Programação linear inteira

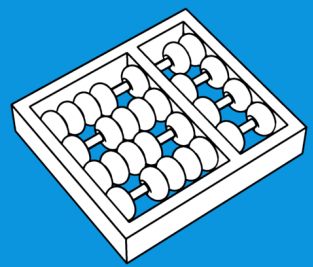
MC558 - Projeto e Análise de Algoritmos II

11/24

23

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

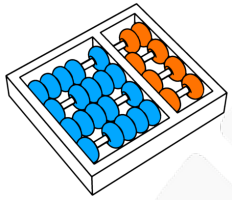
UNICAMP



"A programação linear foi e é talvez o problema do mundo real mais importante."

Keith Devlin.

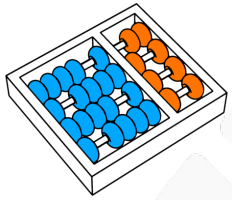
Formulando com PLI



Caminho mínimo

São dados um digrafo $G = (V, E)$ com pesos positivos associados às arestas e dois vértices: um fonte s e um terminal t.

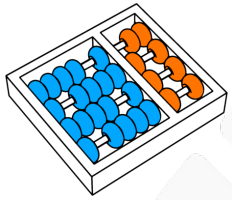
O problema procura por um caminho de s a t que minimize a soma dos pesos das arestas.



Caminho mínimo

Um caminho pode ser visto como uma sequência de arestas, logo a solução pode ser composta pelo conjunto de arestas que pertencem a ela. Assim, por cada aresta, podemos definir uma variável binária que indica se ela está ou não na solução:

$$x_e = \begin{cases} 1 & \text{, se a aresta } e \text{ está na solução} \\ 0 & \text{, em caso contrário} \end{cases}$$



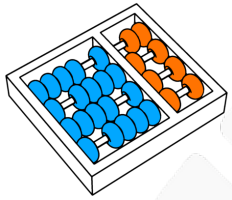
Caminho mínimo

Um caminho pode ser visto como uma sequência de arestas, logo a solução pode ser composta pelo conjunto de arestas que pertencem a ela. Assim, por cada aresta, podemos definir uma variável binária que indica se ela está ou não na solução:

$$\mathbf{x_e} = \begin{cases} 1 & \text{, se a aresta } e \text{ está na solução} \\ 0 & \text{, em caso contrário} \end{cases}$$

O objetivo é minimizar a soma dos pesos das arestas selecionadas:

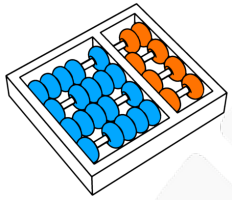
$$\min \quad \sum_{e \in E} \omega(e) \cdot \mathbf{x_a}$$



Caminho mínimo e fluxo

Com exceção de s e t, para todo vértice u do caminho, se uma aresta entra então outra deve sair:

$$\sum_{(u,v)\in E} \mathbf{x_{(u,v)}} - \sum_{(v,u)\in E} \mathbf{x_{(v,u)}} = 0$$



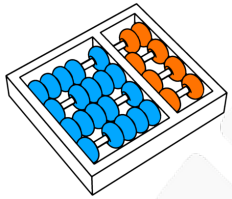
Caminho mínimo e fluxo

Com exceção de s e t, para todo vértice u do caminho, se uma aresta entra então outra deve sair:

$$\sum_{(u,v)\in E} \mathbf{x_{(u,v)}} - \sum_{(v,u)\in E} \mathbf{x_{(v,u)}} = 0$$

Para s, essa diferença deve ser -1, enquanto para t, 1:

$$\sum_{(s,v)\in E} \mathbf{x_{(s,v)}} - \sum_{(v,s)\in E} \mathbf{x_{(v,s)}} = 1 \qquad \sum_{(t,u)\in E} \mathbf{x_{(t,u)}} - \sum_{(u,t)\in E} \mathbf{x_{(u,t)}} = -1$$



Caminho mínimo e fluxo

Com exceção de s e t, para todo vértice u do caminho, se uma aresta entra então outra deve sair:

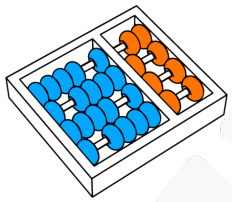
$$\sum_{(u,v)\in E} \mathbf{x_{(u,v)}} - \sum_{(v,u)\in E} \mathbf{x_{(v,u)}} = 0$$

Para s, essa diferença deve ser -1, enquanto para t, 1:

$$\sum_{(s,v)\in E} \mathbf{x_{(s,v)}} - \sum_{(v,s)\in E} \mathbf{x_{(v,s)}} = 1 \qquad \sum_{(t,u)\in E} \mathbf{x_{(t,u)}} - \sum_{(u,t)\in E} \mathbf{x_{(u,t)}} = -1$$

Ademais, por cada vértice d caminho, no máximo uma aresta pode sair (ou entrar):

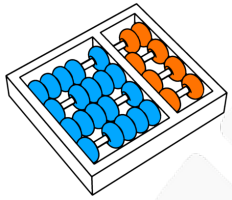
$$\sum_{(v,u)\in E} \mathbf{x_{(v,u)}} \leq 1$$



Caminho mínimo e fluxo

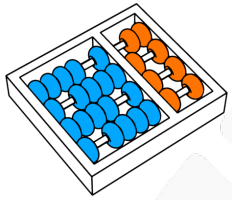
Instância: $G = (V, E)$, $\omega : E \to \mathbb{Q}_+$ e $s, d \in V$.

$$\begin{aligned}
\min \quad & \sum_{e \in E} \omega(e) \cdot \mathbf{x_e} \\
s.a. : \quad & \\
& \sum_{(u,v) \in E} \mathbf{x_{(u,v)}} - \sum_{(v,u) \in E} \mathbf{x_{(v,u)}} = 0 && \forall u \in V \setminus \{s, t\} \\
& \sum_{(s,v) \in E} \mathbf{x_{(s,v)}} - \sum_{(v,s) \in E} \mathbf{x_{(v,s)}} = 1 && \\
& \sum_{(t,v) \in E} \mathbf{x_{(t,v)}} - \sum_{(v,t) \in E} \mathbf{x_{(v,t)}} = -1 && \\
& \sum_{(v,u) \in E} \mathbf{x_{(v,u)}} \leq 1 && \forall u \in V \\
& \mathbf{x_e} \in \{0, 1\} && \forall e \in E
\end{aligned}$$



Caminhos mínimos e ciclos negativos

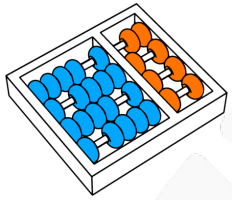
Uma estratégia para garantir que não hajam ciclos na solução, se conhece como **PREVENÇÃO DE SUB-ROTAS** e se baseia no fato de que, em todo caminho simples, o número arestas com dois extremos em qualquer conjunto de vértices S é no máximo $|S| - 1$.



