

Técnicas de Concepção de Algoritmos: Branch and Bound

R. Rossetti, A.P. Rocha, João Pascoal Faria
FEUP, MIEIC, CAL, 2013/2014

Branch and Bound

- O que é?
 - BB ou B&B é uma técnica de concepção de algoritmos genérica para encontrar soluções óptimas em vários problemas de optimização
 - Aplicada especialmente em problemas de optimização discreta e combinatorial
 - Aplica-se geralmente quando uma técnica gananciosa (*greedy*) ou programação dinâmica não são de todo apropriadas
 - É geralmente mais lento que as anteriores; pode levar a complexidades temporais exponenciais nos piores casos
 - Se bem aplicada, pode gerar soluções razoavelmente rápidas nos casos médios

Branch and Bound

■ Princípio

- Consiste na enumeração de todas as soluções candidatas
- Um alargado número de candidatos “não promissores” é descartado, estimando-se limites superiores e inferiores da quantidade a ser otimizada
- Baseia-se numa pesquisa em largura (*breadth-first search*), mas nem todos os vértices filhos são expandidos
 - Há um critério bem definido para dizer que filhos expandir e quando
 - Outro critério determina quando o algoritmo encontrou a solução óptima
- Método proposto por A. H. Land and A. G. Doig (1960) para programação discreta

A. H. Land and A. G. Doig (1960). "An automatic method of solving discrete programming problems". *Econometrica* **28** (3): pp. 497-520.

Aplicações

■ A abordagem é utilizada para resolver alguns problemas conhecidos, ditos *NP-hard*, como:

- Programação Inteira
- Programação não-linear
- Problema do Caixeiro Viajante (*Travelling Salesman*) (TSP)
- Problema de afectação quadrática (QAP)
- *The Maximum Satisfiability Problem* (MAX-SAT)
- Pesquisa do Vizinho mais Próximo (NNS)
- O problema de corte de stock
- False Noise Analysis

B&B: Algoritmo Geral

■ Considerações

- Cada solução é assumida poder ser expressa como um array $X[1:n]$, como por exemplo em técnicas de retrocesso
- Um “preditor”, ou função de custo aproximado CC, deve ser definida

■ Definições importantes

- Um “live node” é um vértice que não foi expandido
- Um “dead node” é um vértice que já foi expandido
- Um “expanded node” (ou “e-node”) é um “live node” com melhor CC

B&B: Algoritmo Geral

```

Procedure B&B()
begin
  E: nodepointer;
  E := new(node); -- this is the root node which
                  -- is the dummy start node
  H: heap;         -- A heap for all the live nodes
                  -- H is a min-heap for minimization problems,
                  -- and a max-heap for maximization problems.
  while (true) do
    if (E is a final leaf) then
      -- E is an optimal solution
      print out the path from E to the root;
      return;
    endif

    Expand(E);
    if (H is empty) then
      report that there is no solution;
      return;
    endif
    E := delete-top(H);
  endwhile
end

```

B&B: Algoritmo Geral

Procedure Expand(E)

begin

- Generate all the children of E;
- Compute the approximate cost value CC of each child;
- Insert each child into the heap H;

end

Funções de Custo Aproximado

■ Como escolher?

- **Definição** da Função de Custo C : para cada vértice x na árvore de solução, a função de custo $C(x)$ é o custo da melhor solução que vai até o vértice x .
- **Teorema:**
No caso de problemas de minimização, se $CC(x) \leq C(x)$ para todo vértice x , e se $CC(x) = C(x)$ para todo vértice folha x , então o primeiro vértice a ser expandido (melhor CC), que é um vértice folha, corresponde à solução ótima do problema!
- *Prove!*

Funções de Custo Aproximado

■ Critérios para escolha da função de custo aproximado CC

- Para problemas de minimização:
 $CC(x) \leq C(x)$ para todo “live node” x
 $CC(x) = C(x)$ para todo vértice folha
- Para problemas de maximização:
 $CC(x) \geq C(x)$ para todo “live node” x
 $CC(x) = C(x)$ para todo vértice folha
- CC é chamada uma subestimação de C (no caso de minimização), e uma superestimação de C (no caso de maximização)
- Quanto mais próximo que CC estiver de C , mais rápido será capaz o algoritmo de encontrar a solução ótima!

Exemplo de Aplicação: B&B

■ Problema

- Uma companhia está a montar uma equipa para levar a cabo uma série de operações
- Há **quatro membros** na equipa: A, B, C e D
- Há **quatro operações** a realizar: 1, 2, 3, e 4
- Cada membro da equipa pode realizar exactamente uma operação, e Todas as quatro operações devem ser executadas com sucesso, para que para que o projecto todo seja considerado concluído satisfatoriamente.
- Entretanto, a probabilidade de um membro particular da equipa ser bem sucedido na realização de uma operação particular varia, como indicado na tabela abaixo:

| | | Operation | | | |
|-------------|---|-----------|-----|------|------|
| | | 1 | 2 | 3 | 4 |
| Team member | A | 0.9 | 0.8 | 0.9 | 0.85 |
| | B | 0.7 | 0.6 | 0.8 | 0.7 |
| | C | 0.85 | 0.7 | 0.85 | 0.8 |
| | D | 0.75 | 0.7 | 0.75 | 0.7 |

Exemplo de Aplicação: B&B

■ Problema (cont...)

- Se aos membros da equipa, ABCD, fossem atribuídas as tarefas 1234, nesta ordem, então a probabilidade do projecto ser concluído na sua totalidade com sucesso seria:

$$(0.9)(0.6)(0.85)(0.7) = 0.3213 \sim 32\%$$

- Pressuposto:** Se houver formas possíveis de organizar a equipa de modo a se obter uma taxa de sucesso para o projecto como um todo que exceda aos **45%**, então o gestor irá aprovar o projecto!
- Questão:** para esta dada equipa, o gestor irá aprovar o projecto?

| | | Operation | | | |
|-------------|---|-----------|-----|------|------|
| | | 1 | 2 | 3 | 4 |
| Team member | A | 0.9 | 0.8 | 0.9 | 0.85 |
| | B | 0.7 | 0.6 | 0.8 | 0.7 |
| | C | 0.85 | 0.7 | 0.85 | 0.8 |
| | D | 0.75 | 0.7 | 0.75 | 0.7 |

Exemplo de Aplicação: B&B

■ Formulação:

- Vértices na árvore: uma atribuição pessoa-tarefa, total ou parcial
- Política de selecção de vértices: melhor valor global da **função de custo aproximado** (*bounding function*)
- Política de selecção de variáveis: escolher a próxima operação na ordem natural, ou seja 1 a 4
- Função de Custo (**bounding function**): para operações não atribuídas, escolher o melhor membro da equipa que esteja disponível, ou seja, o que possui maior probabilidade de sucesso, ainda que a pessoa seja escolhida mais do que uma vez
- Função Objectivo (critério de paragem): quando um estado-solução candidato tiver o valor resultante da função de custo maior do que o valor do estado-candiato corrente
- Estado-solução candidato: quando a função de custo retorna uma solução em que cada operação é atribuída a uma pessoa diferente

| | | Operation | | | |
|-------------|---|-----------|-----|------|------|
| | | 1 | 2 | 3 | 4 |
| Team member | A | 0.9 | 0.8 | 0.9 | 0.85 |
| | B | 0.7 | 0.6 | 0.8 | 0.7 |
| | C | 0.85 | 0.7 | 0.85 | 0.8 |
| | D | 0.75 | 0.7 | 0.75 | 0.7 |

Solução candidata: ?

