Programação Inteira II - *Branch and Bound*

Alexandre Checoli Choueiri

04/02/2024

Motivação

Motivação

Algoritmo B&B

Já aprendemos o poder de representatividade de situações reais que é possível com a modelagem inteira e binária, porém, ainda **não sabemos como encontrar a solução ótima de tais modelos**. Vimos que o algoritmo Simplex não resolve modelos de PI/PIM/PB, devido ao domínio das variáveis (reais). A maioria dos algoritmos de sucesso de para resolução de problemas inteiros é pautado no método **Branch and Bound** (B&B) (ramificar e limitar), o qual iremos aprender.

Para que possamos entender o algoritmo B&B, primeiro veremos os seguintes conceitos:

1. Estrutura de dados: árvore
2. Relaxação linear
3. Relação função objetivo x restrições

Estruturas de dados: árvores

Estrutura de dados: árvore

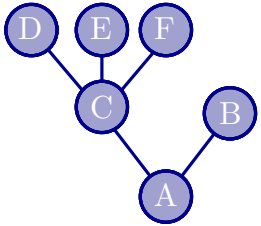
"Trees sprout up just about everywhere in computer science..." Knuth

Uma estrutura de dados muito usada na computação é a **árvore**. Uma **árvore** representa uma estrutura hierárquica, com um conjunto conectado de **nós**. Cada nó da árvore pode ser conectado a muitos **nós filhos**, mas deve ser conectado com exatamente um **nó pai**. O único nó da árvore que não possui pais é chamado de nó raiz.

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

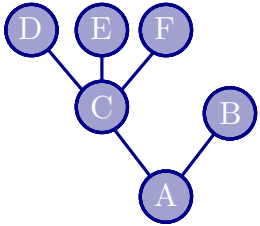
Uma estrutura de dados muito usada na computação é a **árvore**. Uma **árvore** representa uma estrutura hierárquica, com um conjunto conectado de **nós**. Cada nó da árvore pode ser conectado a muitos **nós filhos**, mas deve ser conectado com exatamente um **nó pai**. O único nó da árvore que não possui pais é chamado de nó raiz.Mas por quê é chamada de **árvore**?

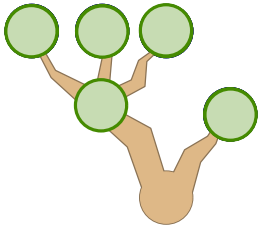


Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Uma estrutura de dados muito usada na computação é a **árvore**. Uma **árvore** representa uma estrutura hierárquica, com um conjunto conectado de **nós**. Cada nó da árvore pode ser conectado a muitos **nós filhos**, mas deve ser conectado com exatamente um **nó pai**. O único nó da árvore que não possui pais é chamado de nó raiz.Mas por quê é chamada de **árvore**?

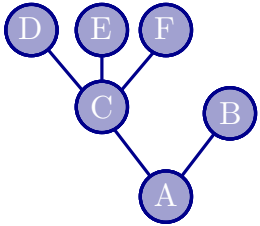


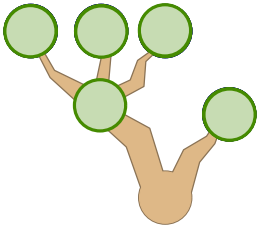


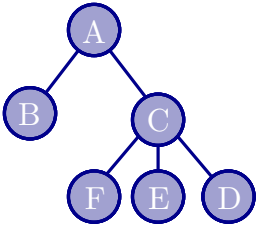
Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Uma estrutura de dados muito usada na computação é a **árvore**. Uma **árvore** representa uma estrutura hierárquica, com um conjunto conectado de **nós**. Cada nó da árvore pode ser conectado a muitos **nós filhos**, mas deve ser conectado com exatamente um **nó pai**. O único nó da árvore que não possui pais é chamado de nó raiz.Mas por quê é chamada de **árvore**?



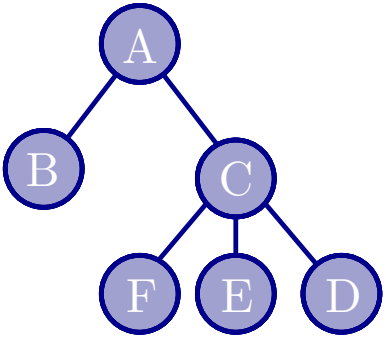




Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Vamos caracterizar alguns nós da árvore ao lado.



Nó raiz:

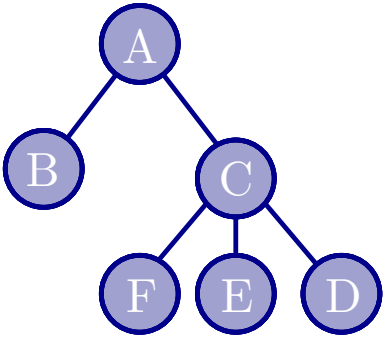
Nós filhos de A:

Nó pai de E:

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Vamos caracterizar alguns nós da árvore ao lado.



Nó raiz:

A

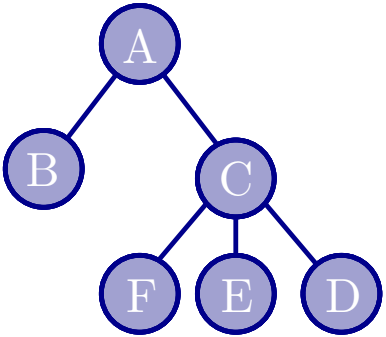
Nós filhos de A:

Nó pai de E:

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Vamos caracterizar alguns nós da árvore ao lado.



Nó raiz:

A

Nós filhos de A:

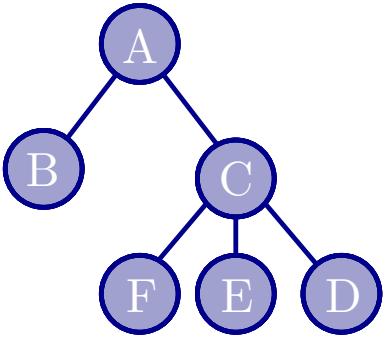
B,C

Nó pai de E:

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Vamos caracterizar alguns nós da árvore ao lado.



Nó raiz:

A

Nós filhos de A:

B,C

Nó pai de E:

C

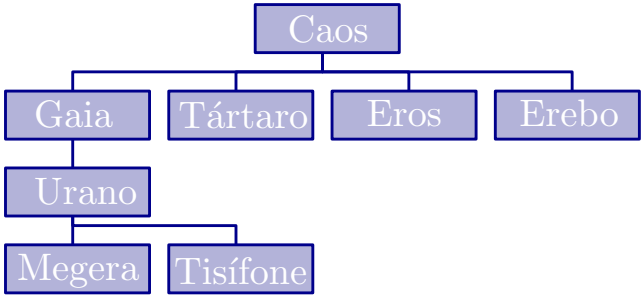
Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Mas para quê utilizamos esta estrutura de dados? Usualmente quando precisamos representar uma **relação-hierárquica**: em uma árvore, os **nós filhos** sempre **"herdam" algo dos nós pais**.

Estrutura de dados: árvore

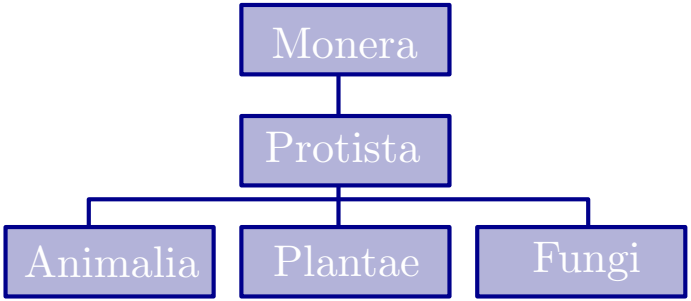
"Trees sprout up just about everywhere in computer science..." Knuth



Como por exemplo em **árvores genealógicas**. A árvore ao lado mostra um pouco da genealogia dos deuses gregos. Note que a relação nó-filho/nó-pai aqui é: "é filho de/gerado por".

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth



A árvore ao lado é chamada de **árvore filogenética das espécies**. Ela mostra a relação de ancestralidade entre os seres vivos.

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

As árvores são muito utilizadas na combinatória, bem como para representar conjuntos disjuntos.

Como podemos encontrar todas as permutações de 4 elementos? (OBS: quantos são esses elementos?)

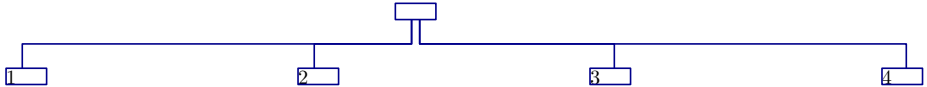
Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth

Começamos escrevendo o conjunto vazio (sem nenhum elemento).

Estrutura de dados: árvore

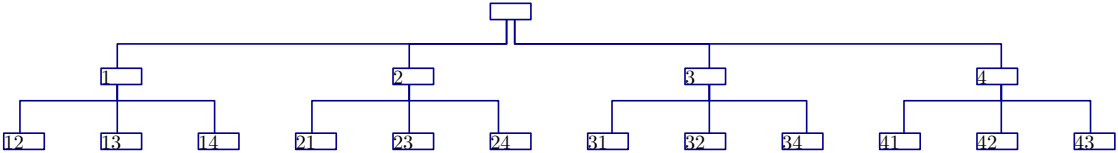
"Trees sprout up just about everywhere in computer science..." Knuth



Em seguida criamos um nó filho para cada possibilidade de preenchimento do primeiro elemento do conjunto. No caso acima temos 4 possibilidades, começar com 1, 2, 3 ou 4.

Estrutura de dados: árvore

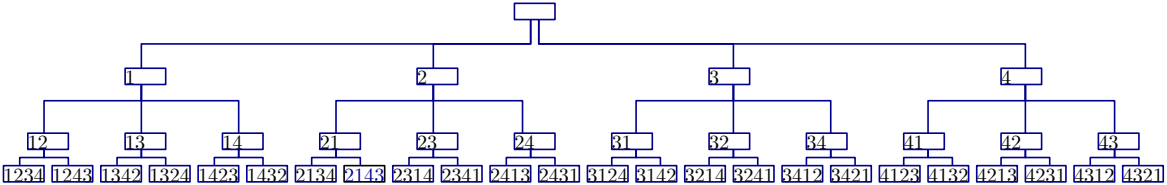
"Trees sprout up just about everywhere in computer science..." Knuth



Da mesma forma, criamos novos nós filhos para cada nó gerado, com as possibilidades restantes para a segunda posição.

