Projeto e Análise de Algoritmos

Aula 28: Busca Inteligente para Problemas Intratáveis

DECOM/UFOP
2012/2 – 5º. Período
Anderson Almeida Ferreira
Material elaborado por:
Andréa Iabrudi Tavares

decom
departamento de computação

Objetivos

- Entender de forma genérica as abordagens para resolver um problema intratável
- Entender os conceitos básicos de busca exaustiva inteligente

- Bibliografia
  - GPV 9.1

Tratando um novo problema de busca

- Encontrar redução/decomposição para um problema P
  - Várias estratégias vistas em sala
  - “Caderno de receitas”
- Provar que é NP-completo
  - NP + Redução
  - Usar o quê para resolver?????

Lidando com a NP-Completude...

"NP-completeness is not a death certificate. It is only the beginning of a fascinating adventure."

- Busca exaustiva inteligente
  - Backtracking (tentativa e erro)
  - Branch-and-bound (ramificação e poda)
- Algoritmos aproximados
  - Garantia de qualidade
- Algoritmos heurísticos

Problemas de Busca

Constraint Satisfaction Problem(CSP)

- Otimização ou Decisão
- Conjunto de variáveis $\{X_1, X_2, \ldots, X_n\}$
  - Cada variável X_i no domínio D_i

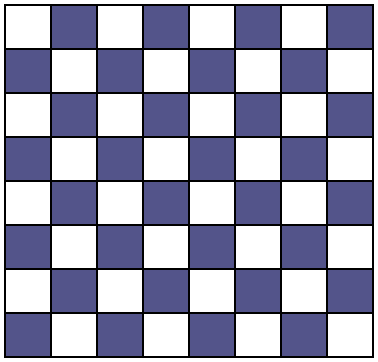
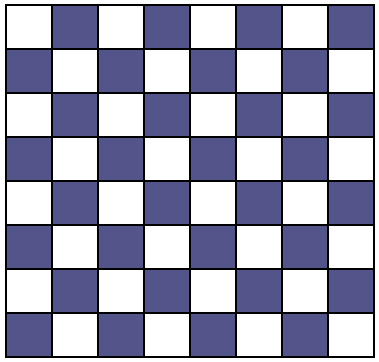
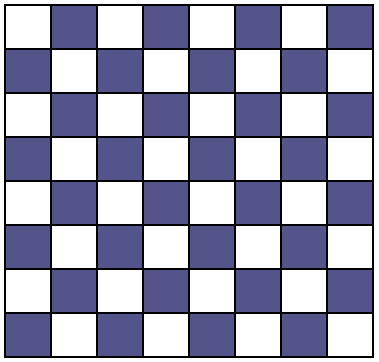
- Conjunto de restrições $\{C_1, C_2, \ldots, C_p\}$
  - Cada restrição relaciona um subconjunto de variáveis e especifica uma combinação viável (consistente) de valores para as mesmas.

Soluções

- Uma solução *s* é uma atribuição de valores para as variáveis: $s = (x_1, \cdots, x_n)$
  $x_i \in D_i \cup \{-\}$
  - **Consistente**: valores fixos não violam restrição.
- Objetivo: Obter solução consistente ótima.
  - Atribuir valor para toda variável de forma que todas as restrições sejam satisfeitas.

Exemplo: 8-rainhas

- Variáveis
- Domínios
- Restrições
- Solução

Exemplo: Mochila

- Variáveis
- Domínios
- Restrições
- Solução

Como resolver de forma exata?

- Geração exaustiva das soluções
  - Viabilidade: Checar restrições
  - Otimalidade: Checar valor da função objetivo

Podemos melhorar?

- Inconsistência pode ser detectada prematuramente!

