TÉCNICO LISBOA

Análise e Síntese de Algoritmos

Programação Linear (cont.).

Solvers de PL / Biblioteca PuLP.

Prof. Pedro T. Monteiro

IST - Universidade de Lisboa

2024/2025

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica [CLRS, Cap.15]
  - Algoritmos greedy [CLRS, Cap.16]
- Análise amortizada [CLRS, Cap.17]
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Árvores abrangentes [CLRS, Cap.23]
  - Fluxos máximos [CLRS, Cap.26]
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais
  - Complexidade Computacional [CLRS, Cap.34]

Resumo

Solvers de PL

Solvers de PL

- `Gurobi` `http://www.gurobi.com/` (Licença Comercial)
- `CPLEX` `https://www.ibm.com/analytics/cplex-optimizer` (Licença Comercial)
- `LP_SOLVE` `http://lpsolve.sourceforge.net/5.5/` (Licença LGPL)
- `GLPK` `https://www.gnu.org/software/glpk/` (Licença GPL)
  (usamos o `GLPK` em ASA)
- ...

Solver LP_Solve

LP_Solve

- Solver de LP e MIP
- Desenvolvido por Michel Berkelaar
- Próprio formato ".lp"
- Licença LGPL
- Primeira versão em 1995
- Versão 5.5 em 2012
- `http://lpsolve.sourceforge.net/5.5/`

Instalação Linux

- Arch: `pacman -S lpsolve`
- Debian: `apt install lp-solve`

Solver LP_Solve

Exemplo de codificação

```
$ cat exemplo.lp
max: 3 x1 + x2 + 2 x3;
  x1 +   x2 + 3 x3 <= 30;
2 x1 + 2 x2 + 5 x3 <= 24;
4 x1 +   x2 + 2 x3 <= 36;
x1 >= 0;
x2 >= 0;
x3 >= 0;
```

```
$ lp_solve -lp exemplo.lp
Value of objective function: 28.00000000

Actual values of the variables:
x1                              8
x2                              4
x3                              0
```


Solver GLPK

GLPK (GNU Linear Programming Kit)

- Desenvolvido por Andrew Makhorin
- Solver de PL escrito em C
- Formato CPLEX ".lp"
- Licença GPL
- Primeira versão em 2000
- Versão 5.0 em 2020
- `https://www.gnu.org/software/glpk/`

Instalação Linux

- Arch: `pacman -S glpk`
- Debian: `apt install glpk-utils`

(utilizado no Mooshak)

Solver GLPK

Exemplo de codificação

```
$ cat exemplo.cplex.lp
Maximize
 Objective: 3 x1 + x2 + 2 x3
Subject To
 Constraint1:   x1 +   x2 + 3 x3 <= 30
 Constraint2: 2 x1 + 2 x2 + 5 x3 <= 24
 Constraint3: 4 x1 +   x2 + 3 x3 <= 36
Bounds
 x1 >= 0
 x2 >= 0
 x3 >= 0
End
```

```
$ glpsol --lp exemplo.cplex.lp -w sol.txt
$ cat sol.txt
c Problem:
c Rows:       3
c Columns:    3
c Non-zeros:  9
c Status:     OPTIMAL
c Objective:  Objective = 28 (MAXimum)
c
s bas 3 3 f f 28
i 1 b 12 0
i 2 u 24 0.166666666666667
i 3 u 36 0.666666666666667
j 1 b 8 0
j 2 b 4 0
j 3 l 0 -0.833333333333333
e o f
```

Biblioteca PuLP

PuLP (Python Linear Programming)

- PuLP é uma biblioteca de Python para modelação de problemas de PL
- Permite definir um problema PL programaticamente
- Wrapper para differentes solvers de PL
- `https://coin-or.github.io/pulp/`

Instalação

- `sudo pip install pulp`
- `sudo pulptest`

`https://coin-or.github.io/pulp/main/installing_pulp_at_home.html`

Biblioteca PuLP - Problema

Problema

- A função `LpProblem` cria um problema de PL
- Recebe como argumentos:
  - O nome do problema
  - O tipo do problema: `LpMaximize` ou `LpMinimize`
- Exemplo:
  - `prob = LpProblem("Exemplo", LpMaximize)`

Biblioteca PuLP - Variáveis

Variáveis

- A função `LpVariable` cria uma variável de um problema de PL
- Recebe como argumentos:
  - O nome da variável
  - O valor mínimo da variável
  - O valor máximo da variável (default: infinito)
  - O tipo da variável: `LpContinuous` (default), `LpInteger` ou `LpBinary`
- Exemplo:
  - `x1 = LpVariable("x1", 0, 1, LpInteger)` $x_1 \in \{0, 1\}$
  - `x2 = LpVariable("x2", 0, cat='Continuous')` $x_2 \in [0, +\infty[$

Biblioteca PuLP - Função objectivo

Expressões lineares

- As expressões lineares são da forma $\sum_{i=1}^{n} c_i x_i$
- A função `lpSum` cria uma expressão linear
  - Recebe como argumento uma lista da forma `[`c_1 `*` x_1`, `c_2 `*` x_2`, ..., `c_n `*` x_n`]`
- Exemplo:
  - `lpSum([2 * x1, -x2, -4 * x5])`
  - `2 * x1 - 2 * x2 -4 * x5`

Função objectivo

- O método += adiciona uma expressão linear à função objectivo
- Exemplo:
  - `prob += lpSum([2 * x1, -x2, -4 * x5])`
  - `prob += 2 * x1 - 2 * x2 -4 * x5`

Biblioteca PuLP - Restrições

Restrições

- As restrições são da forma $\sum_{i=1}^{n} c_i x_i \, (\leq, =, \geq) \, b$
- Podemos acrescentar restrições ao problema usando o método += do problema
- Podemos usar a função `lpSum` para criar expressões lineares
- Exemplo:
  - `prob += lpSum([-3 * x1, 1 * x2]) <= 1`
  - `prob += -3 * x1 + x2 = -4`

Biblioteca PuLP - Resolução

Resolução

- A função `solve` resolve o problema
- Recebe como argumento o solver a usar (que não o default)
- Exemplo:
  - `prob.solve()`
  - `prob.solve(GLPK())` (ex: Solver GLPK)
  - `prob.solve(GLPK(msg=0))` (ex: Solver GLPK sem output)

Biblioteca PuLP - Estado da solução

Estado da solução

- O atributo `status` do problema indica o estado da solução:
  - `Not Solved`: estado inicial antes da resolução
  - `Optimal`: solução óptima encontrada
  - `Infeasible`: sem solução viável
  - `Unbounded`: função objectivo não tem limite superior
  - `Undefined`: problema mal definido, talvez tenha solução
- Exemplo:
  - `print("Status:", LpStatus[prob.status])`

Biblioteca PuLP - Valor da função objectivo

Valor da função objectivo

- O atributo `objective` do problema indica o valor da função objectivo
- Exemplo:
  - `print("Valor da função objectivo:", value(prob.objective))`

Biblioteca PuLP - Valor das variáveis

Valor das variáveis

- O método `variables` do problema devolve o valor das variáveis
- Exemplo:
  - `for v in prob.variables(): print(v.name, "=", v.varValue)`
  - `for v in prob.variables(): print(v.name, "=", value(v))`

Biblioteca PuLP - Verificar codificação

Verificar codificação

- O método `writeLP` do problema escreve o problema num ficheiro
- Exemplo:
  - `prob.writeLP("problema.lp")`

Conselhos gerais (e para o projecto 3)

1. Usar o método `writeLP` para verificar a codificação dos exemplos
2. Restringir o espaço de pesquisa o máximo possível
   - começar com as restrições mais restritivas (lower e upper bounds das variáveis)
   - depois adicionar restrições intermédias
   - finalmente adicionar as restrições menos restritivas
3. Se derem nomes às restrições, a ordem das restrições passa a ser a dada pela ordem alfabética dos nomes (**Atenção:** Altera o 2º conselho !)

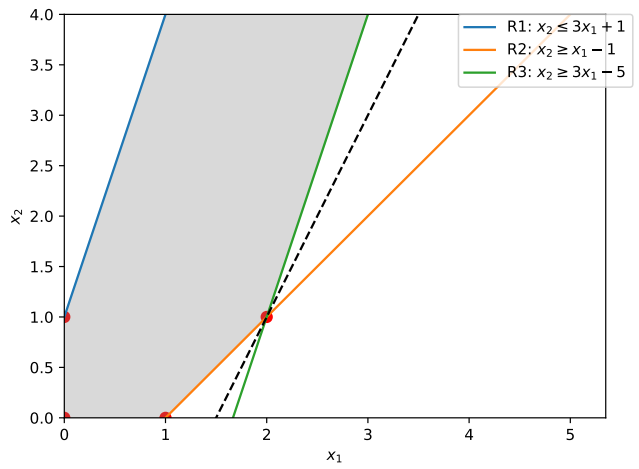
Exemplo

Prática 12 - Q4 (T2 20/21 II.b)
Considere o seguinte programa linear:

$$\begin{array}{lrrcl} \max & 2x_1 & -x_2 & & \\ \text{s.a} & -3x_1 & +x_2 & \leq & 1 \\ & x_1 & -x_2 & \leq & 1 \\ & 3x_1 & -x_2 & \leq & 5 \\ & & x_1, x_2 & \geq & 0 \end{array}$$

- Represente graficamente o problema
- Resolva o problema utilizando a biblioteca PuLP

Exemplo



Pontos

x_1	x_2	z
0	0	0
0	1	-1
1	0	2
2	1	3

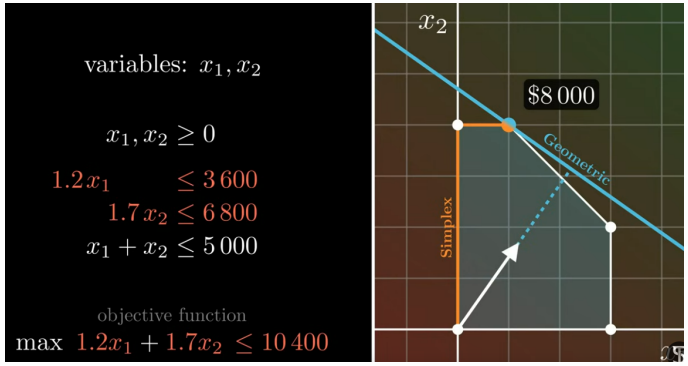
Valor óptimo
$z = 3$

Exemplo

Codificação PuLP

```
from pulp import *
# Problema
prob = LpProblem("Exemplo", LpMaximize)
# Variáveis
x1 = LpVariable("x1", 0)
x2 = LpVariable("x2", 0)
# Função objetivo
prob += 2*x1 - x2
# Restrições
prob += -3*x1 + x2 <= 1
prob += x1 - x2 <= 1
prob += 3*x1 - x2 <= 5
# Solução
status = prob.solve(GLPK(msg=0))
print(LpStatus[status])
print("Valor óptimo: ", value(prob.objective))
for v in prob.variables():
    print(v.name, "=", v.varValue)
```


Video recomendado



https://www.youtube.com/watch?v=E72DWgKP_1Y

Questões?

Dúvidas?