Estrutura de dados: árvore

"Trees sprout up just about everywhere in computer science..." Knuth



Finalmente, definindo as terceiras posições as quartas já ficam também definidas. As permutações do conjunto são todos os **nós folhas** (nós que não possuem filhos) da árvore. Note que $P_n = n! = 24$, exatamente o número de nós-folha da árvore ao lado.

Relaxação linear

Relaxação linear

Relaxação linear

A relaxação linear de um PI (PIM/PB) consiste em se **remover as restrições de integralidade** do problema. Uma propriedade importante da relaxação linear é a de que:
A função objetivo da relaxação linear de um PI é sempre melhor, ou tão boa quanto a função objetivo do próprio PI.

Relaxação linear

Relaxação linear

A relaxação linear de um PI (PIM/PB) consiste em se **remover as restrições de integralidade** do problema. Uma propriedade importante da relaxação linear é a de que:
A função objetivo da relaxação linear de um PI é sempre melhor, ou tão boa quanto a função objetivo do próprio PI.

Sejam X_{PI}, X_{RL} os conjuntos de soluções factíveis de PI e de sua relaxação linear, de um problema de maximização, com função objetivo f.

PROVA

1. $\mathbb{Z}^+ \subset \mathbb{R}^+$
2. $X_{PI} \subset X_{PL}$
3. $f(X_{PI}) \leq f(X_{PL})$

Relaxação linear

Relaxação linear

A relaxação linear de um PI (PIM/PB) consiste em se **remover as restrições de integralidade** do problema. Uma propriedade importante da relaxação linear é a de que: A função objetivo da relaxação linear de um PI é sempre melhor, ou tão boa quanto a função objetivo do próprio PI.

Sejam X_{PI}, X_{RL} os conjuntos de soluções factíveis de PI e de sua relaxação linear, de um problema de maximização, com função objetivo f.

PROVA

1. $\mathbb{Z}^+ \subset \mathbb{R}^+ \rightarrow$ Inteiros contidos nos reais
2. $X_{PI} \subset X_{PL} \rightarrow$ Soluções de PI contidos nas soluções do PL
3. $f(X_{PI}) \leq f(X_{PL}) \rightarrow$ FO PI não ultrapassa FO PL

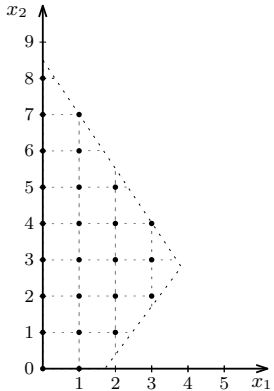
Relaxação linear

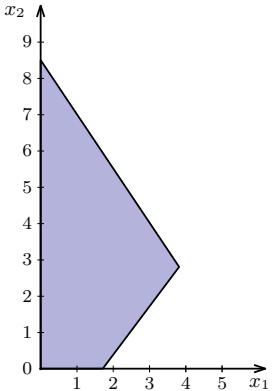
EXEMPLO: Considere o modelo de PI e sua relaxação linear:

$$\begin{aligned} \max z = \; & 5x_1 - x_2 \\ & 7x_1 - 5x_2 \leq 13 \\ & 3x_1 + 2x_2 \leq 17 \\ & x \in \mathbb{Z}^2_+ \end{aligned}$$

$$\begin{aligned} \max z = \; & 5x_1 - x_2 \\ & 7x_1 - 5x_2 \leq 13 \\ & 3x_1 + 2x_2 \leq 17 \\ & x \in \mathbb{R}^2_+ \end{aligned}$$

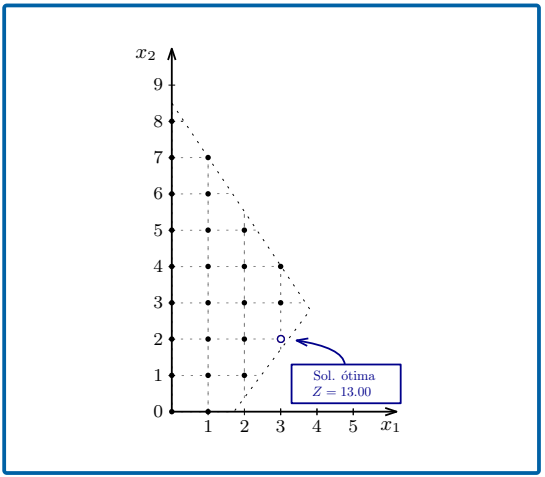
Relaxação linear

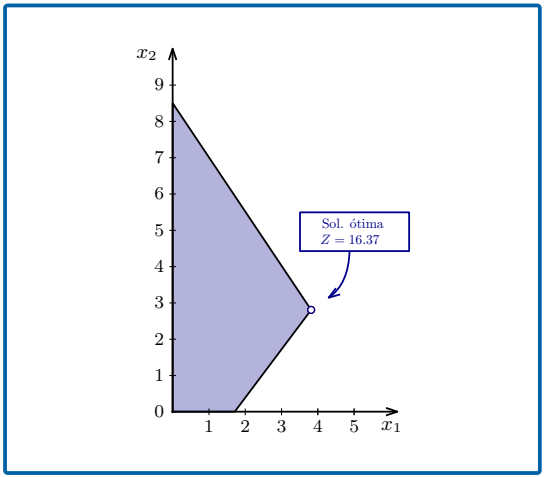




Considere as regiões factíveis do **PI** e da sua **relaxação linear**.

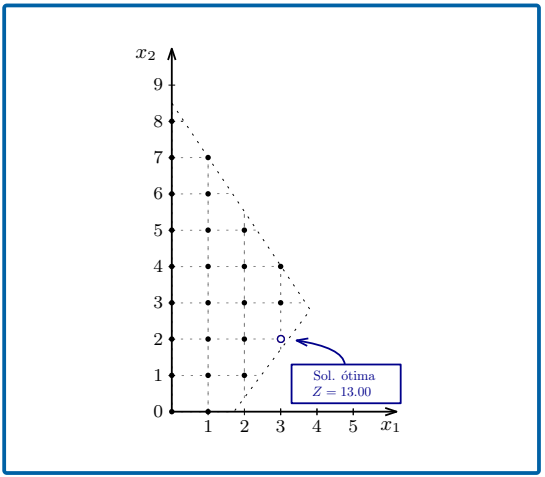
Relaxação linear

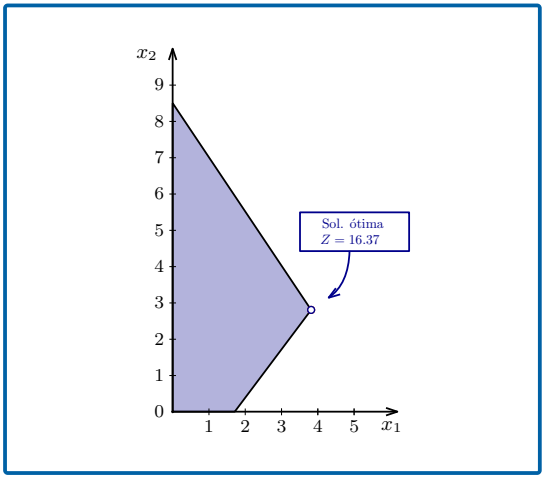




Obviamente a solução inteira está contida na região de soluções reais.

Relaxação linear





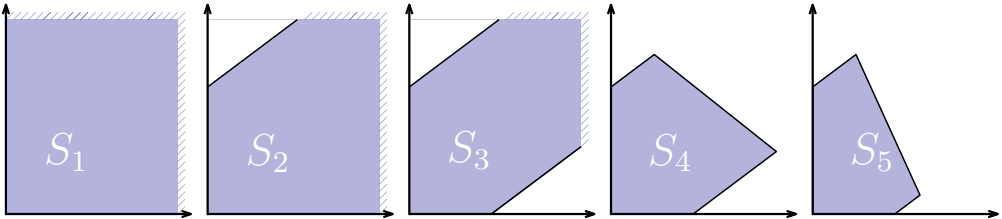
$\therefore$ a FO do PI pode **no máximo ser igual** a da relaxação.

Relação função objetivo x restrições

Relação função objetivo x restrições

Seguindo a mesma lógica da relação de funções objetivo de um PI comparado a sua relaxação linear (**nunca será melhor**, pois um espaço de soluções está contido no outro), podemos inferir a relação de funções objetivo **comparando um modelo ao inserirmos mais restrições no mesmo**.

Relação função objetivo x restrições

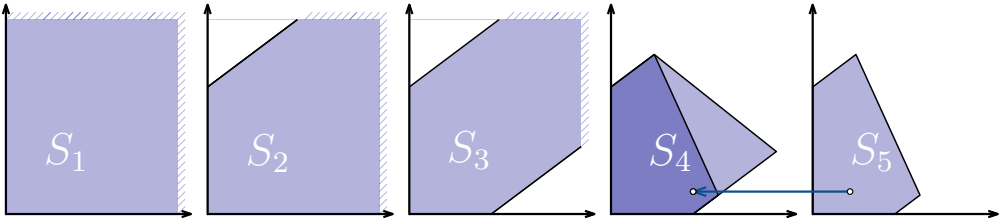


Considere as regiões factíveis acima $\{S_1, S_2, S_3, S_4\}$, em que:

$$S_i = S_{i-1} + r_i \tag{1}$$

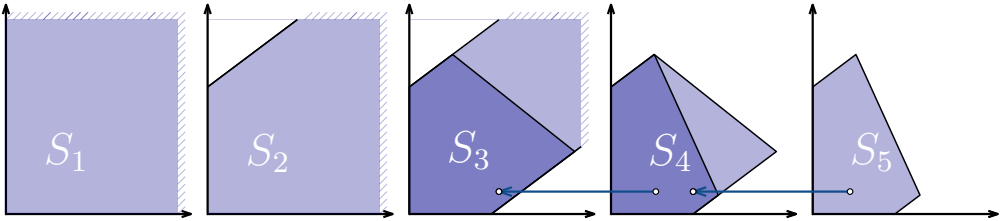
Com r_i sendo uma nova restrição. Ou seja, cada nova região é feita pelo PL da região anterior + uma nova restrição.