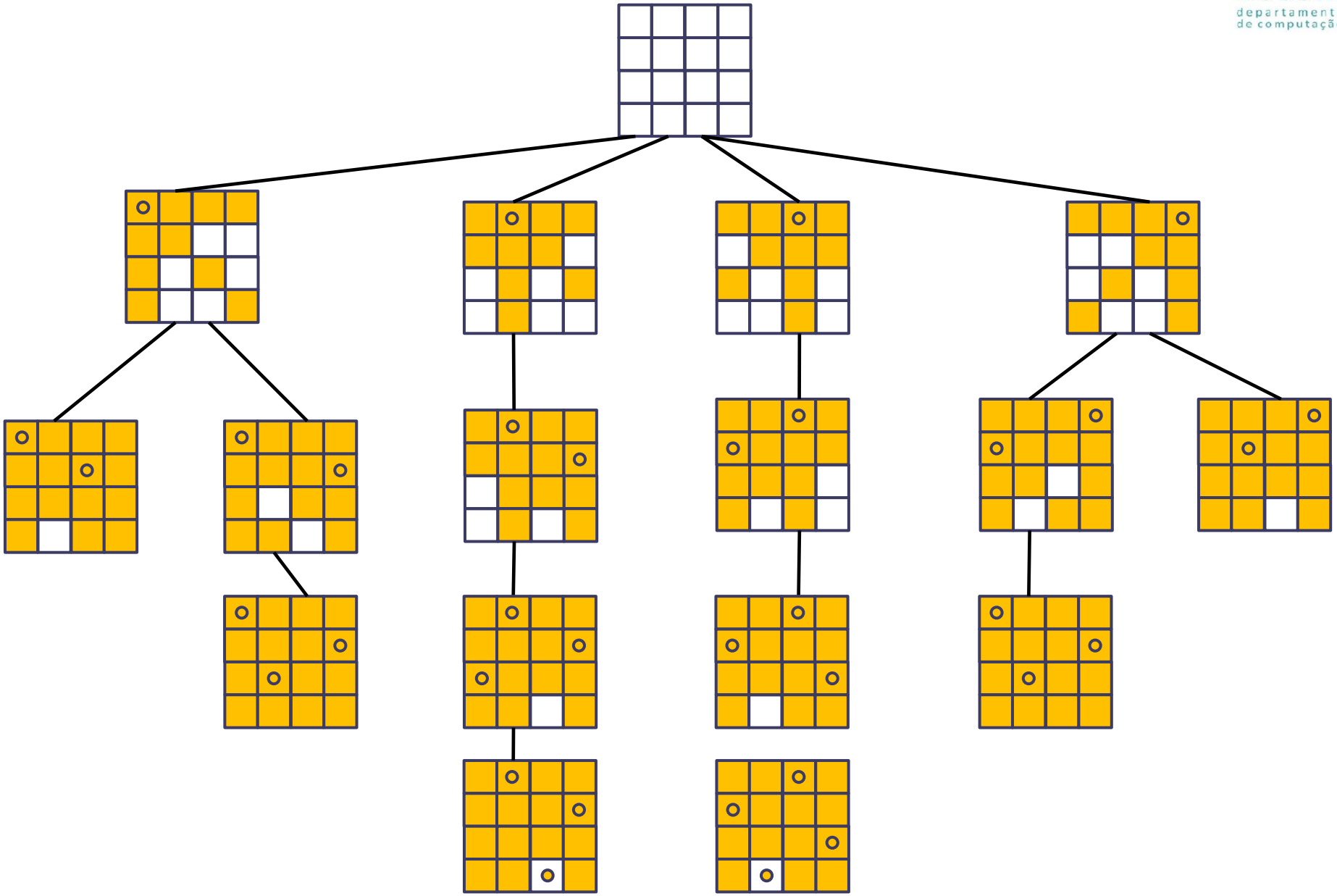
Soluções incrementais

- Uma solução *s* é uma atribuição de valores para as variáveis: $s = (x_1, \cdots, x_n)$

  $x_i \in D_i \cup \{-\}$

  - Se livre, não foi ainda escolhido valor.
  - **Completa(Parcial)**: todos os valores fixos.
  - **Consistente**: valores fixos não violam restrição.

