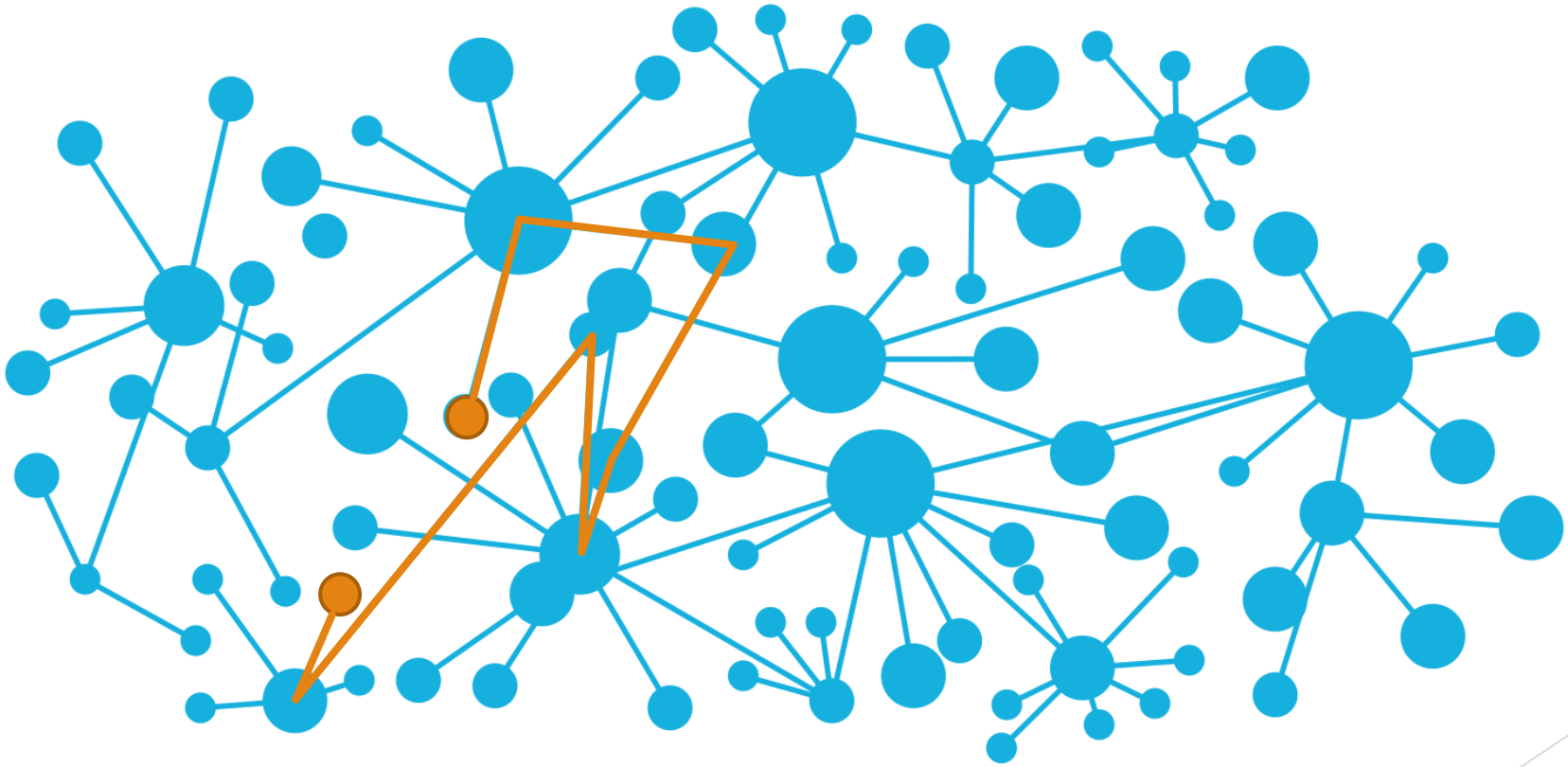
Seleroute.jl, a generic package for network-routing optimisation

Thibaut Cuvelier

What is network routing?



What is network routing?

- **What path should each packet take throughout the network?**
- **Most basic answer: shortest path**
  - Fewest number of routers crossed
  - No prior knowledge of the flows that will traverse the network
- **Traffic engineering**
  - Optimise some metric: network congestion, end-to-end delay, etc.
  - Based on what you know about the users of the network (e.g., traffic)

What are practitioners interested in?