No início, nenhuma decisão foi tomada;
Calcula-se o valor máximo do potencial de sucesso para o projecto:

Se A for escolhido para realizar as 4 operações, então:

AAAA = (0.9)(0.8)(0.9)(0.85) = 0.5508

AAAA 0.5508 root

FEUP Universidade do Porto Faculdade de Engenharia

Branch and Bound - CAL, 2013/14

| | | Operation | | | |
|-------------|---|-----------|-----|------|------|
| | | 1 | 2 | 3 | 4 |
| Team member | A | 0.9 | 0.8 | 0.9 | 0.85 |
| | B | 0.7 | 0.6 | 0.8 | 0.7 |
| | C | 0.85 | 0.7 | 0.85 | 0.8 |
| | D | 0.75 | 0.7 | 0.75 | 0.7 |

Solução candidata: ?

Para a Operação 1, expande-se o vértice raiz (*branching process*).

Cada nó filho corresponde à atribuição da primeira tarefa a um membro específico da equipa...

AAAA 0.5508 root

A

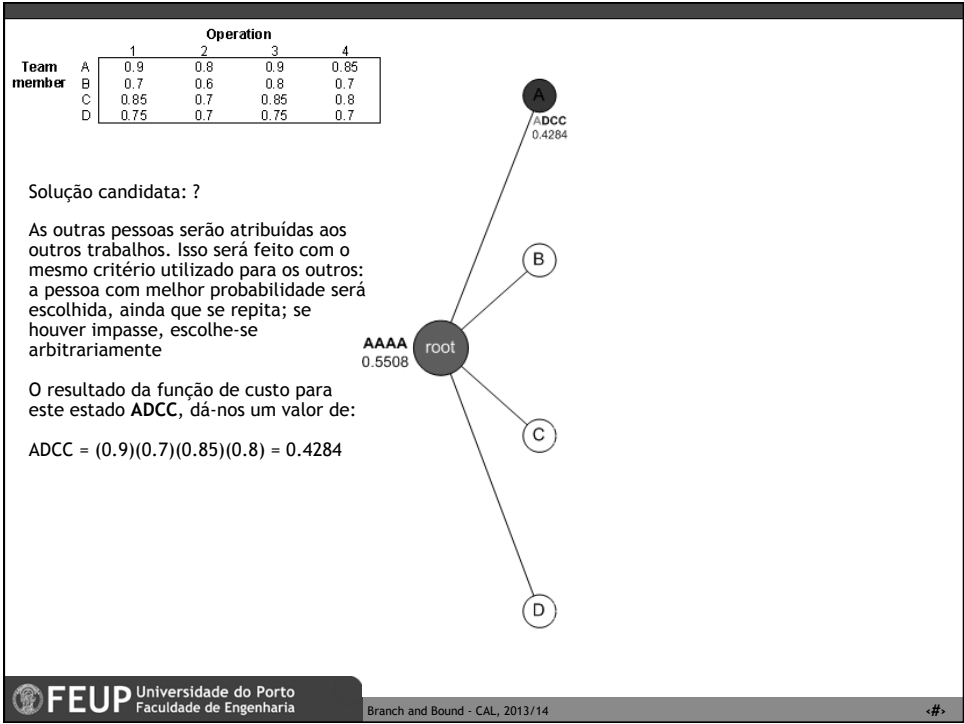
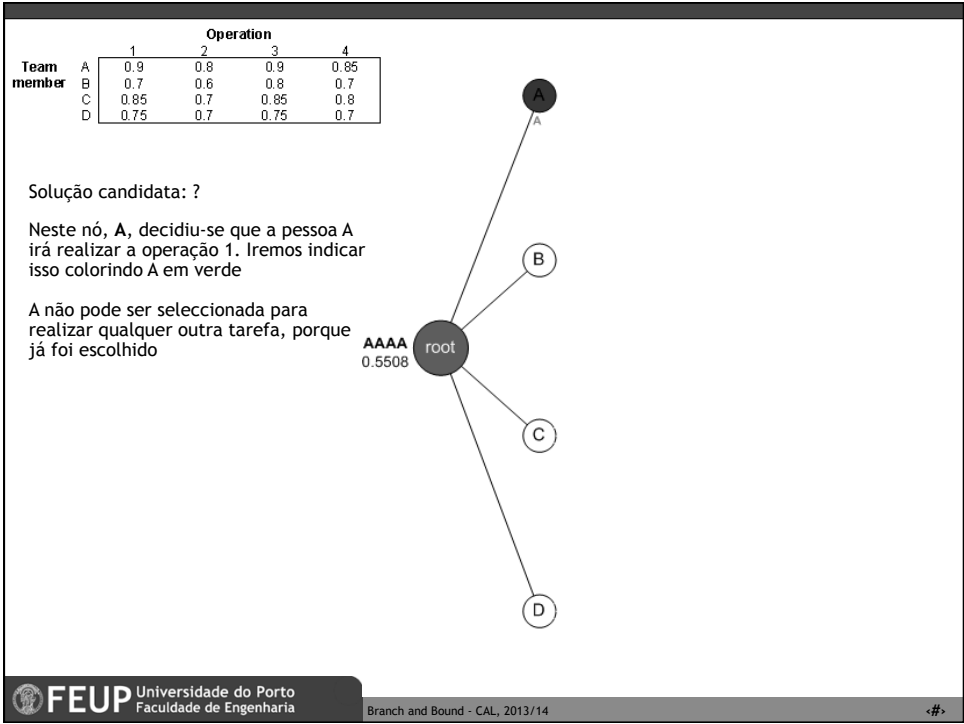
B

