Análise e Técnicas de Algoritmos
Período 2002.2

Aula 11: Backtracking

20/02/2003

Backtracking

- Backtracking é um refinamento da abordagem *força bruta*.
- Sistematicamente busca por uma solução para o problema, considerando todas as opções disponíveis.
- Em geral, avaliamos algumas restrições para minimizar ou maximizar a função de otimização associada.
- O conjunto de soluções possíveis (espaço de solução) é obtido através de soluções parciais que são incrementadas até se chegar a soluções do problema.

Exemplo:

Vamos considerar um problema resolvido por uma estratégia *força bruta*. O espaço de solução para este problema corresponde ao conjunto de todas as soluções viáveis. Vamos supor ainda que esse espaço solução é finito. A estratégia força bruta seria:

1. enumerar todo espaço de solução (em geral muito grande).
2. computar função de otimização.
3. pegar a melhor solução.

Em vez de tentar todas as soluções, backtracking reduz o espaço de solução, através de uma análise de restrições, eliminando aquelas soluções parciais que não têm chance de sucesso (poda do espaço de soluções).

A Idéia Geral: Usando o Espaço de Soluções

As soluções são representadas por n-tuplas ou vetores de solução $\langle v_1, v_2, \cdots, v_n$. Cada v_i é escolhido a partir de um conjunto finito de opções S_i.

Um algoritmo de backtracking inicia com um vetor vazio. Em cada etapa, o vetor é extendido com um novo valor formando um novo vetor que pode representar uma solução parcial do problema. Na avaliação de um vetor $\langle v_1, \cdots, v_i \rangle$, se for constatado que ele não pode representar uma solução parcial, o algoritmo faz o backtrack, eliminando o último valor do vetor, e continua tentando extender o vetor com outros valores alternativos.

Aqui apresentamos um esquema genérico de algoritmos de backtracking:

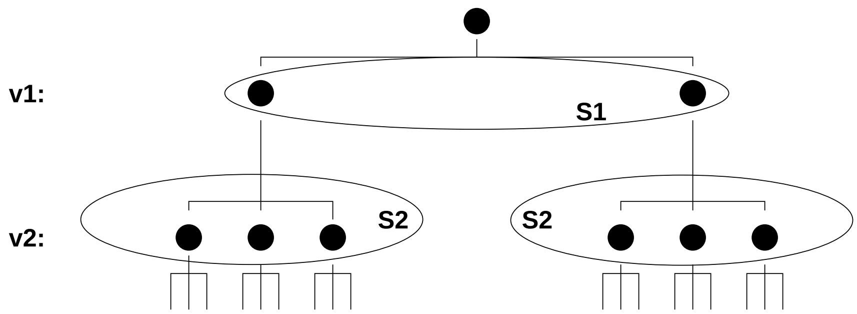
backtrack($v[1..k]$)

$\triangleright v$ é uma solução parcial de tamanho k
if v é uma solução **then**
  escreva v
else

for cada $k+1$ vetor w em que $w[1..k] = v[1..k]$ **do**
backtrack($w[1..k+1]$)

Um aspecto bastante importante nos problemas que são tratados por soluções baseadas em backtracking, é a definição de restrições. Existem dois tipos de restrições: restrições explícitas e restrições implícitas.

Restrições explícitas correspondem às regras que restringem cada v_i em tomar valoes de um determinado conjunto. Isto está relacionado com a representação do problema e as escolhas possíveis.



Restrições implícitas determinam quais das n-tuplas no espaço de soluções correspondem a soluções para um determinado problema.

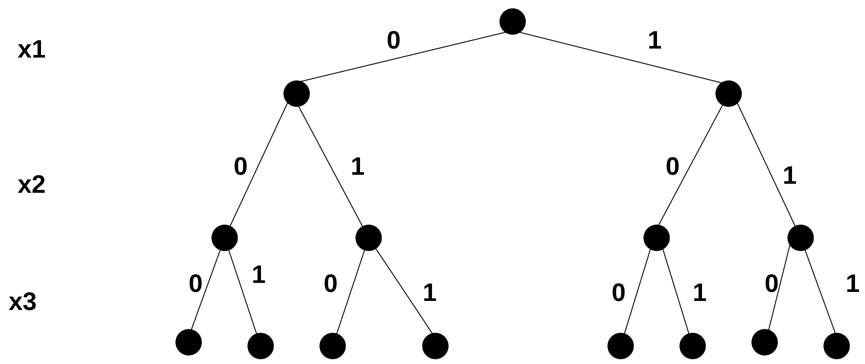
Se S_i é o domínio de v_i, então $S_1 \times \cdots \times S_m$ é o espaço de soluções do problema. O critério de validação (determinado pelas restrições implícitas) determina que parte deste conjunto solução necessita ser considerado.