Relação função objetivo x restrições



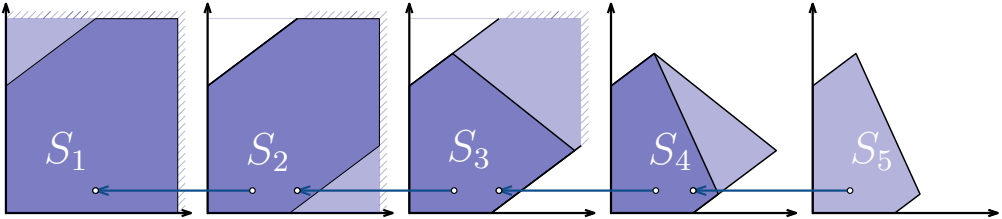
Percebemos que $S_5 \subset S_4$, portanto **não é possível** que a função objetivo em S_5 seja melhor do que em S_4, pois todo o espaço de busca de S_5 também está em S_4.

Relação função objetivo x restrições



O mesmo ocorre agora com S_4 em relação a S_3, ou seja, $S_4 \subset S_3$

Relação função objetivo x restrições



Da mesma forma com as outras regiões. Assim, se o problema for de maximização e $f_1, ..., f_4$ os valores das fo´s para os problemas, temos que:

$$f_1 \geq f_2 \geq f_4 \geq f_5 \tag{2}$$

Relação função objetivo x restrições

Conclusão

Ao adicionarmos restrições em um modelo de PL, a sua função objetivo nunca poderá melhorar. Na melhor das hipóteses ela permanece constante.

O algoritmo *Branch and Bound*

O algoritmo *Branch and Bound*

Agora já possuímos todas as ferramentas para entender o algoritmo B&B (ramifica e limita)

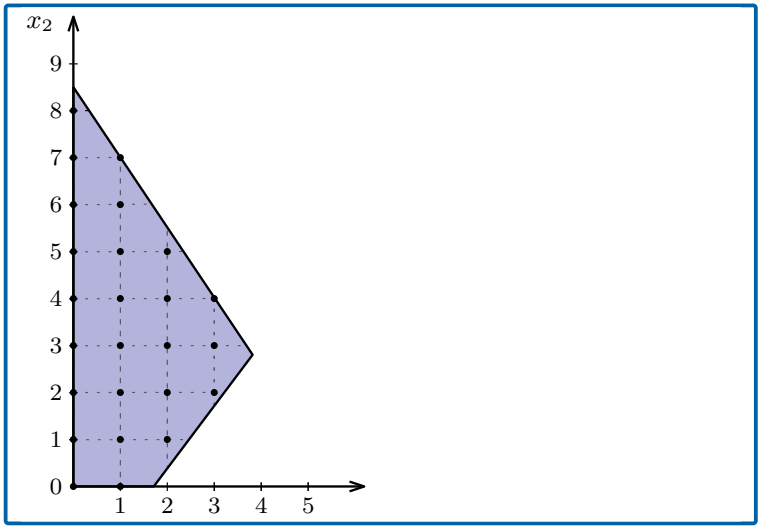
1. Estrutura de dados: árvore
2. Relaxação linear
3. Relação função objetivo x restrições

A ideia do B&B é a de **dividir para conquistar**. A cada iteração dividimos (**branch**) o problema original em partes menores e mais fáceis de serem resolvidas. Dessa forma multiplicamos o número de problemas. Usando as ideias de relaxação linear podemos eliminar alguns desses problemas (**bound**). Os problemas e subproblemas são salvos em uma **estrutura de dados em árvore**.

O algoritmo *Branch and Bound*

A melhor forma de entender o algoritmo é por meio de um **exemplo numérico**. Considere o modelo abaixo, com região factível mostrada ao lado.

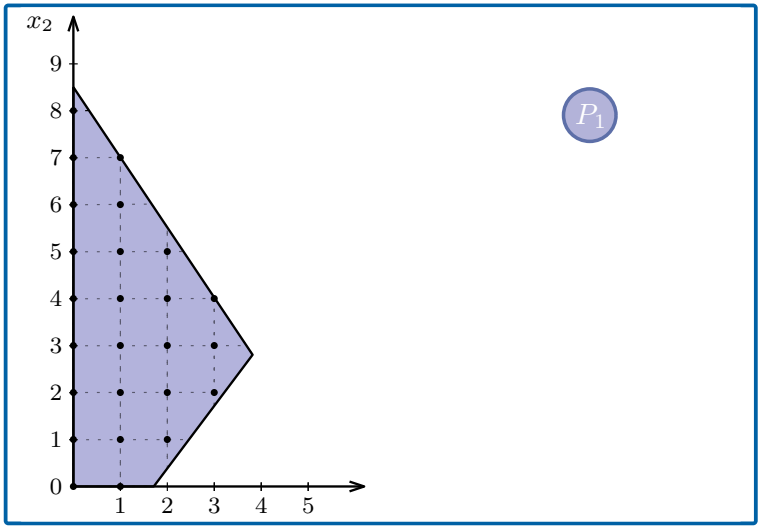
$$\begin{aligned} \max z = {} & 5x_1 - x_2 \\ & 7x_1 - 5x_2 \leq 13 \\ & 3x_1 + 2x_2 \leq 17 \\ & x \in \mathbb{Z}^2_+ \end{aligned}$$



O algoritmo *Branch and Bound*

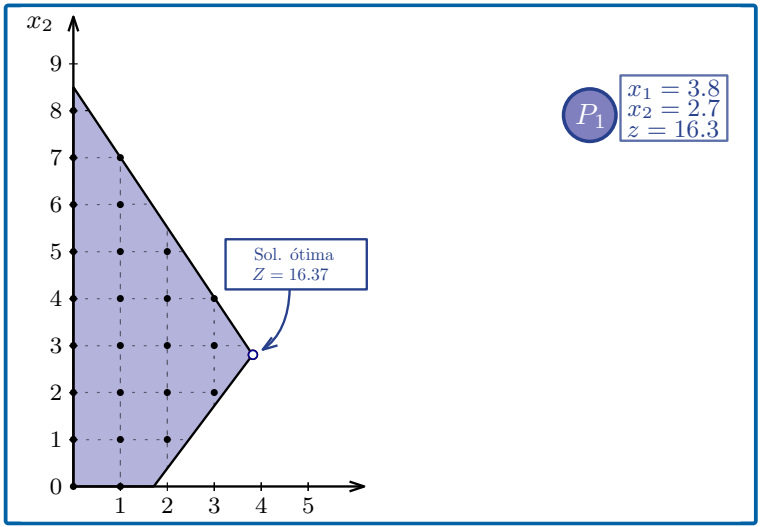
Inicialmente chamamos a relaxação linear do problema original de P_1, e o adicionamos a uma lista de problemas a serem resolvidos $L = \{P_1\}$. Esse problema inicial é a raiz da árvore *Branch and Bound*. A relaxação P_1 fica então:

$$\begin{aligned} \max z = \; & 5x_1 - x_2 \\ & 7x_1 - 5x_2 \leq 13 \\ & 3x_1 + 2x_2 \leq 17 \\ & x \in \mathbb{R}^2_+ \end{aligned}$$



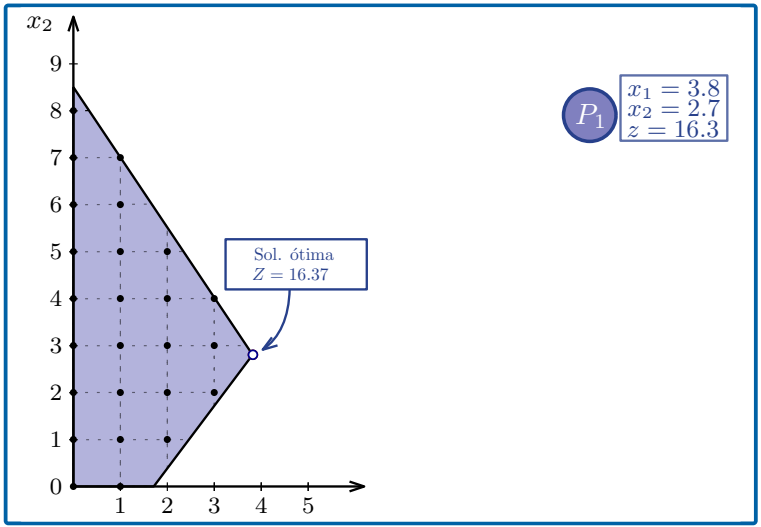
O algoritmo *Branch and Bound*

Em seguida, resolvemos o problema P_1 (pelo próprio algoritmo Simplex). Note que a solução **não é inteira** ($x_1 = 3.8, x_2 = 2.7$), portanto ainda **não possuímos uma solução factível para o problema original P**.



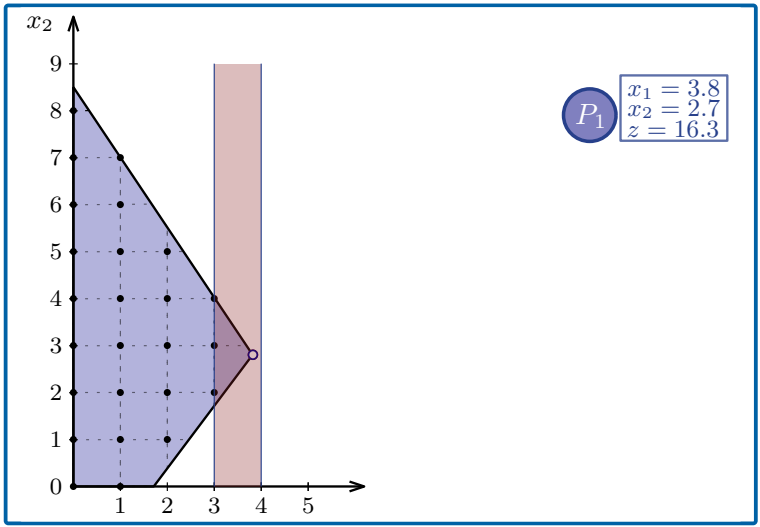
O algoritmo *Branch and Bound*

Como P_1 já foi resolvido, removemos ele da lista de problemas L. $L = \emptyset$. Agora escolhemos uma variável que ainda não é inteira (no caso de P_1, x_1 ou x_2) para tentar corrigi-la. Nesta primeira etapa escolheremos x_1.