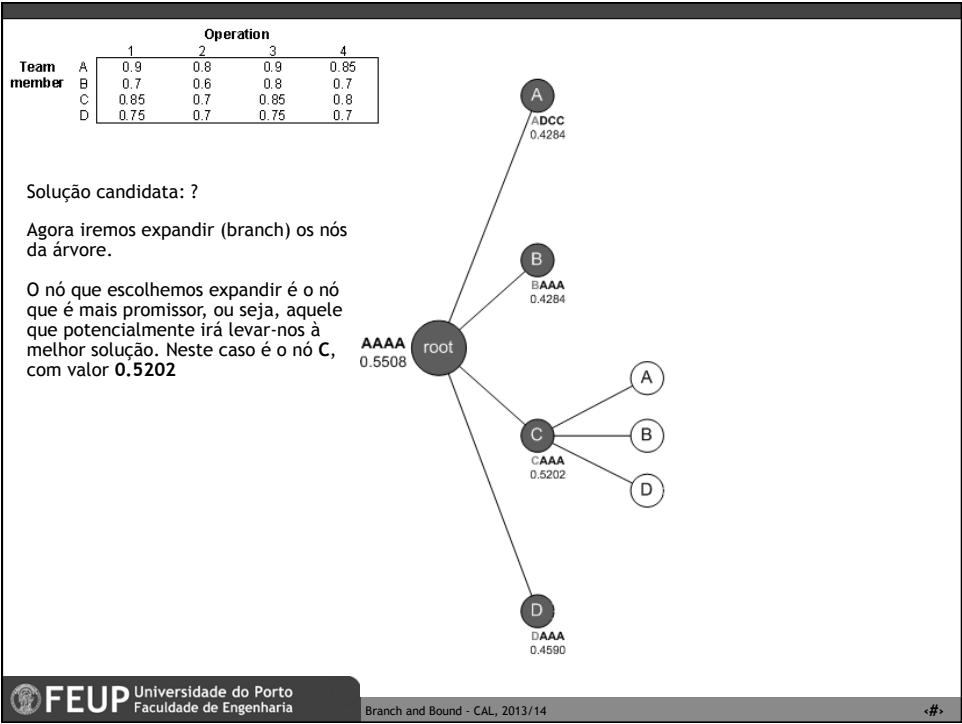
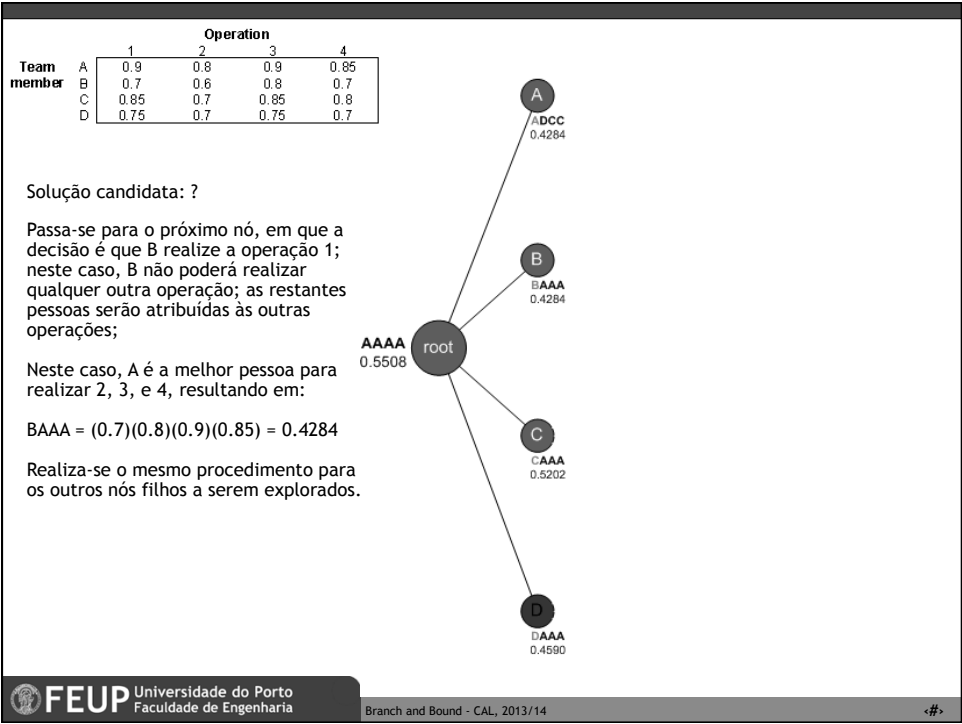
C

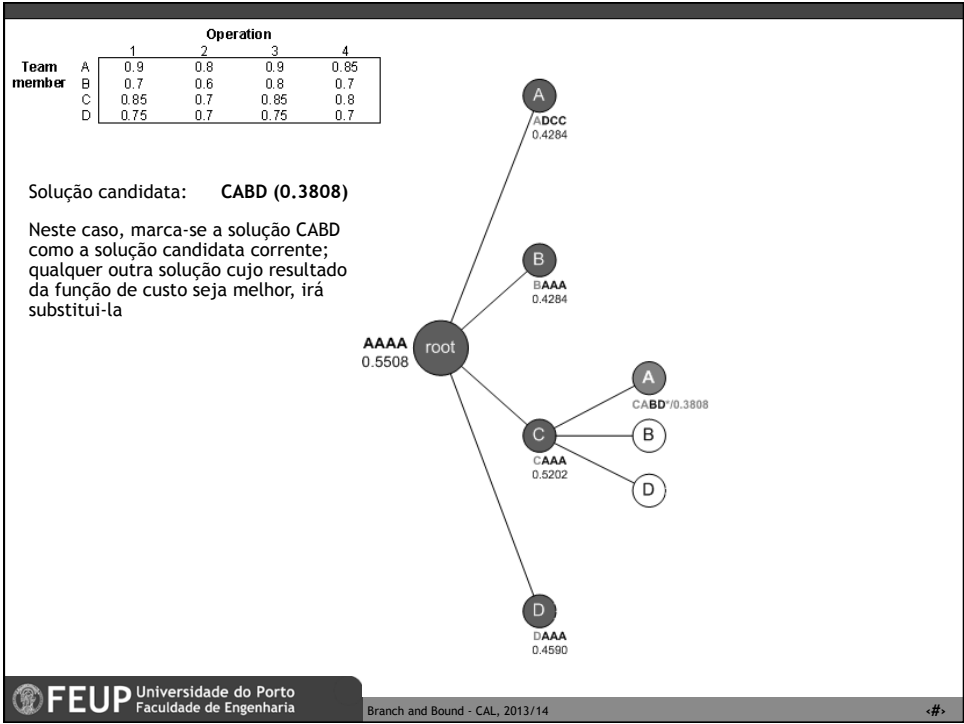
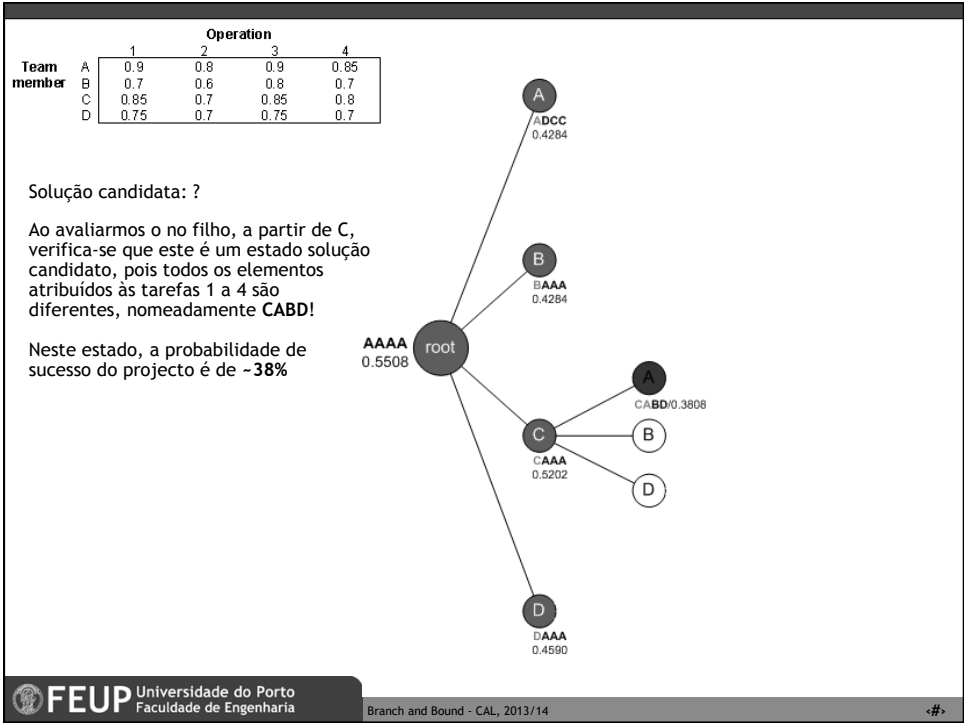
D

