UNIVERSITY OF LIVERPOOL

School of Computer Science and Informatics

Optimal Sequential Flows

Hugo Gimbert Corto Mascle Patrick Totzke

ICALP-a, 2026

From max-flow to a scheduling problem

- Classical max-flow: one network, one assignment of capacities.
- Here we are given a finite set A of capacity assignments.
- Choosing a word $a_1 \cdots a_\ell \in A^*$ creates a "pipeline" graph with ℓ time layers.
- We want the supremum flow from source s to target t over *all* words and *all* times.

From max-flow to a scheduling problem

- Classical max-flow: one network, one assignment of capacities.
- Here we are given a finite set A of capacity assignments.
- Choosing a word $a_1 \cdots a_\ell \in A^*$ creates a "pipeline" graph with ℓ time layers.
- We want the supremum flow from source s to target t over *all* words and *all* times.

Motivation:
This has links to logic and temporal graphs, but our motivation comes from parameterised verification and distributed computing; see
H. Gimbert, C. Mascle, and P. Totzke. "Optimally Controlling a Random Population". In: *ICALP-b*. 2026

Classical max-flow

Fix a directed graph on vertices V with source s and target t, and a capacity assignment $a \in (\mathbb{N} \cup \{\omega\})^{V \times V}$.

A flow $f \in \mathbb{R}_{\geq 0}^{V \times V}$ satisfies

$$0 \leq f(v, v') \leq a(v, v')$$

for all edges (v, v'), and for every inner vertex $v \notin \{s, t\}$:

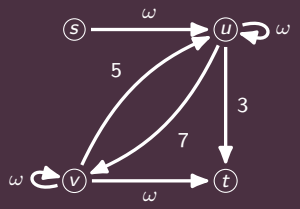
$$\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u).$$

Objective

Maximize the value

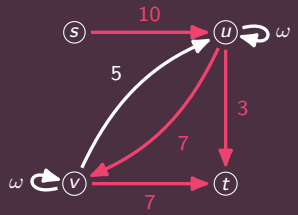
$$|f| = \sum_{u \in V} f(s, u) = \sum_{u \in V} f(u, t).$$

Running Example



Capacity ω means no bound; edges not shown have capacity 0.

Running Example



Capacity ω means no bound; edges not shown have capacity 0.
A maximal flow from s to t is $f(s,u) = 10$, $f(u,v) = f(v,t) = 7$, $f(u,t) = 3$

Sequential flow

Fix a directed graph on vertices V and a finite set $A \subseteq (\mathbb{N} \cup \{\omega\})^{V \times V}$.

A sequential flow consists of

$$f = f_1 f_2 \cdots f_\ell \qquad \text{witnessed by a word} \qquad a_1 a_2 \cdots a_\ell \in A^*,$$

such that for every $1 \leq i \leq \ell$:

$$0 \leq f_i(v, v') \leq a_i(v, v')$$

and for every inner vertex $v \notin \{s, t\}$:

$$\sum_{u \in V} f_i(u, v) = \sum_{u \in V} f_{i+1}(v, u).$$

Objective

Compute

$$\sup\{|f| : f \text{ is a sequential flow over some } a_1 \cdots a_\ell \in A^*\}.$$

Sequential flow

Fix a directed graph on vertices V and a finite set $A \subseteq (\mathbb{N} \cup \{\omega\})^{V \times V}$.

A sequential flow consists of

$$f = f_1 f_2 \cdots f_\ell \qquad \text{witnessed by a word} \qquad a_1 a_2 \cdots a_\ell \in A^*,$$

such that for every $1 \leq i \leq \ell$:

$$0 \leq f_i(v, v') \leq a_i(v, v')$$

and for every inner vertex $v \notin \{s, t\}$:

$$\sum_{u \in V} f_i(u, v) = \sum_{u \in V} f_{i+1}(v, u).$$

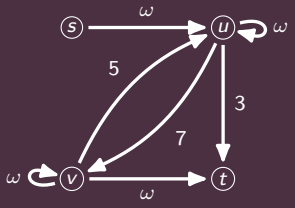
Objective

Compute

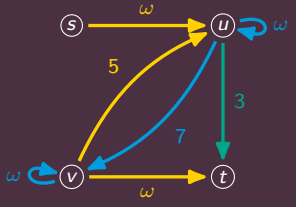
$$\sup\{|f| : f \text{ is a sequential flow over some } a_1 \cdots a_\ell \in A^*\}.$$

$|f| = \sum_{u \in V} f_1(s, u) = \sum_{u \in V} f_\ell(u, t)$

Running example

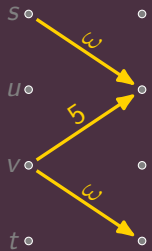


Running example

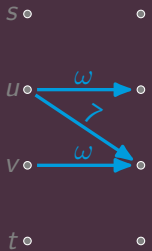


Running example

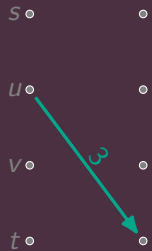
$a =$



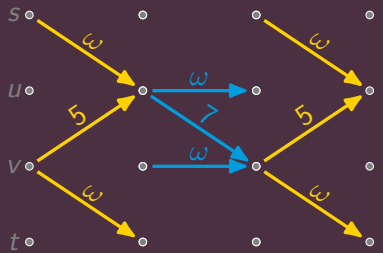
$b =$



$c =$

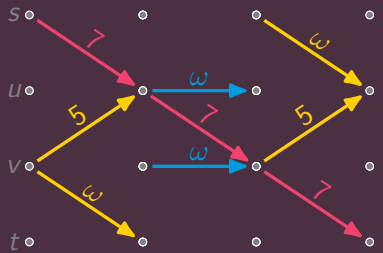


Running example



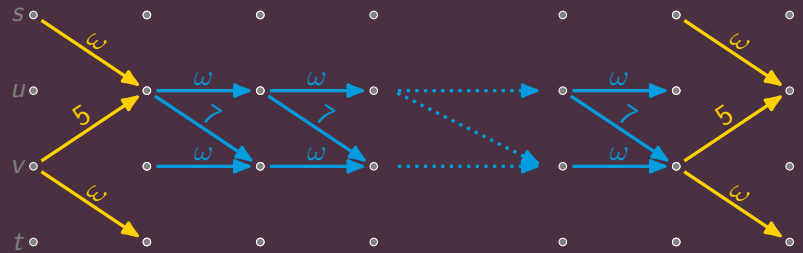
Capacity word $aba \in A^*$ has optimal value 7.

Running example



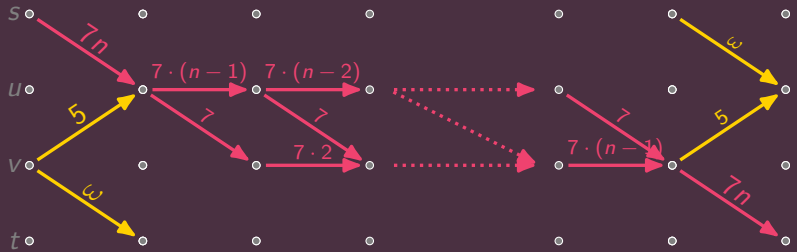
Capacity word $aba \in A^*$ has optimal value 7.

Running example



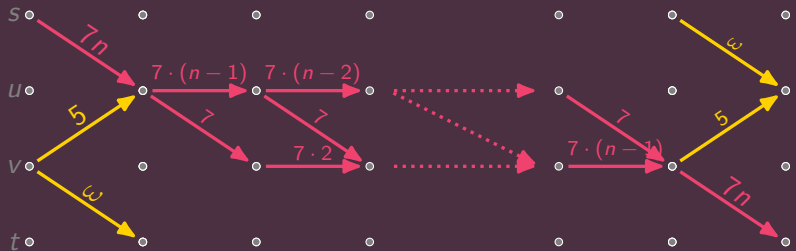
Capacity word $aba \in A^*$ has optimal value 7.
Capacity word $ab^n a \in A^*$ has optimal value $7n$.

Running example



Capacity word $aba \in A^*$ has optimal value 7.
Capacity word $ab^n a \in A^*$ has optimal value $7n$.

Running example



Capacity word $aba \in A^*$ has optimal value 7.
Capacity word $ab^n a \in A^*$ has optimal value $7n$.
The supremum sequential flow is unbounded (ω).

Our Result on Sequential Flows

Questions

1. Is the optimal sequential flow unbounded?
2. If it is finite, what is its exact value?

Main theorem

- Unboundedness is decidable in PSPACE.
- The exact optimal sequential flow is computable in PSPACE.
- This matches the known PSPACE-hardness lower bound [3].

Technical Contribution

An algebraic interpretation of sequential flows and a new factorisation theorem to represent (possibly long) capacity words succinctly.

A finite algebraic abstraction

Abstract each capacity assignment a by a matrix $x_a \in \{0, 1, \omega\}^{V \times V}$ where $0 =$ no capacity, $1 =$ finite positive capacity, $\omega =$ arbitrarily large capacity.

Two operations

- **Composition**: matrix product over $(\max, \min)$ semiring, to summarise effect of two consecutive pipeline segments.
- **Iteration of idempotents**: to capture finite capacity accumulating unboundedly.

Closing the abstract capacities under product and iteration gives the *flow semigroup* $\mathcal{F}$.

A finite algebraic abstraction

Abstract each capacity assignment a by a matrix $x_a \in \{0, 1, \omega\}^{V \times V}$ where $0 =$ no capacity, $1 =$ finite positive capacity, $\omega =$ arbitrarily large capacity.

Two operations

- **Composition**: matrix product over $(\max, \min)$ semiring, to summarise effect of two consecutive pipeline segments.
- **Iteration of idempotents**: to capture finite capacity accumulating unboundedly.

Closing the abstract capacities under product and iteration gives the *flow semigroup* $\mathcal{F}$.

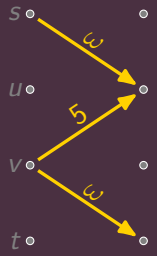
Checking Unboundedness

The sequential flow from s to t is unbounded iff some $x \in \mathcal{F}$ has $x(s, t) = \omega$.

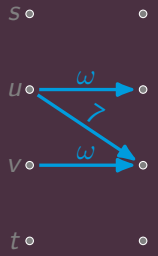
Flow Semigroup: Composition

. . . of capacities into 0, unbounded (ω), or positive and finite (1).

$a =$



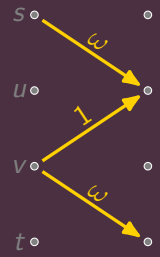
$b =$



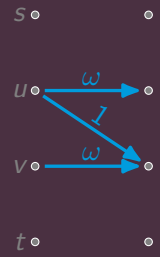
Flow Semigroup: Composition

. . . of capacities into 0, unbounded (ω), or positive and finite (1).

$x_a =$



$x_b =$

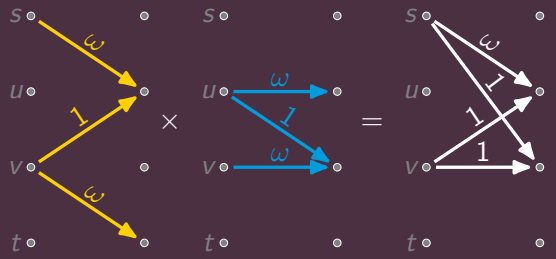


Flow Semigroup: Composition

... of capacities into 0, unbounded (ω), or positive and finite (1).
Composition is just (Matrix) multiplication in the max-min semiring.

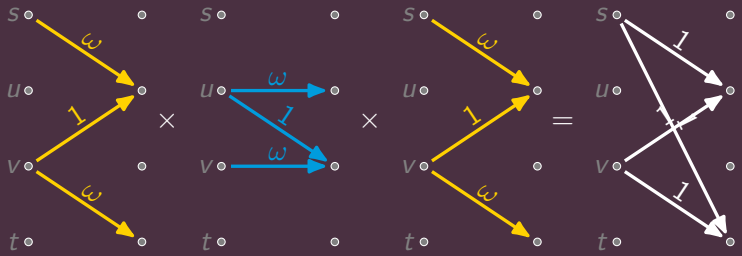
Flow Semigroup: Composition

... of capacities into 0, unbounded (ω), or positive and finite (1).
Composition is just (Matrix) multiplication in the max-min semiring.



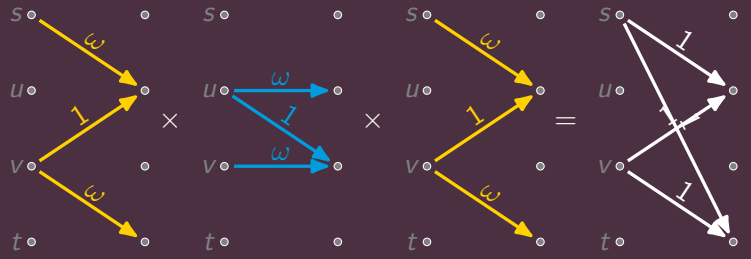
Flow Semigroup: Composition

... of capacities into 0, unbounded (ω), or positive and finite (1).
Composition is just (Matrix) multiplication in the max-min semiring.



Flow Semigroup: Composition

... of capacities into 0, unbounded (ω), or positive and finite (1).
Composition is just (Matrix) multiplication in the max-min semiring.



$$\begin{pmatrix} 0 & \omega & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \omega \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \omega & 1 & 0 \\ 0 & 0 & \omega & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & \omega & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & \omega \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Composition alone is not enough

$$x_b^2 = x_b \qquad \Longrightarrow \qquad x_a x_b^n x_a = x_a x_b x_a$$

$$x_a x_b x_a = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- The source-to-target entry is only 1 (positive but finite seq. flow is achievable)
- But the word $ab^n a$ has optimal value $7n$ (the supremum seq. flow is unbounded)

Composition alone is not enough

$$x_b^2 = x_b \qquad \Longrightarrow \qquad x_a x_b^n x_a = x_a x_b x_a$$

$$x_a x_b x_a = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- The source-to-target entry is only 1 (positive but finite seq. flow is achievable)
- But the word $ab^n a$ has optimal value $7n$ (the supremum seq. flow is unbounded)

Composition does not account for accumulating sequential flow that may be caused by repeating an idempotent !

Iteration

Idempotent stabilization

For an idempotent element $e = e^2$, the element $e^\sharp$ represents:

"use e as many times as needed".

Iterating an idempotent e can only have a few qualitative long-run behaviours:

- if $e(s,t)$ is 0 or ω it has no effect and we set $e^\sharp(s,t) = e(s,t)$;
- if $e(s,t)$ is 1 then the sequential flow is structurally bounded (set $e^\sharp(s,t) = 1$) or accumulates (set $e^\sharp(s,t) = \omega$).

Iteration

Idempotent stabilization

For an idempotent element $e = e^2$, the element $e^\sharp$ represents:

"use e as many times as needed".

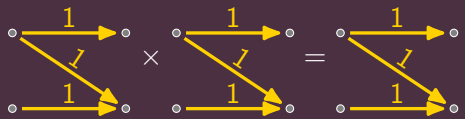
Iterating an idempotent e can only have a few qualitative long-run behaviours:

- if $e(s,t)$ is 0 or ω it has no effect and we set $e^\sharp(s,t) = e(s,t)$;
- if $e(s,t)$ is 1 then the sequential flow is structurally bounded (set $e^\sharp(s,t) = 1$) or accumulates (set $e^\sharp(s,t) = \omega$).

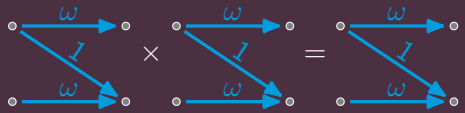
In our running example, $x_a x_b^\sharp x_a$ captures the unbounded behaviour of words $x_a x_b{}^n x_a$ and we will have that $x_a x_b^\sharp x_a(s,t) = \omega$.

Flow Semigroup: Iteration

Compare two idempotents:



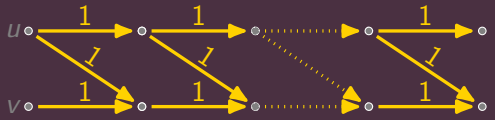
and



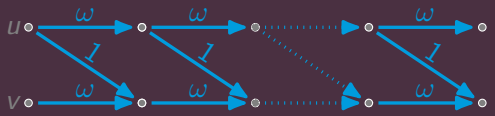
In both, the diagonal entry is 1 (finite and positive). But the sequential flow of iterating the first stabilises to 2 whereas the second accumulates unboundedly.

Flow Semigroup: Iteration

Compare two idempotents:



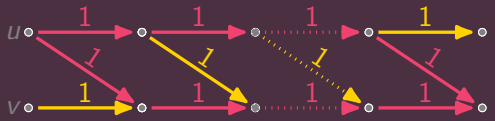
and



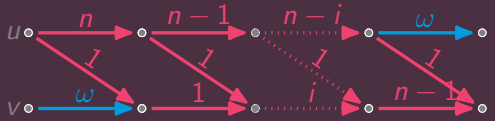
In both, the diagonal entry is 1 (finite and positive). But the sequential flow of iterating the first stabilises to 2 whereas the second accumulates unboundedly.

Flow Semigroup: Iteration

Compare two idempotents:



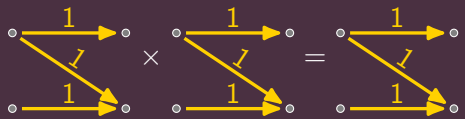
and



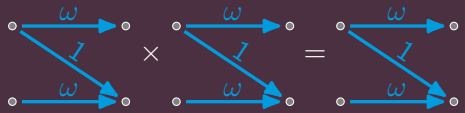
In both, the diagonal entry is 1 (finite and positive). But the sequential flow of iterating the first stabilises to 2 whereas the second accumulates unboundedly.

Flow Semigroup: Iteration

Compare two idempotents:



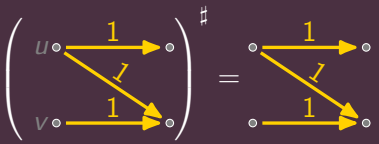
and



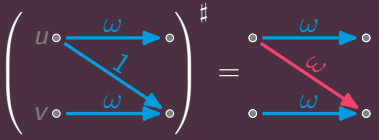
Iterating the first has no effect; iterating the second upgrades the diagonal from 1 to ω.

Flow Semigroup: Iteration

Compare two idempotents:



and



Iterating the first has no effect; iterating the second upgrades the diagonal from 1 to ω.

Small Witnesses for Supremum Sequential Flows

♯-Expressions

- Product nodes concatenate subwords.
- Idempotent / ♯ nodes compress long repeated regions.

Small Witnesses for Supremum Sequential Flows

$\sharp$-Expressions

- Product nodes concatenate subwords.
- Idempotent / $\sharp$ nodes compress long repeated regions.

Theorem
Every $x \in \mathcal{F}$ is generated by a $\sharp$-expression of height at most $2n^4$.

Small Witnesses for Supremum Sequential Flows

♯-Expressions

- Product nodes concatenate subwords.
- Idempotent / ♯ nodes compress long repeated regions.

Theorem
Every $x \in \mathcal{F}$ is generated by a ♯-expression of height at most $2n^4$.

Theorem
For every seq flow f there is an element $x \in \mathcal{F}$ so that for all $u, v \in V$

- *if $x(u,v) = 0$ then $f(u,v) = 0$, and*
- *if $x(u,v) = 1$ then $f(u,v) \leq ||A||_\infty n^{536 \cdot n^{12}}$.*

Computing Supremum Sequential Flows in PSPACE

Supremum sequential flows

- Every capacity word has a poly-bounded summary ($\sharp$-expression)
- Explore such expressions symbolically to find a witness for unboundedness or for finite seq. flow of $\geq k$;
- Binary search over that finite range yields the exact value.

Computing Supremum Sequential Flows in PSPACE

Supremum sequential flows

- Every capacity word has a poly-bounded summary ($\sharp$-expression)
- Explore such expressions symbolically to find a witness for unboundedness or for finite seq. flow of $\geq k$;
- Binary search over that finite range yields the exact value.

Extensions

- Regular-language constraints on admissible capacity words
- Fair simultaneous flow along several edges, and related multi-source variants

Thanks!

References

[1] I. Simon. "Factorization forests of finite height". In: *Theoretical Computer Science* (1990).

[2] E. C. Akrida et al. "Temporal flows in temporal networks". In: *Journal of Computer and System Sciences* 103 (2019), pp. 46–60.

[3] T. Colcombet, N. Fijalkow, and P. Ohlmann. "Controlling a random Population". In: *Logic. Meth. in Comput. Sci.* (2021).

[4] H. Gimbert, C. Mascle, and P. Totzke. "Optimally Controlling a Random Population". In: *ICALP-b*. 2026.

[5] R. Mandel, C. Mascle, and G. Zetzsche. *The complexity of downward closures of indexed languages*. 2026. arXiv: 2601.19466 [cs.FL].