- Performance at scale
  - Find a good routing for a large network (1000s of nodes) in minutes
  - Find the optimum routing for a large network in hours
- Choice of objective function
  - Minimise the worst-case load / delay
  - Maximise the fairness between links
- Choice of uncertainty model
  - Typically, uncertain demand (optical fibre)
  - Sometimes, link capacity (wireless networks)

How to perform traffic engineering

Using mathematical optimisation

How to decide how to route?

- A routing: path from source to destination for all pairs with traffic

- Almost all deployed algorithms are based on mathematical optimisation
  - For variables, use paths (column generation) or flows in the graph

  - **Integer program**: a single path per pair – easy to implement, no load balancing
    Extremely hard to compute!
  - **Linear program**: multiple paths per pair – hard to implement, paths have fractional solution for load balancing
    For instance, 50% of total flow for path 1, 30% for path 2, 20% for path 3
    Still very hard to compute!

Fundamental tool: multicommodity flow (MCF)

- Knowing the traffic the network will face, how to route it?

- How does the model work?
  - **Objective function**: minimise the maximum link utilisation
    - Link utilisation: amount of traffic through the link / link capacity
    - Maximum link utilisation: utilisation of the most loaded link
  - **Usual formulation**: path-based (column generation)
  - **Variables**: fraction of each demand in each path

Fundamental tool: multicommodity flow (MCF)

$$\min \mu$$

$$\sum_{p \in \mathcal{P}_d} routing_{d,p} = 1 \qquad \forall d \in \mathcal{D}$$

$$\sum_{d \in \mathcal{D}} \sum_{\substack{p \in \mathcal{P}_d: \\ e \in p}} demand_d \times routing_{d,p} \leq \mu \times capacity_e \qquad \forall e \in E$$

- Intuition?
  - If μ is 1, at least one edge needs its full capacity, and has a 100% utilisation
  - If μ is 0.5, no edge needs more than half its capacity: the worst utilisation is 50%

Routing and uncertainty

Uncertainty? Where?

- Problem of the previous formulation?
  - Only **one** traffic matrix is considered!
- In practice: the traffic is probably far from this matrix
  - Even from the average matrix
  - Congestion can be arbitrarily far from that of the optimal routing of the average matrix
- **Oblivious routing**: consider **several** traffic matrices
  - Only those that correspond to the **worst traffic conditions**
  - **Only one routing for the network that works reasonably well for any traffic scenario**

The *new* MCF

- Main difference? Multiple traffic matrices, taken in Δ (details later)

$$\min \mu$$

$$\sum_{p \in \mathcal{P}_d} routing_{d,p} = 1 \qquad \forall d \in \mathcal{D}, \boxed{\forall \mathcal{D} \in \Delta}$$

$$\sum_{d \in \mathcal{D}} \sum_{\substack{p \in \mathcal{P}_d: \\ e \in p}} demand_d \times routing_{d,p} \leq \mu \times capacity_e \times \boxed{OPT(D)} \qquad \forall e \in E, \boxed{\forall \mathcal{D} \in \Delta}$$

- OPT(D) is the optimum congestion for D

- In the end: μ is the worst ratio between the oblivious routing and optimal routing
- For the first constraint, no real impact, as long as all matrices $\mathcal{D} \in \Delta$ have the same origin-destination pairs

Maybe oblivious is too extreme

- Oblivious routing keeps everything under control
  - Whatever the uncertainty, it is dealt with
- In practice, not **that** useful
  - Operators monitor their networks and know what traffic they usually see
  - Future traffic can be estimated with a high accuracy (± 5%)
- How about restraining the set of traffic matrices Δ?
  - **Robust routing**

Seleroute.jl

One package to rule them all!

Seleroute.jl

So far, many options to compute a routing:

- Formulation: flows vs. paths
- Uncertainty source: demand, capacity, or both
- Uncertainty formulation: none, robust, oblivious, stochastic
  And several algorithms for each: iterative or reformulation
- Objective functions:
  - Maximise fairness:
    max-min fairness (LP), α-fairness (convex program: LP, SOCP, POW, EXP)
  - Minimise the maximum load:
    standard load (LP), Kleinrock (SOCP), Fortz-Thorup (convex piecewise linear)

Seleroute.jl: software engineering

- Major goal: share as much code as possible between these choices! Second goal: be extensible
- Use Julia's multiple dispatch!

- Encode the choices into a structure, `ModelType`: one field per parameter
- Many abstract types, one per kind of decision to make

- **Result? No performance penalty** compared to a direct implementation for each choice of parameters!