Caminhos mínimos e ciclos negativos

Uma estratégia para garantir que não hajam ciclos na solução, se conhece como **PREVENÇÃO DE SUB-ROTAS** e se baseia no fato de que, em todo caminho simples, o número arestas com dois extremos em qualquer conjunto de vértices S é no máximo $|S| - 1$.

Se $E(S)$ denota o número de arestas com os dois extremos em S, podemos definir a seguinte família de restrições:

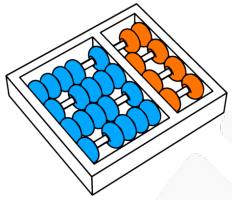


Caminhos mínimos e ciclos negativos

Uma estratégia para garantir que não hajam ciclos na solução, se conhece como **PREVENÇÃO DE SUB-ROTAS** e se baseia no fato de que, em todo caminho simples, o número arestas com dois extremos em qualquer conjunto de vértices S é no máximo $|S| - 1$.

Se $E(S)$ denota o número de arestas com os dois extremos em S, podemos definir a seguinte família de restrições:

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subseteq V \setminus \{s, d\} \text{ e } |S| \geq 2$$



Caminho mínimo em grafos com ciclos negativos

Instância: $G = (V, E)$, $\omega : E \to \mathbb{Q}_+$ e $s, d \in V$.

$$\min \quad \sum_{e \in E} \omega(e) \cdot \mathbf{x_e}$$

s.a. :

$$\sum_{(u,v) \in E} \mathbf{x_{(u,v)}} - \sum_{(v,u) \in E} \mathbf{x_{(v,u)}} = 0 \quad \forall u \in V \setminus \{s, t\}$$

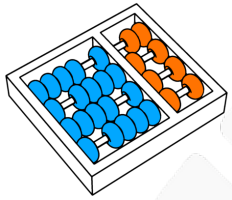
$$\sum_{(s,v) \in E} \mathbf{x_{(s,v)}} - \sum_{(v,s) \in E} \mathbf{x_{(v,s)}} = 1$$

$$\sum_{(t,v) \in E} \mathbf{x_{(t,v)}} - \sum_{(v,t) \in E} \mathbf{x_{(v,t)}} = -1$$

$$\sum_{(v,u) \in E} \mathbf{x_{(v,u)}} \leq 1 \quad \forall u \in V$$

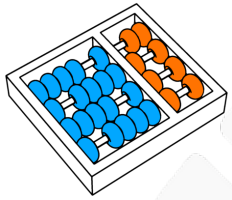
$$\sum_{e \in E(S)} \mathbf{x_e} \leq |S| - 1 \quad \forall S \subseteq V \setminus \{s, d\}, |S| \geq 2$$

$$\mathbf{x_e} \in \{0, 1\} \quad \forall e \in E$$



Caminho máximo

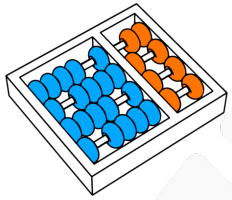
São dados um digrafo $G = (V, E)$ com pesos positivos associados às arestas e dois vértices: um fonte s e um terminal t.



Caminho máximo

São dados um digrafo $G = (V, E)$ com pesos positivos associados às arestas e dois vértices: um fonte s e um terminal t.

O problema procura por um caminho de s a t que maximize a soma dos pesos das arestas.



Caminho máximo

Instância: $G = (V, E)$, $\omega : E \to \mathbb{Q}_+$ e $s, d \in V$.

$$\max \quad \sum_{e \in E} \omega(e) \cdot \mathbf{x_e}$$

$s.a.:$

$$\sum_{(u,v) \in E} \mathbf{x_{(u,v)}} - \sum_{(v,u) \in E} \mathbf{x_{(v,u)}} = 0 \quad \forall u \in V \setminus \{s, t\}$$

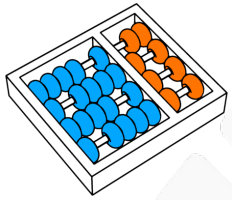
$$\sum_{(s,v) \in E} \mathbf{x_{(s,v)}} - \sum_{(v,s) \in E} \mathbf{x_{(v,s)}} = 1$$

$$\sum_{(t,v) \in E} \mathbf{x_{(t,v)}} - \sum_{(v,t) \in E} \mathbf{x_{(v,t)}} = -1$$

$$\sum_{(v,u) \in E} \mathbf{x_{(v,u)}} \leq 1 \quad \forall u \in V$$

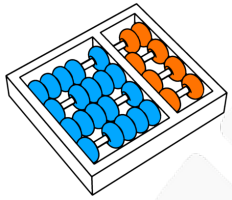
$$\sum_{e \in E(S)} \mathbf{x_e} \leq |S| - 1 \quad \forall S \subseteq V \setminus \{s, d\}, |S| \geq 2$$

$$\mathbf{x_e} \in \{0, 1\} \quad \forall e \in E$$



Árvore geradora mínima

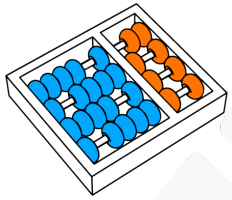
É dado um grafo $G = (V, E)$ com pesos positivos associados às arestas.



Árvore geradora mínima

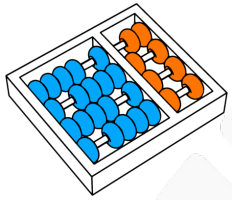
É dado um grafo $G = (V, E)$ com pesos positivos associados às arestas.