- Objetivo: Obter solução (ótima,) completa e consistente
  - Atribuir valor para toda variável de forma que todas as restrições sejam satisfeitas.



```
bool finished = FALSE; (* found all solutions yet? *)

backtrack(int a[], int k, data input)
{
        int c[MAXCANDIDATES]; (* candidates for next position *)
        int ncandidates; (* next position candidate count *)
        int i; (* counter *)

        if (is_a_solution(a,k,input))
                process_solution(a,k,input);
        else {
                k = k+1;
                construct_candidates(a,k,input,c,&ncandidates);
                for (i=0; i<ncandidates; i++) {
                        a[k] = c[i];
                        backtrack(a,k,input);
                        if (finished) return; (* terminate early *)
                }
        }
}
```

n-rainhas

```
construct_candidates(int a[], int k, int n, int c[], int *ncandidates)
{
        int i,j; (* counters *)
        bool legal_move; (* might the move be legal? *)

        *ncandidates = 0;
        for (i=1; i<=n; i++) {
                legal_move = TRUE;
                for (j=1; j<k; j++) {
                        if (abs((k)-j) == abs(i-a[j])) (* diagonal threat *)
                                legal_move = FALSE;
                        if (i == a[j]) (* column threat *)
                                legal_move = FALSE;
                }
                if (legal_move == TRUE) {
                        c[*ncandidates] = i;
                        *ncandidates = *ncandidates + 1;
                }
        }
}
```

Soluções parciais e subproblemas

- As variáveis livres de uma solução parcial geram um subproblema equivalente ao original, mas menor.

Árvore de busca

- Cada nó, uma solução.
- Cada folha, uma solução completa.
- Cada nível, uma variável é fixada.

- Subárvore: todas as soluções completas derivadas da solução parcial.

Árvore para Problema da Mochila

Valor	Peso	
45	3	
45	9	
30	5	
10	2	

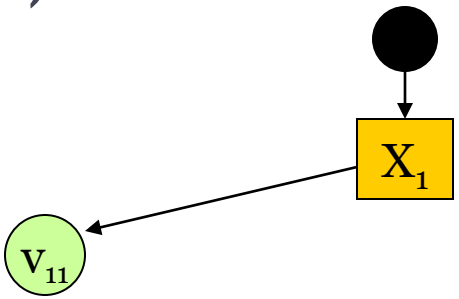
Algoritmo Backtracking

- Busca em profundidade na árvore de busca
- Permitir somente valores que levem a soluções parciais consistentes

Exemplo de Backtracking (3 variáveis)

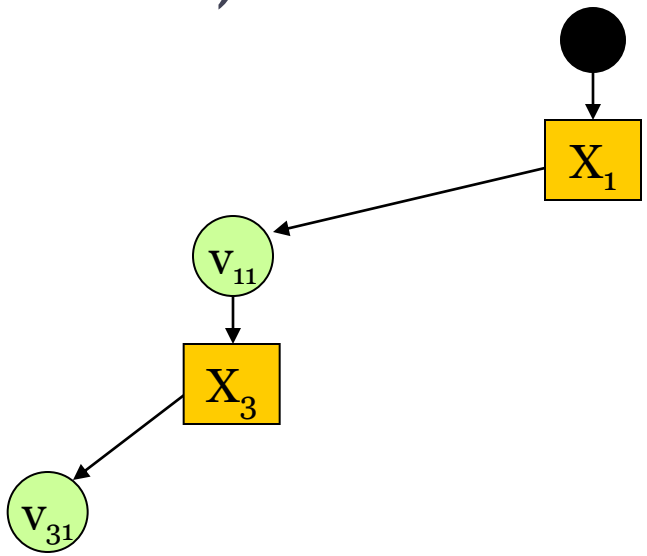
s = {-,-,-}

Busca Backtracking (3 variáveis)



