Flows in Temporal Graphs

Solving Methods and Open Questions

Mathilde Vernet
{firstname.lastname}@univ-avignon.fr

LIA, Avignon Université

Poitiers
November 25, 2022

The work of many people

- From my PhD thesis and after with Yoann Pigné and Éric Sanlaville
- Join work with Maciej Drozdowski
- Ongoing work

Outline

In static graphs

- Maximum flow
- Minimum cost flow

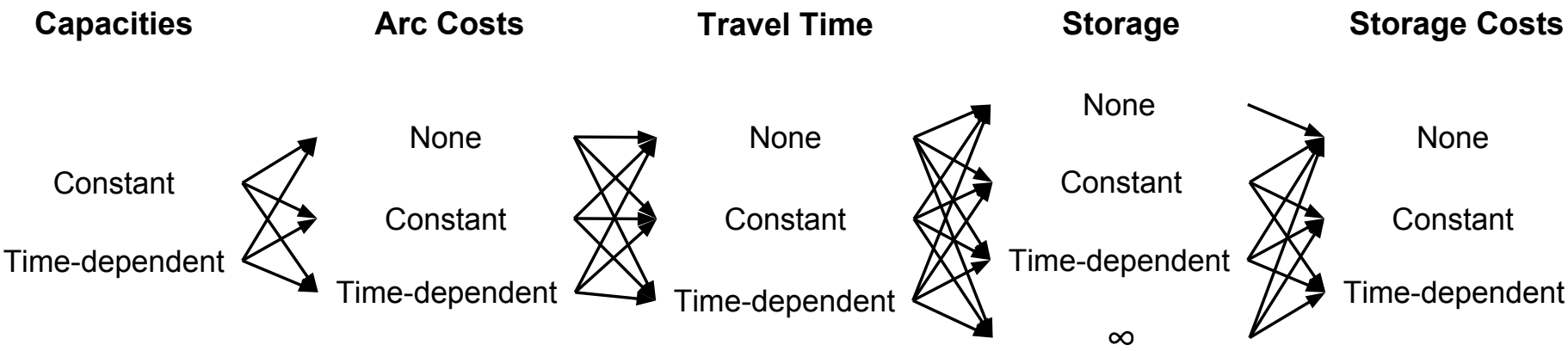
In temporal graphs

- Previous problems still exist
  - Maximum flow
  - Minimum cost flow
- Time creates more flow problems
  - Quickest flow
  - Earliest flow

How to solve them?

- Use the time-expanded graph
- Provide new method

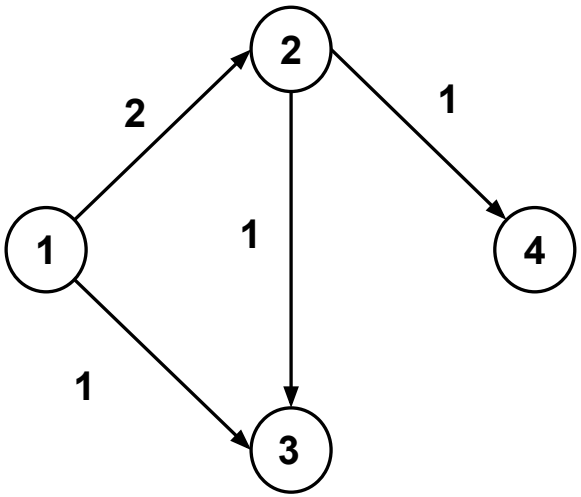
Various models

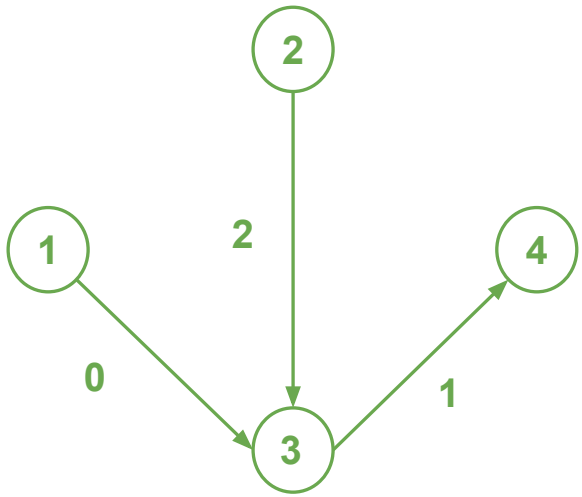


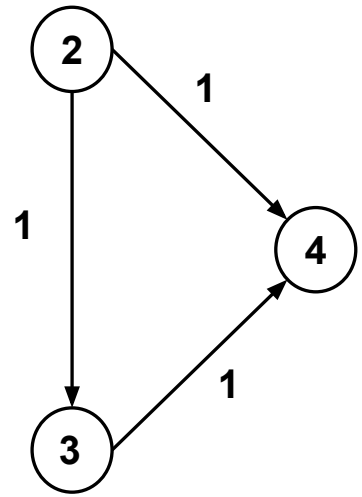
Remark

- The flow problem and its solution depend on the model
  - *Max flow and fixed capacities* $\Rightarrow$ *repeating flow*
  - *Max flow and no storage* $\Rightarrow$ *solve each time step separately*
  - *Max flow and fixed/Infinite storage* $\Rightarrow$ *different solutions*

Definition



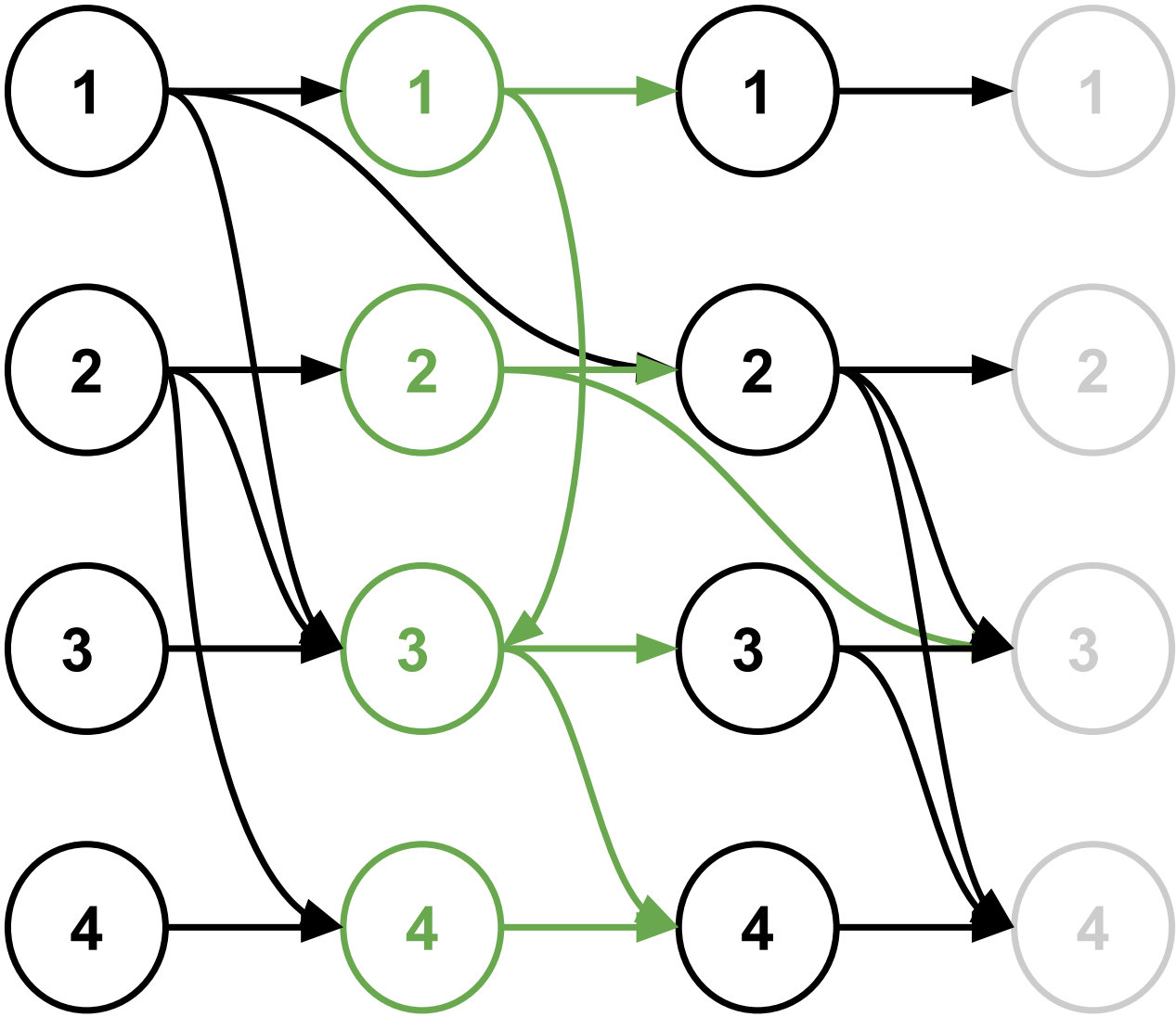




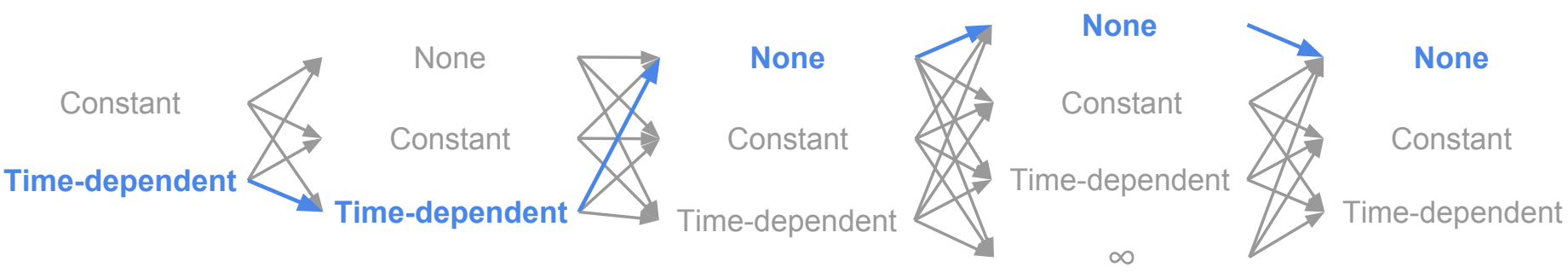
Building the time-expanded graph

- Duplicate each vertex i for each time step : $i_1 \dots i_T$
- For each arc ij at time t with travel time τ, create arc $i_t j_{t+\tau}$ of same capacity