O problema procura por uma árvore geradora de G que minimize a soma dos pesos das arestas.



Árvore geradora mínima

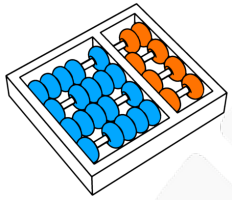
Uma árvore pode ser vista como um conjunto de arestas. Logo, podemos definir uma variável binárias por cada aresta que indica se ela está ou não na árvore:



Árvore geradora mínima

Uma árvore pode ser vista como um conjunto de arestas. Logo, podemos definir uma variável binárias por cada aresta que indica se ela está ou não na árvore:

$$x_e = \begin{cases} 1 & \text{, se a aresta } e \text{ forma parte da solução} \\ 0 & \text{, em caso contrário} \end{cases}$$

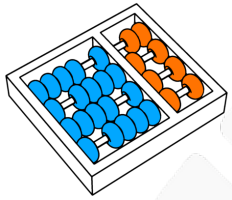


Árvore geradora mínima

Uma árvore pode ser vista como um conjunto de arestas. Logo, podemos definir uma variável binárias por cada aresta que indica se ela está ou não na árvore:

$$x_e = \begin{cases} 1 & \text{, se a aresta } e \text{ forma parte da solução} \\ 0 & \text{, em caso contrário} \end{cases}$$

O objetivo é minimizar a soma das arestas selecionadas:



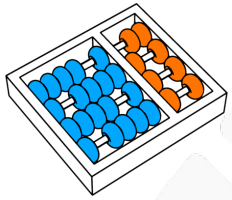
Árvore geradora mínima

Uma árvore pode ser vista como um conjunto de arestas. Logo, podemos definir uma variável binárias por cada aresta que indica se ela está ou não na árvore:

$$\mathbf{x_e} = \begin{cases} 1 & \text{, se a aresta } e \text{ forma parte da solução} \\ 0 & \text{, em caso contrário} \end{cases}$$

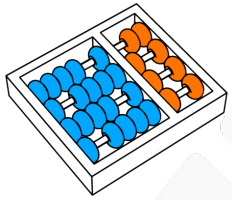
O objetivo é minimizar a soma das arestas selecionadas:

$$\min \quad \sum_{e \in E} \omega(e) \cdot \mathbf{x_e}$$



Árvore geradora mínima

Árvores geradoras devem conectar todos os vértices e não ter ciclos:

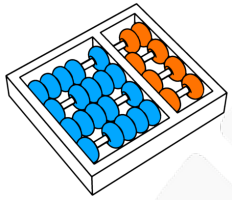


Árvore geradora mínima

Árvores geradoras devem conectar todos os vértices e não ter ciclos:

- A conectividade é garantida se existe pelo menos uma aresta no corte de cada subconjunto (próprio e não vazio) de vértices:

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall \emptyset \subset S \subset V$$



Árvore geradora mínima

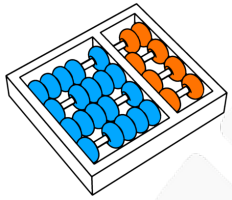
Árvores geradoras devem conectar todos os vértices e não ter ciclos:

- A conectividade é garantida se existe pelo menos uma aresta no corte de cada subconjunto (próprio e não vazio) de vértices:

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall \emptyset \subset S \subset V$$

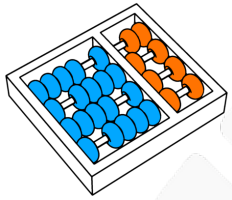
- A ausência de ciclos é garantida se para cada subconjunto de vértices $S \subseteq V$ não houver mais do que $|S| - 1$ arestas com as duas pontas em S ($|E(S)| \leq |S| - 1$):

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subseteq V, |S| \geq 2$$



Árvore geradora mínima

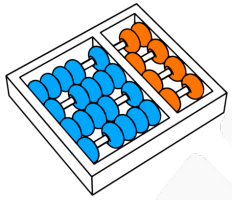
Definições equivalentes de árvores são:



Árvore geradora mínima

Definições equivalentes de árvores são:

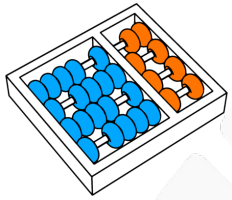
- Grafo conexo com $|V| - 1$ arestas.



Árvore geradora mínima

Definições equivalentes de árvores são:

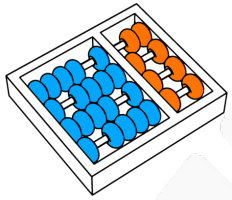
- Grafo conexo com $|V| - 1$ arestas.
- Grafo acíclico com $|V| - 1$ arestas.



Árvore geradora mínima

Definições equivalentes de árvores são:

- Grafo conexo com $|V| - 1$ arestas.
- Grafo acíclico com $|V| - 1$ arestas.



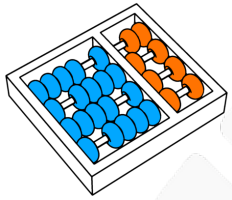
Árvore geradora mínima

Definições equivalentes de árvores são:

- Grafo conexo com $|V| - 1$ arestas.
- Grafo acíclico com $|V| - 1$ arestas.