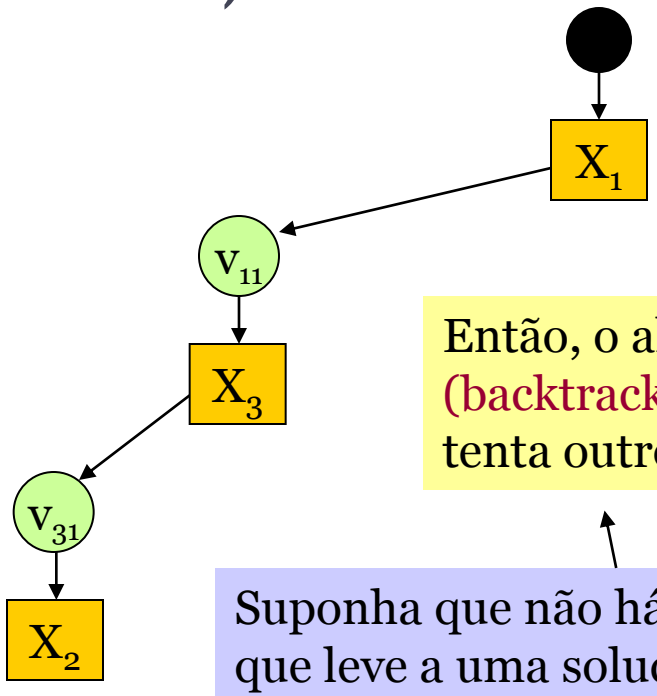
$s = \{v_{11}, -, -\}$

Busca Backtracking (3 variáveis)



$s = \{v_{11}, -, v_{31}\}$

Busca Backtracking (3 variáveis)

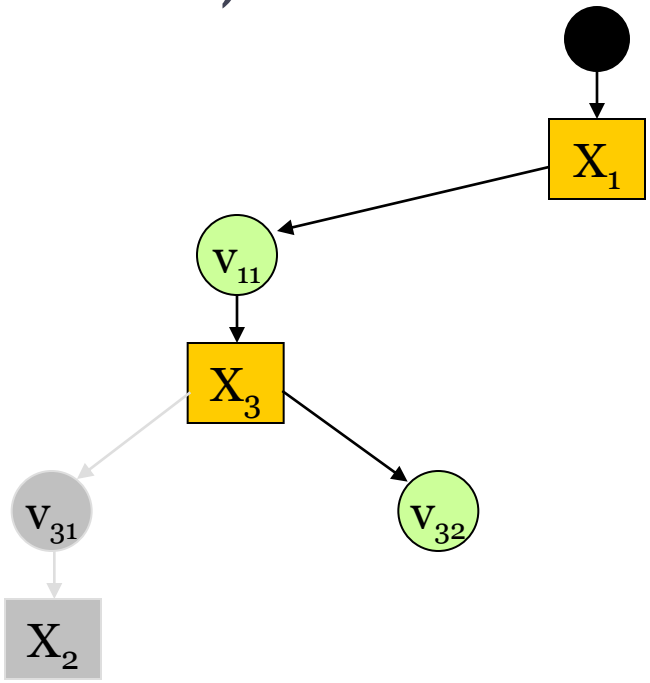


Então, o algoritmo de busca volta atrás (backtracks) para a variável anterior (X_3) e tenta outro valor

Suponha que não há valor de X_2 que leve a uma solução consistente

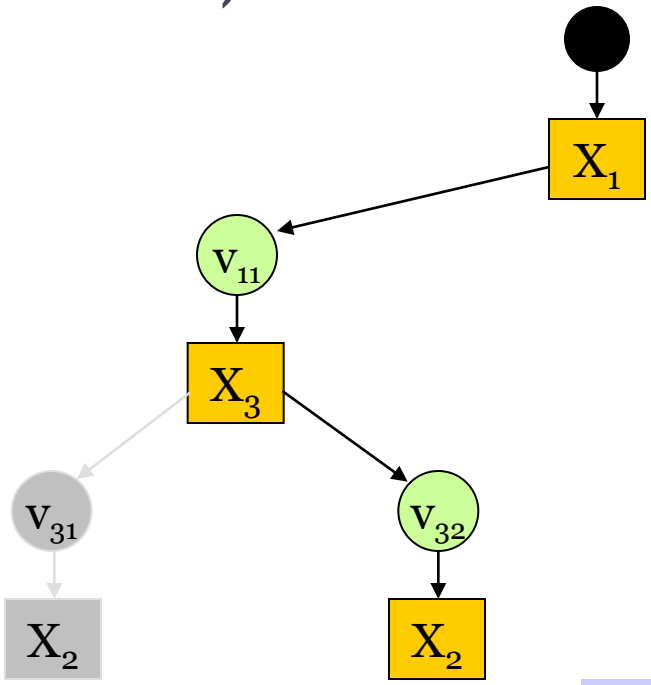
$s = \{v_{11}, -, v_{31}\}$

Busca Backtracking (3 variáveis)