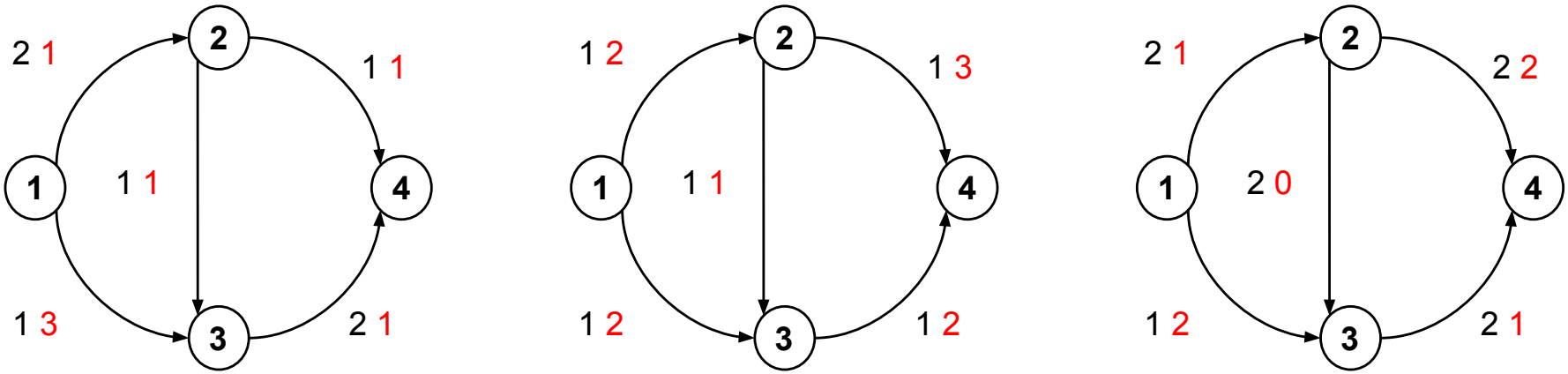
Example



Model Parameters



Example



Arc Capacities Arc Unit Costs

Problem Parameters

- Time horizon fixed
- Unique source and unique sink fixed
- Amount of flow to be sent fixed

Constraints

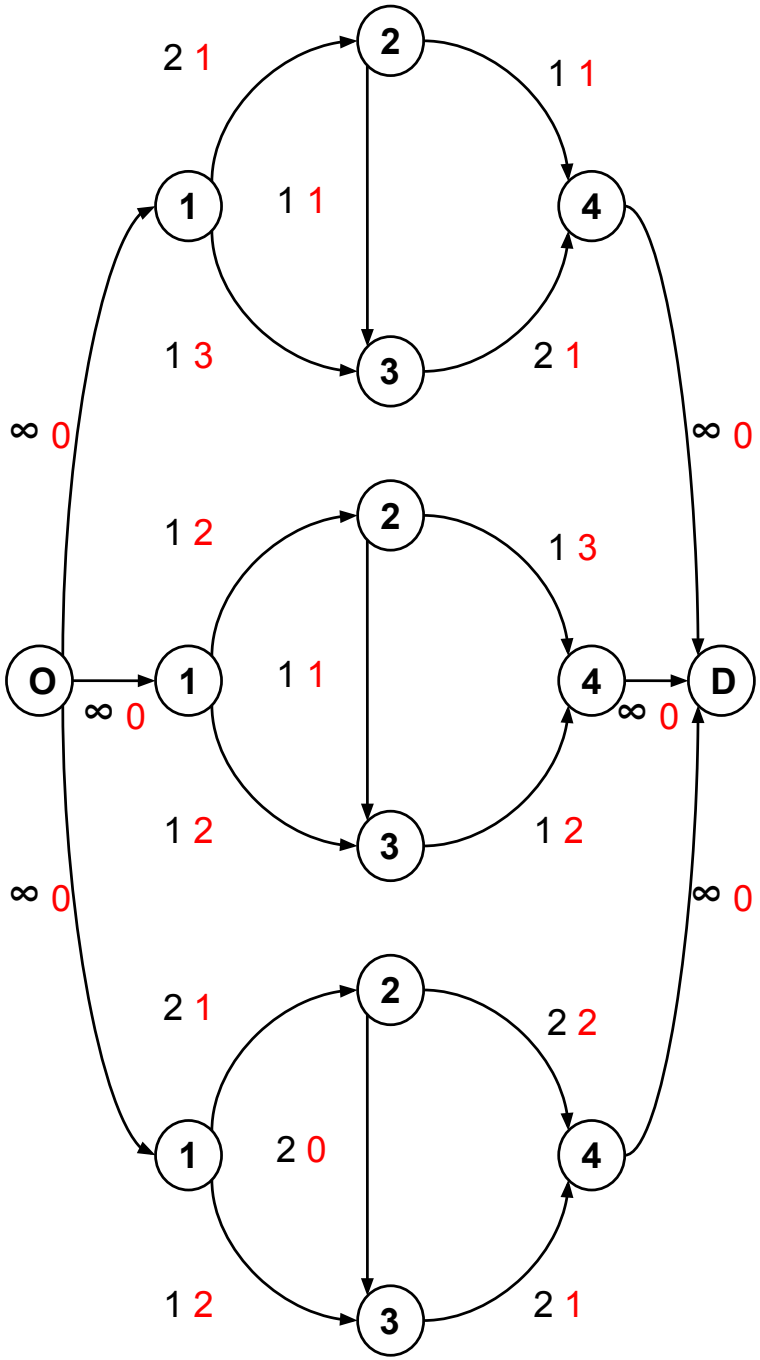
- Respect flow conservation on vertices
- Respect capacities on arcs
- Respect time horizon

Goal

- Send flow the cheapest possible way

Relevance

- Optimum $\neq$ Solving at each time step
- Be more efficient



Size of the Time-Expanded Graph

- $T \cdot n + 2$ vertices
- $T \cdot m + 2 \cdot T$ arcs

Complexity

- Successive Shortest Path Algorithm on Static Graph
  - $U \cdot (m + n \cdot log(n))$
- Successive Shortest Path on Time-Expanded Graph
  - $U \cdot T \cdot (m + n \cdot log(n \cdot T))$

CAN IT BE IMPROVED ?

Presentation

Idea

- Based on the Successive Shortest Path Algorithm (SSP)

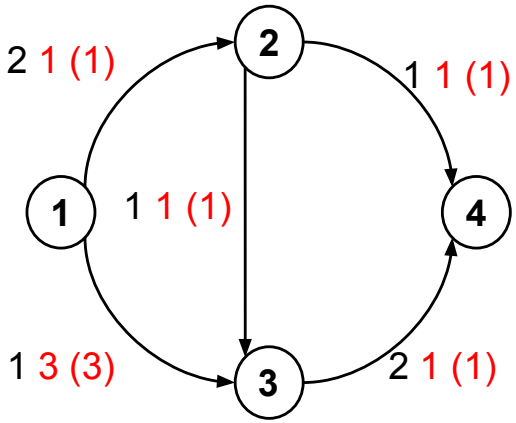
Outline of the Algorithm

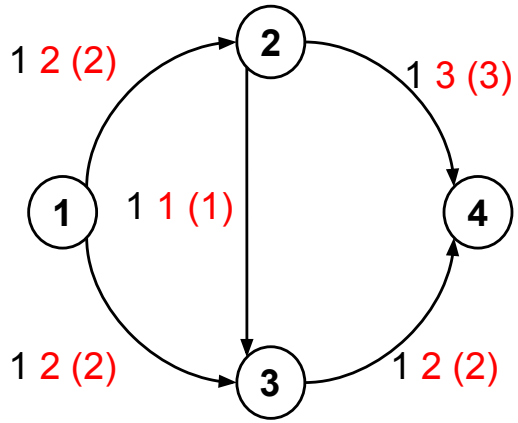
1. Initialization : one iteration of SSP on each t-graph G_t
2. Iteration : one iteration of SSP on the best t-graph G_t
3. Back to 2 if there are flow units left to be sent

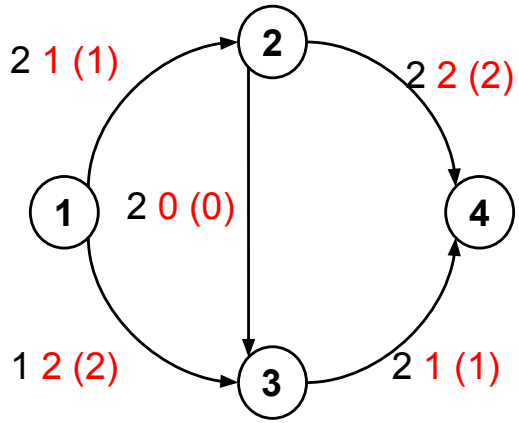
Example

Algorithm's Execution on Residual Graph

- Objective : $U = 4$







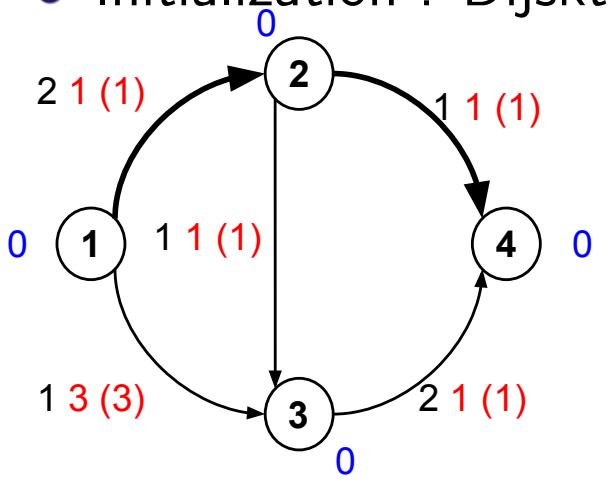
Algorithm's Result

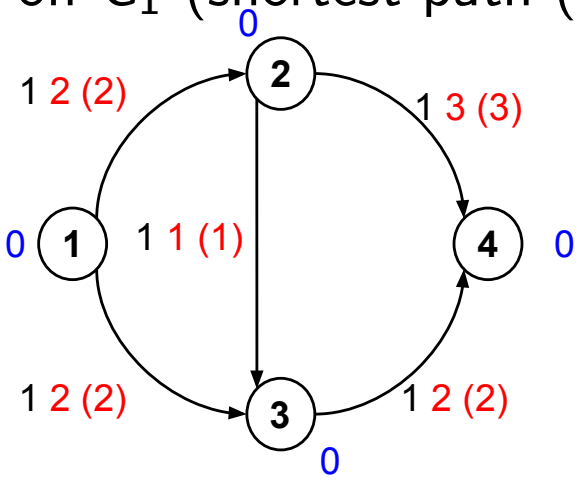
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

