

**TUGAS 1 OPTIMISASI  
MATAKULIAH OPTIMISASI  
NETWORK FLOW PROBLEM**



NAMA : RENDI YUDHA FRENDIKA  
NPM : G1D021002

**PROGRAM STUDI TEKNIK ELEKTRO  
FAKULTAS TEKNIK  
UNIVERSITAS BENGKULU  
2024**

Nama : Rendi Yudha Frendika  
NPM : G1D021002  
Matakuliah : Optimisasi

1. Link YouTube

<https://youtu.be/j253oTJX5ek>

2. Link GitHub

[https://github.com/rendiyudhafrendika/Optimisasi\\_part1/blob/main/task\\_optimization](https://github.com/rendiyudhafrendika/Optimisasi_part1/blob/main/task_optimization)

## Tugas 1 Optimisasi Network Flow Problem



## Tugas 1 Optimisasi Program Studi Teknik Elektro Universitas Bengkulu

Dosen Pengampu: Novalio Daratha

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.

**Selamat berjuang!**

### 1. Example of network flow

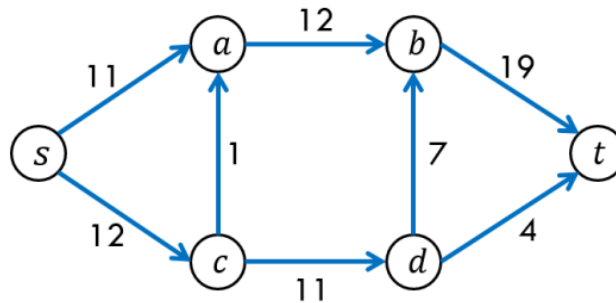


Figure 1. Network Flow

Gambar ini menggambarkan sebuah network flow yang umum digunakan dalam algoritma terkait teori graf, seperti algoritma Ford-Fulkerson untuk menemukan aliran maksimum dalam sebuah jaringan.

Pada network flow diatas dapat dideskripsikan sebagai berikut dari jaringan tersebut:

- Simpul (Node/Vertex): s, a, b, c, d, t

Dengan s sebagai simpul sumber yang merupakan sebagai simpul awal dalam network flow dan t sebagai simpul tujuanyang merupakan titik akhir dari network flow berhenti.

- **Tepi (Edges):** Menunjukkan arah dan kapasitas aliran antara simpul-simpul.
  - $s \rightarrow a$  (Kapasitas: 11)
  - $s \rightarrow c$  (Kapasitas: 12)
  - $a \rightarrow b$  (Kapasitas: 12)
  - $a \rightarrow c$  (Kapasitas: 1)
  - $c \rightarrow d$  (Kapasitas: 11)
  - $b \rightarrow d$  (Kapasitas: 7)
  - $b \rightarrow t$  (Kapasitas: 19)
  - $d \rightarrow t$  (Kapasitas: 4)

Dari network flow diatas langkah yang digunakan adalah untuk menentukan max\_flow. Max\_flow adalah optimisasi dari seberapa besar aliran maksimum yang dapat ditransfer dari s ke t melalui jaringan, dengan memperhatikan kapasitas maksimum dari setiap tepi (*edge*) yang menghubungkan simpul-simpul di dalam jaringan. Untuk