$s = \{v_{11}, -, v_{32}\}$

Busca Backtracking (3 variáveis)

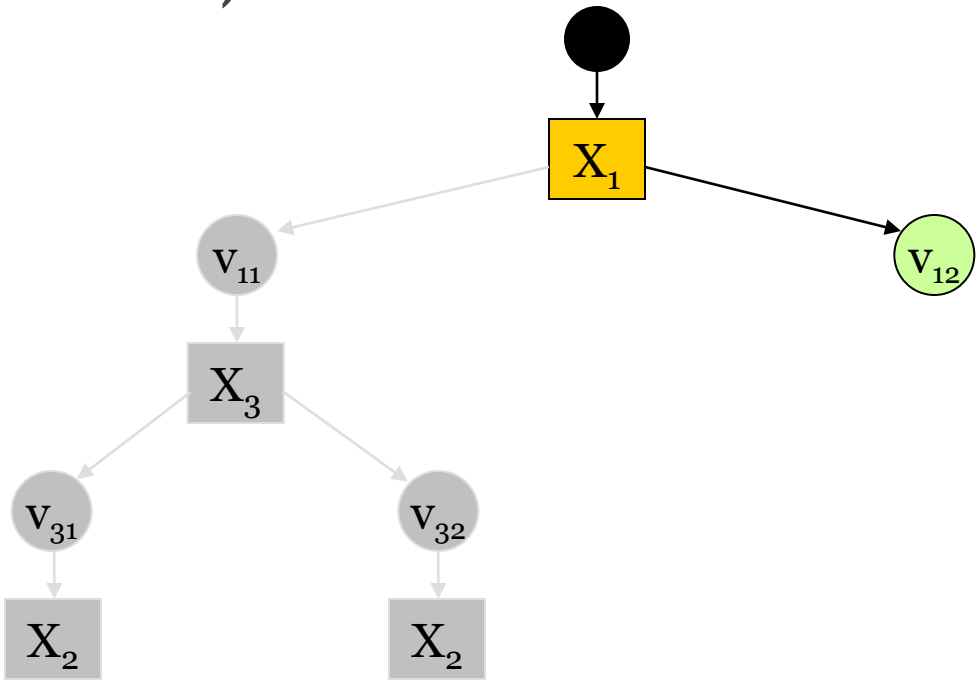


O algoritmo de busca volta atrás para a variável anterior (X_3) e tenta outro valor. Suponha que só existam dois valores, o algoritmo volta atrás novamente para X_1