A busca pelo espaço de soluções pode ser representado por um caminhamento em profundidade de uma árvore.

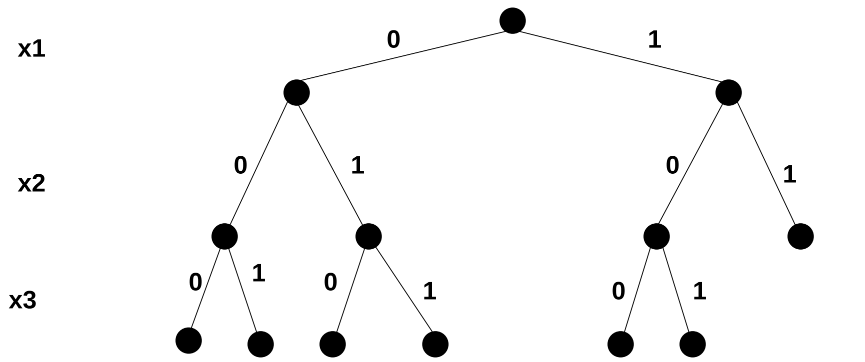
Exemplo 1: Mochila Binária

O exemplo da mochila binária já é bastante conhecido. A restrição explícita, neste caso, é: $x_i = \{0, 1\}$.

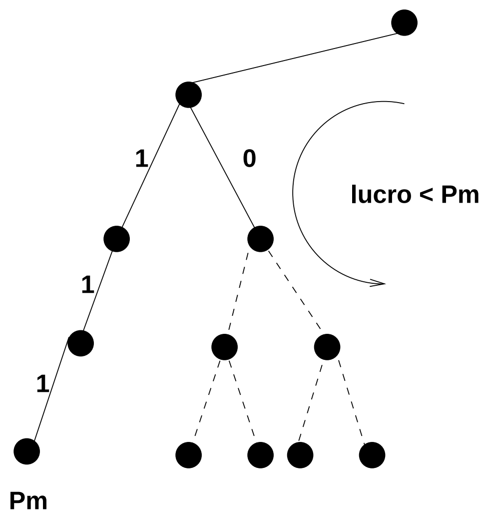
Assim, se fizermos a enumeração exaustiva do espaço de soluções para $n = 3$, teremos a seguinte árvore:



Uma outra restrição explícita é o fato de que $\Sigma x_i.w_i \leq M$, que indica a capacidade da mochila. Supondo que $w_1 + w_2 > M$, podemos usar esta restrição para podar nossa árvore.



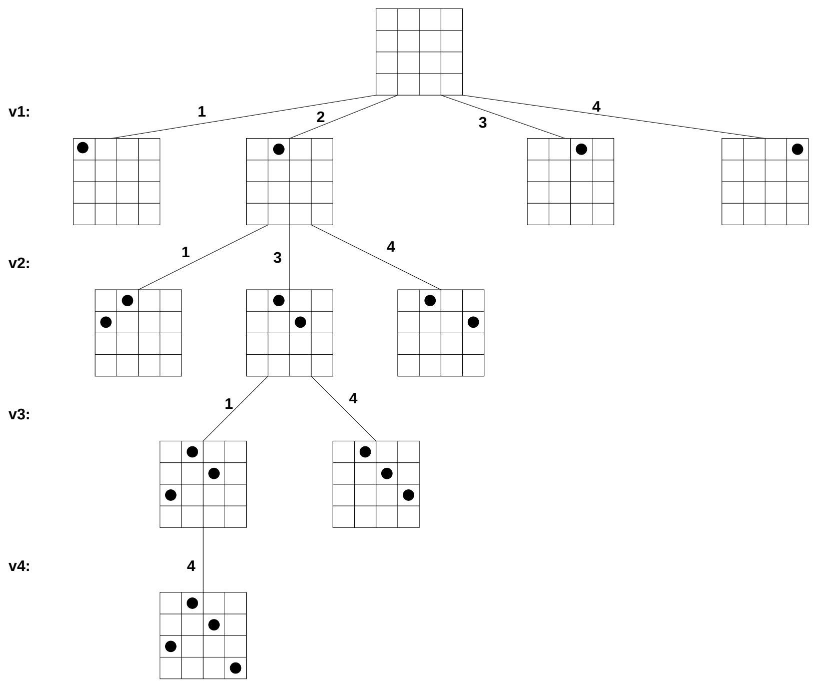
Finalmente, podemos utilizar restrições implícitas para podar ainda mais a árvore. Se estamos interssados em ter o maior lucro e, em uma determinada solução temos lucro P_m, não adianta considerar aquelas soluções parciais que não levem a um lucro de pelo menos P_m.



