Optimal Sequential Flows

Hugo Gimbert
CNRS, LaBRI, Université de Bordeaux, Talence, France

Corto Mascle
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Patrick Totzke
University of Liverpool, UK

Abstract

We provide a new algebraic technique to solve the sequential flow problem in polynomial space. The task is to maximise the flow through a graph where edge capacities can be changed over time by choosing a sequence of capacity labelings from a given finite set. Our method is based on a novel factorization theorem for finite semigroups that, applied to a suitable flow semigroup, allows to derive small witnesses. This generalises to multiple in/output vertices, as well as regular constraints.

2012 ACM Subject Classification Theory of computation → Network flows; Computing methodologies → Symbolic and algebraic algorithms; Computing methodologies → Algebraic algorithms; Theory of computation → Algebraic language theory

Keywords and phrases Network Flow, Sequential Flow, Semigroup Factorization

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.98

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2511.13806`

Funding *Patrick Totzke*: EPSRC, grant no.: EP/X042596/1.

1 Introduction

Determining the maximal flow through a network under capacity constraints is a classical optimisation problem [10]. The Sequential flow problem is a dynamic variant in which the labelling of edges by capacities is not static but can change over (discrete) time. We are given a finite set $A \subseteq (\mathbb{N} \cup \{\omega\})^{V \times V}$ of which each element $a \in A$ prescribes capacities for every edge in the directed graph[1]. Every length-ℓ capacity word $a_1 a_2 \cdots a_\ell \in A^*$ uniquely determines a *pipeline*, a graph of size $(\ell+1) \cdot |V|$ together with edge capacities where at time $1 \leq i \leq \ell$, the edge $v \to v'$ has capacity $a_i(v, v')$. The Sequential flow problem asks to determine the supremum of flow values through any such pipeline.

Notice that there is no limit on the length ℓ of the capacity words and therefore, the optimal sequential flow can be unbounded even if there is no finite word witnessing this.

▶ **Example 1.** Consider the graph with vertices $V = \{v_1, v_2, v_3, v_4\}$, with source $v_s = v_1$ and target $v_t = v_4$, and capacity constraints $A = \{a, b\}$ as depicted on the left in Figure 1. In both capacities a and b, there is no path from the source v_1 to the target v_4. It is therefore necessary to combine them sequentially in order to enable positive flow from v_1 to v_4. This can be achieved using the capacity word $abba$, which has a maximal flow value 2.

Even more flow can be transported from v_1 to v_4 through longer pipelines. For every $n > 0$ the pipeline for capacity word $ab^n a$ has a flow of (maximal) value n, as depicted in Figure 2. The *sequential* flow for A is therefore unbounded.

[1] A capacity / flow value ω means unbounded, i.e., finite but arbitrarily large.

53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026).
Editors: Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis;
Article No. 98; pp. 98:1–98:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

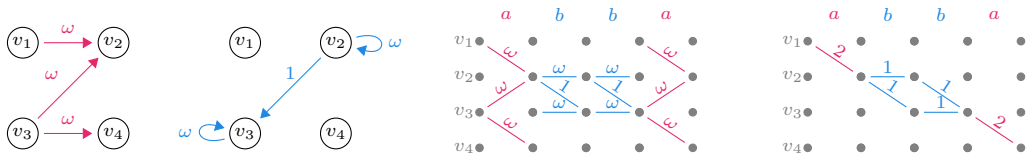


Figure 1 Two capacity constraints, a pipeline, and an optimal flow through it with value 2.

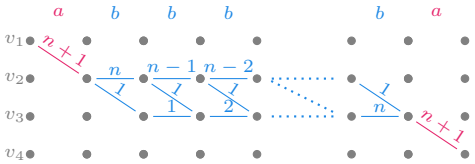


Figure 2 A flow of value $n+1$ through the pipeline $ab^{n+1}a$.

Background. Early works on *dynamic flows* consider flows in directed networks where edges have fixed capacities as well as transit times, and where associated costs are minimised [9, 2].

