Nama : Nico Arie Angga Barus

NPM : G1D021065

Mata kuliah : Optimisasi

Link Youtube : https://youtu.be/K_Mv6K97l4A

Link GitHub : https://github.com/nicobarus65/Tugas-1-Optimisasi_Nico-Arie-Angga-Barus_G1D021065/commit/1370f9222940e40682554a52e0e06cf27018e489

Soal

1. Install Julia
    a. Install JuMP pada julia
    b. Install solver HiGHS dan Ipopt pada julia
2. Pelajari tentang *network flow problem*. Misalnya dengan memahami link ini.
3. Buat atau cari contoh *network flow problem*.
4. Pecahkan soal tersebut dengan menggunakan JuMP Dan HiGHS.
5. Buat video YouTube yang membahas kegiatan 3 dan 4.
6. Upload *source code* yang Anda gunakan ke akun GitHub masing-masing.
7. Buat laporan singkat tentang kegiatan 3 Dan 4. Lampirkan hal berikut:
    a. *Source code* yang digunakan.
    b. Link video YouTube yang menjelaskan tahapan 3 Dan 4.
    c. Link GitHub terkait.
8. Upload laporan ke GitHub Anda.
9. Kegiatan 1 dimulai pukul 8:00 Rabu, 28 Agustus 2024. Kegiatan 8 selesai sebelum pukul 8:00 Rabu, 4 September 2024.

Contoh Soal :

Menyelesaikan masalah aliran jaringan dengan tujuan meminimalkan total biaya aliran berdasarkan matriks biaya G dan menunjukkan jalur optimal dalam bentuk biner dan total biaya optimal.

Source Code

```
julia> G = [
    0 10 20 0 0
    0 0 15 0 0
    0 0 0 40 35
    0 25 0 0 40
    0 0 0 0 0
]
```

G adalah matriks biaya untuk setiap arc dalam jaringan. Elemen G[i, j] menunjukkan biaya untuk melintasi dari node i ke node j. Jika nilai adalah 0, maka tidak ada arc antara node i dan j.

```
5×5 Matrix{Int64}:
 0  10  20   0   0
 0   0  15   0   0
 0   0   0  40  35
 0  25   0   0  40
 0   0   0   0   0
```

```
julia> n=size(G)[1]
5
```

n adalah jumlah baris (atau kolom) dari matriks G, yang menunjukkan jumlah node dalam jaringan. Di sini, n adalah 5.

```
julia> b=[2, -2, 0, 0, 0]
```

b menunjukkan permintaan atau penyediaan di setiap node. Misalnya, elemen ke-1 adalah 2 (sumber aliran), elemen ke-2 adalah -2 (tujuan aliran), dan node lainnya memiliki nilai 0 (node biasa).

```
5-element Vector{Int64}:
  2
 -2
  0
  0
  0
julia> shortest_path=Model(HiGHS.Optimizer)
 # shortest_path adalah model optimisasi menggunakan solver HiGHS.
A JuMP Model
├ solver: HiGHS
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none

julia> set_silent(shortest_path)
```

set_silent(shortest_path) membuat solver tidak menampilkan output selama proses optimisasi.

```
julia> @variable(shortest_path, x[1:n, 1:n], Bin)
```

Mendefinisikan variabel binari x[i, j] yang menunjukkan apakah ada aliran dari node i ke node j (1 jika ada, 0 jika tidak).

```
5×5 Matrix{VariableRef}:
 x[1,1]  x[1,2]  x[1,3]  x[1,4]  x[1,5]
 x[2,1]  x[2,2]  x[2,3]  x[2,4]  x[2,5]
 x[3,1]  x[3,2]  x[3,3]  x[3,4]  x[3,5]
 x[4,1]  x[4,2]  x[4,3]  x[4,4]  x[4,5]
```

x[5,1] x[5,2] x[5,3] x[5,4] x[5,5]

julia> #Arc with zero cost are not a part the path as they do no exist

julia> @constraint(shortest_path,[i=1:n, j=1:n; G[i,j]==0], x[i, j]==0)

memastikan bahwa jika G[i, j] adalah 0 (tidak ada biaya), maka x[i, j] harus 0 (tidak ada aliran).