Logo, podemos substituir restrições de conexidade ou acíclicas por:

$$\sum_{e \in E} x_e = |V| - 1$$



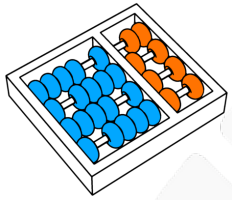
Árvore geradora mínima spanning tree

Instância: $G = (V, E)$, $\omega : E \to \mathbb{Q}$.

$$
\begin{array}{lll}
\min & \sum_{e \in E} \omega(e) \cdot x_e & \\
s.a.: & & \\
& \sum_{e \in \delta(S)} x_e \geq 1 & \forall \emptyset \subset S \subset V \\
& \sum_{e \in E} x_e = |V| - 1 & \\
& x_v \in \{0, 1\} & \forall v \in V
\end{array}
$$

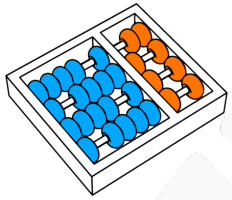
$$
\begin{array}{lll}
\min & \sum_{e \in E} \omega(e) \cdot x_e & \\
s.a.: & & \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 & \forall S \subseteq V, |S| \geq 2 \\
& \sum_{e \in E} x_e = |V| - 1 & \\
& x_v \in \{0, 1\} & \forall v \in V
\end{array}
$$

Sobre formulações



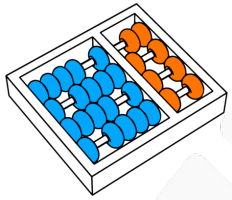
Passos para formular

1. Selecionar variáveis de decisão.



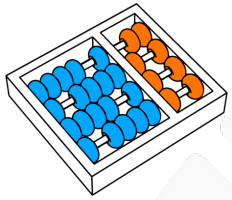
Passos para formular

1. Selecionar variáveis de decisão.
2. Escrever a função objetivo.



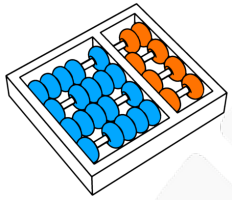
Passos para formular

1. Selecionar variáveis de decisão.
2. Escrever a função objetivo.
3. Escrever as restrições.



Passos para formular

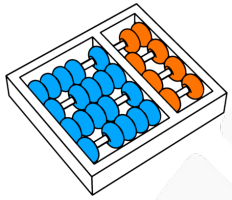
1. Selecionar variáveis de decisão.
2. Escrever a função objetivo.
3. Escrever as restrições.



Passos para formular

1. Selecionar variáveis de decisão.
2. Escrever a função objetivo.
3. Escrever as restrições.

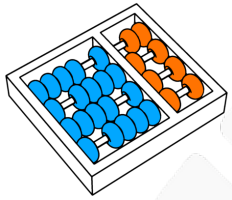
Observação 1. Se for mais fácil escrever as restrições com funções não-lineares ou com expressões lógicas, então faça-o. Depois, pode tentar linearizar.



Passos para formular

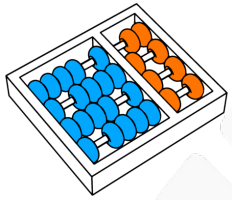
1. Selecionar variáveis de decisão.
2. Escrever a função objetivo.
3. Escrever as restrições.

Observação 1. Se for mais fácil escrever as restrições com funções não-lineares ou com expressões lógicas, então faça-o. Depois, pode tentar linearizar.
Observação 2. Se perceber que precisa mais variáveis, então adicione-as na formulação.



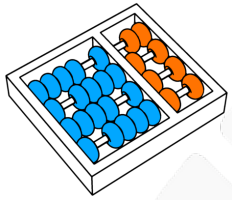
Comentários

- Usualmente há muitas formas de modelar um problema.



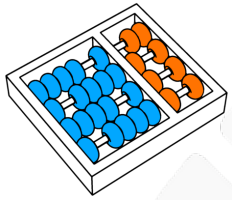
Comentários

- Usualmente há muitas formas de modelar um problema.
- As estrategias atuais para solucionar programas lineares inteiros são extremamente sensíveis à formulação:



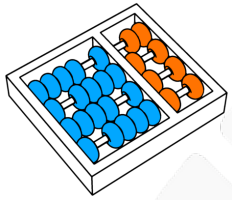
Comentários

- Usualmente há muitas formas de modelar um problema.
- As estrategias atuais para solucionar programas lineares inteiros são extremamente sensíveis à formulação:
  - Alguns programas lineares inteiros são fáceis de resolver, mesmo quando o número de variáveis e restrições estão na ordem dos milhões.



Comentários

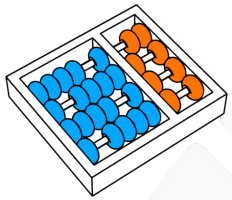
- Usualmente há muitas formas de modelar um problema.
- As estrategias atuais para solucionar programas lineares inteiros são extremamente sensíveis à formulação:
  - Alguns programas lineares inteiros são fáceis de resolver, mesmo quando o número de variáveis e restrições estão na ordem dos milhões.
  - Outros programas lineares inteiros são muito difíceis de solucionar, sendo desafiadores mesmo com centenas de variáveis.



Comentários

- Usualmente há muitas formas de modelar um problema.
- As estrategias atuais para solucionar programas lineares inteiros são extremamente sensíveis à formulação:
  - Alguns programas lineares inteiros são fáceis de resolver, mesmo quando o número de variáveis e restrições estão na ordem dos milhões.
  - Outros programas lineares inteiros são muito difíceis de solucionar, sendo desafiadores mesmo com centenas de variáveis.
  - Requer habilidade para identificar qual é qual e, geralmente, é uma tarefa difícil.

Uso de resolvedores em Python

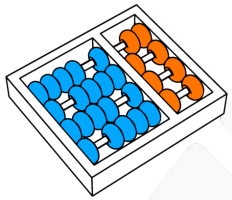


Programas comerciais

IBM CPLEX

FICO™

GUROBI OPTIMIZATION



Programas livres

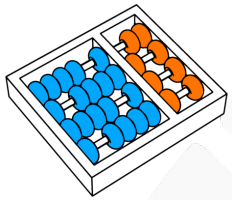
Home / Browse / Development / Algorithms / lpsolve

lpsolve

Mixed Integer Linear Programming (MILP) solver

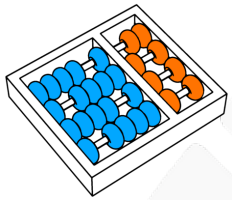
Brought to you by: keikland, peno64

GLPK (GNU Linear Programming Kit)



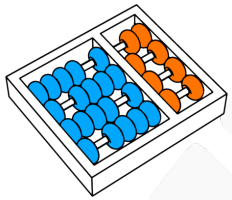
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.