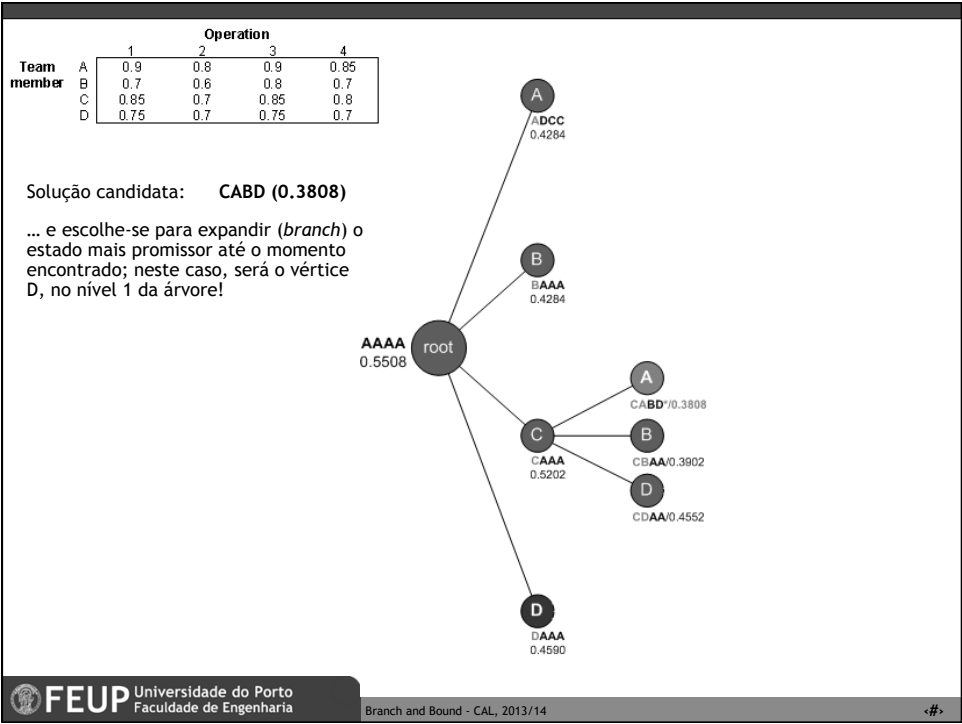
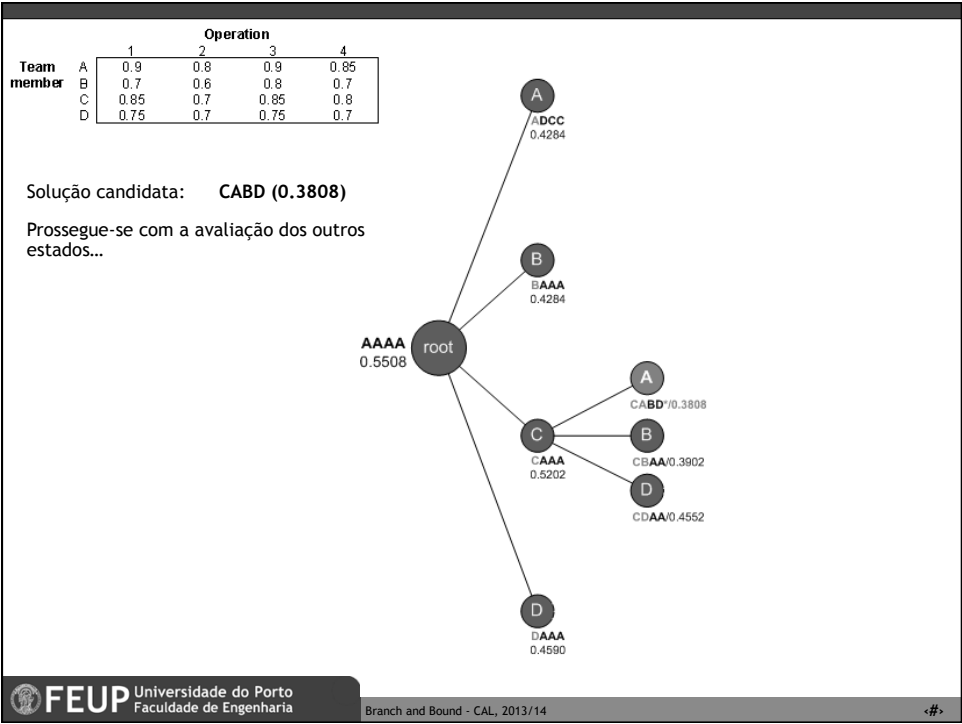
FEUP Universidade do Porto Faculdade de Engenharia