menyelesaikan optimisasi dari network flow diatas maka digunakanlah pemrograman dengan bahasa Julia.

```
Julia 1.10.4 | Documentation: https://docs.julialang.org
Type "?" for help, "]" for Pkg help.
Version 1.10.4 (2024-06-04)
Official https://julialang.org/ release

julia> using JuMP
julia> import HiGHS

julia> x = [
    0 11 0 12 0 0
    0 0 12 0 0 0
    0 0 0 0 19
    0 1 0 0 11 0
    0 0 7 0 0 4
    0 0 0 0 0 0
]
6x6 Matrix{Int64}:
 0 11 0 12 0 0
 0 0 12 0 0 0
 0 0 0 0 19
 0 1 0 0 11 0
 0 0 7 0 0 4
 0 0 0 0 0 0

julia> n = size(x)[1]
6

julia> max_flow = 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 1.10.4

julia> @variable(max_flow, f[1:n, 1:n] >= 0)
6x6 Matrix{VariableRef}:
 f[1,1] f[1,2] f[1,3] f[1,4] f[1,5] f[1,6]
 f[2,1] f[2,2] f[2,3] f[2,4] f[2,5] f[2,6]
 f[3,1] f[3,2] f[3,3] f[3,4] f[3,5] f[3,6]
 f[4,1] f[4,2] f[4,3] f[4,4] f[4,5] f[4,6]
 f[5,1] f[5,2] f[5,3] f[5,4] f[5,5] f[5,6]
 f[6,1] f[6,2] f[6,3] f[6,4] f[6,5] f[6,6]

julia> @constraint(max_flow, [i = 1:n, j = 1:n], f[i, j] <= x[i, j])
6x6 Matrix{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.LessThan{Float64}}, ScalarShape}}:
 f[1,1] <= 0 f[1,2] <= 11 f[1,3] <= 0 f[1,4] <= 12 f[1,5] <= 0 f[1,6] <= 0
 f[2,1] <= 0 f[2,2] <= 0 f[2,3] <= 12 f[2,4] <= 0 f[2,5] <= 0 f[2,6] <= 0
 f[3,1] <= 0 f[3,2] <= 0 f[3,3] <= 0 f[3,4] <= 0 f[3,5] <= 0 f[3,6] <= 19
 f[4,1] <= 0 f[4,2] <= 1 f[4,3] <= 0 f[4,4] <= 0 f[4,5] <= 11 f[4,6] <= 0
 f[5,1] <= 0 f[5,2] <= 0 f[5,3] <= 7 f[5,4] <= 0 f[5,5] <= 0 f[5,6] <= 4
 f[6,1] <= 0 f[6,2] <= 0 f[6,3] <= 0 f[6,4] <= 0 f[6,5] <= 0 f[6,6] <= 0

julia> @constraint(max_flow, [i = 1:n; i != 1 && i != 6], sum(f[i, :]) == sum(f[:, i]))
JuMP.Containers.SparseAxisArray{ConstraintRef{Model, MathOptInterface.ConstraintIndex{MathOptInterface.ScalarAffineFunction{Float64}, MathOptInterface.EqualTo{Float64}}, ScalarShape}, 1, Tuple{Int64}} with 4 entries:
 [2] = f[2,1] - f[1,2] - f[3,2] - f[4,2] - f[5,2] - f[6,2] + f[2,3] + f[2,4] + f[2,5] + f[2,6] == 0
 [3] = f[3,1] + f[3,2] - f[1,3] - f[2,3] - f[4,3] - f[5,3] - f[6,3] + f[3,4] + f[3,5] + f[3,6] == 0
 [4] = f[4,1] + f[4,2] + f[4,3] - f[1,4] - f[2,4] - f[3,4] - f[5,4] - f[6,4] + f[4,5] + f[4,6] == 0
 [5] = f[5,1] + f[5,2] + f[5,3] + f[5,4] - f[1,5] - f[2,5] - f[3,5] - f[4,5] - f[6,5] == 0

julia> @objective(max_flow, Max, sum(f[1, :]))
f[1,1] + f[1,2] + f[1,3] + f[1,4] + f[1,5] + f[1,6]

julia> optimize!(max_flow)
Running HiGHS 1.7.2 (git hash: 5ce7a2753): Copyright (c) 2024 HiGHS under MIT licence terms
Coefficient ranges:
  Matrix [1e+00, 1e+00]
  Cost   [1e+00, 1e+00]
  Bound  [0e+00, 0e+00]
  RHS    [1e+00, 2e+01]
Presolving model
1 rows, 4 cols, 3 nonzeros 0s
0 rows, 0 cols, 0 nonzeros 0s
```

```
Julia 1.10.4
0 rows, 0 cols, 0 nonzeros 0s
Presolve : Reductions: rows 0(-48); columns 0(-36); elements 0(-76) - Reduced to empty
Solving the original LP from the solution after postsolve
Model status      : Optimal
Objective value    : 2.3800000000000e+01
HiGHS run time     : 0.02

julia> @assert is_solved_and_feasible(max_flow)

julia> objective_value(max_flow)
23.0

julia>
```

Untuk menyelesaikan masalah max\_flow dengan menggunakan bahasa pemrograman Julia.

- a. Baris kode **using JuMP** dengan **Import HiGHS** Untuk dapat mendefinisikan variabel keputusan, fungsi tujuan yang akan digunakan, di kasus ini adalah network flow
- b. Langkah kedua adalah dengan memodelkan network flow tersebut dengan menggunakan matriks, yaitu untuk merepresentasikan jaringan tersebut dalam bentuk matriks kapasitas. Matriks tersebut menunjukkan kapasitas flow antara setiap pasangan simpul dalam jaringan. Berdasarkan jumlah node pada network flow maka matriksnya dapat membentuk matriks ordo 6x6 yang merepresentasikan kapasitas antar node sebagai berikut

	s	a	b	c	d	e
s	0	11	0	12	0	0
a	0	0	12	0	0	0
b	0	0	0	0	0	19
c	0	1	0	0	11	0
d	0	0	7	0	0	4
t	0	0	0	0	0	0

Dari matriks tersebut kemudian dimasukkan ke Julia untuk dieksekusi sehingga Julia mendeklarasikan bahwa user memasukkan nilai matriks.

- c. Baris kode **n = size(x)[1]** adalah mendeklarasikan variabel n sebagai bentuk ukuran dari matriks x dengan ordo 6x6
- d. Baris kode mendeklarasikan sebuah model variabel **max\_flow** solver optimasi program
- e. Baris kode **@variable(max\_flow, f[1:n, 1:n] >= 0)** mendefinisikan variabel keputusan **f[i, j]** dalam model **max\_flow**, yang merupakan matriks berukuran **n x n**. Setiap elemen dari matriks ini merepresentasikan aliran antara dua node dalam sebuah jaringan. Kondisi **>= 0** memastikan bahwa aliran ini adalah non-negatif, sesuai dengan aturan dalam network flow problem

- f. baris kode **@constraint(max\_flow, [i = 1:n, j = 1:n], f[i, j] <= G[i, j])** mendefinisikan setiap edge pada model max\_flow di network flow dari satu untuk setiap pasangan (i, j) dari node dalam jaringan.
- g. Baris kode **@constraint(max\_flow, [i = 1:n; i != 1 && i != 6], sum(f[i, :]) == sum(f[:, i]))** ini mendeklarasikan bahwa untuk setiap node i yang tidak sama dengan 1 hingga 6, jumlah aliran yang keluar dari node i harus sama dengan jumlah aliran yang masuk ke node i.
- h. Baris kode **@objective(max\_flow, Max, sum(f[1, :]))** ini mendeklarasikan fungsi tujuan dalam model max\_flow untuk memaksimalkan total flow dari node sumber (node s) ke semua node lainnya.
- i. Baris kode **optimize!(max\_flow)** ini mendeklarasikan perintah untuk menjalankan solver dan menyelesaikan model optimasi
- j. Baris kode **objective\_value(max\_flow)** memberikan nilai terbaik dari fungsi tujuan yang dicapai setelah optimasi model max\_flow. Dari pemrograman diatas didapatkanlah nilai maksimal terbaik setelah optimasi max\_flow adalah 23.0