Suponha novamente que não há valor de X_2 que leve a uma solução consistente

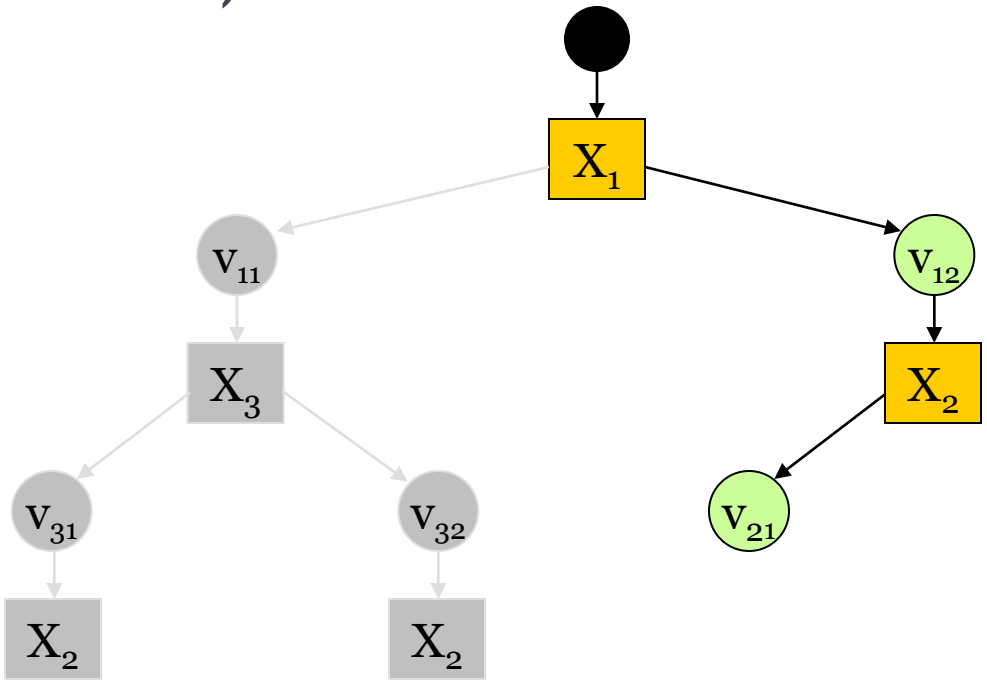
$S = \{v_{11}, -, v_{32}\}$

Busca Backtracking (3 variáveis)



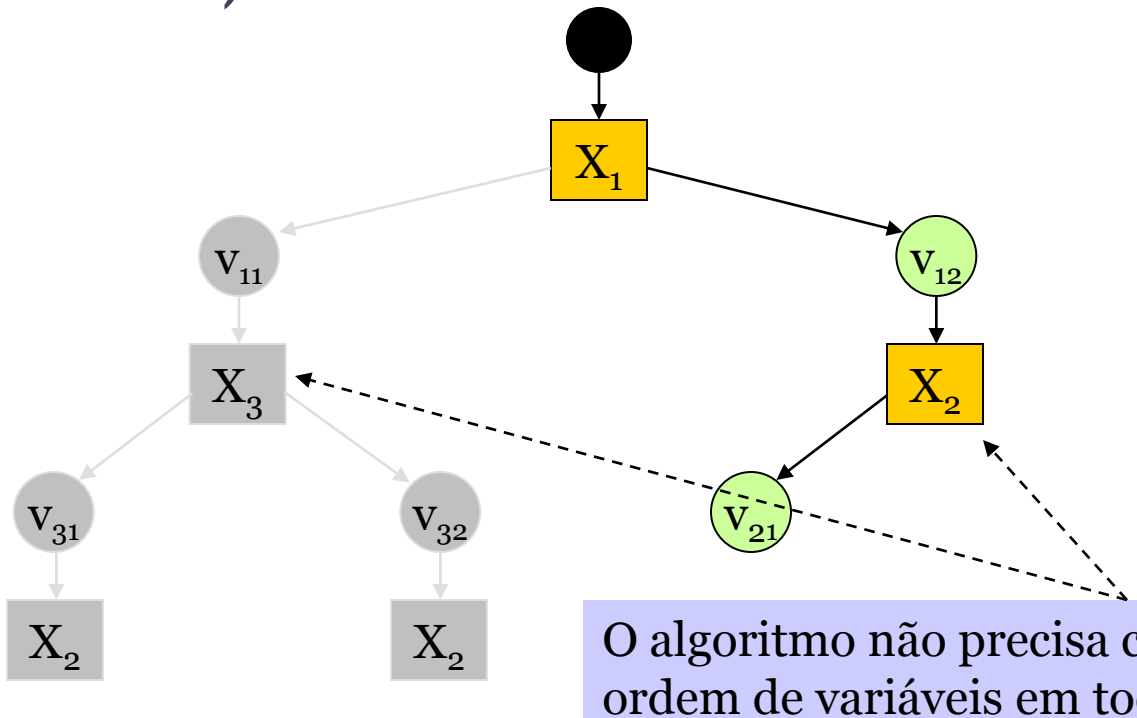
$s = \{v_{12}, -, -\}$

Busca Backtracking (3 variáveis)