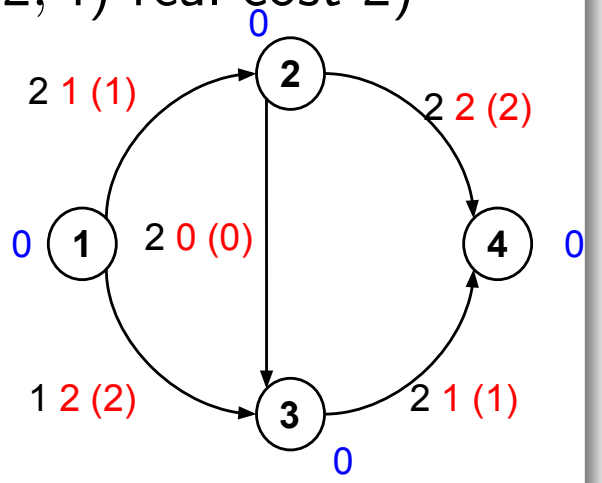
Example

Algorithm's Execution on Residual Graph

- Initialization : Dijsktra on G_1 (shortest path $(1, 2, 4)$ real cost 2)







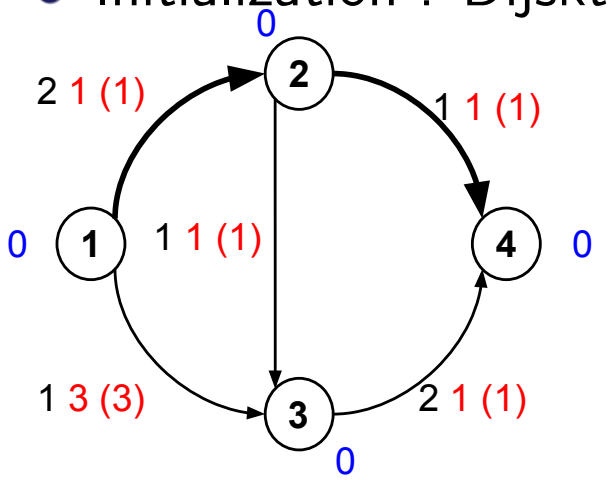
Algorithm's Result

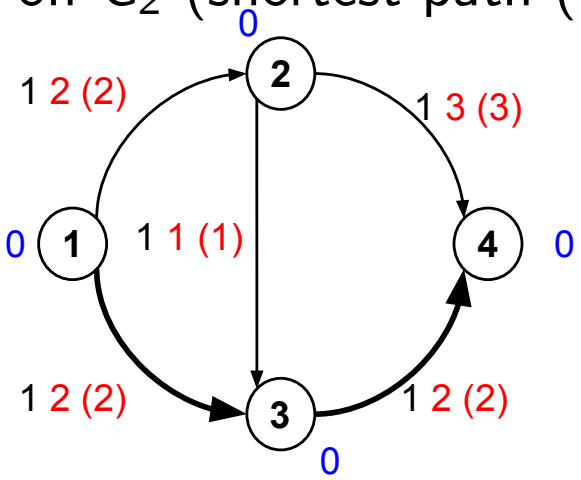
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

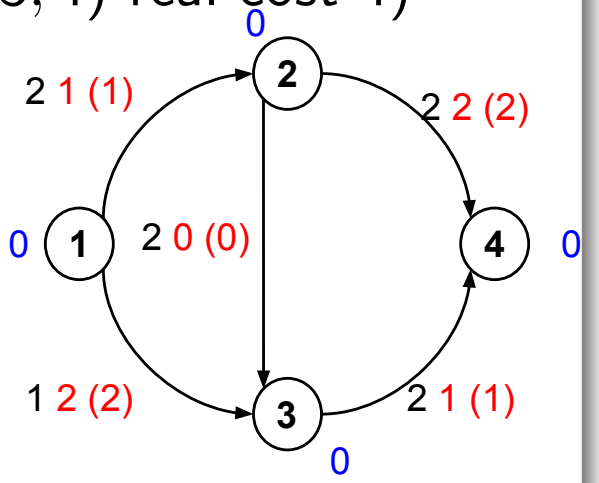
Example

Algorithm's Execution on Residual Graph

- Initialization : Dijsktra on G_2 (shortest path $(1, 3, 4)$ real cost 4)







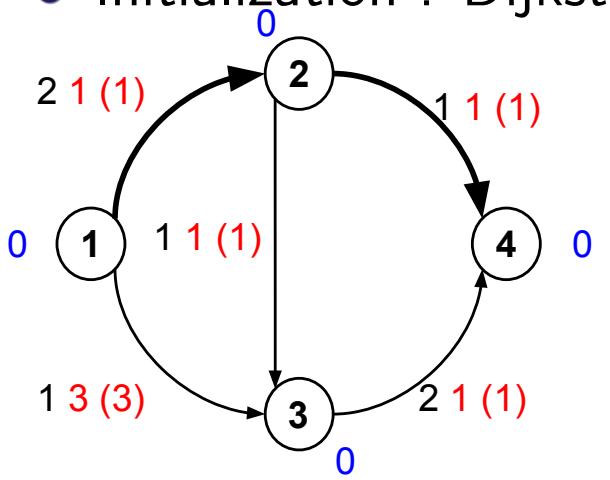
Algorithm's Result

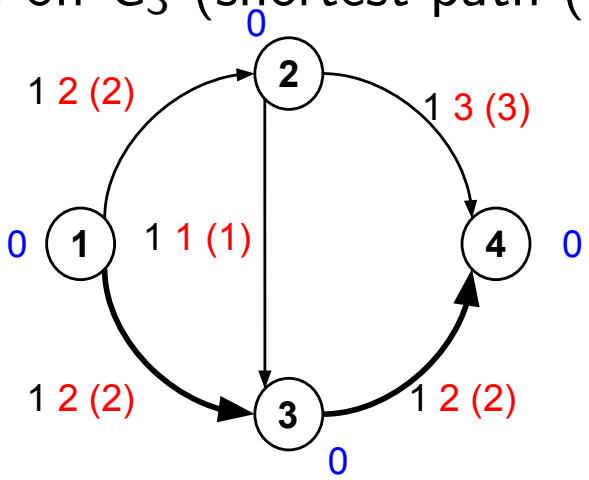
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

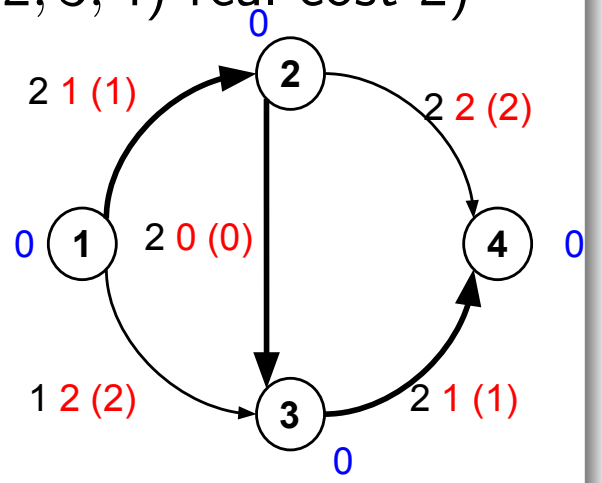
Example

Algorithm's Execution on Residual Graph

- Initialization : Dijkstra on G_3 (shortest path $(1, 2, 3, 4)$ real cost 2)







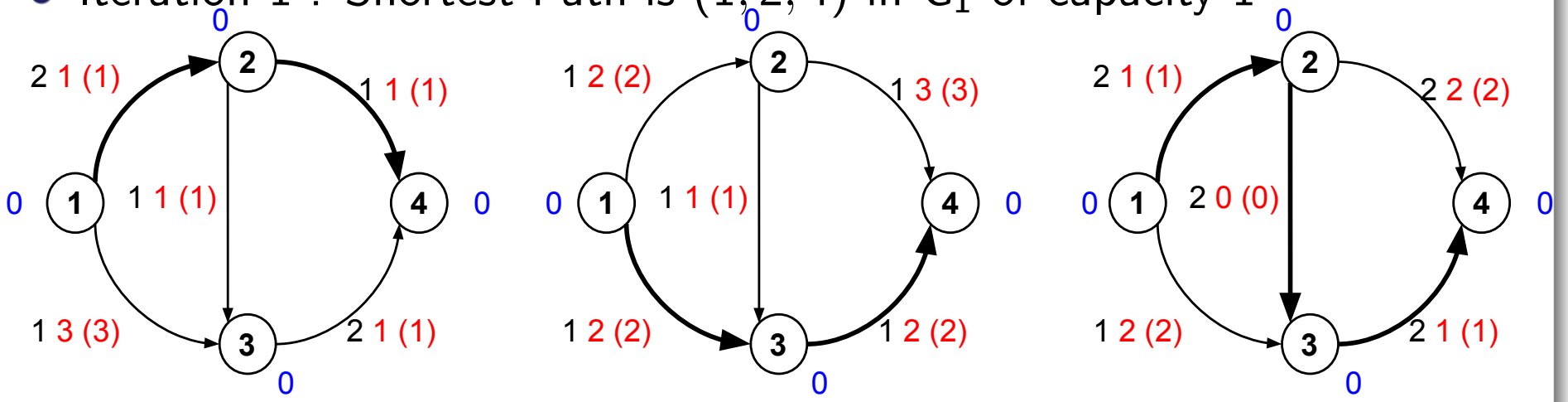
Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Example

Algorithm's Execution on Residual Graph

- Iteration 1 : Shortest Path is $(1, 2, 4)$ in G_1 of capacity 1



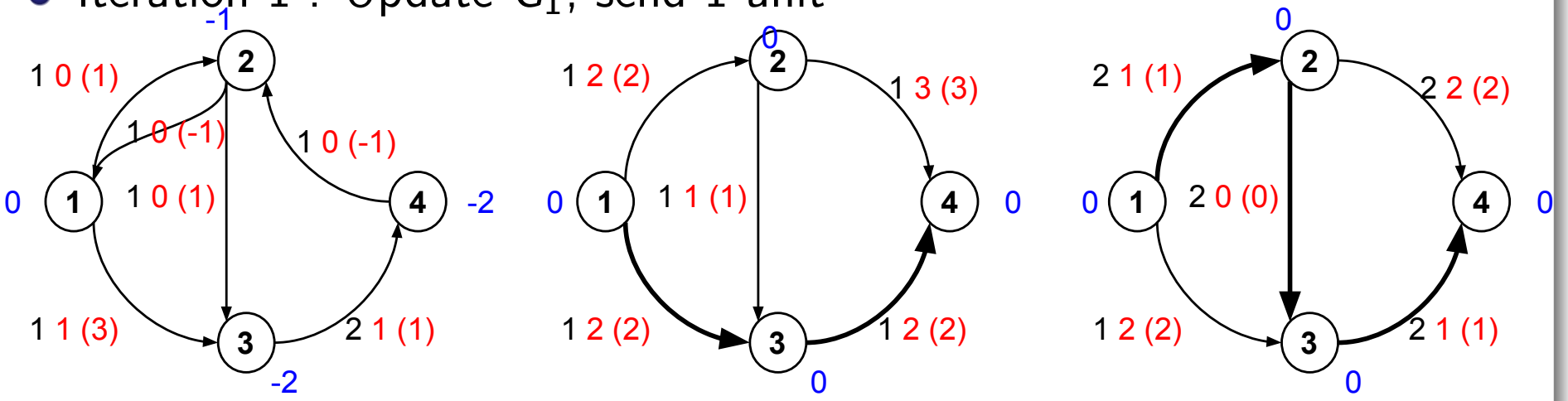
Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Example

Algorithm's Execution on Residual Graph

- Iteration 1 : Update G_1, send 1 unit



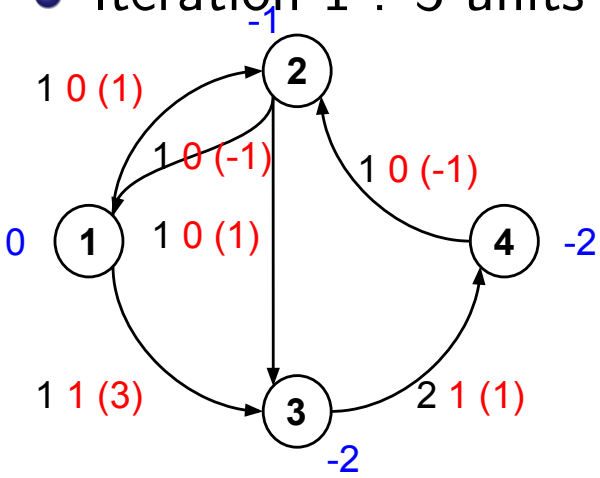
Algorithm's Result

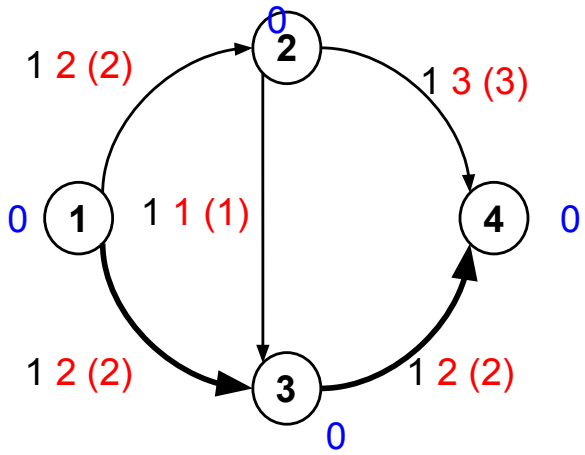
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

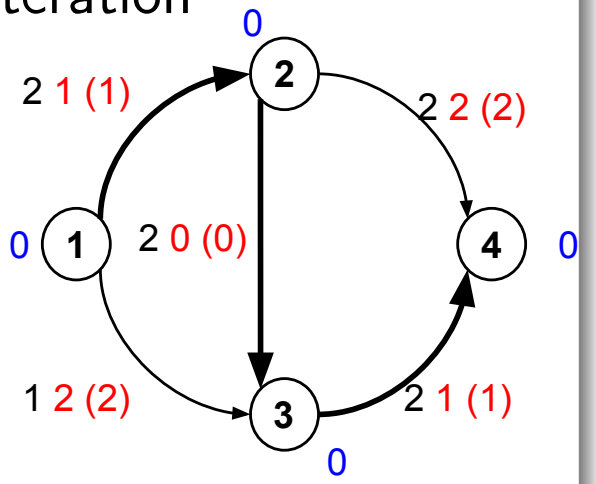
Example

Algorithm's Execution on Residual Graph

- Iteration 1 : 3 units left to be sent, do another iteration







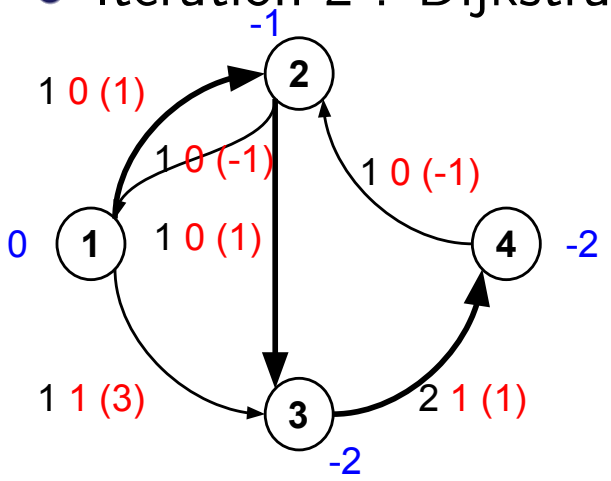
Algorithm's Result

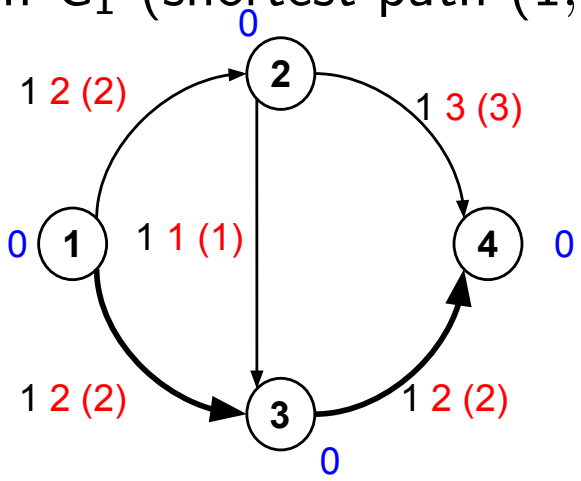
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

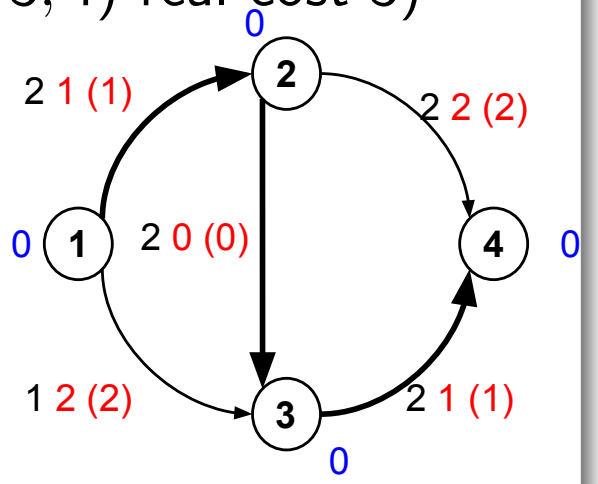
Example

Algorithm's Execution on Residual Graph

- Iteration 2 : Dijkstra on G_1 (shortest path $(1, 2, 3, 4)$ real cost 3)







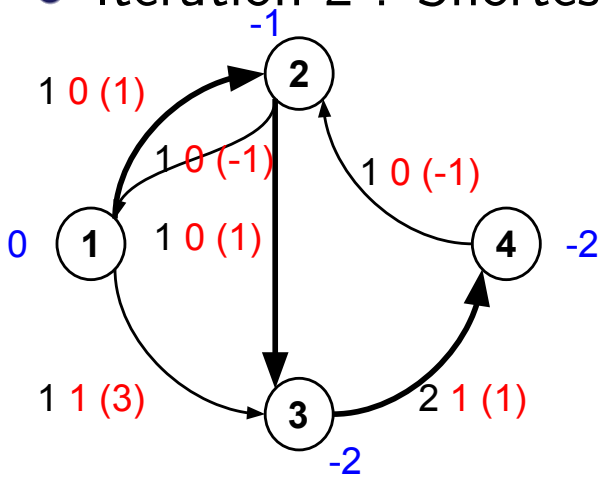
Algorithm's Result

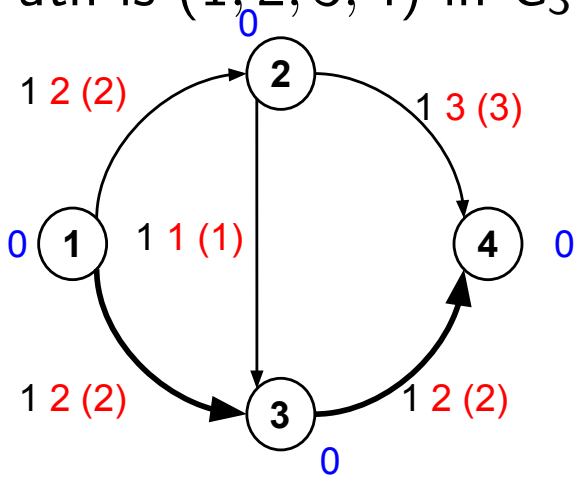
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

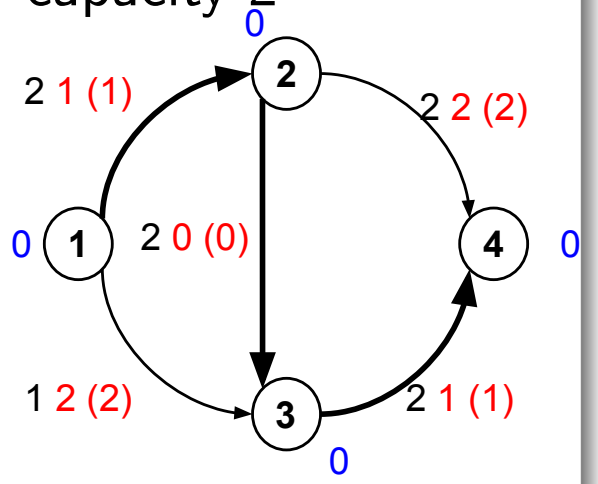
Example

Algorithm's Execution on Residual Graph

- Iteration 2 : Shortest Path is $(1, 2, 3, 4)$ in G_3 of capacity 2







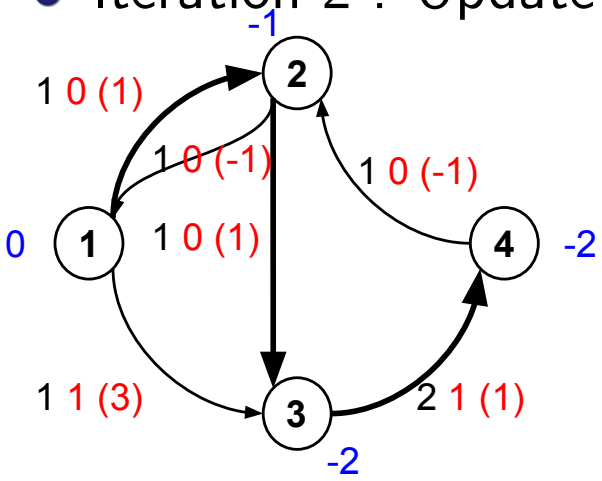
Algorithm's Result

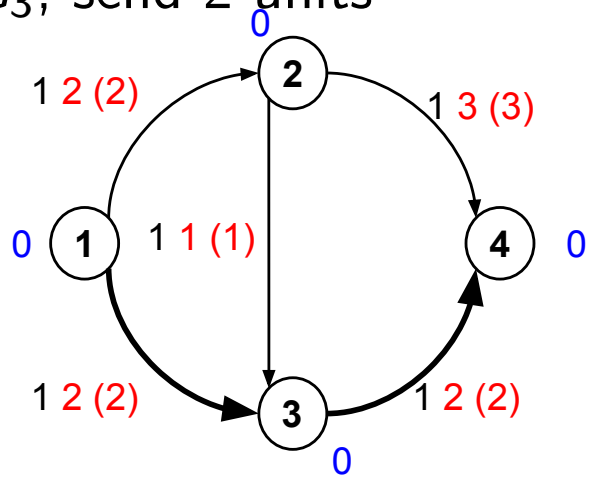
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

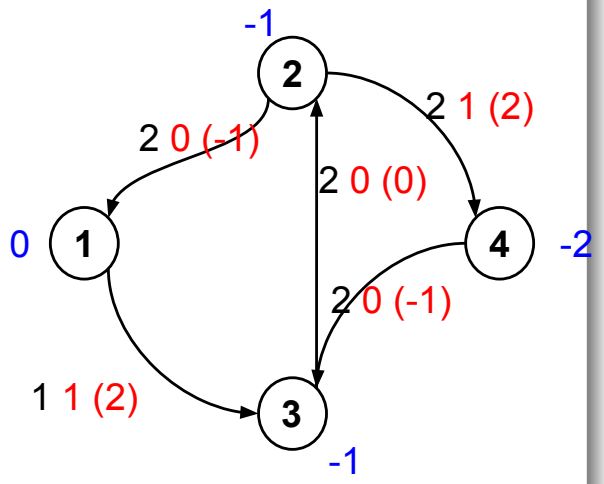
Example

Algorithm's Execution on Residual Graph

- Iteration 2 : Update G_3, send 2 units







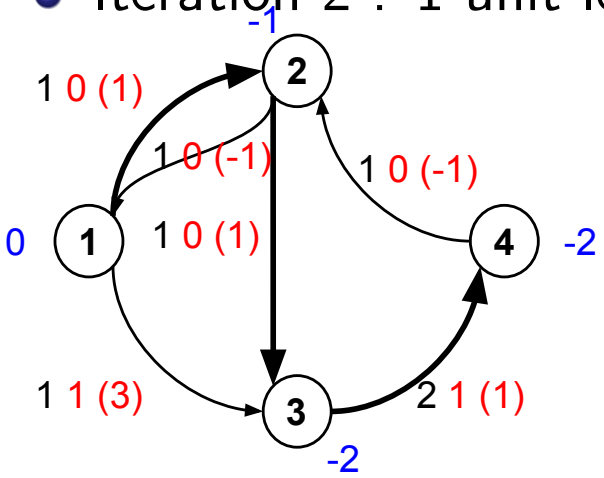
Algorithm's Result

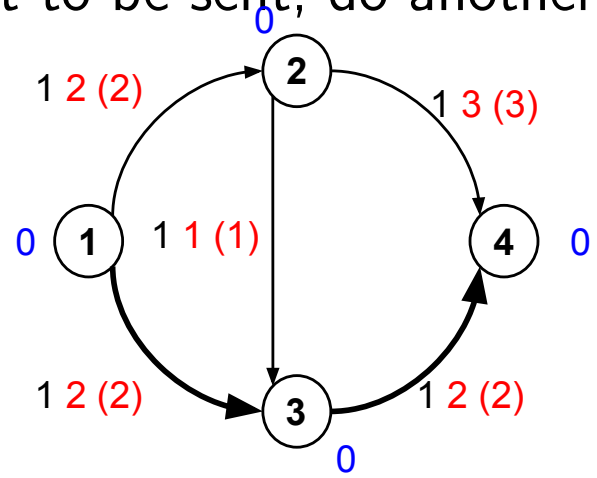
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

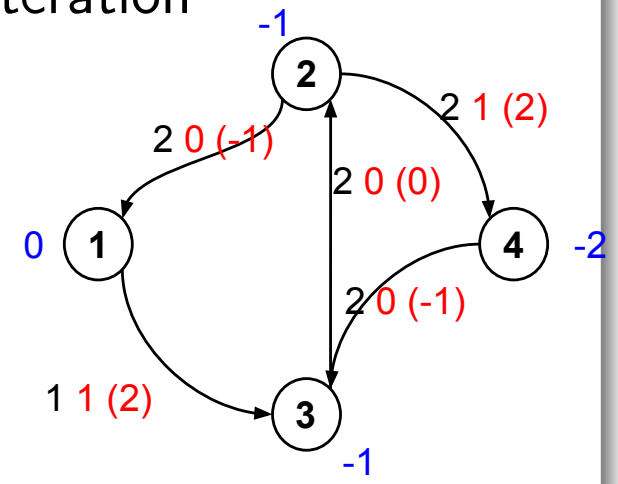
Example

Algorithm's Execution on Residual Graph

- Iteration 2 : 1 unit left to be sent, do another iteration







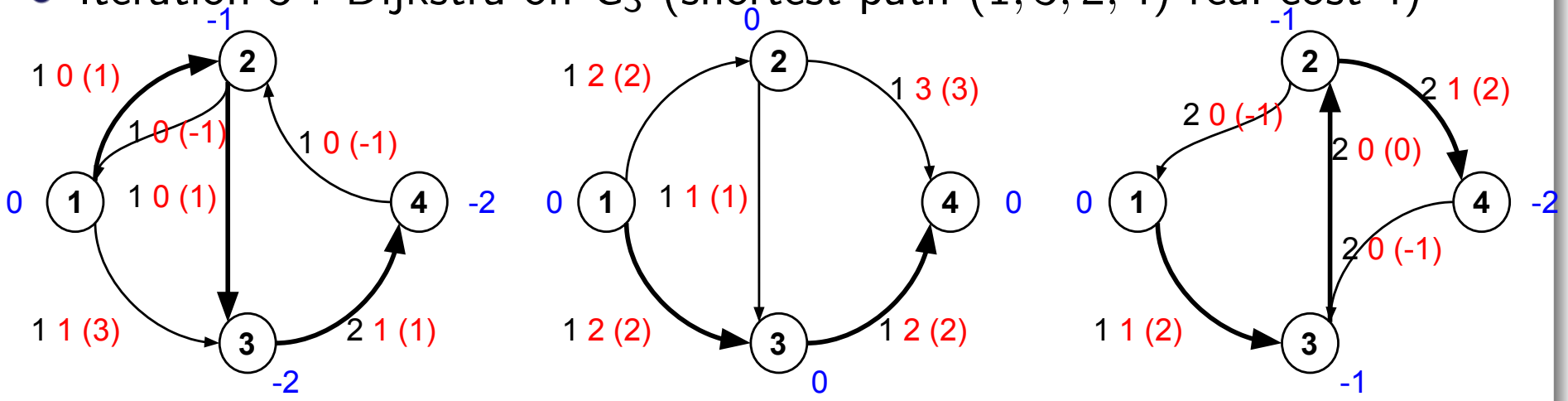
Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Example

Algorithm's Execution on Residual Graph

- Iteration 3 : Dijkstra on G_3 (shortest path $(1, 3, 2, 4)$ real cost 4)



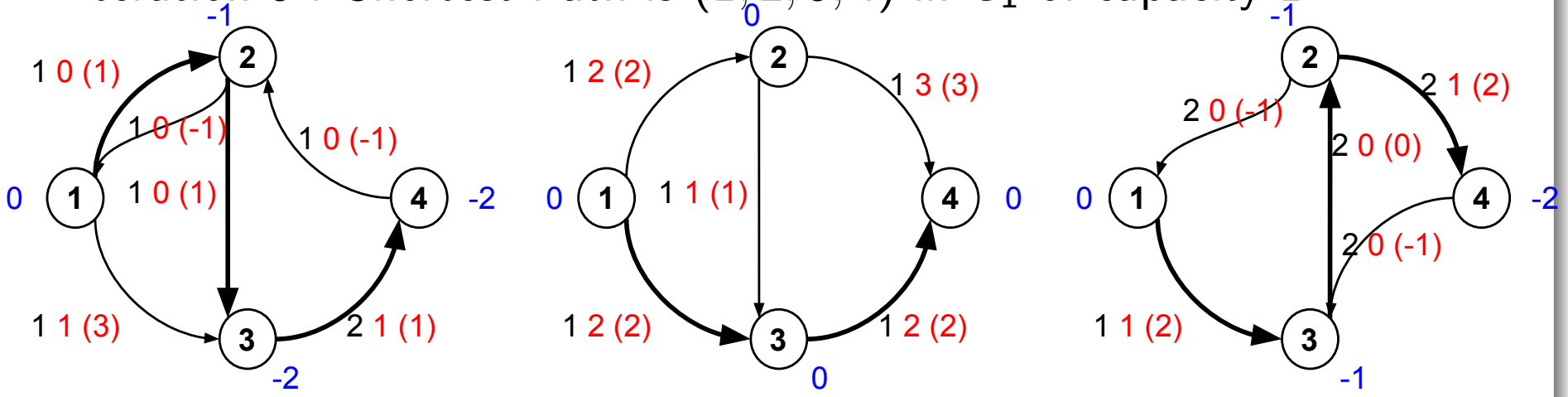
Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Example

Algorithm's Execution on Residual Graph

- Iteration 3 : Shortest Path is $(1, 2, 3, 4)$ in G_1 of capacity 1



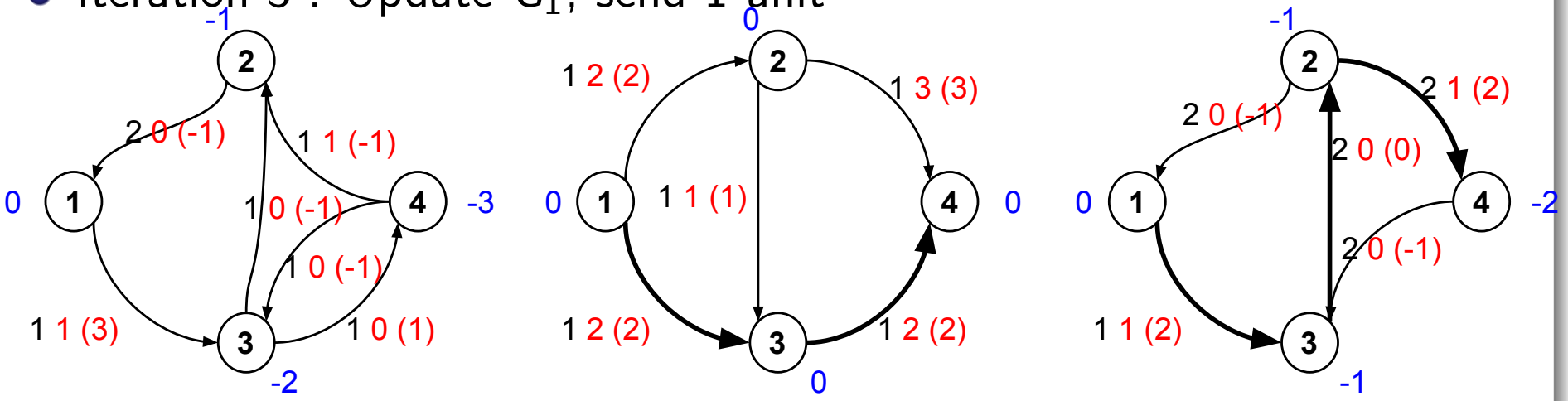
Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Example

Algorithm's Execution on Residual Graph

- Iteration 3 : Update G_1, send 1 unit



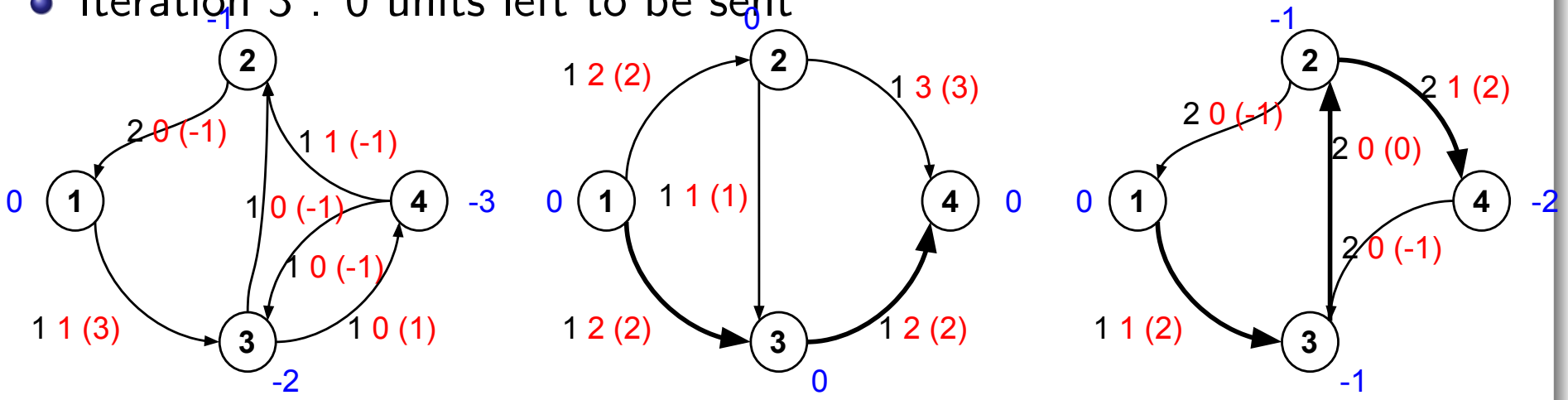
Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Example

Algorithm's Execution on Residual Graph

- Iteration 3 : 0 units left to be sent



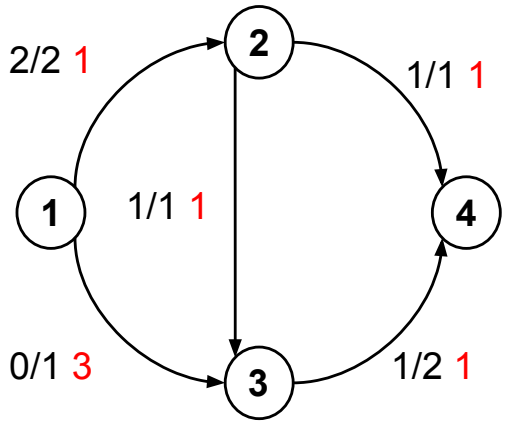
Algorithm's Result

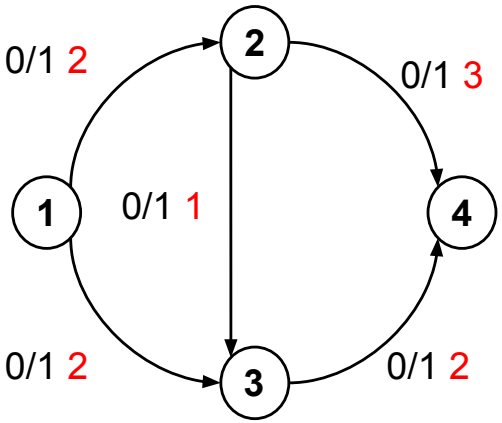
- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

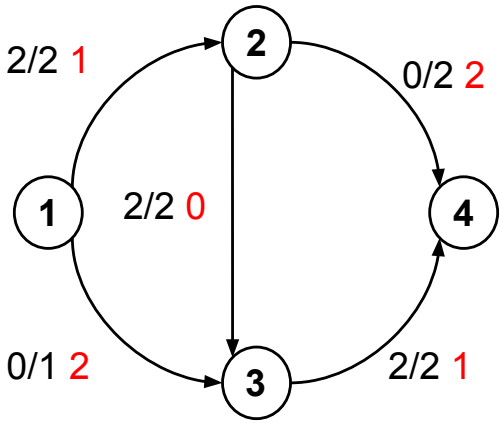
Example

Algorithm's Execution on Residual Graph

- End : 4 units sent with cost 9







Algorithm's Result

- Objective : $U = 4$
- Iteration 1 : Send 1 unit on path $(1, 2, 4)$ in G_1 at cost 2
- Iteration 2 : Send 2 units on path $(1, 2, 3, 4)$ in G_3 at cost 4
- Iteration 3 : Send 1 unit on path $(1, 2, 3, 4)$ in G_1 at cost 3
- End : 4 units sent with cost 9

Complexity

Outline of the Algorithm

1. Initialization : one iteration of SSP on each t-graph G_t
   $\boldsymbol{T \cdot (m + n \cdot log(n))}$
2. Iteration : one iteration of SSP on the best t-graph G_t
   $\boldsymbol{m + n \cdot log(n) + log(T)}$
3. Back to 2 if there are flow units left to be sent
   At most $\boldsymbol{U}$ times

Complexity

- $\boldsymbol{O((U + T) \cdot (m + n \cdot log(n)) + U \cdot log(T))}$

Varying Number of Time Steps

Random Graph

- 500 vertices
- $\approx 1\%$ density
- $\boldsymbol{T}$ **time steps** $(10 \leq \boldsymbol{T} \leq 1000)$
- Randomly varying capacities and costs

On Time-Expanded (EXP)

- $U \cdot T \cdot (m + n \cdot log(n \cdot T))$
- U linear in T, n fixed, hence m fixed

$\Rightarrow \boldsymbol{T^2 \cdot log(T)}$

On Temporal (TEMP)

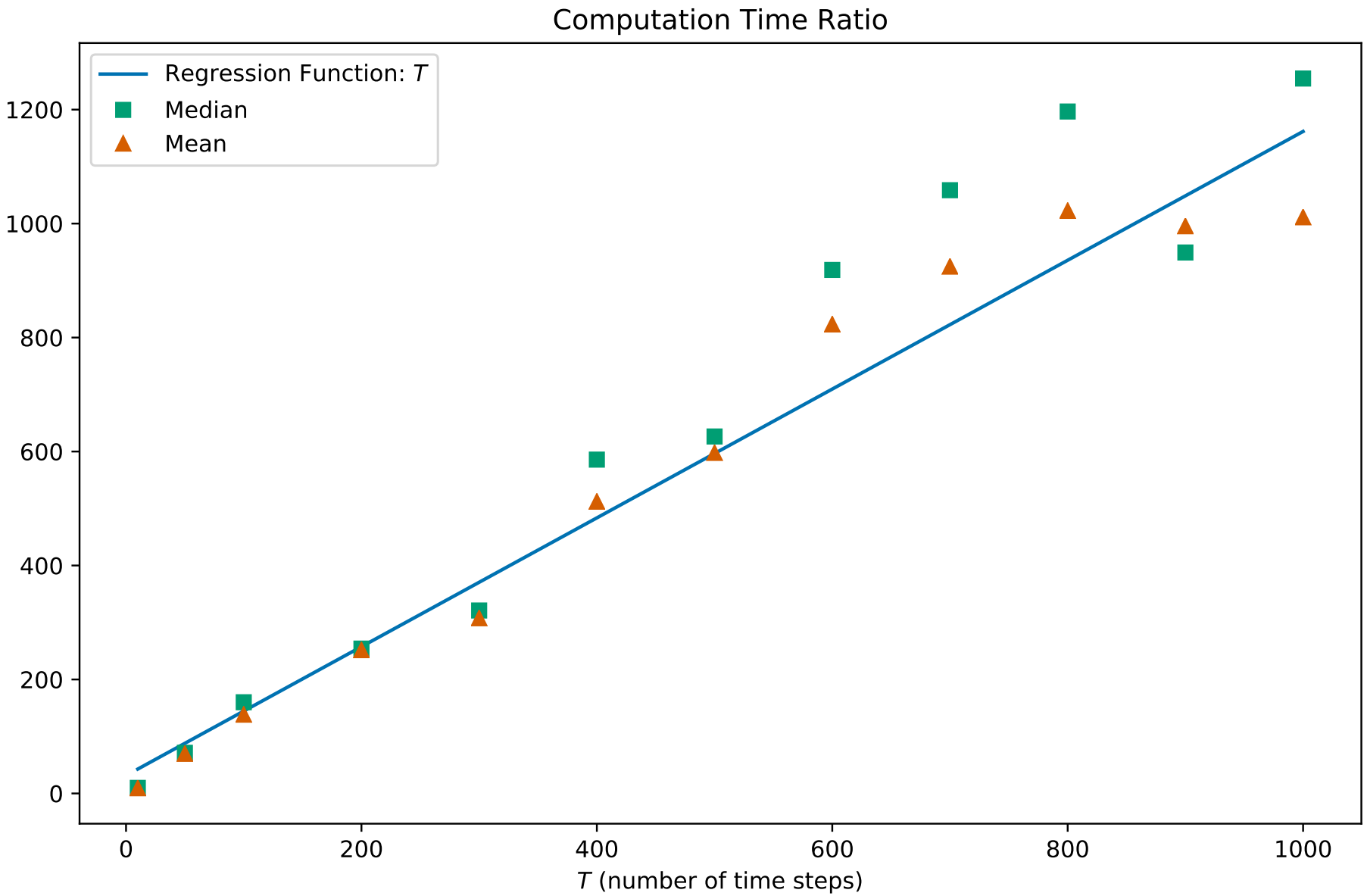
- $(U + T) \cdot (m + n \cdot log(n)) + U \cdot log(T)$
- U linear in T, n fixed, hence m fixed

$\Rightarrow \boldsymbol{T \cdot log(T)}$

Ratio

- $\frac{EXP}{TEMP} \sim \frac{T^2 \cdot log(T)}{T \cdot log(T)} \sim \mathbf{T}$

Varying Number of Time Steps



Varying Number of Vertices

Random Graph

- $\boldsymbol{n}$ **vertices** ($500 \leq \boldsymbol{n} \leq 2000$)
- $\approx 1\%$ density
- $T = 100$ time steps
- Randomly varying capacities and costs

On Time-Expanded (EXP)

- $U \cdot T \cdot (m + n \cdot log(n \cdot T))$
- U linear in n, T fixed, $m \sim n^2$

$\Rightarrow \boldsymbol{n^3}$

On Temporal (TEMP)

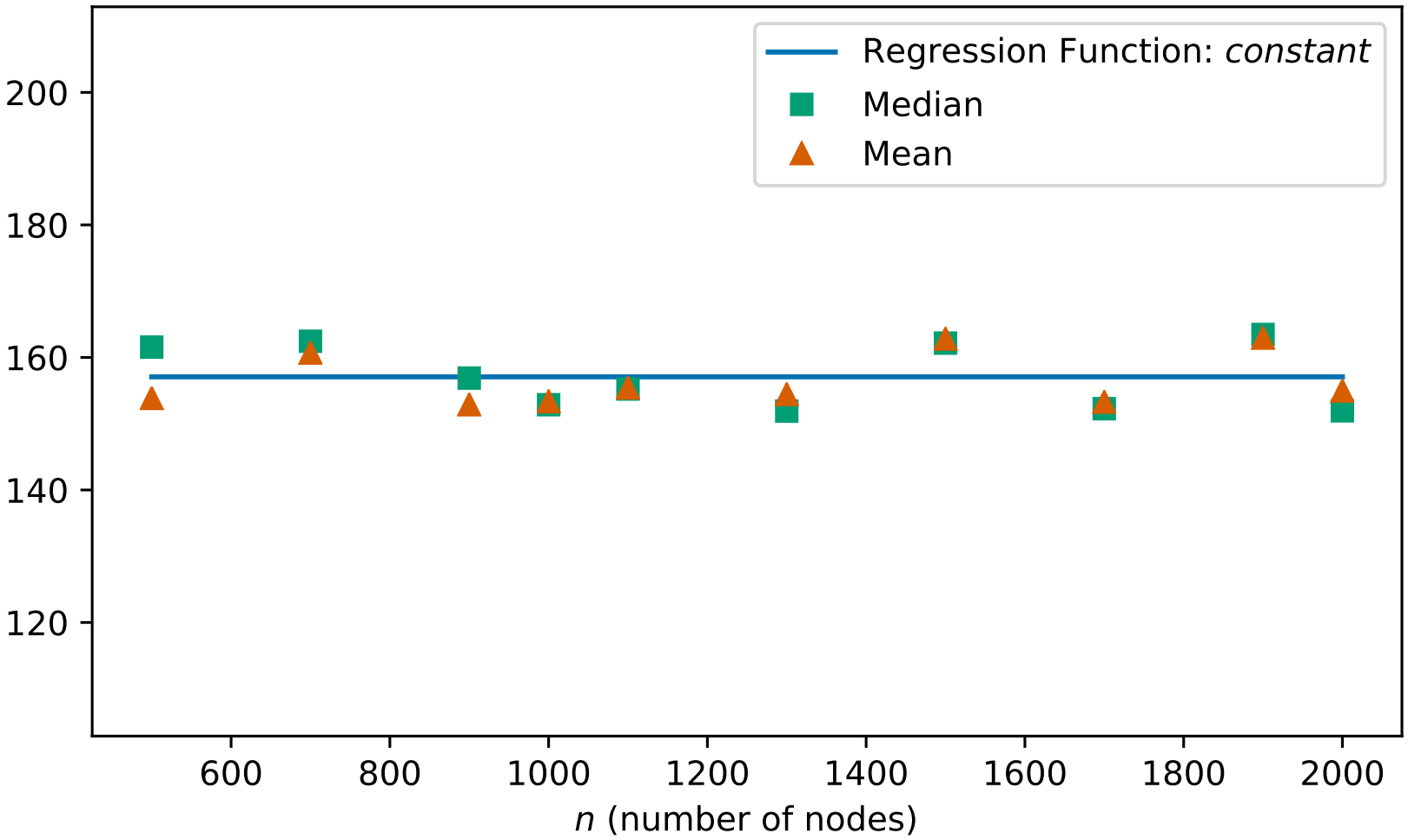
- $(U + T) \cdot (m + n \cdot log(n)) + U \cdot log(T)$
- U linear in n, T fixed, $m \sim n^2$

$\Rightarrow \boldsymbol{n^3}$

Ratio

- $\frac{EXP}{TEMP} \sim \frac{n^3}{n^3} \sim \mathbf{1}$

Varying Number of Vertices



Graph model

- Discrete Time $\mathcal{T}=\{1,\dots,T\}$
- Temporal Graph $\mathcal{G} = G_1,\dots,G_T$
- Each snapshot $G_\theta = (V, E_\theta)$
- Each time-arc $e = uv_\theta \in E_\theta$ with $u, v \in V$ has a capacity $u_\theta(uv)$
- Source vertex s and sink vertex t

Maximum Flow Problem

- Send the maximum amount of flow from s to t on $\mathcal{T}$
- No travel time
- Infinite and free storage

Example

- Unit capacities as a capacity vector

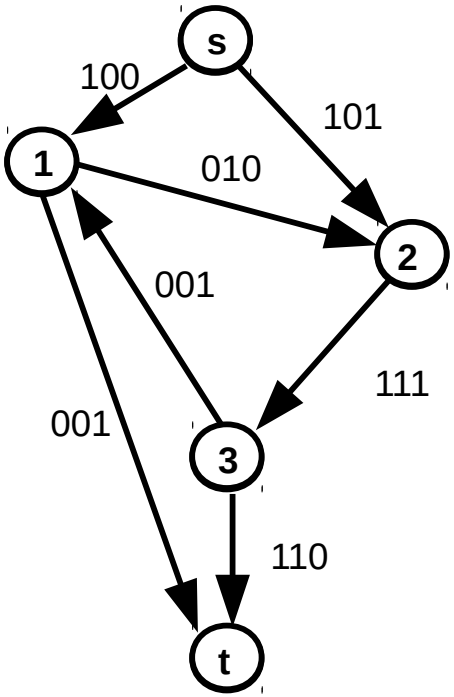


Figure: Compact Representation

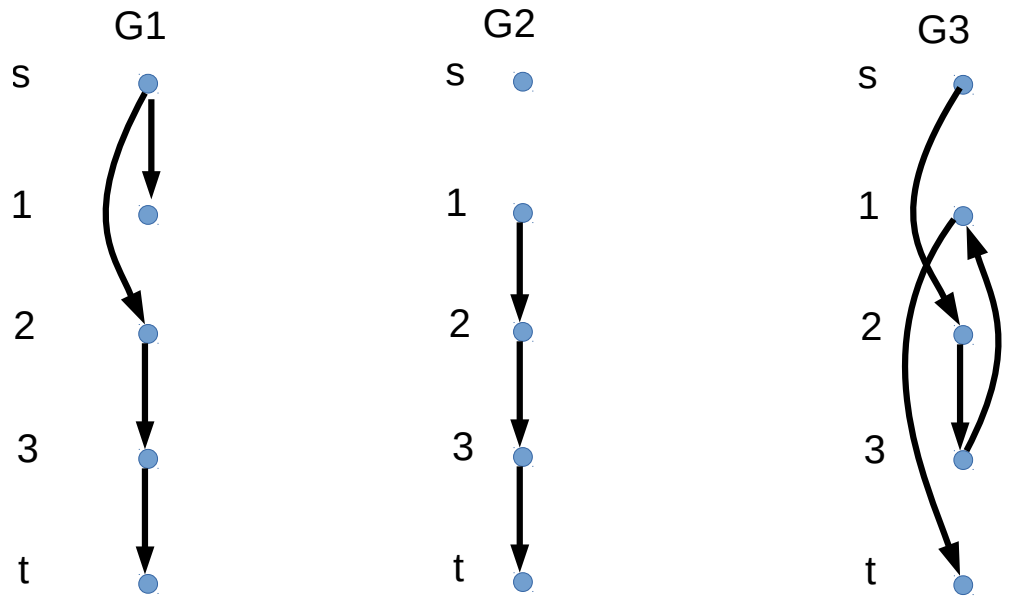


Figure: Sequence of snapshots

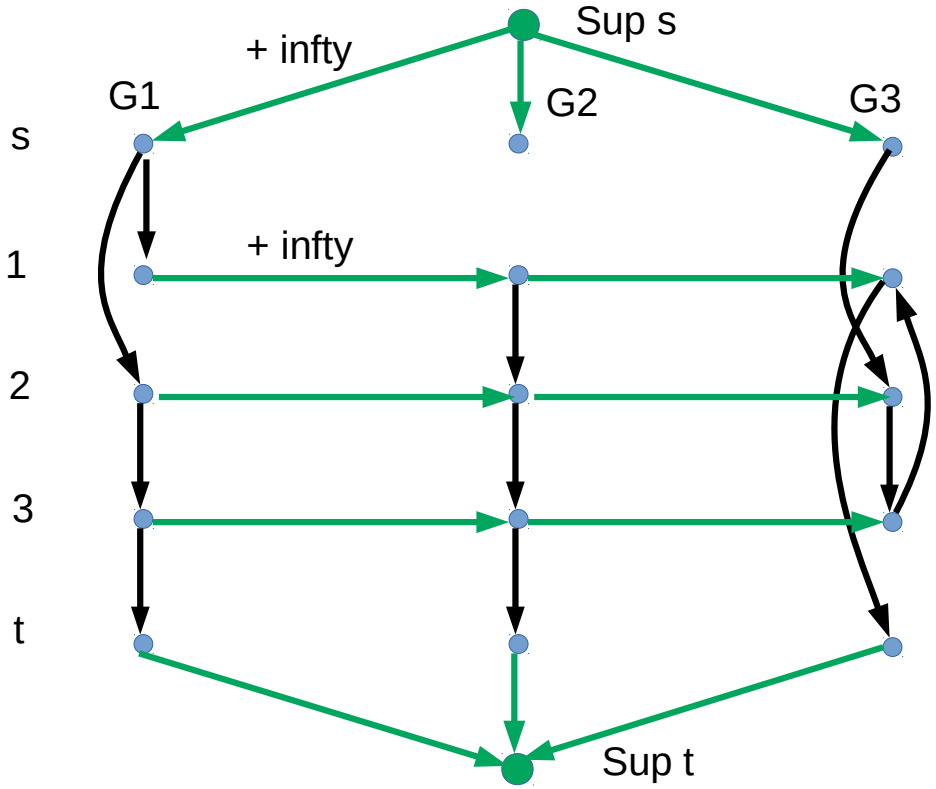


Figure: Time-expanded Graph

Time-expanded Graph

- A super-source and a super-sink are added
- storage arcs are added as horizontal links, with infinite capacity

Size

- Number of vertices: $n \cdot T + 2 = \Theta(n \cdot T)$
- Number of arcs: $m \cdot T + (n-2) \cdot T + 2 \cdot T = \Theta(m \cdot T)$ if $n = O(m)$

Efficient algorithms for maximum flows on static graphs

- Based on two ideas
  - SSP: compute a sequence of augmenting paths in the residual graph
  - PF: compute Pre-Flows, try to remove excess by local operations

Complexity of various algorithms

Algorithm	*Static complexity*	*"Expanded" complexity*
Edmonds Karp (SSP)	$n^2 m$	$n^2 m T^3$
Generic pre flow	$n^2 m$	$n^2 m T^3$
PF, FIFO order	n^3	$n^3 T^3$
PF, *Highest distance order*	$n^2\sqrt{m}$	$n^2\sqrt{m} T^2\sqrt{T}$

Remarks

- Most known algorithms have a T^3 factor on the time-expanded graph
- Except for the Highest Label (distance to the sink) first implementation of Pre-Flow: $T^2\sqrt{T}$

Can we do better than $T^{5/2}$?

- Ideas
  - Better use of the time-expanded graph?
  - Use capacity vectors on the compact representation?
- Wrong ideas
  - No direct result with structural consideration on expanded graph
  - Examples can be built, for which augmenting paths are arbitrary long
  - No optimum algorithm (so far) keeping compact representation

Idea

- Considering an arc:
  - How much flow can get here?
  - How much flow can be sent from here?
  - Sum capacities

Remark

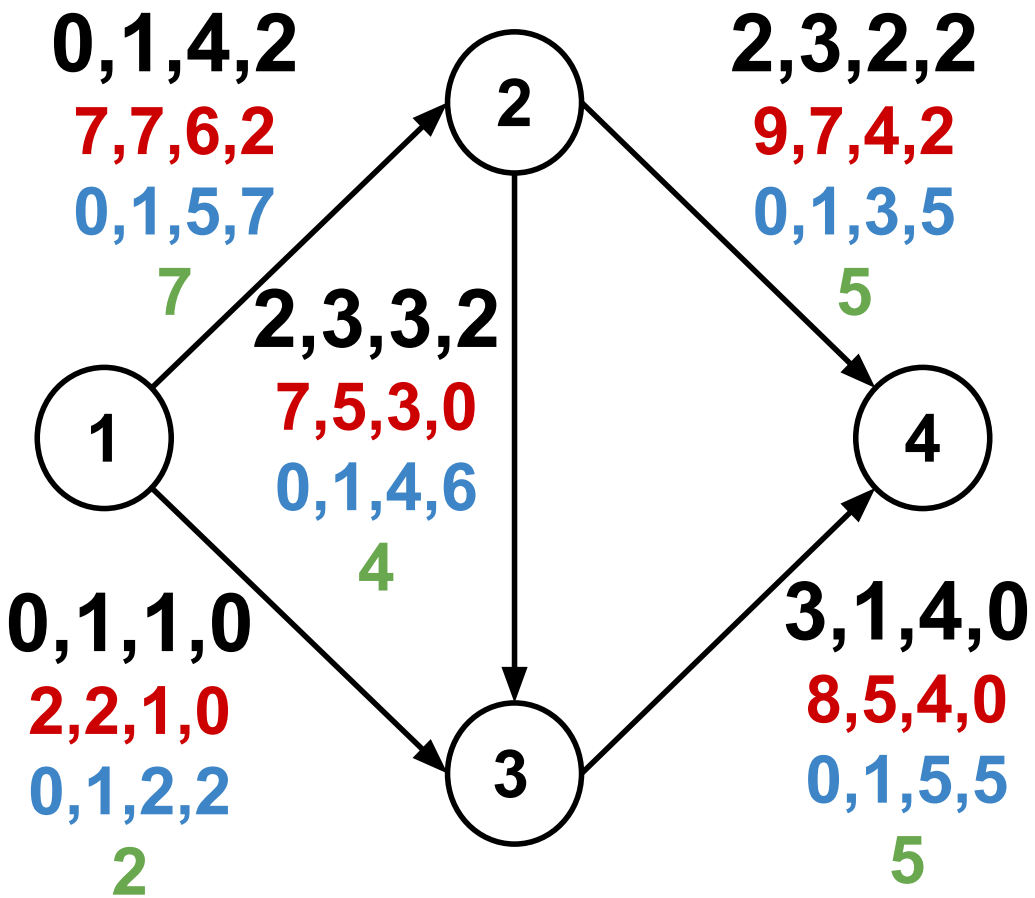
- Using what happens before and what happens after means we have to consider DAGs

Upper Bound on Arc Capacity

- Capacities
- Backward Useful Cumulated Capacities
- Forward Useful Cumulated Capacities
- Upper Bound on global flow on each arc
- Preprocessing $O(m \cdot T)$

Upper Bound on max flow

- Use upper bound on capacities
- Compute static flow
- Total cost $O(m \cdot T + m + n \cdot \log(n))$



Concluding remarks

- Many different flow problems on temporal graphs
- Time-expanded graph is a safe method but unfortunately costly
- Need to develop specific algorithm for the temporal case
- Sometimes we succeed *(Minimum cost flow)*
- Sometimes we fail *(Maximum flow)*
- Still many open questions on many problems and whether it is possible or not to improve the complexity

Thank you for your attention

Flows in Temporal Graphs: Solving Methods and Open Questions

Mathilde Vernet
{firstname.lastname}@univ-avignon.fr

LIA, Avignon Université

Poitiers
November 25, 2022