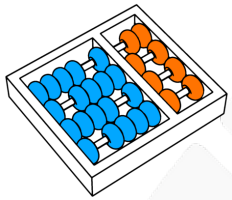
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.
- Popularidade e constante crescimento na comunidade:



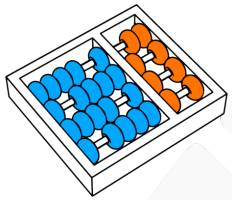
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.
- Popularidade e constante crescimento na comunidade:
  - 1^{ra} no índice TIOBE.



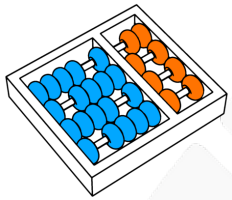
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.
- Popularidade e constante crescimento na comunidade:
  - 1^{ra} no índice TIOBE.
  - 2^{da} no GitHub.



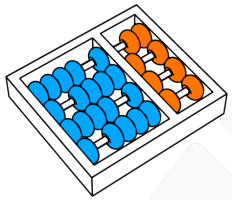
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.
- Popularidade e constante crescimento na comunidade:
  - 1^{ra} no índice TIOBE.
  - 2^{da} no GitHub.
- Interface comum para diferentes resolvedores de programação linear inteira e mista(e.g., **PuLP/DipPy**):



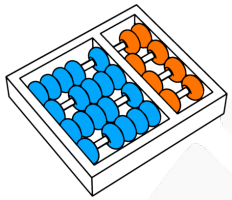
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.
- Popularidade e constante crescimento na comunidade:
  - 1^{ra} no índice TIOBE.
  - 2^{da} no GitHub.
- Interface comum para diferentes resolvedores de programação linear inteira e mista(e.g., **PuLP/DipPy**):
  - Simplifica a codificação de um modelo.



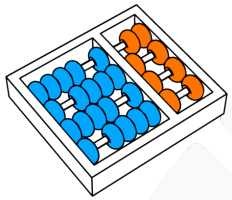
Por que Python?

- Linguagem de programação muito intuitiva e fácil para programar.
- Popularidade e constante crescimento na comunidade:
  - 1^{ra} no índice TIOBE.
  - 2^{da} no GitHub.
- Interface comum para diferentes resolvedores de programação linear inteira e mista(e.g., **PuLP/DipPy**):
  - Simplifica a codificação de um modelo.
  - Facilita chamados aos resolvedores durante a execução de algoritmos ou aplicações.



Pulp. Descrição

PuLP é um modelador de programas lineares escrito em Python e permite integrar diferentes resolvedores.



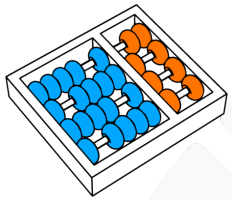
Pulp. Descrição

PuLP é um modelador de programas lineares escrito em Python e permite integrar diferentes resolvedores.

As principais classes para codificar formulações são **LpProblem**, **LpVariable**, **LpConstraint** e **LpConstraintVar**.

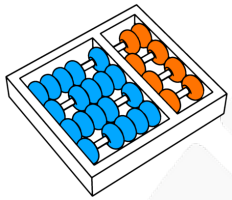
As interfaces para os resolvedores são dadas pelas classes **LpSolver** e **LpSolver_CMD**.

Acesse a documentação aqui



Codificando formulações

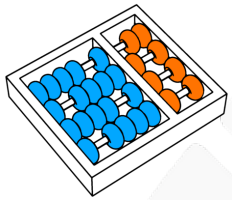
Importando **PuLP** com suas classes e funções: `from pulp import *`



Codificando formulações

Importando **PuLP** com suas classes e funções: `from pulp import *`

Definindo as variáveis:

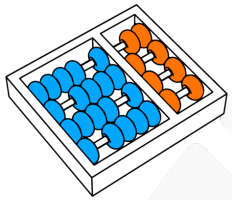


Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)

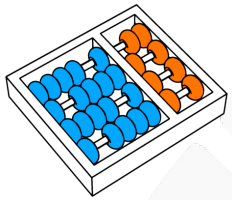


Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)



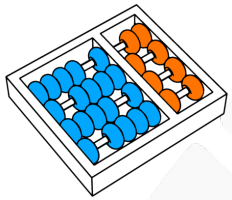
Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

Definindo o problema:



Codificando formulações

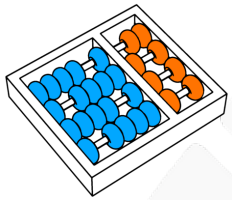
Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)



Codificando formulações

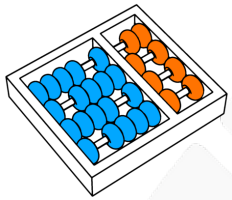
Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)
- meuProblema = LpProblem(nome, LpMaximize)



Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

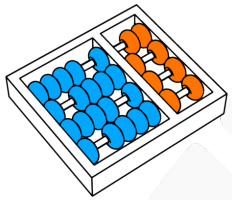
Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)
- meuProblema = LpProblem(nome, LpMaximize)

Definindo a função objetivo: meuProblema += expressão, nome



Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

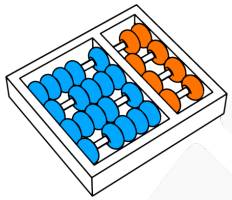
- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)
- meuProblema = LpProblem(nome, LpMaximize)

Definindo a função objetivo: meuProblema += expressão, nome

Definindo as restrições:



Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

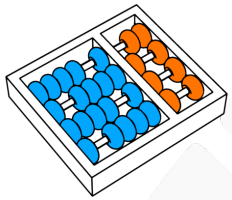
Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)
- meuProblema = LpProblem(nome, LpMaximize)

Definindo a função objetivo: meuProblema += expressão, nome

Definindo as restrições:

- meuProblema += expressão <= valor



Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

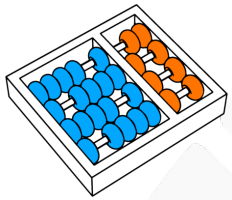
Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)
- meuProblema = LpProblem(nome, LpMaximize)

Definindo a função objetivo: meuProblema += expressão, nome

Definindo as restrições:

- meuProblema += expressão <= valor
- meuProblema += expressão == valor



Codificando formulações

Importando **PuLP** com suas classes e funções: from pulp import *

Definindo as variáveis:

- minhaVariavel = LpVariable(nome, lowBound = None, upBound = None, cat = 'Continuous', e = None)
- minhaVariavel = LpVariable.dicts(nome, índices, lowBound = None, upBound = None, cat = 0)

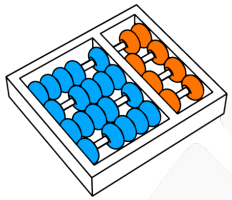
Definindo o problema:

- meuProblema = LpProblem(nome, LpMinimize)
- meuProblema = LpProblem(nome, LpMaximize)

Definindo a função objetivo: meuProblema += expressão, nome

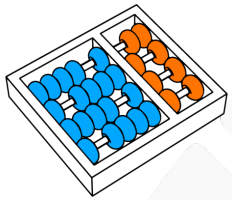
Definindo as restrições:

- meuProblema += expressão <= valor
- meuProblema += expressão == valor
- meuProblema += expressão >= valor



Um exemplo simples

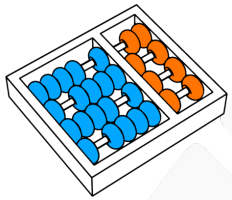
Considere o seguinte problema:



Um exemplo simples

Considere o seguinte problema:

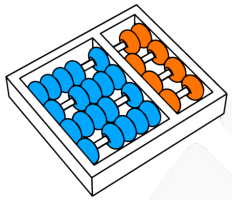
$$\max \quad 3\mathbf{x} + 2\mathbf{y}$$



Um exemplo simples

Considere o seguinte problema:

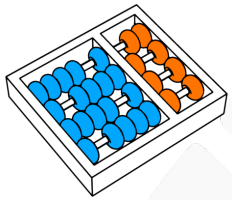
$$\begin{aligned} \max \quad & 3\mathbf{x} + 2\mathbf{y} \\ s.a: \quad & \end{aligned}$$



Um exemplo simples

Considere o seguinte problema:

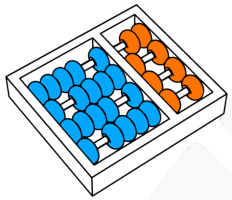
$$\begin{aligned} \max \quad & 3\mathbf{x} + 2\mathbf{y} \\ s.a: \quad & \\ & \mathbf{x} - \mathbf{y} + \mathbf{z} = 1 \end{aligned}$$



Um exemplo simples

Considere o seguinte problema:

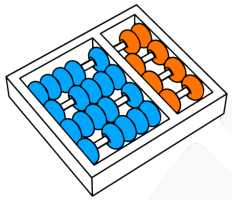
$$
\begin{aligned}
\max \quad & 3\mathbf{x} + 2\mathbf{y} \\
s.a: \quad & \\
& \mathbf{x} - \mathbf{y} + \mathbf{z} = 1 \\
& \mathbf{x} + 2\mathbf{y} \leq 14
\end{aligned}
$$



Um exemplo simples

Considere o seguinte problema:

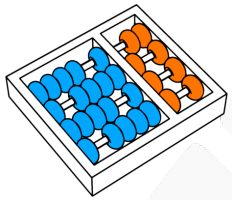
$$
\begin{aligned}
\max \quad & 3\mathbf{x} + 2\mathbf{y} \\
s.a: \quad & \\
& \mathbf{x} - \mathbf{y} + \mathbf{z} = 1 \\
& \mathbf{x} + 2\mathbf{y} \leq 14 \\
& 4\mathbf{x} + \mathbf{y} \leq 20
\end{aligned}
$$



Um exemplo simples

Considere o seguinte problema:

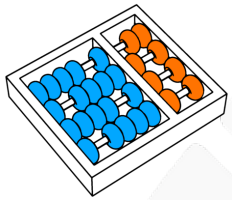
$$
\begin{aligned}
\max \quad & 3\mathbf{x} + 2\mathbf{y} \\
s.a: \quad & \\
& \mathbf{x} - \mathbf{y} + \mathbf{z} = 1 \\
& \mathbf{x} + 2\mathbf{y} \leq 14 \\
& 4\mathbf{x} + \mathbf{y} \leq 20 \\
& \mathbf{x}, \mathbf{y} \in \mathbb{Z}_+, \mathbf{z} \in \mathbb{Q}_+
\end{aligned}
$$



Um exemplo simples

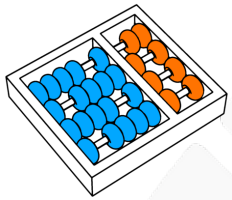
$$
\begin{aligned}
\max \quad & 3x + 2y \\
s.a: \quad & \\
& x - y + z = 1 \\
& x + 2y \leq 14 \\
& 4x + y \leq 20 \\
& x, y \in \mathbb{Z}_+, z \in \mathbb{Q}_+
\end{aligned}
$$

```
1  from pulp import *
2
3  x = LpVariable("x", lowBound = 0, cat = 'Integer')
4  y = LpVariable("y", lowBound = 0, cat = 'Integer')
5  z = LpVariable("z", lowBound = 0)
6
7  problema = LpProblem("ProblemaSimples",
   ↪  LpMaximize)
8
9  problema += 3 * x + 2 * y, "objetivo"
10
11 problema += x - y + z == 1
12 problema += x + 2 * y <= 14
13 problema += 4 * x + y <= 20
14
15
```



Exemplo. Mochila binária

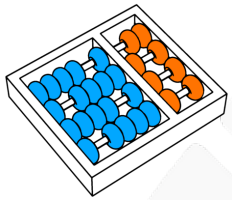
Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.