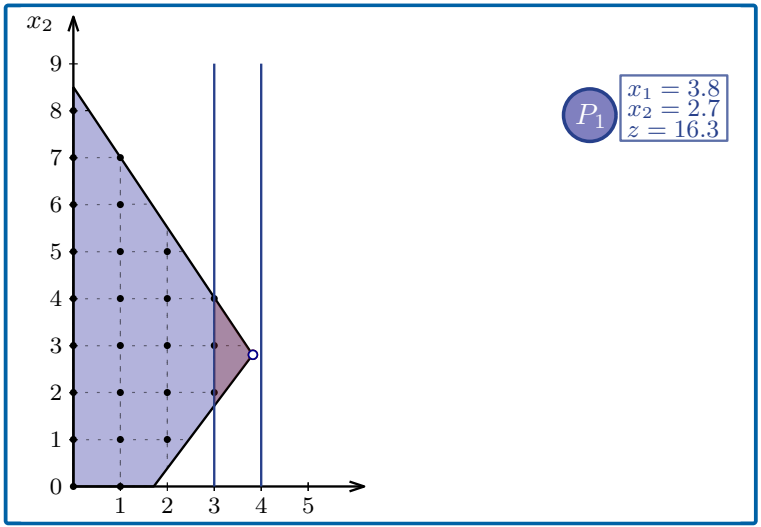
O algoritmo *Branch and Bound*

Olhando na região factível de P_1, percebemos que x_1 está entre 2 números inteiros: 3 e 4. Mais ainda, sabemos que o valor ótimo de x_1 para o problema inteiro, **nunca estará no intervalo** $]3, 4[$.



O algoritmo *Branch and Bound*

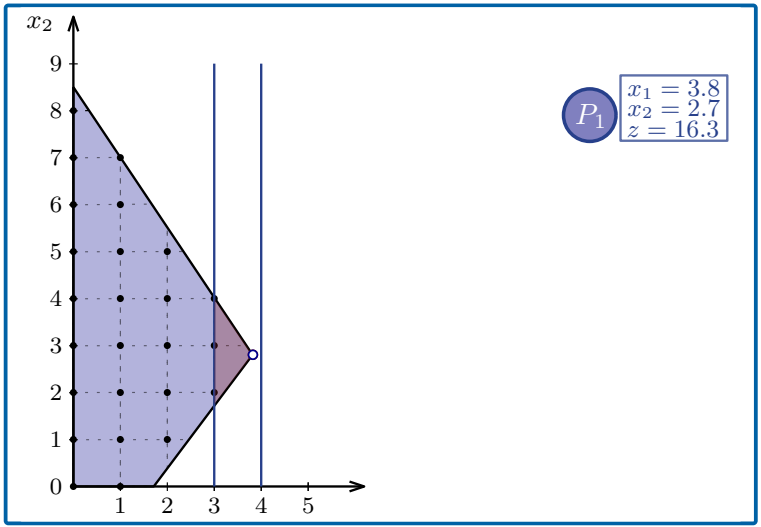
O que é equivalente a dizer quer a área em vermelho ao lado pode ser removida do problema original.



O algoritmo *Branch and Bound*

O que é equivalente a dizer quer a área em vermelho ao lado pode ser removida do problema original.

Mas como podemos fazer isso?

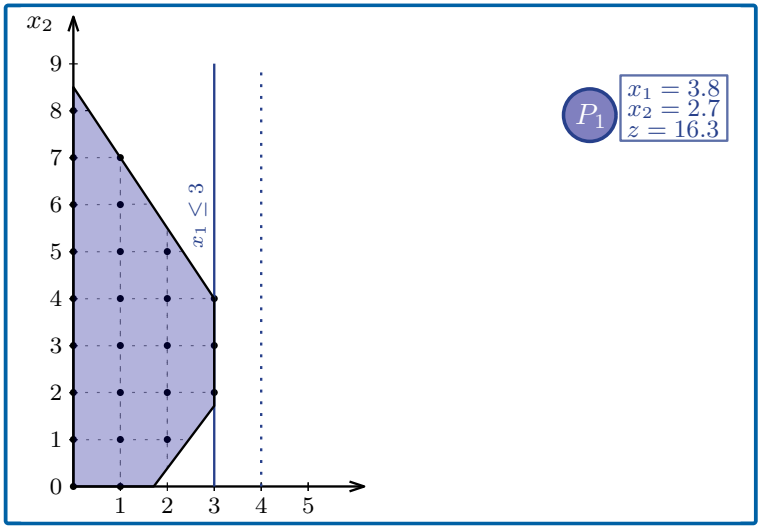


O algoritmo *Branch and Bound*

Com restrições! Para garantir que x_1 seja menor ou igual a 3, adicionamos a restrição:

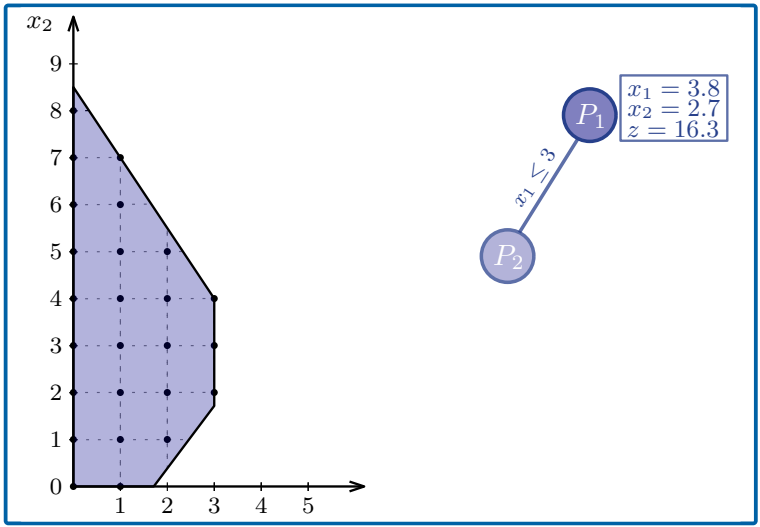
$$x_1 \leq 3$$

Note que essa nova restrição deve ser adicionada ao problema P_1, ou seja, criamos um novo subproblema (P_2) que herda tudo de P_1 + essa nova restrição.



O algoritmo *Branch and Bound*

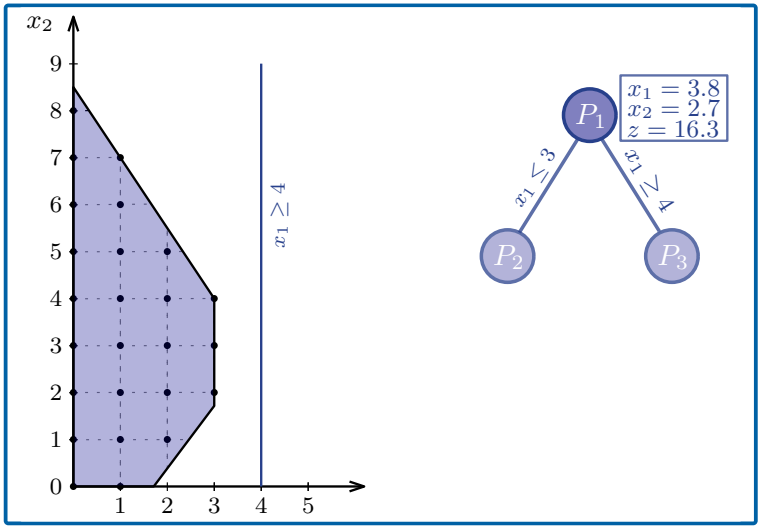
A melhor forma de representar essa relação hierárquica entre os problemas é justamente por meio de uma árvore. Criamos um novo nó filho de P_1, com o problema P_2. Como criamos um novo problema, temos que adicioná-lo a lista de problemas a serem verificados. $L = \{P_2\}$.



O algoritmo *Branch and Bound*

Mas ainda não acabamos. Lembre que decidimos que $x_1 \notin]3, 4[$. Só geramos o subproblema da "esquerda" (menor ou igual a 3). Da mesma forma, a solução ótima pode estar contida na outra região. Adicionamos um novo subproblema P_3, que é P_1 + a restrição:

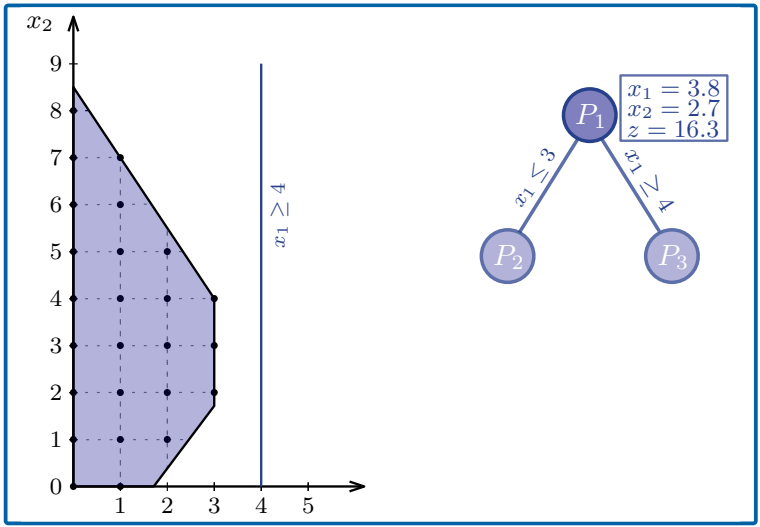
$$x_1 \geq 4$$



O algoritmo *Branch and Bound*

Sabemos só olhando a região que P_4 é infactível, porém computacionalmente não teria como saber isso antes de resolver o problema. Também adicionamos P_4 na lista de problemas a serem resolvidos. A lista fica então:

