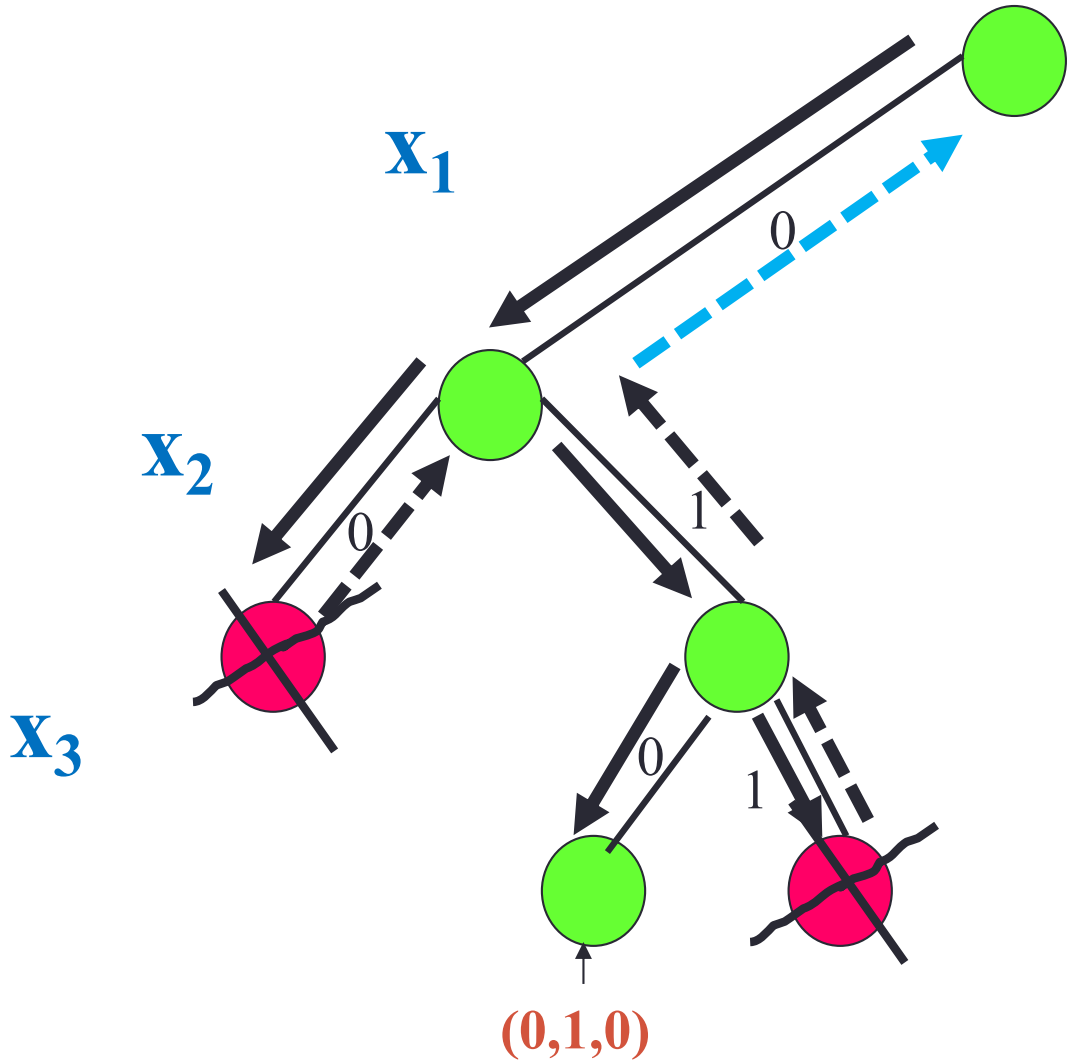
Nivel = 1



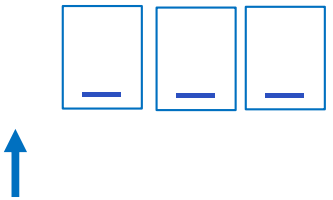
Soluciones
0 1 0

Ejemplo: Solución por Backtracking

- Con backtracking evitamos combinaciones no válidas



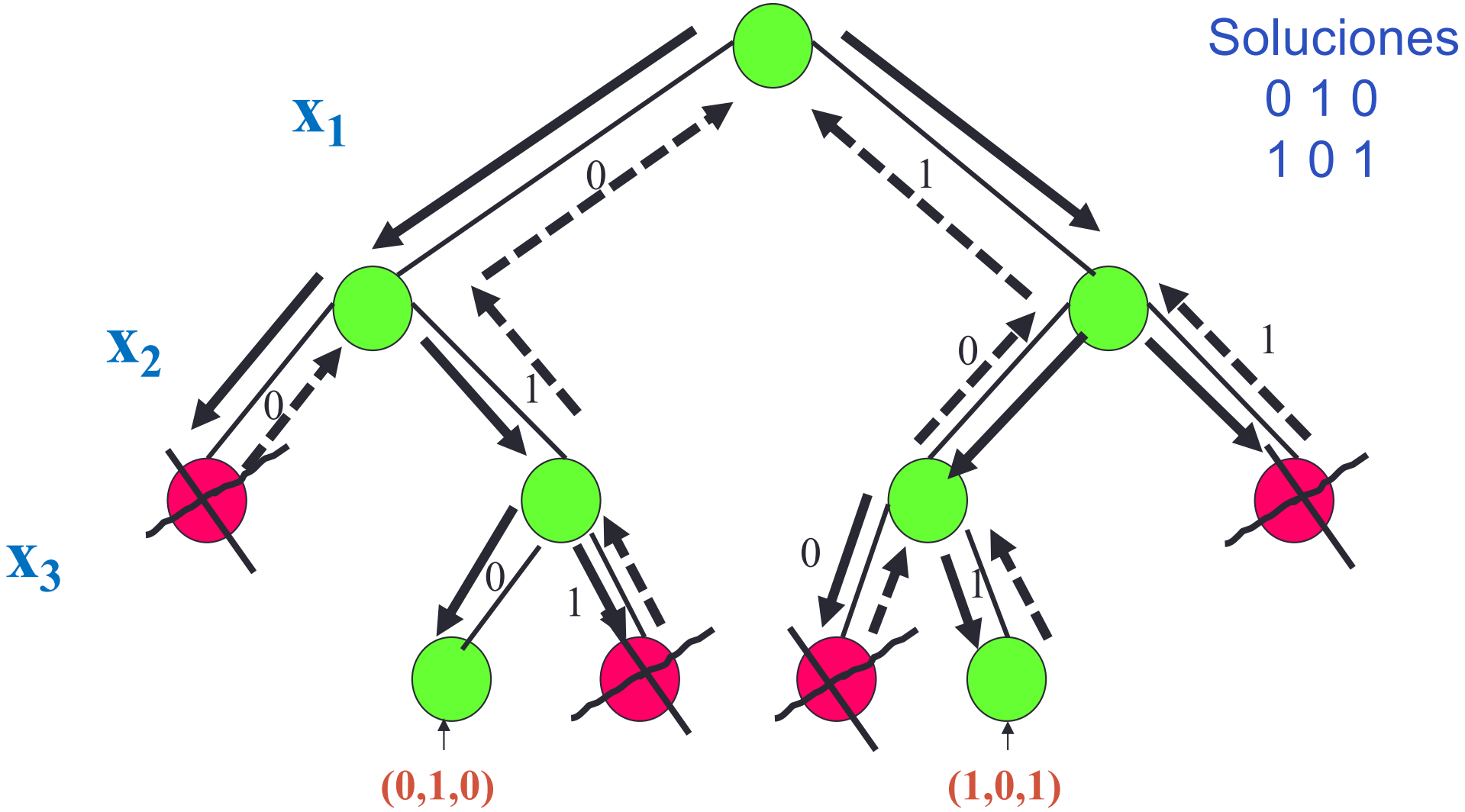
Nivel = 0



Soluciones
0 1 0

Ejemplo: Solución por Backtracking

- Completando el recorrido …



Vuelta atrás (*Backtracking*)

- Estrategia general de búsqueda de soluciones en problemas de optimización (*constraint satisfaction problems*)

- Características
  - Construye de formal incremental candidatos parciales
  - Cada candidato parcial añade 1 componente a la solución
  - Conceptualmente los candidatos parciales se representan como nodos de una estructura árbol (*search tree*)
  - Descarta (*poda del árbol*) los candidatos que no llevan a la solución
  - Los nodos hoja del árbol son los candidatos que no pueden extenderse

https://en.wikipedia.org/wiki/Backtracking

Vuelta atrás (*Backtracking*)

Algoritmo Genérico

1. Generamos una combinación componente a componente
2. Evaluamos si nos puede llevar hacia la solución
3. Si no satisface alguna restricción del problema descartamos esta solución (y todas las que dependan de ella)
4. Volvemos al paso 1

Puede programarse con código recursivo o con código iterativo

Ejemplo: Solución recursiva mediante Backtracking

```
def solve(num_digits):

    def is_valid_solution(solution, level):
        # Recorre la 'solucion' de izquierda a derecha
        # comprobando que no tiene digitos consecutivos
        # iguales.
        ...

    solution = [-1] * num_digits

    def dfs(level):
        ...

    # Comenzamos el recorrido en profundidad
    dfs(level=0)
    return ...      # retornamos las soluciones
```