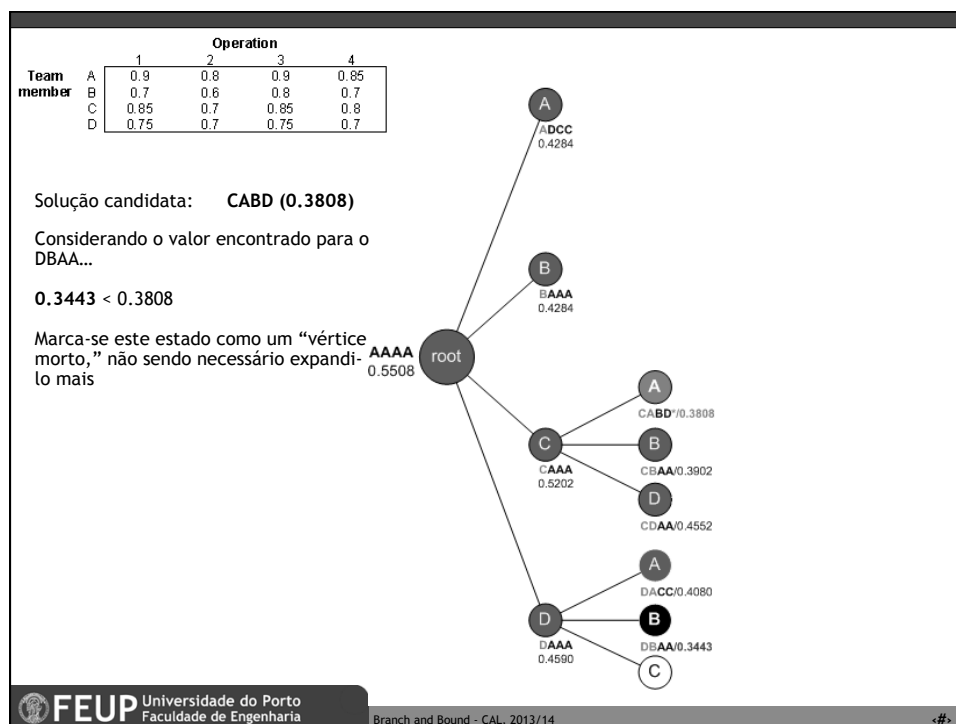
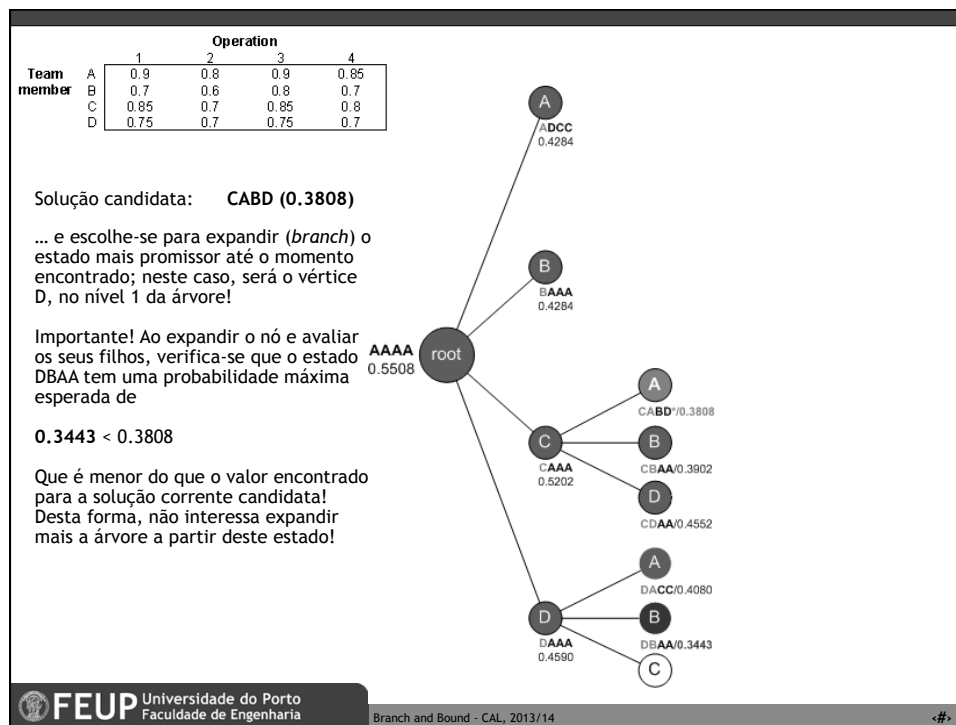
Branch and Bound - CAL, 2013/14

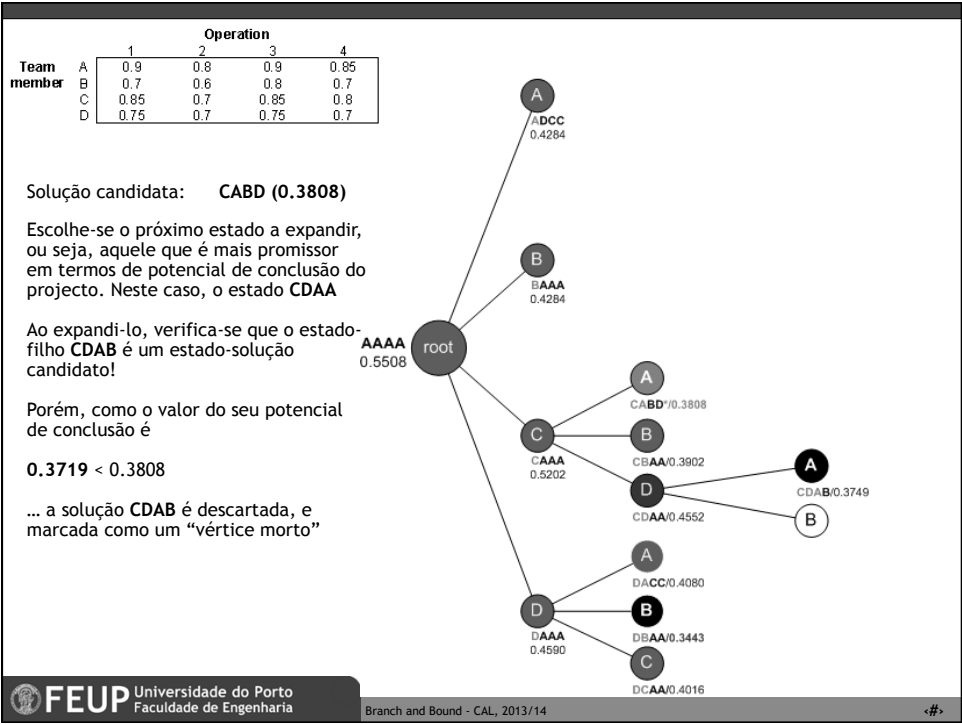
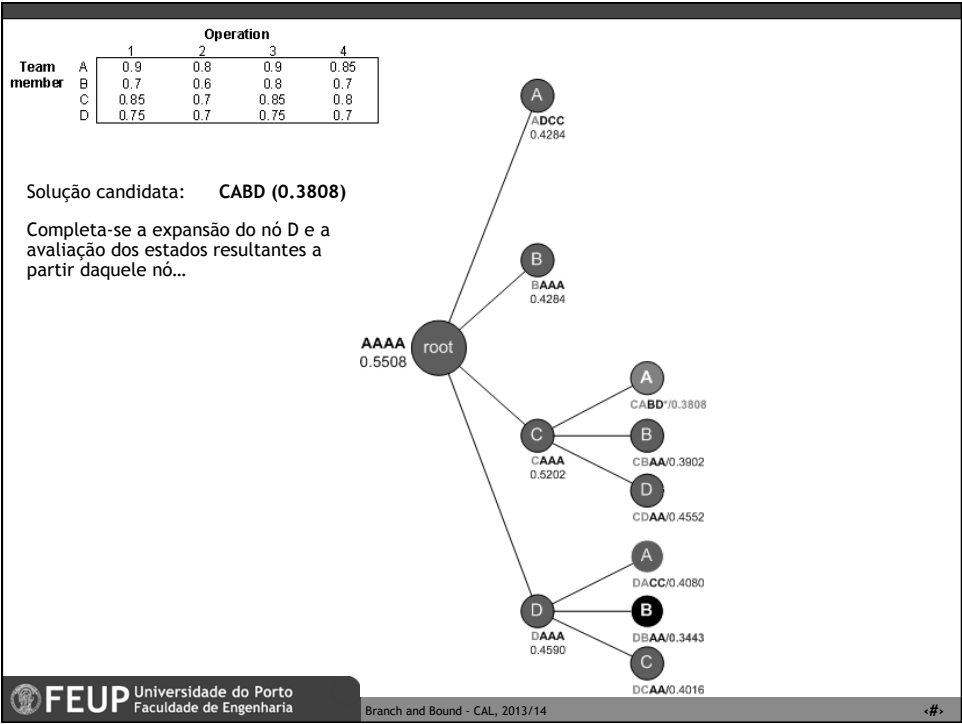