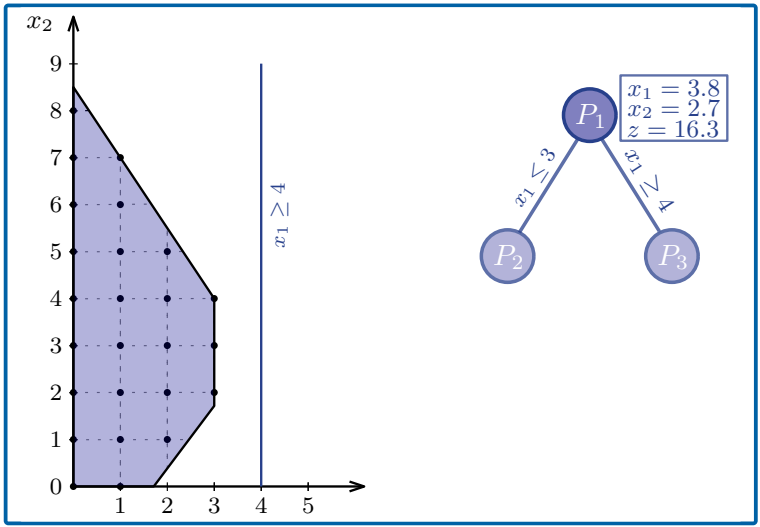
$$L = \{P_2, P_3\}$$



O algoritmo *Branch and Bound*

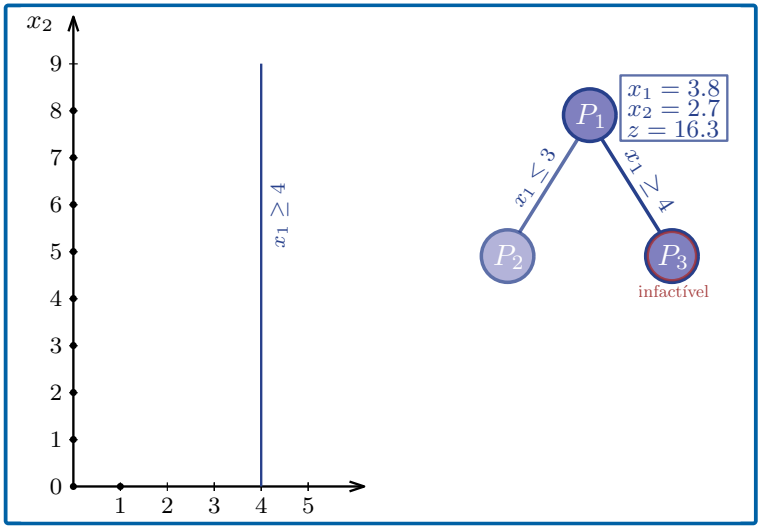
Esta etapa é chamada de *branching* (**ramificação**), pois estamos ramificando a árvore em novos subproblemas.

Temos então 2 nós da árvore B&B que ainda não foram visitados (problemas da lista L). O próximo passo então é selecionar um nó para resolvê-lo.



O algoritmo *Branch and Bound*

Seja o nó selecionado P_3. Resolvemos P_3 e o removemos da lista L. Como P_3 é infactível, esse nó é considerado eliminado, ou seja, **nenhuma ramificação será feita nele**. Esse é o primeiro dentre 3 critérios para a parada de exploração dos nós: infactibilidade.

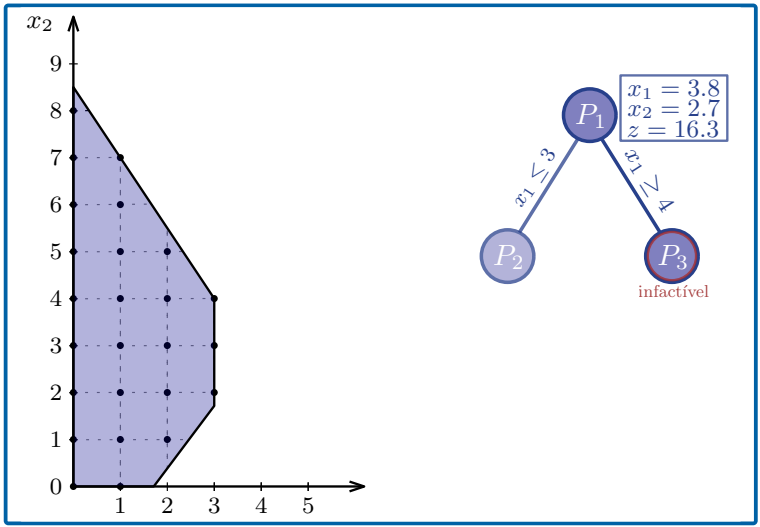


O algoritmo *Branch and Bound*

O critério de parada do algoritmo é quando não existirem mais nós a serem explorados na árvore, ou seja: $L = \emptyset$. Temos então que escolher um nó da árvore para ser explorado, dentre:

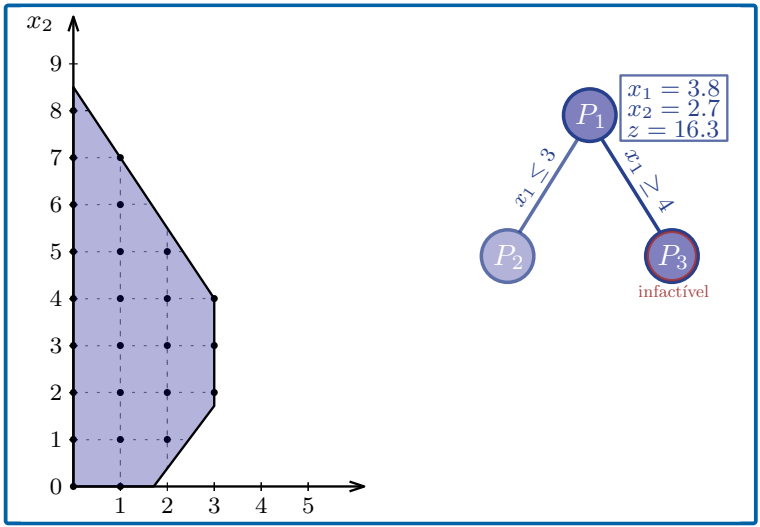
$$L = \{P_2\}$$

Por falta de opção, só podemos selecionar P_2.



O algoritmo *Branch and Bound*

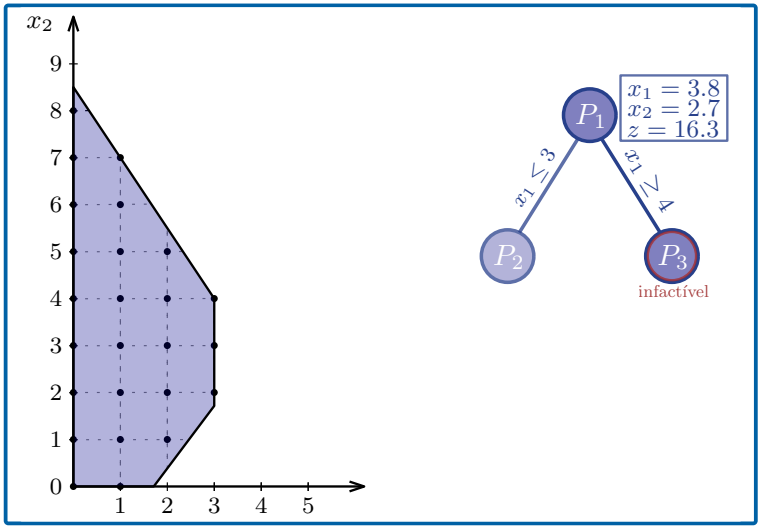
PERGUNTA: O que podemos afirmar sobre o valor da fo de P_2 em relação ao valor da fo de P_1, mesmo antes de resolvermos?



O algoritmo *Branch and Bound*

PERGUNTA: O que podemos afirmar sobre o valor da fo de P_2 em relação ao valor da fo de P_1, mesmo antes de resolvermos?

$$\mathbf{z_2 \leq z_1} \qquad (3)$$

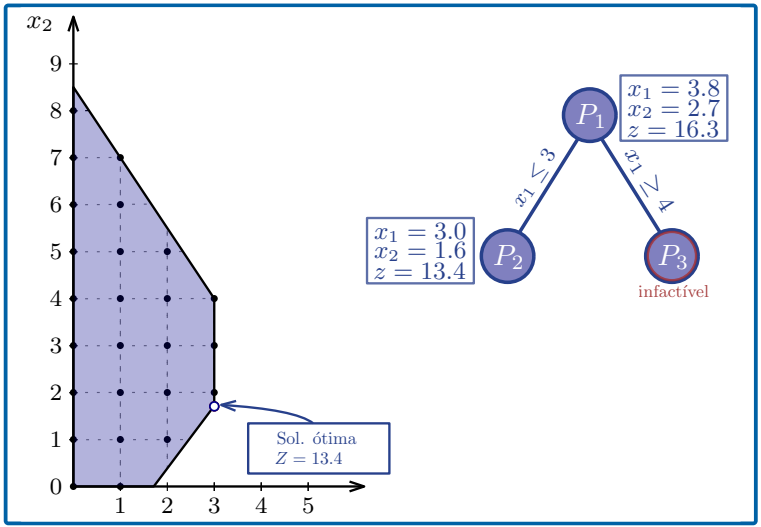


O algoritmo *Branch and Bound*

Resolvendo P_2 (e removendo-o da lista L) temos a solução:

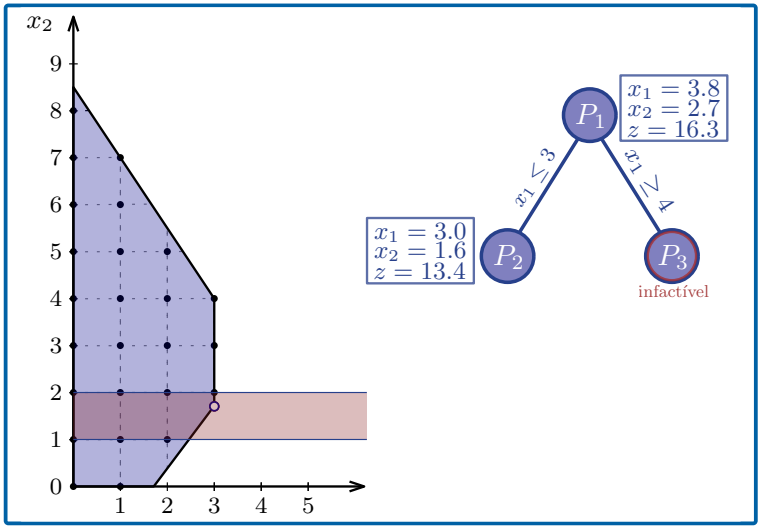
$$\begin{cases} x_1 = 3.0 \\ x_2 = 1.6 \end{cases}$$

Note que já conseguimos uma variável inteira (x_1). Mas ainda não temos uma solução factível, pois $x_2 \in \mathbb{R}$.