$S = \{v_{12}, v_{21}, -\}$

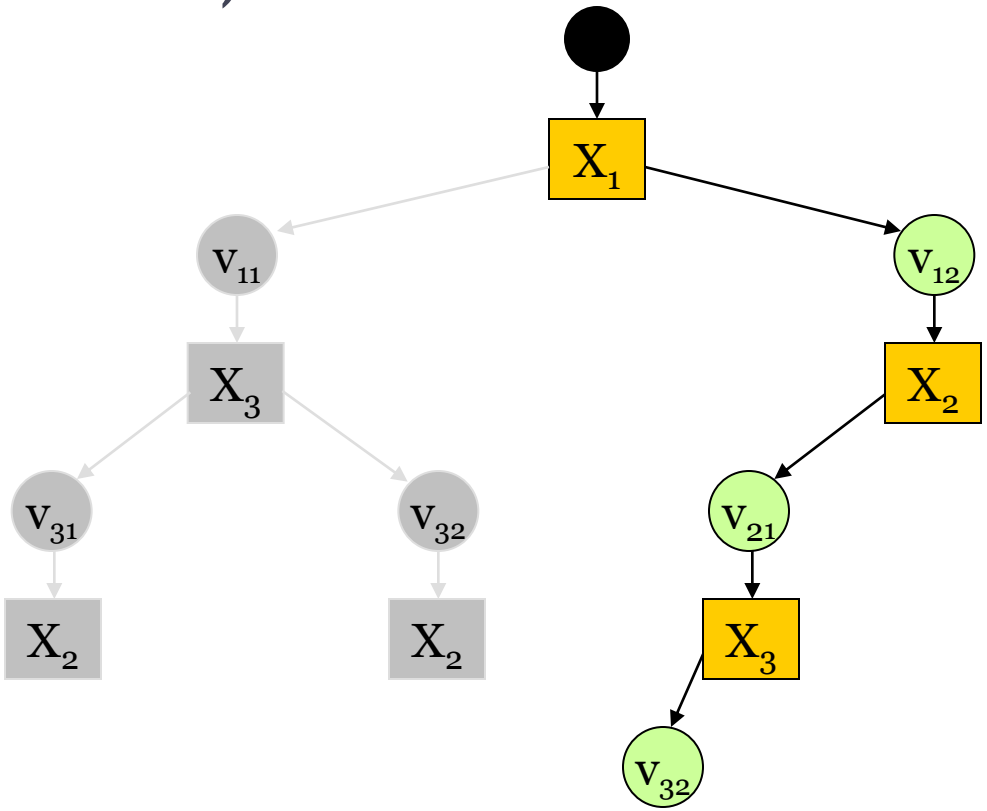
Busca Backtracking (3 variáveis)



O algoritmo não precisa considerar a mesma ordem de variáveis em todos os ramos...