Seleroute.jl: software engineering

Many abstract types:

- How to measure the load per edge: `EdgeWiseObjectiveFunction`
  standard load, Kleinrock, Fortz-Thorup, α-fairness
- How to aggregate the load per edge: `AggregationObjectiveFunction`
  sum, maximum/minimum, max-min fairness
- What formulation to use: `FormulationType`
  flows or path (iterative with column generation)
- What algorithm to use: `AlgorithmChoice`
  reformulation or cutting planes (iterative)
- What parameters are uncertain: `UncertainParameters`
  demand or capacity
- What uncertainty model to use: `UncertaintyHandling`
  none, robust, oblivious, stochastic

Seleroute.jl: software engineering

- Code-wise, the major decision is: the algorithm!
  Seleroute.jl provides only four:
  - Without uncertainty
  - Max-min fairness
  - Oblivious/robust: iterative or reformulation
- Implementation split in basic blocks: capacity constraints, solving main problem, modifying it with one iteration, etc.
- Several variants for each: use (multiple) dispatch to choose

Seleroute.jl: software engineering

- Minor decisions:
  - Flows vs. paths
  - Objective function
- The algorithms call helpers:
  - Build the main mathematical formulation: `basic_routing_model`, `total_flow_in_edge`
  - Build a term of the objective function: `objective_edge_expression`

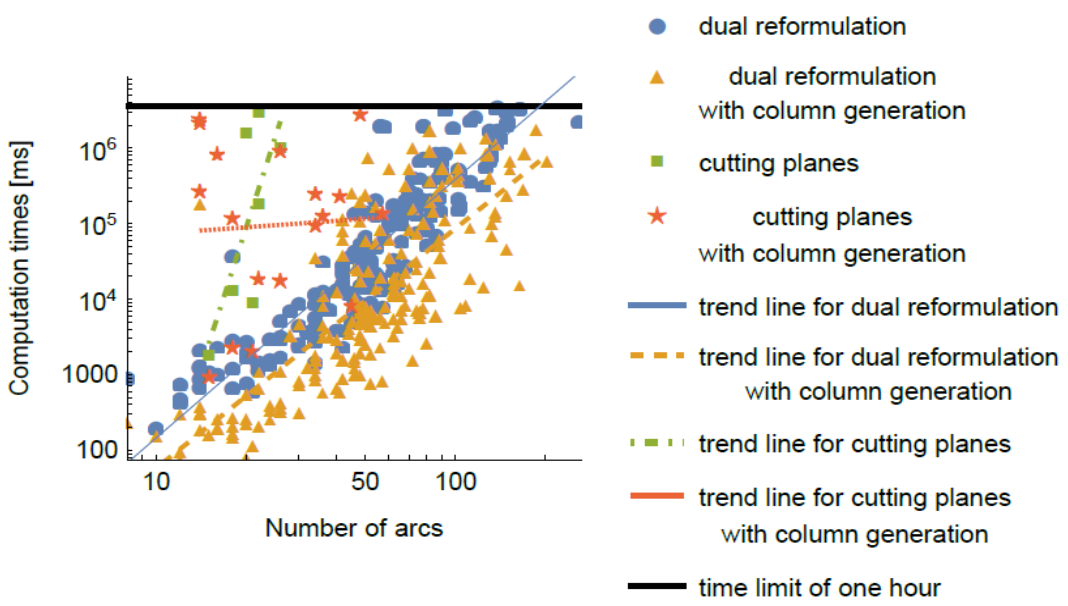
Seleroute.jl: what does it look like?

Compute a routing without uncertainty:

```
function compute_routing(rd::RoutingData, edge_obj::EdgeWiseObjectiveFunction,
        agg_obj::Union{MinimumTotal, MaximumTotal}, ::FormulationType,
        ::Val{false}, ::Automatic, ::NoUncertaintyHandling, ::NoUncertainty)
    m = _create_model(rd)
    rm = basic_routing_model_unitary(m, rd)
    capacity_constraints(rm, rd.traffic_matrix)

    obj = sum(objective_edge_expression(rm, edge_obj, e) for e in edges(rd))
    if agg_obj <: MinimumTotal
        @objective(m, Min, obj)
    # ...
    end    optimize!(m)
    return RoutingSolution(…)
end
```

Seleroute.jl: numerical results



- Having several implementation allows comparing their results
- JuMP makes it easy to change the underlying solver!

Conclusions

Conclusion

- Julia has a good ecosystem for mathematical optimisation and graphs
- Julia has interesting features and allows efficient software engineering with virtually no performance impact

- Very little open-source software for traffic engineering:
  - Focus on infrastructure (P4, OpenFlow, and their implementations)
  - Or on protocols (Quagga, Facebook Open/R, BIRD, OpenBPGD, etc.)


DO YOU HAVE ANY QUESTIONS FOR ME?