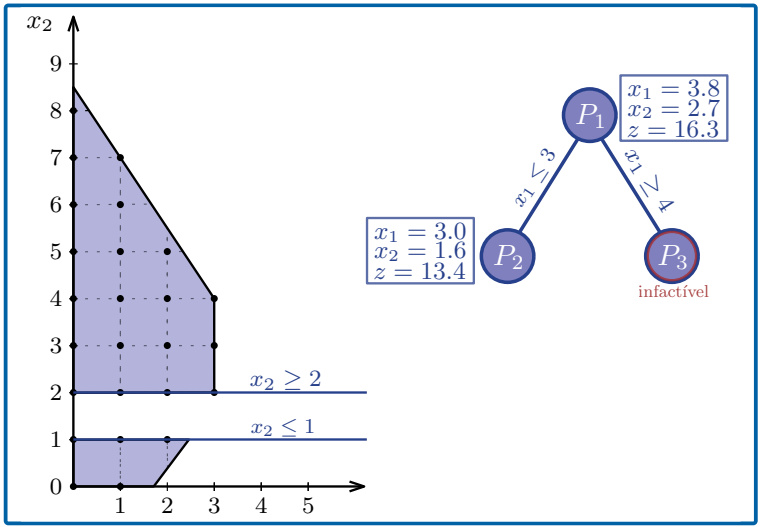
O algoritmo *Branch and Bound*

O próximo passo então é a ramificação do nó P_2 em outros 2 subproblemas. Como só temos uma variável real, selecionamos x_2 para criar os subproblemas.



O algoritmo *Branch and Bound*

Da mesma forma que anteriormente feito com x_1, percebemos que a solução ótima para x_2 não está no intervalo $]1, 2[$, assim criamos as restrições mostradas ao lado, com os dois subproblemas gerados, P_4 e P_5.

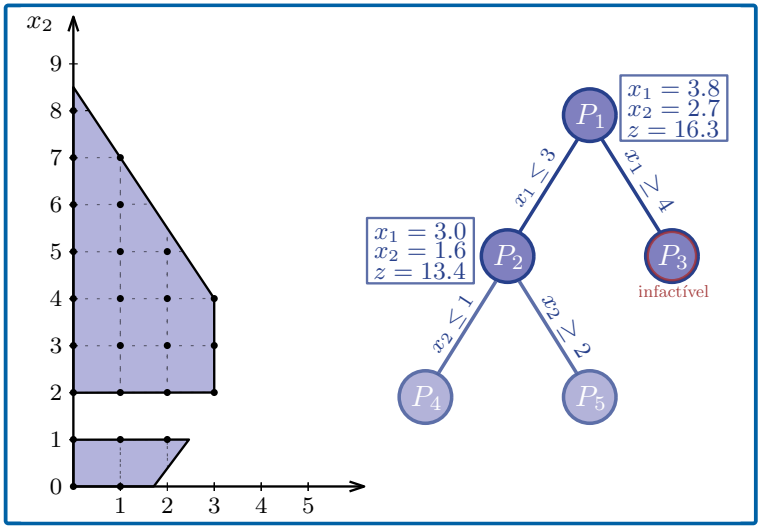


O algoritmo *Branch and Bound*

Os dois subproblemas são adicionados á lista de problemas L:

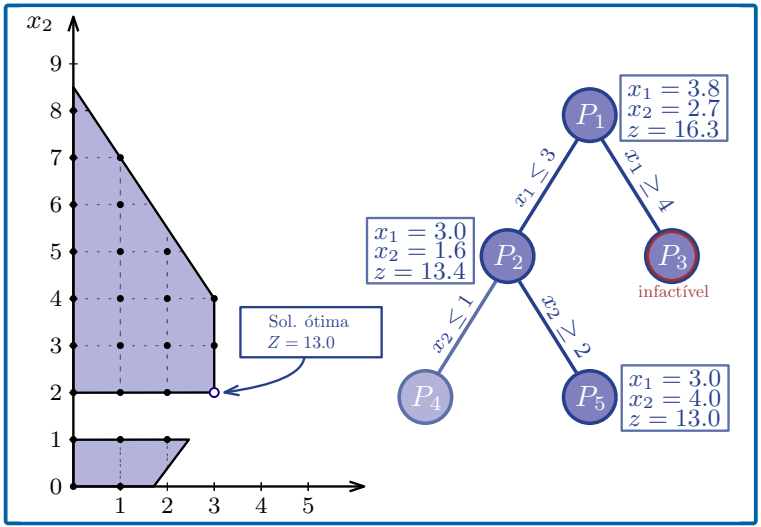
$$L = \{P_4, P_5\}$$

Novamente, escolhemos um subproblema para ser resolvido. Escolhendo o nó P_5, e o eliminando da lista L.



O algoritmo *Branch and Bound*

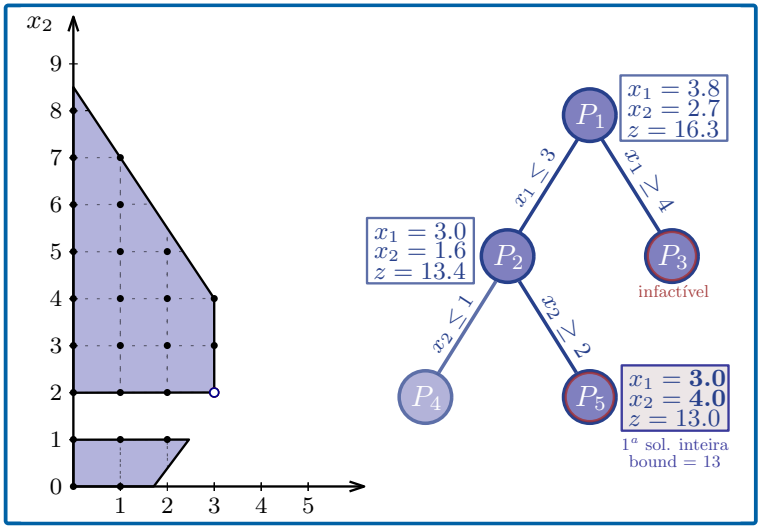
Ao resolvermos P_5, temos a primeira solução inteira.



O algoritmo *Branch and Bound*

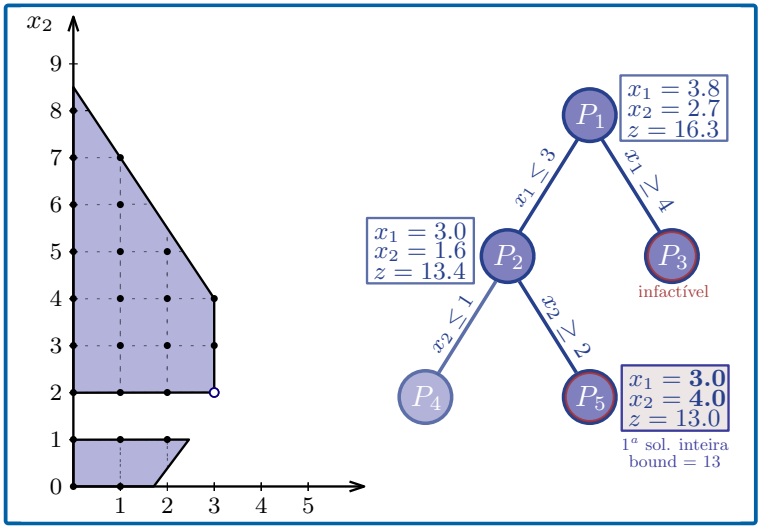
Com essa solução, atualizamos os valores da melhor solução inteira até o momento, e do valor da função objetivo desta solução:

$$\begin{cases} x^* = x_{P5} \\ z^* = z_{P5} = 13 \end{cases}$$



O algoritmo *Branch and Bound*

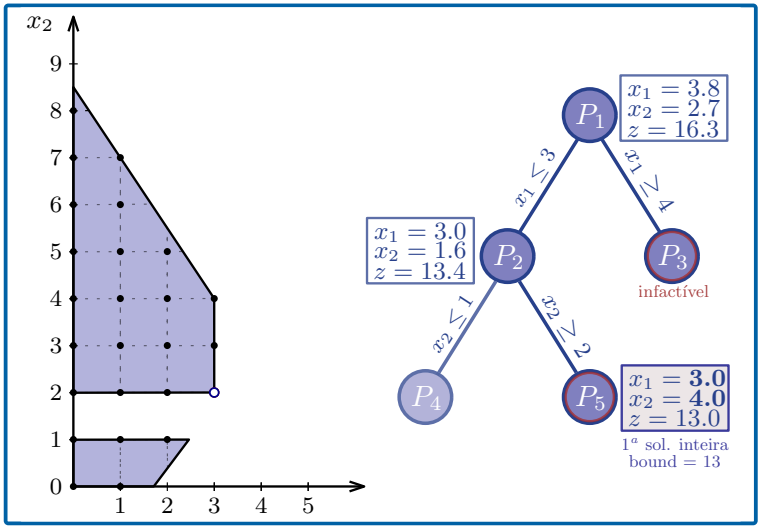
O valor de z^* é chamado de **bound** (limitante). Ou seja, sabemos que a solução ótima **não será menor do que 13** (pois já temos uma solução inteira com valor de 13).



O algoritmo *Branch and Bound*

O valor de z^* é chamado de **bound** (limitante). Ou seja, sabemos que a solução ótima **não será menor do que 13** (pois já temos uma solução inteira com valor de 13).

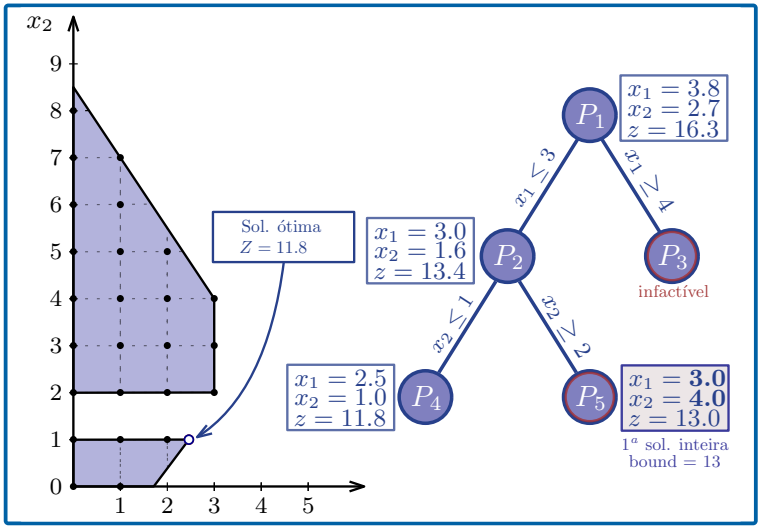
Sempre que uma solução inteira é encontrada, **atualizamos os valores de** x^* e z^* se eles forem melhores que os atuais.