JuMP.Containers.SparseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float64}}, ScalarShape}, 2, Tuple{Int64, Int64}} with 18 entries:

[1, 1] = x[1,1] == 0
[1, 4] = x[1,4] == 0
[1, 5] = x[1,5] == 0
[2, 1] = x[2,1] == 0
[2, 2] = x[2,2] == 0
[2, 4] = x[2,4] == 0
[2, 5] = x[2,5] == 0
[3, 1] = x[3,1] == 0
[3, 2] = x[3,2] == 0
[3, 3] = x[3,3] == 0
[4, 1] = x[4,1] == 0
[4, 3] = x[4,3] == 0
[4, 4] = x[4,4] == 0
[5, 1] = x[5,1] == 0
[5, 2] = x[5,2] == 0
[5, 3] = x[5,3] == 0
⋮
[5, 4] = x[5,4] == 0
[5, 5] = x[5,5] == 0

```
julia> # Flow conservation constraint
# kendala dalam model aliran jaringan (network flow problem) yang memastikan bahwa jumlah aliran yang masuk sama dengan permintaan atau penyediaan di node

julia> @constraint(shortest_path, [i = 1:n], sum(x[i, :]) - sum(x[:, i]) == b[i],)
#  aliran masuk ke node dikurangi aliran keluar dari node sama dengan permintaan atau penyediaan.
5-element Vector{ConstraintRef{Model,
MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}
, MathOptInterface.EqualTo{Float64}}, ScalarShape}}:
 -x[2,1] - x[3,1] - x[4,1] - x[5,1] + x[1,2] + x[1,3] + x[1,4] + x[1,5] == 2
 x[2,1] - x[1,2] - x[3,2] - x[4,2] - x[5,2] + x[2,3] + x[2,4] + x[2,5] == -2
 x[3,1] + x[3,2] - x[1,3] - x[2,3] - x[4,3] - x[5,3] + x[3,4] + x[3,5] == 0
 x[4,1] + x[4,2] + x[4,3] - x[1,4] - x[2,4] - x[3,4] - x[5,4] + x[4,5] == 0
 x[5,1] + x[5,2] + x[5,3] + x[5,4] - x[1,5] - x[2,5] - x[3,5] - x[4,5] == 0

julia> @objective(shortest_path, Min, sum(G .* x))
# Fungsi objektif untuk meminimalkan total biaya aliran berdasarkan matriks G dan variabel x.
10 x[1,2] + 25 x[4,2] + 20 x[1,3] + 15 x[2,3] + 40 x[3,4] + 35 x[3,5] + 40 x[4,5]

julia> optimize!(shortest_path)
 #solusi yang ditemukan oleh solver

julia> @assert is_solved_and_feasible(shortest_path)
 #kode  ini digunakan untuk memastikan bahwa model optimisasi yang telah dipecahkan memenuhi syarat-syarat tertentu
```

#Feasible menunjukan bahwa solusi memenuhi semua kendala yang ditetapkan dalam model.

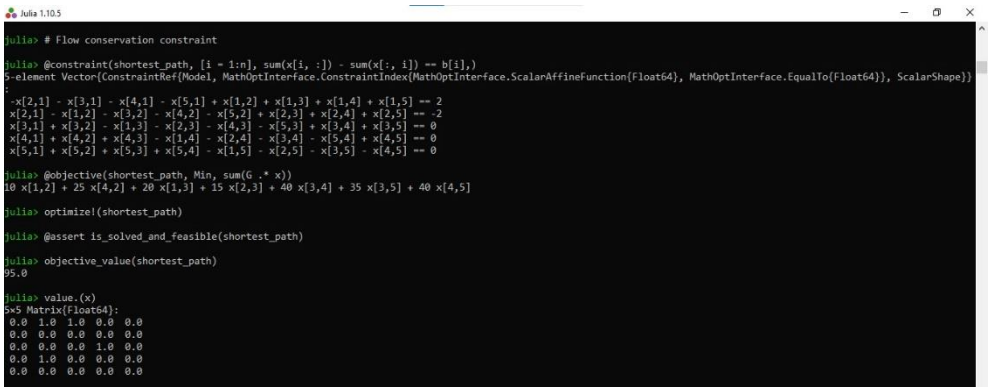
```
julia> objective_value(shortest_path)
 # mengembalikan nilai dari fungsi objektif untuk solusi optimal.
95.0

julia> value.(x)
 # Mengembalikan nilai dari variabel x setelah optimisasi. Di sini, matriks x menunjukkan jalur optimal dalam bentuk binari:
5×5 Matrix{Float64}:
 0.0  1.0  1.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
 0.0  0.0  0.0  1.0  0.0
 0.0  1.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0  0.0
```