Exemplo 2: O problema das n Rainhas

Considere um tabuleiro de xadrez $n \times n$. O problema consiste em colocar n rainhas no tabuleiro sem que nenhuma delas possa atacar uma outra rainha.

O espaço de solução é $\{1, 2, 3, \cdots, n\}^n$. Tentando todas as possibilidades possíveis temos n^n possibilidades. Como as rainhas devem estar em diferentes colunas, o número de possibilidades se reduz a $n!$.

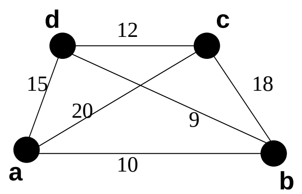


Usando backtracking e considerando que duas rainhas não podem estar na mesma coluna, linha ou diagonal, podemos reduzir o espaço de busca.

Exemplo 3: O Caixeiro-Viajante

Este problema assume um conjunto de n cidades e um caixeiro-viajante que necessita visitar cada uma das cidades. Por questão de economia, ele deve visitar uma única vez cada cidade e por fim retornar a cidade de origem. Portanto, o problema consiste em determinar uma turnê de custo mínimo.

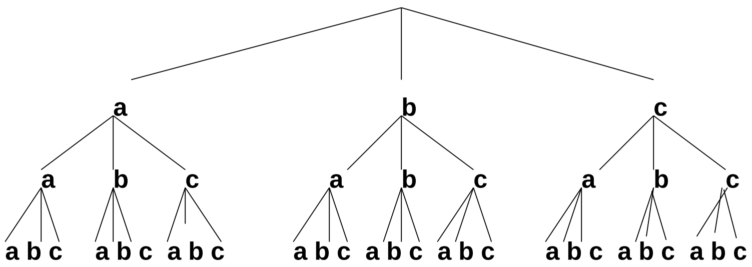
No exemplo abaixo, a rota (a, b, d, c) é a de custo mínimo (51).



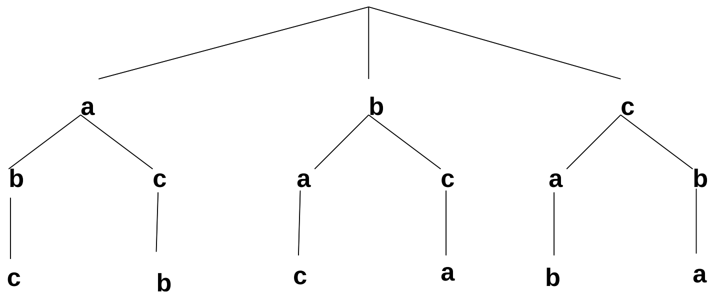
O problema do caixeiro viajante é NP-hard (veremos isso mais na frente). isso implica dizer que não existe uma solução em tempo polinomial para ele. A solução por backtracking define um vetor de cidades $(v_1, \cdots, v_n)$ que representa a melhor rota.

Como critério de poda podemos utilizar o número de cidade na rota obtida.

esse número não pode ser maior do que $|V|$. Neste caso, o algoritmo deve buscar $|V|^{|V|}$ possibilidades.



Se o critério de poda verificar a repetição de cidades, o número de possibilidades é reduzido para $|V|!$.



Considerações Adicionais

- Forma eficiente de implementar busca exaustiva.
- Linguagens da área de programação em lógica geralmente trazem algum mecanismo que dá suporte ao baxktracking. Exemplo: Prolog, OPS5, etc.
- Programas em backtracking são, por natureza, combinatórios.
- Requer muita memória, já que a quantidade de variáveis locais transportada em cada chamada recursiva é diretamente proporcional ao tamanho do problema.

Branch-and-Bound

Branch-and-bound (BB) é uma estratégia bastante similar a backtracking. Também utiliza uma estrutura de árvore para explorar todas as possíveis soluções. A principal diferença entre as duas estratégias está na forma como a árvore é explorada. Como vimos, backtracking utiliza uma busca em profundidade. Na estratégia BB, utiliza busca em aplitude. Através de computação auxiliar, BB decide em cada etapa qual dos nós deve ser o próximo a ser explorado. Uma lista de prioridade mantém os nós que foram gerados mas que ainda permanecem inexplorados.