O algoritmo *Branch and Bound*

Como o nó P_5 foi eliminado por ser inteiro, ele não gera filhos. Continuamos explorando a árvore, temos que explorar o nó P_4. Resolvendo P_4 temos (e removendo ele da lista L):

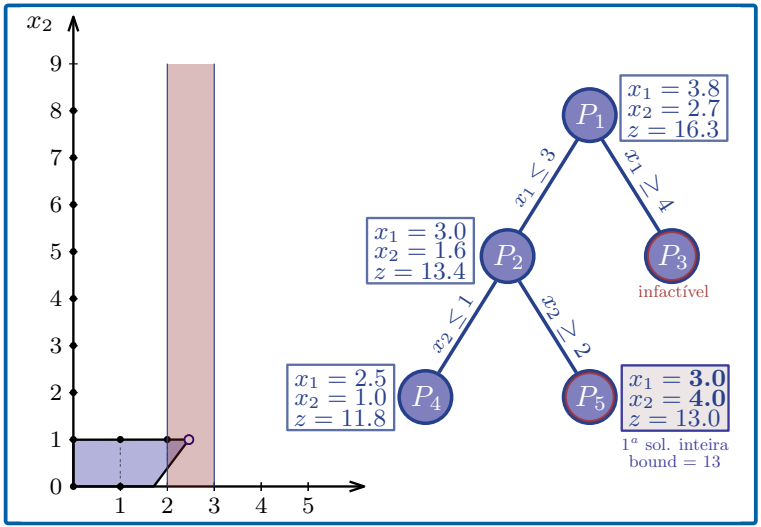
$$\begin{cases} x_1 = 3.0 \\ x_2 = 1.6 \end{cases}$$



O algoritmo *Branch and Bound*

A solução de P_4 não é inteira em x_1. O que devemos fazer? Continuar ramificando, criando mais dois subproblemas em x_1?

Agora usamos a informação do nosso **melhor bound** z^*

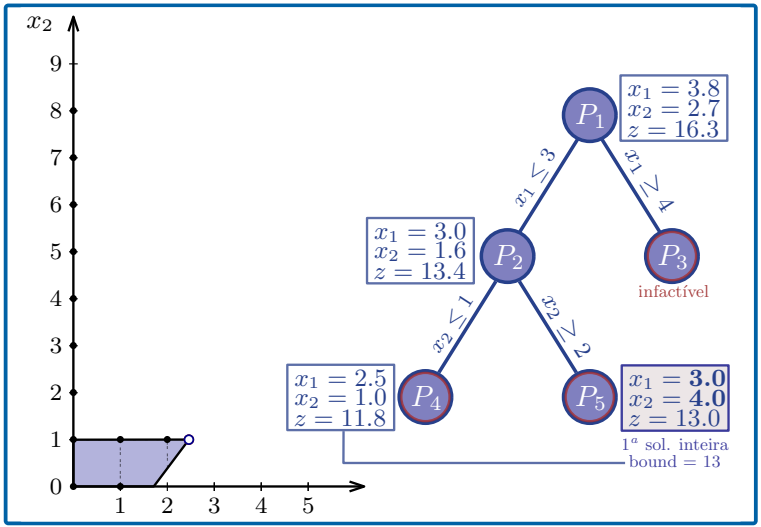


O algoritmo *Branch and Bound*

Temos um **bound** em $z^* = 13$, ou seja, sabemos que a solução ótima não pode ser menor do que 13. Como $z_{P4} = 11.8$, temos que:

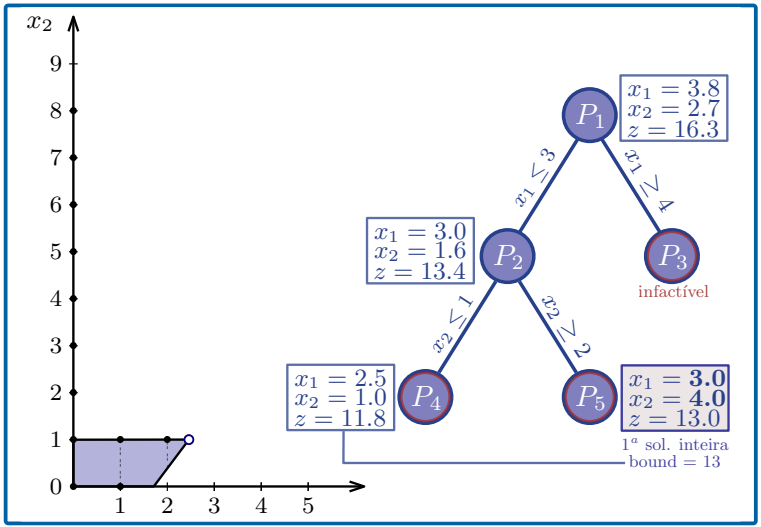
$$z_{P5} \leq z^*$$

Sabemos então que **nenhuma solução inteira que possa existir ramificando P_4 será melhor do que x^*.**



O algoritmo *Branch and Bound*

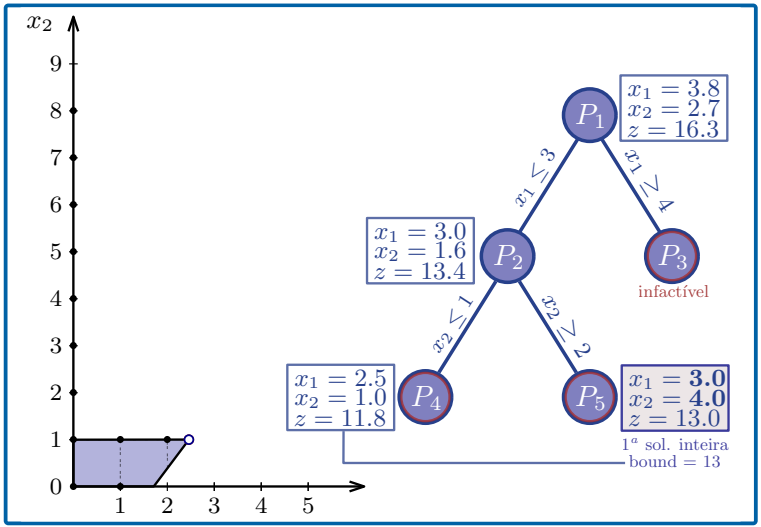
Portanto, podemos eliminar o nó P_4 de ramificação, pelo terceiro motivo: eliminação por bound.



O algoritmo *Branch and Bound*

Como não temos mais nós a serem explorados (não existem problemas na lista L), **o algoritmo chega a um fim**, e a nossa solução ótima é a melhor solução inteira que encontramos:

$$\begin{cases} x^* = (3, 4) \\ z^* = 13 \end{cases}$$



O algoritmo *Branch and Bound*

O algoritmo BB funciona com a criação de muitos subproblemas (**Branching**) a partir do problema inicial, que são resolvidos pelo método Simplex (ou dual-simplex). Nem todos os subproblemas precisam ser resolvidos, devido aos 3 critérios de eliminação de nós (**Bound**).

Vamos formalizar o algoritmo de forma sistemática.

O algoritmo *Branch and Bound*

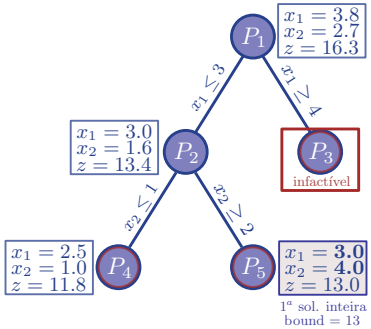
Considere a seguinte notação:

- P: PI original.
- $F(i)$: região factível do problema P_i.
- z_i: valor ótimo do problema P_i.
- x_i: valor ótimo do problema P_i.
- x^*: melhor solução inteira.
- z^*: valor objetivo da melhor solução inteira x^*.
- L: lista de nós a serem explorados (árvore BB).

O algoritmo *Branch and Bound*

Como vimos no exemplo, o algoritmo não resolve todos os subproblemas gerados, contando com um método de eliminação. Um problema P_i é eliminado da árvore se satisfizer alguma das condições:

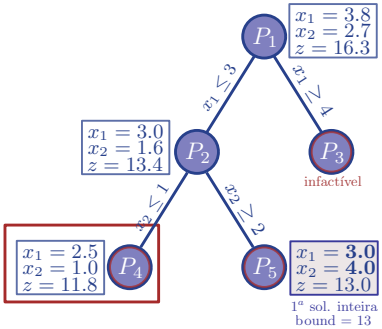
1. $F(i) = \emptyset$: o nó é eliminado por infactibilidade



O algoritmo *Branch and Bound*

Como vimos no exemplo, o algoritmo não resolve todos os subproblemas gerados, contando com um método de eliminação. Um problema P_i é eliminado da árvore se satisfizer alguma das condições:

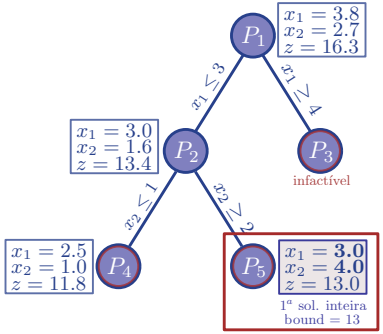
1. $F(i) = \emptyset$: o nó é eliminado por infactibilidade
2. $z_i \leq z^*$: o nó é eliminado por qualidade (propriedade da relaxação linear)



O algoritmo *Branch and Bound*

Como vimos no exemplo, o algoritmo não resolve todos os subproblemas gerados, contando com um método de eliminação. Um problema P_i é eliminado da árvore se satisfizer alguma das condições:

1. $F(i) = \emptyset$: o nó é eliminado por infactibilidade
2. $z_i \leq z^*$: o nó é eliminado por qualidade (propriedade da relaxação linear)
3. $x_i \in \mathbb{Z}$: a solução atual já é inteira.