Veremos este código en la siguiente página

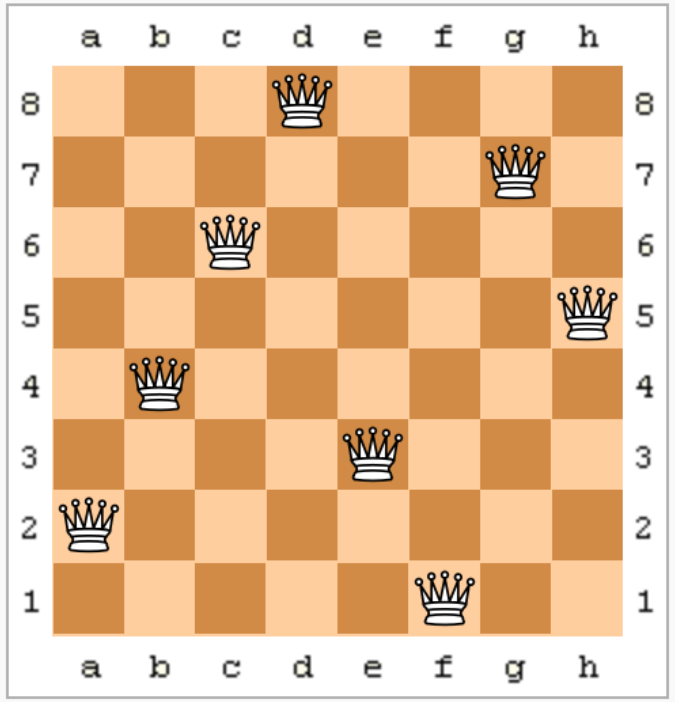
```
def dfs(level):
    # Si la solución que tengo construida hasta este nivel
    # no es válida subimos al nivel anterior ('backtrack')
    if not is_valid_solution(solution, level):
        return

    # Si tengo todos los digitos de una solución, la proceso
    # y continúo el recorrido DFS.
    elif level == num_digits:
        # ... (código que guarda o imprime esta solución)
        return


    else:
        # Continúo con el recorrido en profundidad
        for digit in ...
            solution[level] = digit   # Coloco un dígito
            dfs(level + 1)            #   Llamada recursiva

        solution[level] = -1;         # Quito el digito
        return
```

Resumen



Una posible solución entre las 92 posibles soluciones en un tablero de 8×8.

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

A Sudoku solved by backtracking.

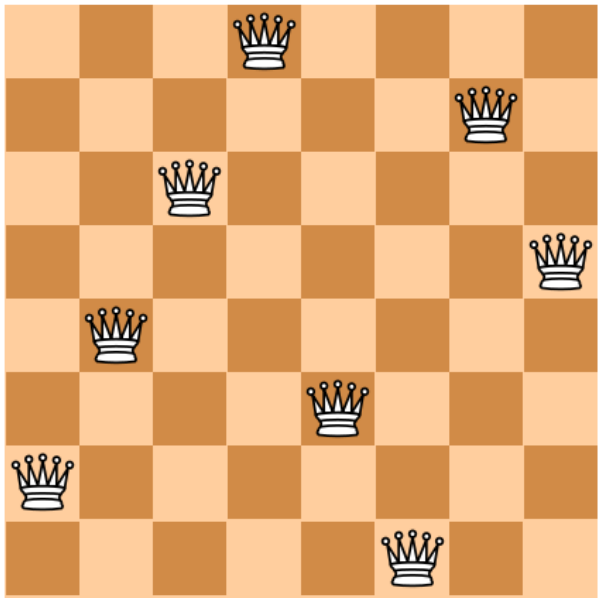
- Es básicamente una estrategia de fuerza bruta con poda
- Puede implementarse recursivamente o iterativamente

https://es.wikipedia.org/wiki/Problema_de_las_ocho_reinas
https://en.wikipedia.org/wiki/Sudoku_solving_algorithms

Fuerza Bruta

```
...
[6, 4, 2, 0, 5, 7, 1, 3]
[7, 1, 3, 0, 6, 4, 2, 5]
[7, 1, 4, 2, 0, 6, 3, 5]
[7, 2, 0, 5, 1, 4, 6, 3]
[7, 3, 0, 2, 5, 1, 6, 4]
92 soluciones

time: 0m10.904s
```



N-Queens

Vuelta atrás (*Backtracking*)

```
...
[6, 4, 2, 0, 5, 7, 1, 3]
[7, 1, 3, 0, 6, 4, 2, 5]
[7, 1, 4, 2, 0, 6, 3, 5]
[7, 2, 0, 5, 1, 4, 6, 3]
[7, 3, 0, 2, 5, 1, 6, 4]
92 soluciones

time: 0m0.049s
```

Este ejercicio será nuestra siguiente práctica de laboratorio