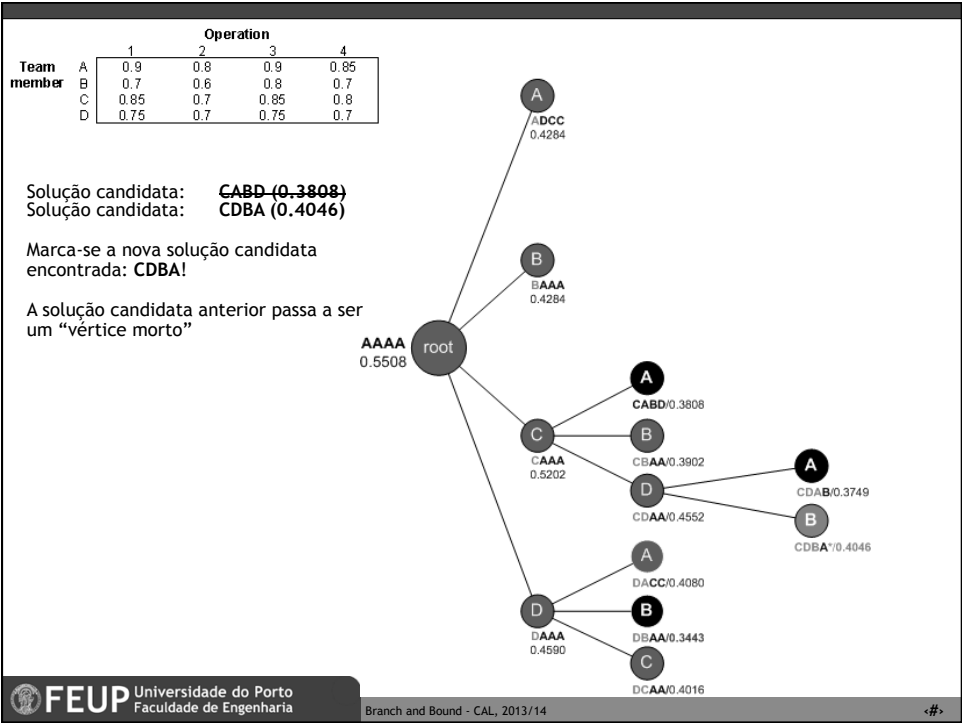
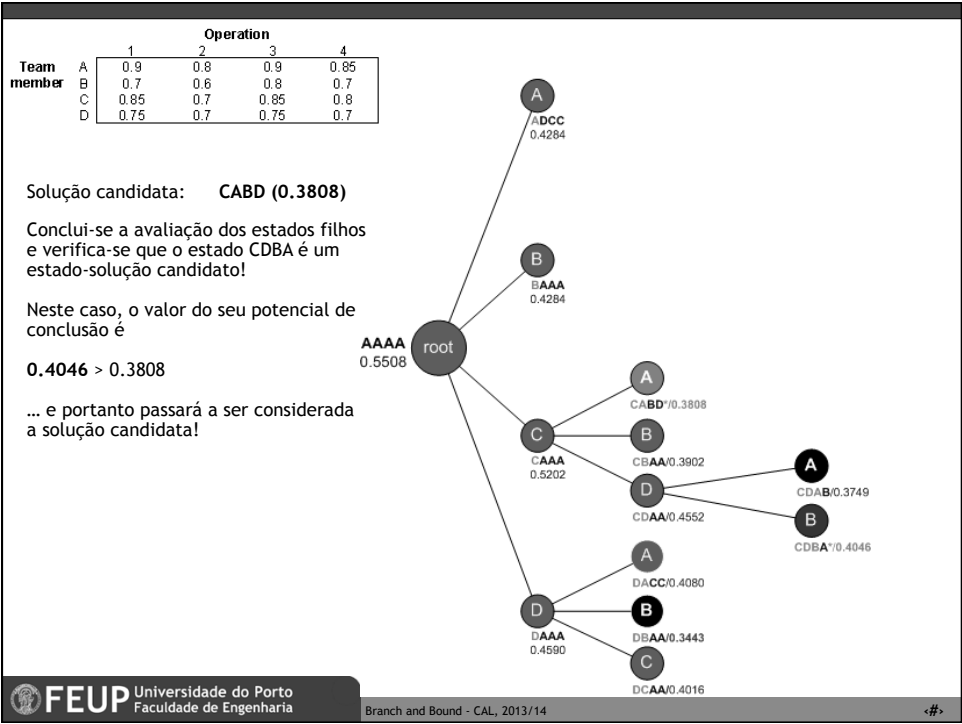


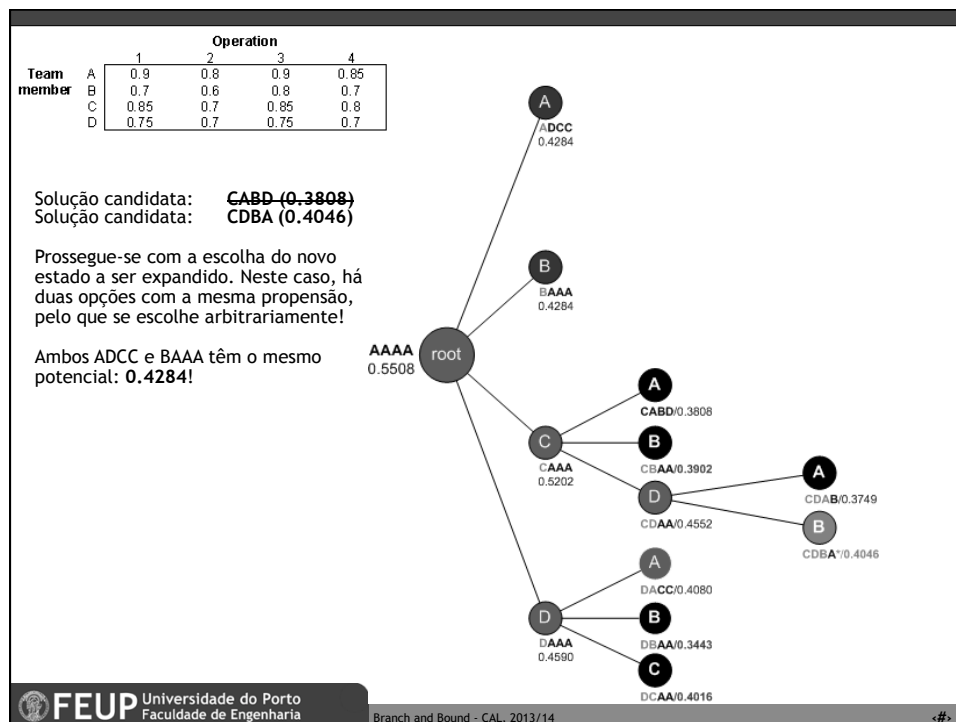
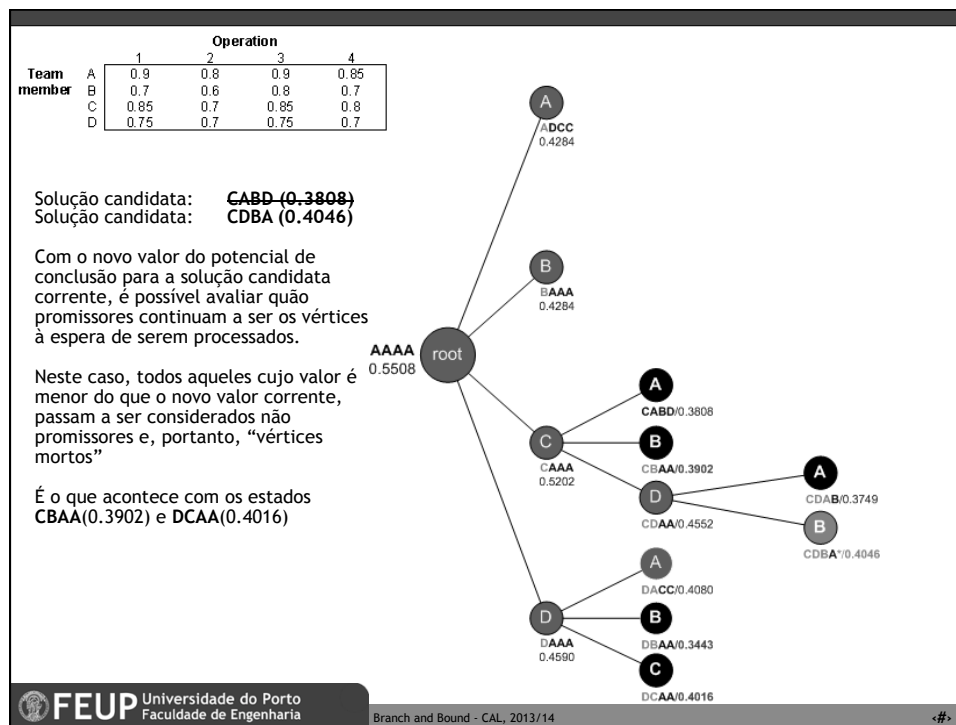


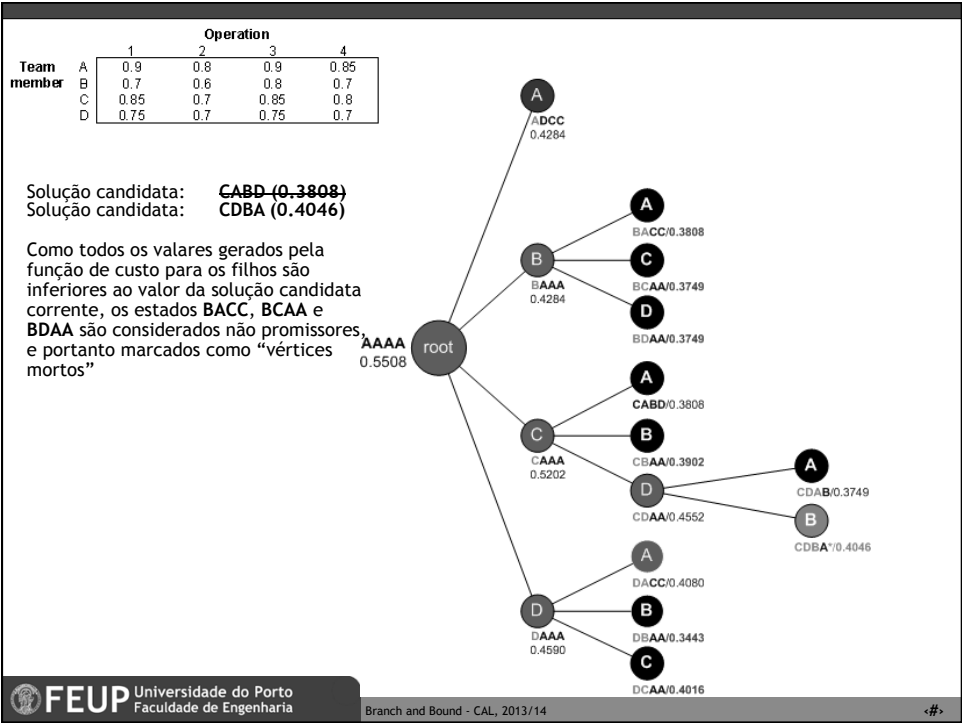
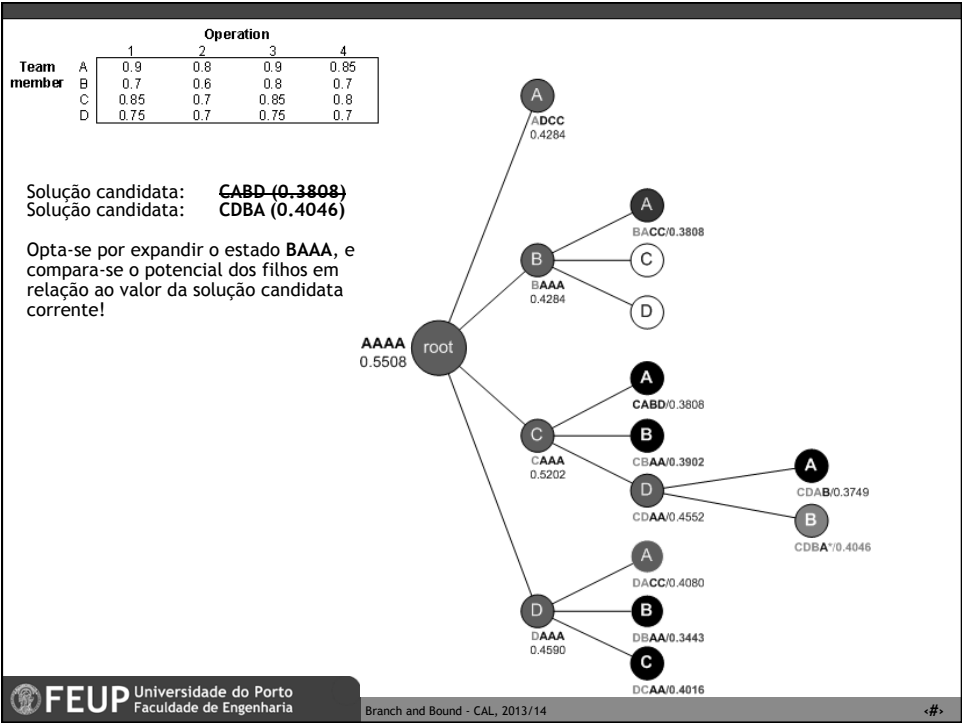


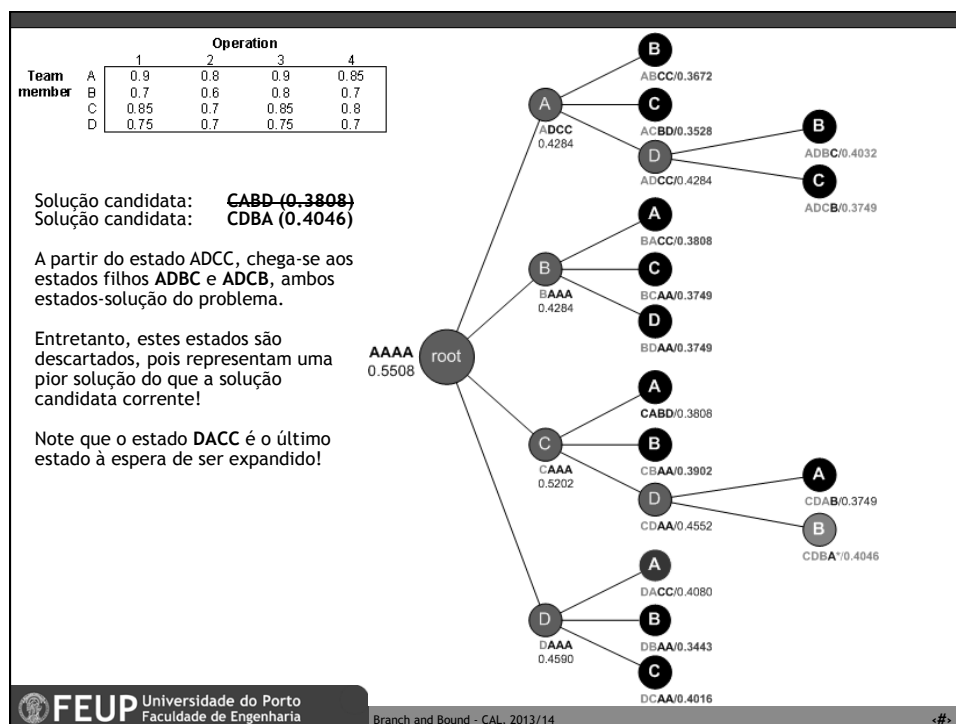
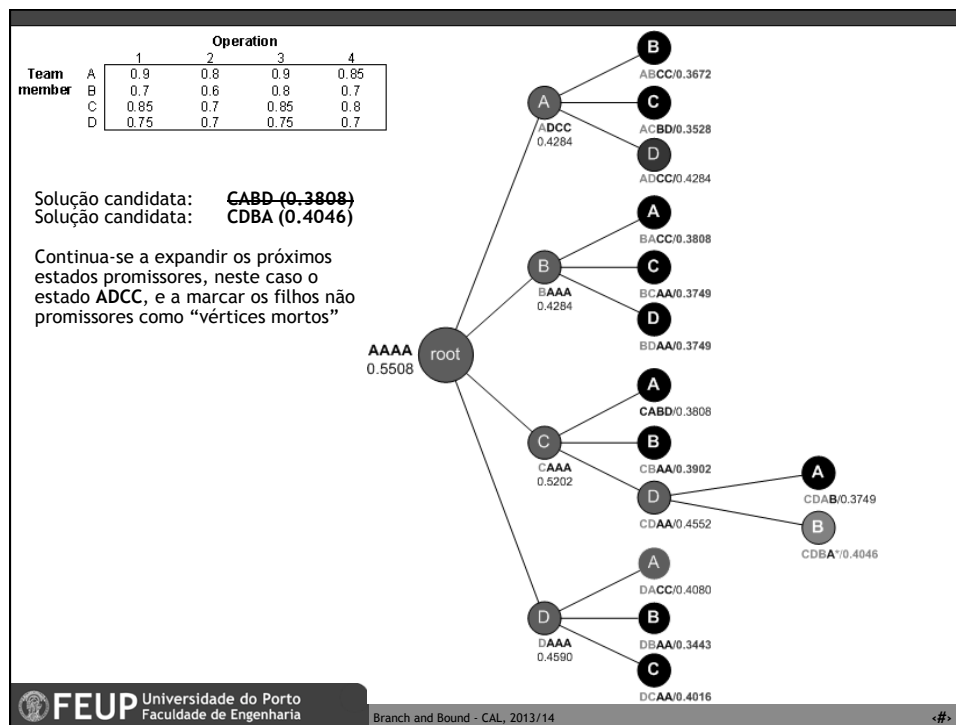


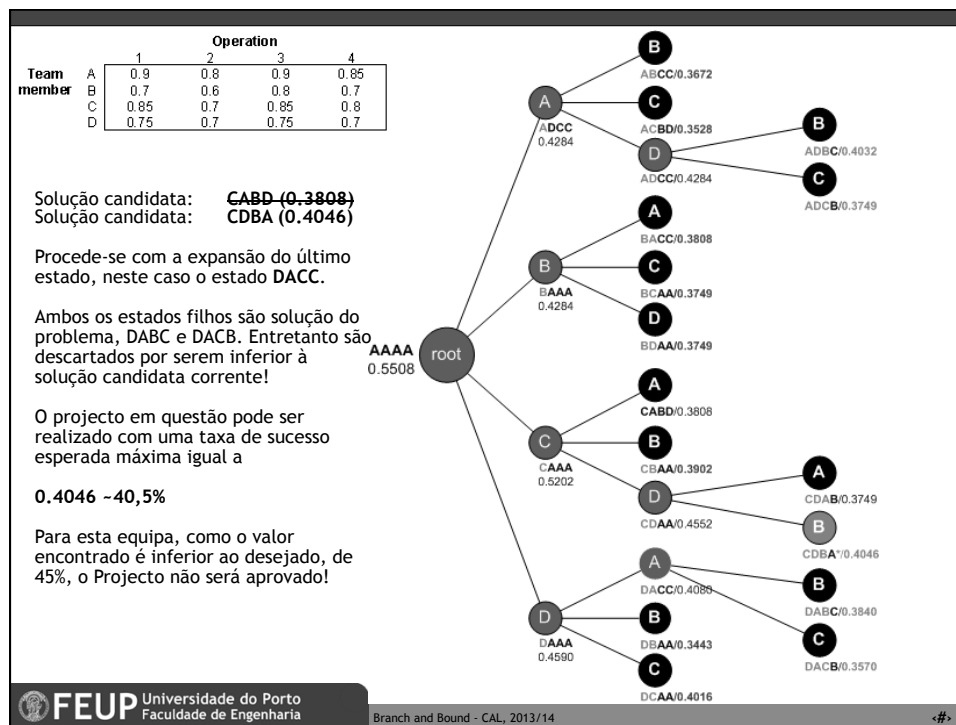












Alguns cuidados a ter em B&B

- Para utilizar B&B com relativo sucesso, é necessário:
 - Ter uma boa função de custo (bounding function): deve ser uma função optimística, mas tão viável quanto possível
 - Reconhecer um estado-solução candidato cedo: é importante para iniciar o processo de poda da árvore, diminuindo o número de estados a serem expandidos
 - Encontrar formas de identificar estados cujos descendentes não são promissores: Este procedimento também evita que vértices sejam expandidos desnecessariamente
 - Ordenar restrições e variáveis, de forma a testar primeiro as mais restritivas: a ideia geral é favorecer que os vértices da árvore sejam considerados não promissores o mais rapidamente possível. O mais próximo da raiz que um vértice é podado, mais a árvore poderá ser simplificada.

Referências e mais informação

- “Introduction to Algorithms”, Second Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2001
- “The Algorithm Design Manual”, Steven S. Skiena, Springer-Verlag, 1998
- A. H. Land and A. G. Doig (1960). "An automatic method of solving discrete programming problems". *Econometrica* 28 (3): pp. 497-520
- John W. Chinneck (2010) “Practical Optimization: a Gentle Introduction”, Carleton University, CA.