O algoritmo *Branch and Bound*

O algoritmo completo fica então:

O algoritmo *Branch and Bound*

1. (Inicialização) $z^* = -\infty$, $x^* = \emptyset$, $L = \{P\}$
2. (Seleção) Se $L \neq \emptyset$, remova um nó $P_i \in L$, resolva a relaxação linear e vá para 3. Senão, vá para 7
3. (Teste 1) Se $F(i) = \emptyset$, volte para 2
4. (Teste 2) Se $z_i \leq z^*$, volte para 2
5. (Teste 3) Se $x_i \in \mathbb{Z}$
   5.1 Se $z_i \geq z^*$, $x^* = x_i$, $z^* = z_i$, volte para 2
6. (Ramificação) Selecione uma variável da solução ótima da relaxação com valor não inteiro e divida em dois subproblemas. Adicione-os a lista L e volte para 2.
7. (Fim) Se $z^* = -\infty$, pare, não existe solução. Caso contrário, x^* é a solução ótima.

Algumas questões de implementação

Questões de implementação

A etapa de seleção de um nó no algoritmo diz o seguinte:

> 2. (Seleção) Se $L \neq \emptyset$, remova um nó $P_i \in L$, resolva a relaxação linear e vá para 3. Senão, vá para 7

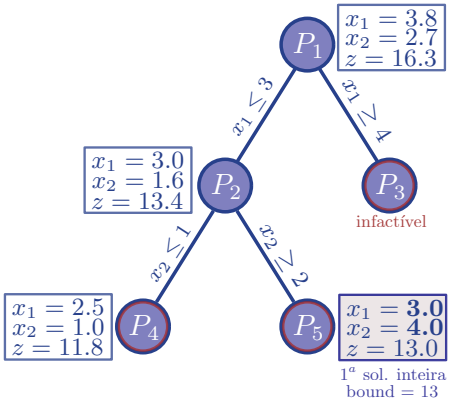
Ou seja, a cada iteração precisamos escolher um nó da árvore para ser visitado (verificar se é ramificado ou não.) **Como escolher o próximo nó**? Será que isso influencia no **número de iterações** que o algoritmo precisa para terminar?

Questões de implementação

No exemplo numérico exploramos os nós pela ordem:

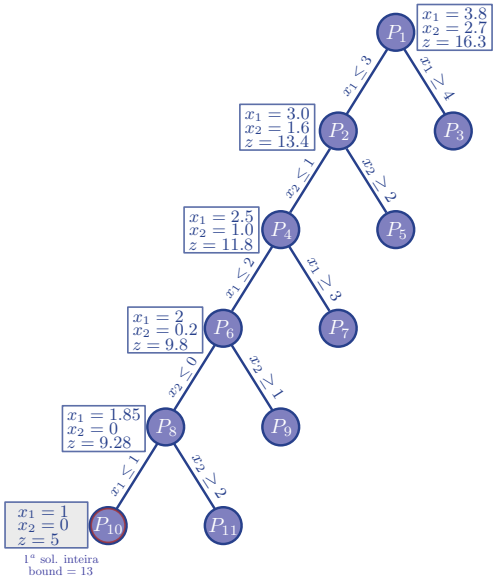
$$P_1 \to P_3 \to P_2 \to P_5 \to P_4$$

Note que eliminamos P_4 de uma possível ramificação, pois já havíamos encontrado um **Bound** inteiro em P_5. O que aconteceria se tivéssemos optado por expandir P_4 **antes** de P_5?



Questões de implementação

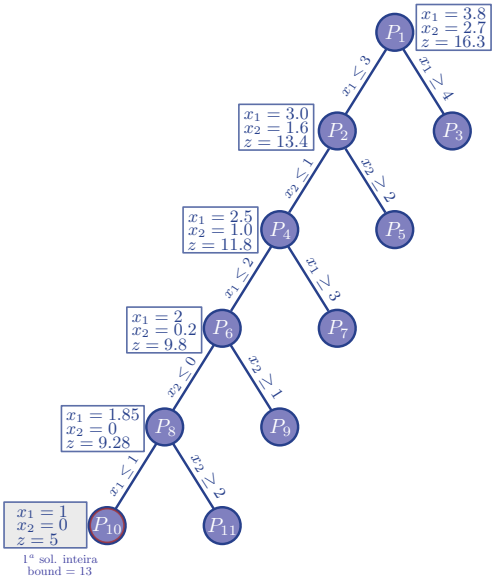
A árvore ao lado foi expandida na ordem: $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_6 \rightarrow P_8 \rightarrow P_{10}$. Ainda, existem 5 nós a serem explorados $(P_3, P_5, P_7, P_9, P_{11})$, **que poderiam gerar mais ramificações...**



Questões de implementação

Ou seja, **a ordem em que os nós são visitados importa e afeta o desempenho do algoritmo!**

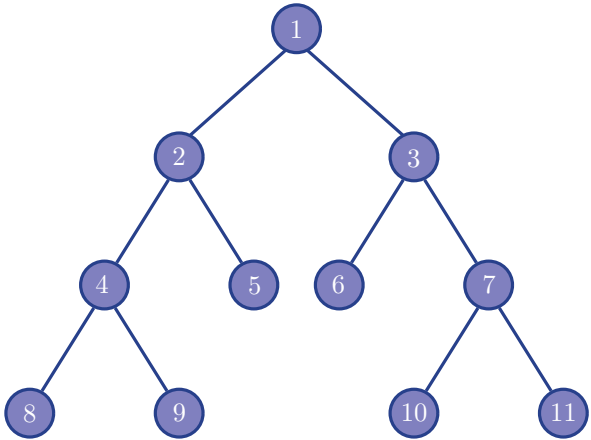
Veremos duas formas clássicas de busca em árvores na computação: expansão por **largura** (*Breadth-First Search*) e por **profundidade** (*Depth-First Search*).



Questões de implementação

Busca em largura

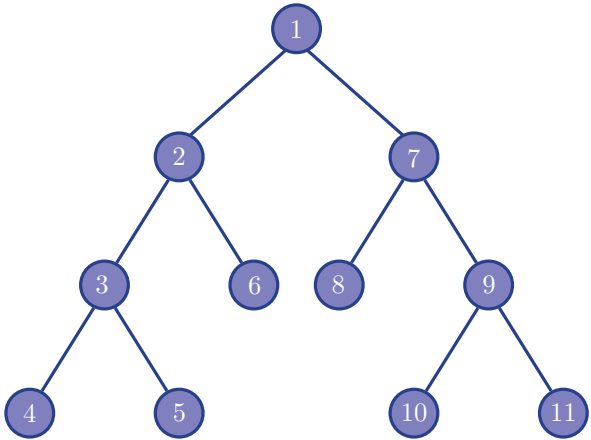
A busca em largura visita os nós de uma árvore **por níveis**, ou seja, primeiro ela visita todos os nós do nível 1, depois do nível 2, e assim sucessivamente. A árvore ao lado seria expandida na ordem dos números em cada nó.



Questões de implementação

Busca em profundidade

Ja a busca em profundidade escolhe um nó e **desce o máximo possível em seus níveis**. Em seguida escolhe o nó de nível mais próximo do último expandido e novamente expande até o fim.

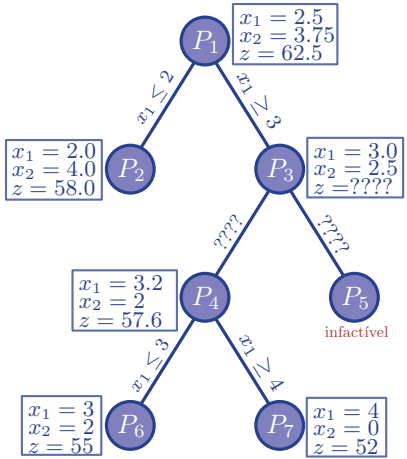


Exercícios

Exercícios

1 - Considerando o seguinte modelo de PI e a sua árvore BB (incompleta), responda o que se pede:

$$
\begin{aligned}
\max z = \; & 13x_1 - 8x_2 \\
& x_1 - 2x_2 \;\; \leq 10 \\
& 5x_1 + 2x_2 \;\; \leq 20 \\
& x \in \mathbb{Z}^2_+
\end{aligned}
$$



Exercícios

1.1 - Quais são as restrições faltantes entre $P_3 - P_4$ e $P_3 - P_5$?

1.2 - Represente a região factível de P_1, P_2, P_3, P_4, P_5

1.3 - Determine limitantes para a função objetivo do problema P_3.

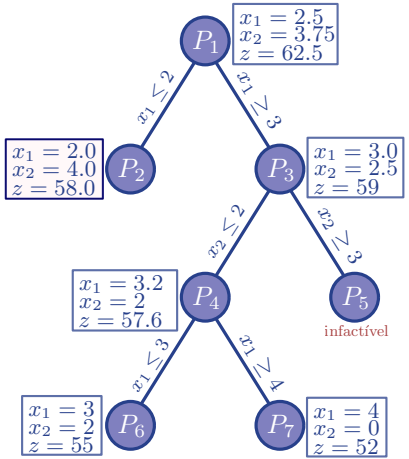
1.4 - Qual é o PL que está sendo resolvido em P_5? Escreva o modelo com todas as restrições.

1.4 - Qual é a solução ótima do problema inteiro?

Exercícios

2 - Considerando o seguinte modelo de PI e a sua árvore BB, responda o que se pede:

$$
\begin{aligned}
\max z = \; & 13x_1 - 8x_2 \\
& x_1 - 2x_2 \leq 10 \\
& 5x_1 + 2x_2 \leq 20 \\
& x \in \mathbb{Z}^2_+
\end{aligned}
$$



Exercícios

2.1 - Considerando que você já sabe o estado final da árvore. Qual seria a melhor ordem para exploração dos nós? Por quê? E a pior?

3 - Considere o modelo abaixo:

$$\begin{aligned} \max z = \; & 3x_1 + 2x_2 \\ & 2x_1 + 5x_2 \leq 9 \\ & 4x_1 + 2x_2 \leq 9 \\ & x \in \mathbb{Z}^2_+ \end{aligned}$$

3.1 - Represente a região factível do problema.

3.2 - Resolva o problema usando o algoritmo *Branch and Bound*. Resolva a relaxação linear inicial (P_1), e pelo menos uma das primeiras duas ramificações usando cálculos aprendidos em sala de aula (quais algoritmos utilizar?). As outras ramificações podem ser resolvidas usando o **GUSEK**. (Excel e modelos aqui).