Exemplo. Mochila binária

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

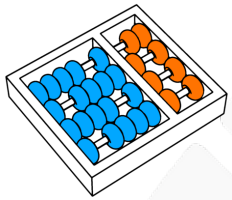


Exemplo. Mochila binária

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\max \quad \sum_{o \in O} \nu(o) \mathbf{x_o}$$



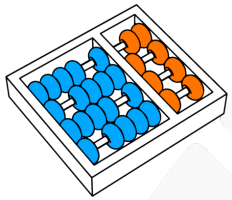
Exemplo. Mochila binária

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\max \quad \sum_{o \in O} \nu(o) \mathbf{x_o}$$

$$s.a :$$

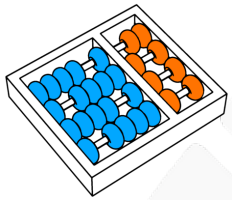


Exemplo. Mochila binária

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \nu(o) \mathbf{x_o} \\ s.a: \quad & \\ & \sum_{o \in O} \omega(o) \mathbf{x_o} \leq W \end{aligned}$$

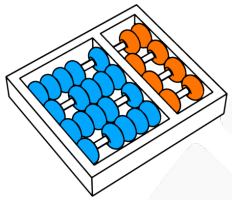


Exemplo. Mochila binária

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, peso máximo $W \in \mathbb{Q}_+$.

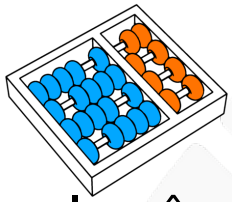
Formulação:

$$\begin{aligned} \max \quad & \sum_{o \in O} \nu(o) \mathbf{x_o} \\ s.a: \quad & \\ & \sum_{o \in O} \omega(o) \mathbf{x_o} \leq W \\ & \mathbf{x_o} \in \{0,1\}, \forall o \in O \end{aligned}$$



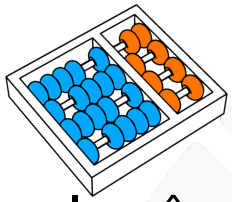
Exemplo. Mochila binária

```
1  from pulp import *
2
3  def mochila(lucros, pesos, W):
4      x = LpVariable.dicts("x", [o for o in range(len(lucros))], lowBound = 0, upBound = 1,
       ↪  cat='Integer')
5
6      problema = LpProblem("MochilaBinaria", LpMaximize)
7
8      problema += lpSum(lucros[o] * x[o] for o in range(len(lucros))), "lucro"
9
10     problema += lpSum(pesos[o] * x[o] for o in range(len(lucros))) <= W
11
12
```



Exemplo. Mochila múltipla

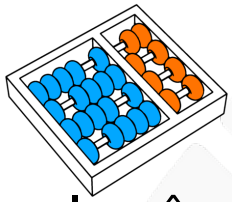
Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.



Exemplo. Mochila múltipla

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

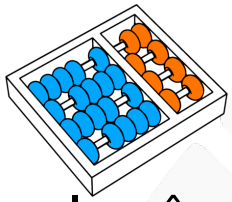


Exemplo. Mochila múltipla

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\max \quad \sum_{o \in O} \sum_{i=1}^{m} \nu(o) \mathbf{x_{oi}}$$



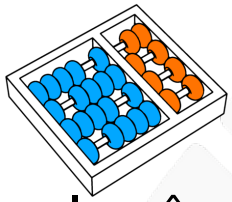
Exemplo. Mochila múltipla

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\max \quad \sum_{o \in O} \sum_{i=1}^{m} \nu(o) x_{oi}$$

$$s.a :$$



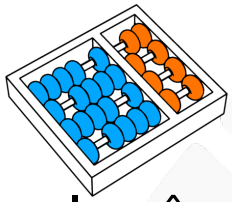
Exemplo. Mochila múltipla

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

$$\max \quad \sum_{o \in O} \sum_{i=1}^{m} \nu(o) \mathbf{x_{oi}}$$

$$s.a: \quad \sum_{o \in O} \omega(o) \mathbf{x_{oi}} \leq W_i \quad \forall 1 \leq i \leq m$$



Exemplo. Mochila múltipla

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

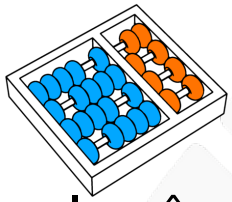
Formulação:

$$\max \quad \sum_{o \in O} \sum_{i=1}^{m} \nu(o) \mathbf{x_{oi}}$$

$$s.a:$$

$$\sum_{o \in O} \omega(o) \mathbf{x_{oi}} \leq W_i \quad \forall 1 \leq i \leq m$$

$$\sum_{i=1}^{m} \mathbf{x_{oi}} \leq 1 \quad \forall o \in O$$



Exemplo. Mochila múltipla

Instância: conjunto de objetos O, função de lucro $\nu : O \to \mathbb{Q}_+$, função de pesos $\omega : O \to \mathbb{Q}_+$, vetor de pesos máximos $W \in \mathbb{Q}_+^m$.

Formulação:

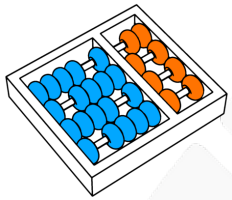
$$\max \quad \sum_{o \in O} \sum_{i=1}^{m} \nu(o) \mathbf{x_{oi}}$$

$$s.a:$$

$$\sum_{o \in O} \omega(o) \mathbf{x_{oi}} \leq W_i \quad \forall 1 \leq i \leq m$$

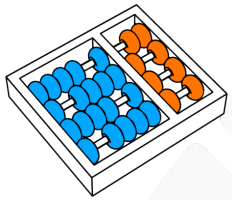
$$\sum_{i=1}^{m} \mathbf{x_{oi}} \leq 1 \quad \forall o \in O$$

$$\mathbf{x_{oi}} \in \{0, 1\}, \forall o \in O, 1 \leq i \leq m$$



Exemplo. Mochila múltipla

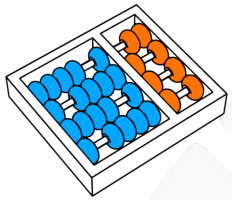
```
1  from pulp import *
2
3  def mochilaMultipla(lucros, pesos, W):
4      x = LpVariable.dicts("x", [(o, i) for o in range(len(lucros)) for i in range(len(W))],
       ↪  lowBound = 0, upBound = 1, cat='Integer')
5
6      problema = LpProblem("MochilaMultipla", LpMaximize)
7
8      problema += lpSum(lucros[o] * x[o, i] for o in range(len(lucros)) for i in
       ↪  range(len(W))), "lucro"
9
10     for i in range(len(W)):
11         problema += lpSum(pesos[o] * x[o, i] for o in range(len(lucros))) <= W[i]
12
13     for o in range(len(W)):
14         problema += lpSum(x[o, i] for i in range(len(W))) <= 1
15
16
```



Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

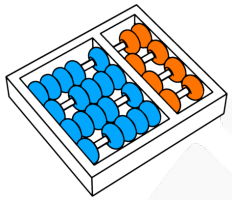


Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:



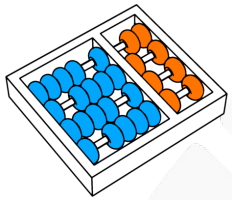
Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- e.g., solver = GUROBI(parameters).