Lampiran

```
julia> G = [
       0 10 20 0 0
       0 0 15 0 0
       0 0 0 40 35
       0 25 0 0 40
       0 0 0 0 0
       ]
5×5 Matrix{Int64}:
 0  10  20   0   0
 0   0  15   0   0
 0   0   0  40  35
 0  25   0   0  40
 0   0   0   0   0

julia> n=size(G)[1]
5

julia> b=[2, -2, 0, 0, 0]
5-element Vector{Int64}:
  2
 -2
  0
  0
  0

julia> shortest_path=Model(HiGHS.Optimizer)
A JuMP Model
├ solver: HiGHS
├ objective_sense: FEASIBILITY_SENSE
├ num_variables: 0
├ num_constraints: 0
└ Names registered in the model: none

julia> set_silent(shortest_path)
```

Julia 1.10.5

```
julia> @variable(shortest_path, x[1:n, 1:n], Bin)
5×5 Matrix{VariableRef}:
 x[1,1]  x[1,2]  x[1,3]  x[1,4]  x[1,5]
 x[2,1]  x[2,2]  x[2,3]  x[2,4]  x[2,5]
 x[3,1]  x[3,2]  x[3,3]  x[3,4]  x[3,5]
 x[4,1]  x[4,2]  x[4,3]  x[4,4]  x[4,5]
 x[5,1]  x[5,2]  x[5,3]  x[5,4]  x[5,5]

julia> #Arc with zero cost are not a part the path as they do no exist

julia> @constraint(shortest_path,[i=1:n, j=1:n; G[i,j]==0], x[i, j]==0)
JuMP.Containers.SparseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float64}}
, ScalarShape}, 2, Tuple{Int64, Int64}} with 18 entries:
  [1, 1]  =  x[1,1] == 0
  [1, 4]  =  x[1,4] == 0
  [1, 5]  =  x[1,5] == 0
  [2, 1]  =  x[2,1] == 0
  [2, 2]  =  x[2,2] == 0
  [2, 4]  =  x[2,4] == 0
  [2, 5]  =  x[2,5] == 0
  [3, 1]  =  x[3,1] == 0
  [3, 2]  =  x[3,2] == 0
  [3, 3]  =  x[3,3] == 0
  [4, 1]  =  x[4,1] == 0
  [4, 3]  =  x[4,3] == 0
  [4, 4]  =  x[4,4] == 0
  [5, 1]  =  x[5,1] == 0
  [5, 2]  =  x[5,2] == 0
  [5, 3]  =  x[5,3] == 0
  [5, 4]  =  x[5,4] == 0
  [5, 5]  =  x[5,5] == 0
```


Julia 1.10.5
julia> # Flow conservation constraint
julia> @constraint(shortest_path, [i = 1:n], sum(x[i, :]) - sum(x[:, i]) == b[i],)
5-element Vector{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float64}}, ScalarShape}}
:
-x[2,1] - x[3,1] - x[4,1] - x[5,1] + x[1,2] + x[1,3] + x[1,4] + x[1,5] == 2
x[2,1] - x[3,2] - x[3,2] - x[4,2] - x[5,2] + x[2,3] + x[2,4] + x[2,5] == -2
x[3,1] + x[3,2] - x[1,3] - x[2,3] - x[4,3] - x[5,3] + x[3,4] + x[3,5] == 0
x[4,1] + x[4,2] + x[4,3] - x[1,4] - x[2,4] - x[3,4] - x[5,4] + x[4,5] == 0
x[5,1] + x[5,2] + x[5,3] + x[5,4] - x[1,5] - x[2,5] - x[3,5] - x[4,5] == 0
julia> @objective(shortest_path, Min, sum(G .* x))
10 x[1,2] + 25 x[4,2] + 20 x[1,3] + 15 x[2,3] + 40 x[3,4] + 35 x[3,5] + 40 x[4,5]
julia> optimize!(shortest_path)
julia> @assert is_solved_and_feasible(shortest_path)
julia> objective_value(shortest_path)
95.0
julia> value.(x)
5×5 Matrix{Float64}:
0.0 1.0 1.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0
0.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0