ENG1467 - OTIMIZAÇÃO - PUC-Rio, 2022.1

Lista 5: Simplex BenchMark

20/05/2022

Professor: Alexandre Street *Aluno: Thiago Novaes*

O código completo desenvolvido durante essa atividade pode ser encontrado em `https://github.com/Thiago-NovaesB/MestradoPuc.jl/tree/main/Programa%C3%A7%C3%A3o%20Linear/Lista%204/Simplex`

1 Motivação

Esse trabalho visa implementar algumas regras que são necessárias em casos degenerados. A primeira delas é a regra de Bland, ela impede que casos degenerados (os vértices não possuem bases únicas) entrem em ciclos intermináveis. Outra regra apresentada no Bertisimas seção 3.5, é o caso onde o resultado no simplex fase 1 possui variáveis artificiais na base.

Por fim, um benchmark do solver desenvolvido será apresentado.

2 Regra de Bland

Choose the lowest-numbered (i.e., leftmost) nonbasic column with a negative (reduced) cost.

```
B = view(A,:,base)
N = view(A,:,nbase)
xB = B \ b
y = B' \ c[base]
midterm.red_cost = c[nbase] - N'*y
val = maximum(midterm.red_cost)

if val <= tol
    midterm.termination_status = 1
    return midterm #optimal
end
midterm.j = findfirst(x->x>tol,midterm.red_cost)
```

Now among the rows, choose the one with the lowest ratio between the (transformed) right hand side and the coefficient in the pivot tableau where the coefficient is greater than zero. If the minimum ratio is shared by several rows, choose the row with the lowest-numbered column (variable) basic in it.

```
d_base = B \ N[:,midterm.j]
d_base = max.(d_base, 0)
r = max.(xB, tol) ./ d_base
val, midterm.i = findmin(r)
```

Sobre a implementação, como estamos usando os vetores *base* e *nbase* para realizar os pivotiamentos, eles ja guardam a correta ordem do tableau, restando apenas usar a função *findfirst* para achar a primeira ocorrência quando necessário.

3 Remoção de variáveis artificiais da base

Quando o ponto ótimo do Simplex fase 1 é um ponto degenerado, é possível que a variável artificial usada para inflar o poliedro, fique na base. Neste caso, é preciso ter cuidado ao voltar o problema para a forma original. Na seção 3.5 do Bertisimas ele explica a seguinte forma:

1. Identifique a coordenada l, onde a coluna artificial está na base.
2. Percorra as colunas fora da base. Para cada coluna A_j, resolva o sistema $Bx = A_j$, se $x_l = 0$, significa que A_j é combinação linear das colunas básicas não artificiais, logo A_j não pode entrar na base. Caso $x_l \neq 0$, A_j é linearmente independente das colunas não artificiais, neste caso, remova a coluna artificial do problema e passe A_j para a base.
3. Se nenhuma coluna não básica atender essa condição, então remova a linha do tableau.

```
cache = findfirst(x->x==input.n + 1,midterm.nbase)
input.c = c_mem
midterm.termination_status = 0

if cache !== nothing
    deleteat!(midterm.nbase, cache)
else
    l = findfirst(x->x==input.n + 1,midterm.base)
    B = view(input.A,:,midterm.base)
    for (k,w) in enumerate(midterm.nbase)
        sol = B \ input.A[:,w]
        if abs.(sol[l]) > input.tol
            push!(midterm.base, w)
            deleteat!(midterm.nbase, k)
            deleteat!(midterm.base, l)
            break
        end
        if k == length(midterm.nbase)
            input.A = input.A[1:end .!= l, :]
            input.b = input.b[1:end .!= l]
            deleteat!(midterm.base, l)
            input.m -= 1
            break
        end
    end
end
```

4 Escolha do problema para o BenchMark

O problema escolhido foi retirado do capítulo 16 do manual *Optimization Modeling* da AIMMS `https://manual.aimms.com/_downloads/AIMMS_modeling.pdf`.

The following symbols will be used. *Notation*

Indices:

p	*plant types*
k	*demand categories*

Parameters:

e_p	*existing capacity of plant type p [GW]*
cc_p	*daily fraction of capital cost of plant p [10^3 \$/GW]*
oc_p	*daily operating cost of plant p [10^3 \$/GWh]*
ic_k	*electricity import cost for category k[10^3 \$/GWh]*
d_k	*instantaneous electricity demand for category k [GW]*
du_k	*duration of demand for category k [h]*
r_k	*required electricity for category k [GWh]*

Variables:

x_p	*new design capacity of plant type p [GW]*
y_{pk}	*allocation of capacity to demand [GW]*
z_k	*import of electricity for category k [GWh]*

where the parameter r_k is defined as $r_k = (d_k - d_{k-1})du_k$.

The mathematical description of the model can be stated as follows. *Mathematical model statement*

Minimize:

$$\sum_p cc_p(e_p + x_p) + \sum_k \left(ic_k z_k + du_k \sum_p oc_p y_{pk} \right)$$

Subject to:

$$\sum_k y_{pk} \le e_p + x_p \qquad \forall p$$

$$z_k + du_k \sum_p y_{pk} = r_k \qquad \forall k$$

$$x_p \ge 0 \qquad \forall p$$

$$y_{pk} \ge 0 \qquad \forall (p,k)$$

$$z_k \ge 0 \qquad \forall k$$

Figure 1: Problema de expansão.

5 Script para criar o problema do BenchMark

```
function build_problem(p::Integer, k::Integer)
    Random.seed!(123)
```

```
    e_p = rand(1:5,p)
    cc_p = rand(1:9,p)
    oc_p = rand(1:2,p)
    ic_k = rand(1:30,k)
    dd_k = rand(1:5,k)
    du_k = rand(1:1,k)
    r_k = [dd_k[i]*du_k[i] for i in 1:k]

    lenvars = [p, p*k, k];
    n = sum(lenvars)

    I_pp = Matrix(I,p,p)
    I_kk = Matrix(I,k,k)

    Z_pk = zeros(p,k)
    Z_kp = zeros(k,p)

    D = zeros(p,p*k)
    for i in 1:p
        D[i,1+(i-1)*k:i*k] .= 1
    end

    E = zeros(k,p*k)
    for i in 1:p
        E[:,1+(i-1)*k:i*k] = I_kk
    end

    A = [-I_pp D Z_pk; Z_kp E I_kk; Z_kp -E -I_kk]

    b = vcat(e_p, r_k, -r_k)

    c_y = zeros(p*k)

    for i in 1:p, j in 1:k
        c_y[(i-1)*k+j] = du_k[j]*oc_p[i]
    end

    c = -[cc_p' c_y' ic_k']

    m = p+2*k

    A = [A Matrix(I,m,m)]
    c = vcat(c',zeros(m))
    return A, b, c, n, m
end
```

6 BenchMark

Para o benchmark foram escolhidos os solvers *Open-Source* GLPK e HiGHS. O pacote BenchmarkTools foi usado para para o benchmark, esse pacote roda diversas vezes o mesmo problema e calcula as estáticas principais dos dados.

6.1 p = 2, k =3

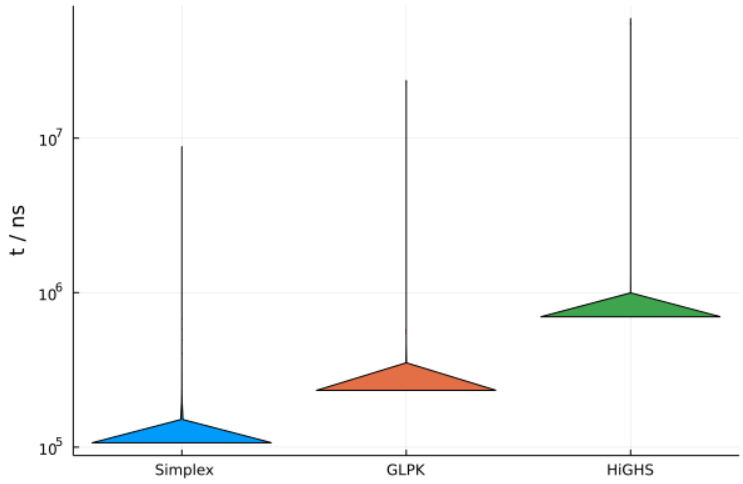

Figure 2: Benchmark p=2,k=3

Neste primeiro caso, o problema possui 11 variáveis e 8 restrições. O solver implementado se saiu melhor que os 2 solvers *open-source*. Sendo cerca de 10 vezes mais rápido que o HiGHS.

6.2 p = 30, k =10

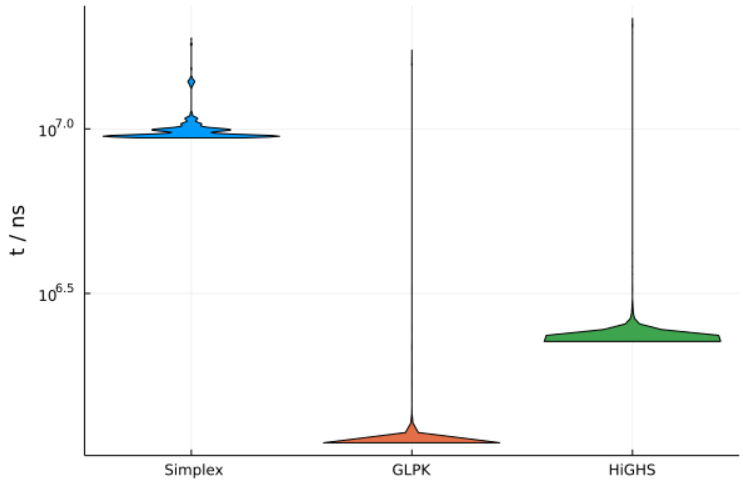

Figure 3: Problema de expansão.

No segundo caso, o problema possui 340 variáveis e 50 restrições. O solver implementado foi o pior dentro os 3. Sendo da ordem de 10 vezes mais lento.

6.3 p = 100, k =50

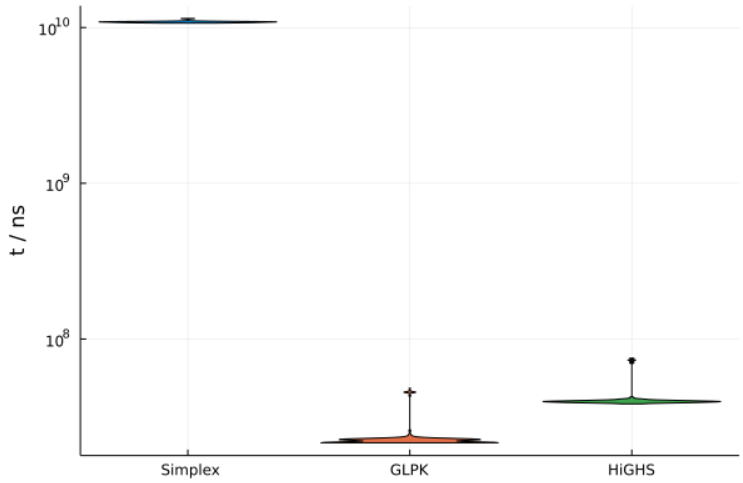

Figure 4: Problema de expansão.

No terceiro caso, o problema possui 5150 variáveis e 200 restrições. O solver implementado foi o pior dentro os 3. Sendo centenas de vezes mais lento que os outros.

7 Conclusão

A implementação do algoritmo Simplex foi concluída com sucesso, ele pode ser usado sem a necessidade de se passar um vértice inicial, não possui ciclos e pode remover variáveis artificiais da primeira etapa com sucesso.

Além disso alguns parâmetros do solver foram adicionados, como tolerância, numero máximo de iterações e verbose.

O BenchMark realizado apontou que a implementação do Simplex feita pelo aluno é muito mais rápida em casos pequenos, porém muito mais menta em casos grandes, quanto comparada com implementações mais profissionais.

Este resultado é esperado uma vez que GLPK e HiGHS contam com diversas rotinas e heurísticas para acelerar a resolução. Por exemplo, eles possuem um "pre-solver", que pode acelerar muito o algoritmo em casos grandes e esparsos. Porém em problemas pequenos, pode ser melhor utilizar o algoritmo base do Simplex.

Além disso, a maior parte gasta no Simplex é resolvendo sistemas lineares, para isso o pacote implementado usa apenas o pacote base do Julia com o MKL da Intel. Usar rotinas especificas para a resolução de grandes sistemas lineares teria um bom impacto no tempo do solver.