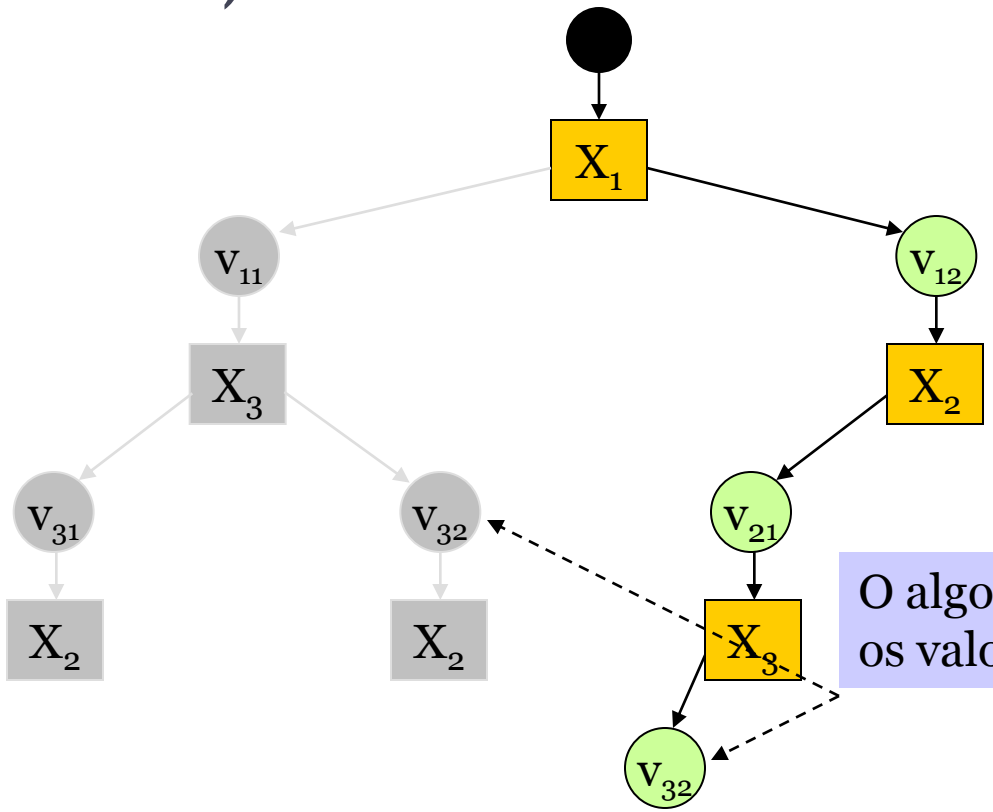
$S = \{v_{12}, v_{21}, -\}$

Busca Backtracking (3 variáveis)



$S = \{v_{12}, v_{21}, v_{32}\}$

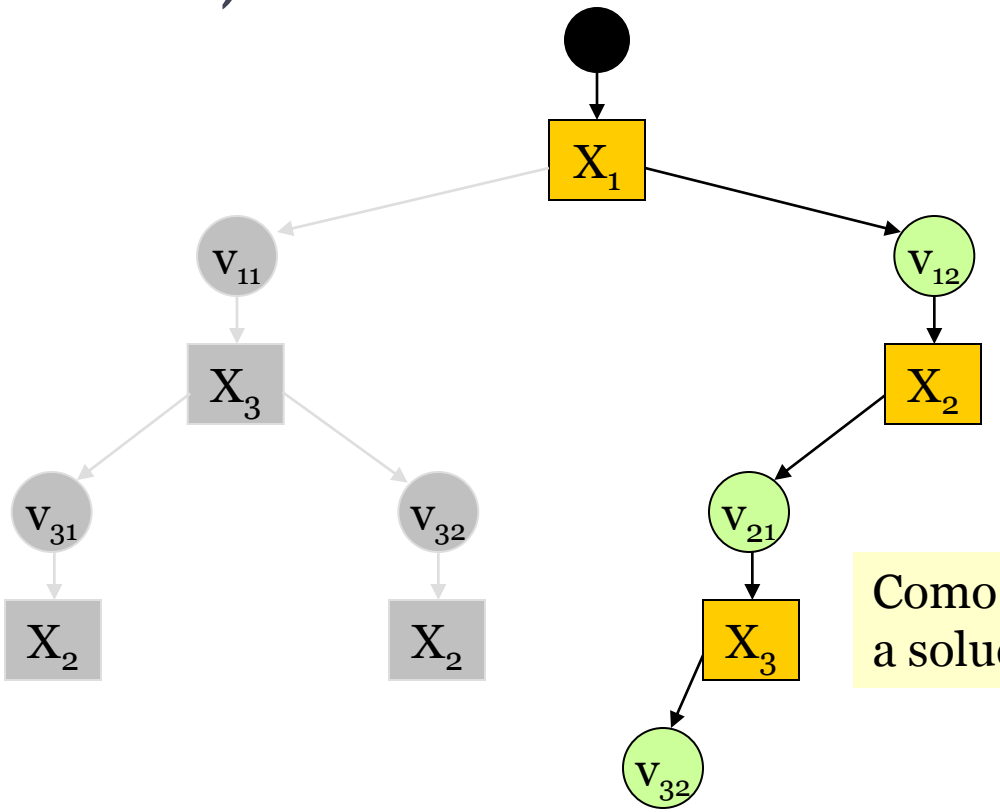
Busca Backtracking (3 variáveis)



O algoritmo não precisa conside
os valores na mesma ordem

$S = \{v_{12}, v_{21}, v_{32}\}$

Busca Backtracking (3 variáveis)



Como só há três variáveis, a solução é completa.

$S = \{v_{12}, v_{21}, v_{32}\}$

Backtracking

- **Exploração** sistemática do espaço de busca
  - Geração recursiva (implícita) da árvore
  - Semelhante à exaustiva
  - Busca em profundidade

- **Eliminação** de soluções parciais inviáveis
  - Construtivo
  - Restrições implícitas
  - Pode usar heurísticas para definir poda