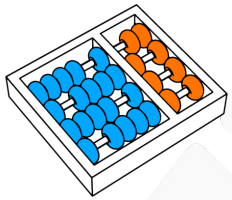
Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- e.g., solver = GUROBI(parameters).
- Alguns parâmetros podem ser incluídos: timeLimit in seconds, Cuts, Heuristics e Presolve para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e preprocessamento.



Solucionando formulações

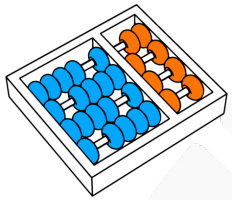
A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- e.g., solver = GUROBI(parameters).
- Alguns parâmetros podem ser incluídos: timeLimit in seconds, Cuts, Heuristics e Presolve para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e preprocessamento.

Solucionando o problema:



Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

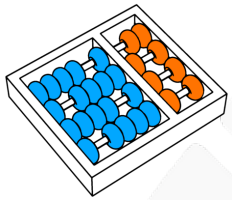
- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- e.g., solver = GUROBI(parameters).
- Alguns parâmetros podem ser incluídos: timeLimit in seconds, Cuts, Heuristics e Presolve para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e preprocessamento.

Solucionando o problema:

- meuProblema.solve(solver)



Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

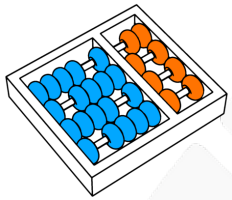
- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

- e.g., solver = GUROBI(parameters).
- Alguns parâmetros podem ser incluídos: timeLimit in seconds, Cuts, Heuristics e Presolve para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e preprocessamento.

Solucionando o problema:

- meuProblema.solve(solver)
- solver.buildSolverModel(meuProblema), solver.callSolver(meuProblema) e solver.findSolutionValues(meuProblema)



Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

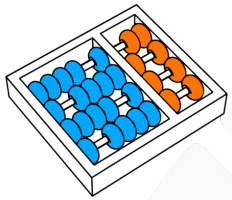
Definindo o resolvedor:

- e.g., solver = GUROBI(parameters).
- Alguns parâmetros podem ser incluídos: timeLimit in seconds, Cuts, Heuristics e Presolve para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e preprocessamento.

Solucionando o problema:

- meuProblema.solve(solver)
- solver.buildSolverModel(meuProblema), solver.callSolver(meuProblema) e solver.findSolutionValues(meuProblema)

Obtendo o valor da solução: value(meuProblema.objective).



Solucionando formulações

A função **list_solvers()** retorna os resolvedores disponíveis:

- e.g., ['GLPK_CMD', 'PYGLPK', 'CPLEX_CMD', 'CPLEX_PY', 'GUROBI', 'GUROBI_CMD', 'XPRESS', 'COIN_CMD', 'SCIP_CMD']

Definindo o resolvedor:

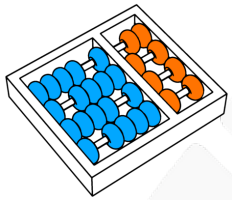
- e.g., solver = GUROBI(parameters).
- Alguns parâmetros podem ser incluídos: timeLimit in seconds, Cuts, Heuristics e Presolve para indicar, respectivamente, se serão usados: geração de cortes padrão, heurísticas do resolvedor e preprocessamento.

Solucionando o problema:

- meuProblema.solve(solver)
- solver.buildSolverModel(meuProblema), solver.callSolver(meuProblema) e solver.findSolutionValues(meuProblema)

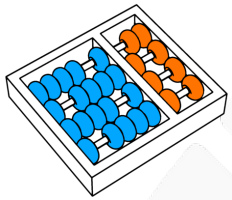
Obtendo o valor da solução: value(meuProblema.objective).

As variáveis são elementos de meuProblema.variables() cada um com os atributos name e varValue.



Um exemplo simples

```
1  from pulp import *
2
3  x = LpVariable("x", lowBound = 0, cat = 'Integer')
4  y = LpVariable("y", lowBound = 0, cat = 'Integer')
5  z = LpVariable("z", lowBound = 0)
6
7  problema = LpProblem("Problema", objetivo)
8
9  problema += 3 * x + 2 * y, "objetivo"
10
11 problema += x - y + z == 1
12 problema += x + 2 * y <= 14
13 problema += 4 * x + y <= 20
14
15 problema.solve(GUROBI_CMD())
16
17 print('Valor otimo: ' + str(value(problema.objective)))
18 print('Solucao otima: ')
19 for variavel in problema.variables():
20     print('     ' + variavel.name + " = " + str(variavel.varValue))
21
```



Exemplo. Mochila binária

```
1  from pulp import *
2
3  def mochila(lucros, pesos, W):
4      x = LpVariable.dicts("x", [o for o in range(len(lucros))], lowBound = 0, upBound = 1,
   ↪   cat='Integer')
5
6      problema = LpProblem("MochilaBinaria", LpMaximize)
7
8      problema += lpSum(lucros[o] * x[o] for o in range(len(lucros))), "lucro"
9
10     problema += lpSum(pesos[o] * x[o] for o in range(len(lucros))) <= W
11
12     solver = GUROBI(timeLimit = 3600)
13
14     solver.buildSolverModel(problema)
15     solver.callSolver(problema)
16     solver.findSolutionValues(problema)
17
18     print('Valor otimo: ' + str(value(problema.objective)))
19     print('Solucao otima: ')
20     for variavel in problema.variables():
21         print('      ' + variavel.name + " = " + str(variavel.varValue))
22
23
```

Programação linear inteira

MC558 - Projeto e Análise de Algoritmos II

11/24

23

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP