Tratamento de Problemas NP-Difíceis: BackTracking

Cid C. Souza
Eduardo C. Xavier

Instituto de Computação/Unicamp

9 de maio de 2011

Introdução

Casos de aplicação:

- Problemas cujas soluções podem ser representadas por tuplas (vetores) de tamanho fixo ou variável da forma $(x_1, \ldots, x_n)$.
- Solucionar o problema equivale a encontrar uma tupla que otimiza ou satisfaz uma *função critério* $P(x_1, \ldots, x_n)$
- As vezes também queremos encontrar todas as tuplas que satisfaçam $P(x_1, \ldots, x_n)$.

Restrições:

- *Explícitas*: especificam os domínios (finitos) das variáveis na tupla.
- *Implícitas*: relações entre as variáveis da tupla que especificam quais delas respondem ao problema, satisfazendo P.

Exemplo: Oito Rainhas

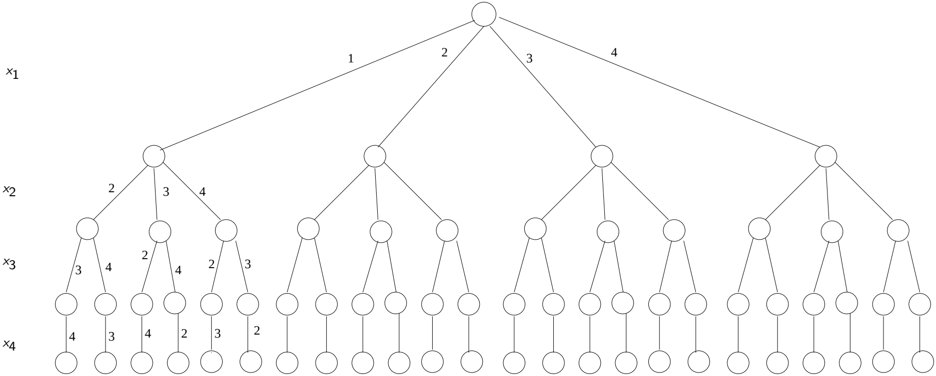
- Devemos dispor 8 rainhas em um tabuleiro de tal forma que não se ataquem.
- Cada rainha deve estar em uma coluna diferente.
- Váriáveis $x_1, \ldots, x_8$ que indicam em qual coluna a rainha da linha i está.
  - **Restrições Explícitas:** Variáveis assumem valores em $\{1, 2, \ldots, 8\}$
  - **Restrições Implícitas:** Duas rainhas não podem se atacar.
- Note que o número de tuplas é da ordem de 8!.

Conceitos Básicos

- **Algoritmo Força Bruta**: enumera todas tuplas de estados e verifica quais delas satisfazem às restrições *implícitas e explícitas*.
- **Algoritmo Backtracking**: busca sistemática no espaço de estados do problema que é organizado segundo uma *estrutura de árvore*, denominada **árvore de espaço de estados**.
  - uso de *funções limitantes* para restringir a busca na árvore.

Conceitos Básicos

Exemplo de árv. para 4 rainhas:



Conceitos Básicos

- **Espaço de Estados do Problema**: conjunto de todas as subseqüências das tuplas.
  - No exemplo são todos os nós.
- **Espaço de soluções**: conjunto de todas as tuplas satisfazendo as restrições *explícitas* e *implícitas*.

Conceitos Básicos

Durante a execução o algoritmo explora nós. Consideramos a seguinte terminologia:

- *nós ativos*: aqueles que ainda têm filhos a serem gerados.
- *nós amadurecidos*: aqueles em que todos os filhos já foram gerados ou não devam ser mais expandidos de acordo com a *função limitante*.
- *nó corrente*: aquele que está sendo explorado.

Conceitos Básicos

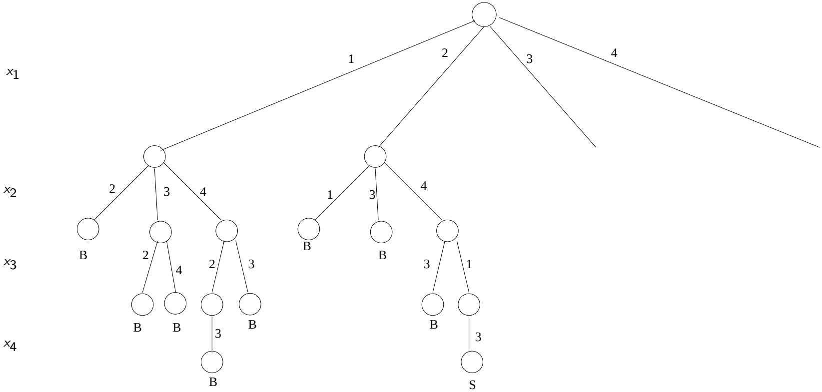
Métodos de exploração do espaço de estados (EE):

- *Backtracking*: busca no EE é feita em *profundidade*. Assim que um filho F de um nó R é gerado, o nó filho F passa a ser o nó corrente.
  - É feito um *backtracking* para retornar ao nó R.
- *Branch-and-Bound*: durante a busca no EE a geração de todos os filhos do nó corrente assim como o cálculo da função limitante em cada um deles é feita de uma vez só.

```
BACK (k)
Entrada: x_1, x_2, ..., x_{k-1} (já escolhidos).
Saída: todas as soluções serão impressas.
    (T, ℓ) ← Domínio (x_1, x_2, ..., x_{k-1}); //Valores válidos para x_k
    Para i = 1 até ℓ faça //Testa todos valores do domínio
        x_k ← T[i];
        Se ÉSolução(x_1, ..., x_k) então
            Imprima(x_1, ..., x_k);
        Se SatisfazRestrições(x_1, ..., x_k) então //Se puder alcançar
                                                   //uma solução
            BACK(k + 1);
    fim-para.
```

Conceitos Básicos

Exemplo de backtracking para 4 rainhas:



Sub-Set Sum (SuS)

- SuS: dado um conjunto $S = \{w_1, \ldots, w_n\}$ de n valores inteiros positivos e um valor inteiro positivo W, existe $U \subseteq \{1, \ldots, n\}$ tal que $\sum_{i \in U} w_i = W$?
- Exemplo: Se $n = 4$, $S = \{11, 13, 24, 7\}$ e $W = 31$ tem-se $U = \{1, 2, 4\}$ e $U = \{3, 4\}$.
- Representação das soluções:
  1. tupla de tamanho variável: como no exemplo acima.
  2. tupla de tamanho fixo n: $x_i = 1$ se $i \in U$ e $x_i = 0$ caso contrário.

Sub-Set Sum (SuS)

Componentes do algoritmo:

- Tuplas de tamanho n (fixo) onde todo x_i está em $\{0, 1\}$.
- Hipóteses: $0 < w_1 \leq w_2 \leq \ldots \leq w_n < W$ e $\sum_{i=1}^{n} w_i \geq W$.
- Para uma tupla-parcial $(x_1, \ldots, x_{k-1})$ já determinada, usar parâmetros:
  - $s = \sum_{i=1}^{k-1} w_i x_i$. Soma dos pesos da sub-solução.
  - $r = \sum_{i=k}^{n} w_i$. Soma dos pesos restantes.

Sub-Set Sum (SuS)

Funções Limitantes:

1. SatisfazRestrições$(x_1, \ldots, x_k) =$ `true` se e somente se

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq W.$$

2. Suponha que $0 < w_1 \leq w_2 \leq \ldots \leq w_n$. Então $(x_1, \ldots, x_k)$ não pode levar a uma <u>nova</u> solução se $\sum_{i=1}^{k} w_i x_i + w_{k+1} > W$. Logo, uma outra função limitante seria: SatisfazRestrições$(x_1, \ldots, x_k) =$ `true` sse

$$\sum_{i=1}^{k} w_i x_i + \sum_{i=k+1}^{n} w_i \geq W$$

e

$$\sum_{i=1}^{k} w_i x_i + w_{k+1} \leq W.$$

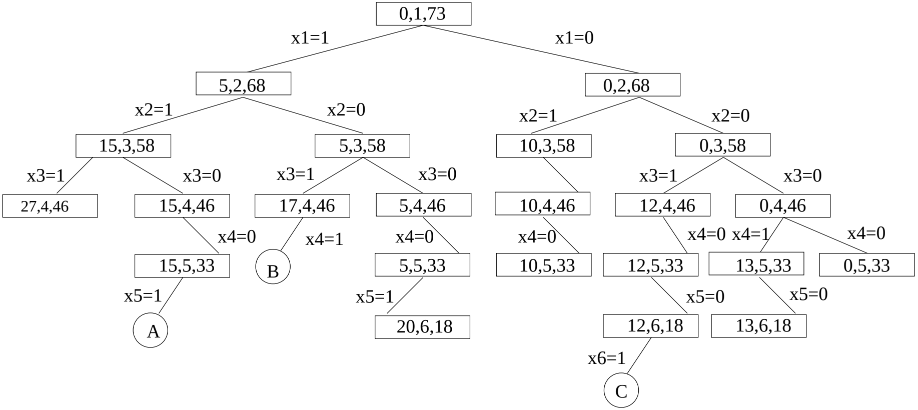
Sub-Set Sum (SuS)

```
SuS (k, s, r) //Domínio de x_k será sempre {0, 1}.
    x_k ← 1;   // Primeiro caso: x_k = 1.
    Se (s + w_k = W) então
        IMPRIMA (x_1, ..., x_k, 0, ..., 0)
    se não
        Se SatisfazRestrições(x_1, ..., x_k) então
            SuS(k + 1, s + w_k, r − w_k);
    fim-se
    x_k ← 0; // Segundo caso: x_k = 0
    Se SatisfazRestrições(x_1, ..., x_k) então
        SuS(k + 1, s, r − w_k);
    fim-se
fim.
```

Sub-Set Sum (SuS)

- Exemplo: $n = 6$, $W = 30$, $S = \{5, 10, 12, 13, 15, 18\}$.
- Espaço de estados total na árvore: $2^{6+1} - 1 = 127$.
- Parte da árvore de espaço de estados gerada por `SuS(0,1,73)` (próxima transparência ...)
- Legenda:
  - triplas (s, k, r);
  - $\bigcirc$ são os nós-resposta.

Sub-Set Sum (SuS)



p-coloração de grafos (COR)

- COR: dado um grafo não-orientado G com n vértices e representado por sua matriz de adjacências A, encontrar todas as colorações de G com p cores ou menos.
- OBS: A versão de decisão é $\mathcal{NP}$-Completo.
- Representação das soluções: tuplas de tamanho n (fixo) onde $x_i \in \{0, 1, \ldots, p\}$ representa a cor do vértice i.
- A cor *zero* significa que o vértice ainda não está colorido.
- O algoritmo inicializará a tupla com *zeros*.
- *Função Limitante*: recursão só é interrompida se não for possível alocar uma cor para o k-ésimo vértice.

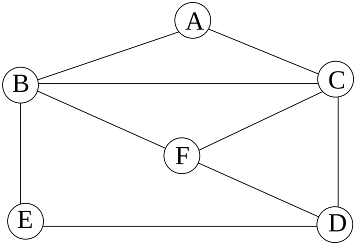
p-coloração de grafos (COR)

```
DOMÍNIO (x_1, ..., x_{k-1})
    Para i = 1 até p faça pode[i] = 1;
    Para j = 1 até k − 1 faça
        Se (A[k, j] = 1 ) então
            pode[x_j] ← 0; // não pode ter cor igual a um vizinho
    fim-para;
    ℓ ← 0;   // Constrói o domínio
    Para i = 1 até p faça
        Se (pode[i] = 1) então
            ℓ ← ℓ + 1;    T[ℓ] ← i;
        fim-se;
    fim-para;
    Retornar (T, ℓ).
fim.
```

p-coloração de grafos (COR)

COR(k);
  $(T, \ell) \leftarrow$ Domínio$(x_1, x_2, \ldots, x_{k-1})$;
  Para $i = 1$ **até** ℓ **faça**
    $x_k \leftarrow T[i]$;
    Se $k = n$ **então** (* todos vértices estão coloridos *)
      `IMPRIMA`$(x_1, \ldots, x_k)$
    se não `COR`$(k + 1)$;
  fim-para.

p-coloração de grafos (COR)



p-coloração de grafos (COR)