Closest to our setting, Akrida et al. [1] compute maximal flows through temporal networks [15, 16], where the edge capacities are a function of (discrete and bounded) time, i.e., a fixed capacity word. They devise a polynomial-time algorithm to compute the maximal *temporal* flow under such constraints, and prove a temporal version of the max-flow min-cut theorem.

In contrast to these works, in the Sequential flow problem neither the time horizon nor the capacities at given times are fixed. Instead, akin to planning or scheduling problems, one can choose the capacity word to maximise the flow through the network. Sequential flows were introduced in [5, Section 3] in the context of distributed computing. They have applications for controlling populations of Markov Decision Processes [6, 11] and Logics: it can be observed that the *commutative lossy tiling problem* defined and shown decidable in [3] is inter-reducible with the Sequential flow problem. Colcombet et al. showed that it is PSPACE-hard [6, Section 7], and decidable in exponential space [6, Theorem 5.1] whether the optimal sequential flow is unbounded[2]. The upper bound was achieved by an exponential reduction to the unboundedness problem of distance automata, for which a PSPACE upper bound is known [17].

Contributions. We provide a new, simple and optimal solution for the Sequential flow problem. We show how to compute the precise maximal sequential flow values in polynomial space (Theorem 22), thus matching the lower bound of [6] with an upper bound for a much more general problem. The approach in [6] is based on *regular cost functions*, a formalism used to express boundedness properties of functions from words to integers. They use a model of weighted automata adapted to this setting to prove their upper bound. There is however an exponential blow-up in the construction of this automata, leading to an exponential-space algorithm. We go through an algebraic approach instead, by exhibiting a suitable ♯-monoid (another way to represent regular cost functions), whose elements are matrices of dimension the number of vertices. While a ♯-monoid could be obtained from the automata in [6], the exponential blow-up would remain.

[2] This is called the "simple sequential flow problem" in [6].

Our technique is adapted to further generalisations: We derive the same upper bounds for the Sequential flow problem where sequential flows must be witnessed by capacity words from a given regular language (Theorem 37). This directly generalises the classical Max flow problem as well as the setting in Akrida et al. [1]. We also show how to solve a generalisation called the Fair sequential flow problem, where the flow should be routed equally along a set of given edges (Theorem 35).

Our contributions are based on new algebraic contributions of independent interest: In particular, we provide a factorization technique for general finite semigroups, where we show the existence of small *summaries* (Theorem 26). We also show the existence of $\sharp$-*summaries* of polynomial height for elements of the *flow semigroup*, a particular stabilization monoid [4, Section 3.1] used to witness unbounded sequential flows. This allows to obtain optimal upper bounds on the value of a finite solution to the Sequential flow problem (Theorem 16).

Missing proofs can be found in the long version of this paper.

2 Sequential Flows

We first recall the definition of flows and their optimisation problem.

Flows. Fix a finite set of vertices V and two distinct vertices v_s and v_t referred to as *source* and *target*, respectively. A *flow* $f \in \mathbb{R}^{V\times V}$ is a mapping of edges to non-negative reals which satisfies capacity– and flow conservation constraints as follows.

A *capacity constraint* $a \in (\mathbb{N} \cup \{\omega\})^{V\times V}$ assigns to each edge a capacity, respecting that all incoming edges to the source and from the target have capacity 0.

$$\forall (v, v') \in V^2, (v' = v_s \vee v = v_t) \implies a(v, v') = 0 \tag{1}$$

A flow f satisfies the capacity constraint a if

$$\forall v, v' \in V, \quad f(v, v') \leq a(v, v') \tag{2}$$

It satisfies the *flow conservation* constraints if

$$\forall v \in V \setminus \{v_s, v_t\}, \quad \mathrm{out}(f)(v) = \mathrm{in}(f)(v) \tag{3}$$

where $\mathrm{out}(f)(v) = \sum_{v'\in V} f(v, v')$ and $\mathrm{in}(f)(v) = \sum_{v'\in V} f(v', v)$.

Intuitively, a flow determines rate of goods flowing along each of the edges, and the capacity of an edge (v, v') is a predetermined bound on the admissible rate that can flow from v to v'. A capacity of $a(v, v') = 0$ means that nothing at all can flow, and a capacity of $a(v, v') = \omega$ means that an arbitrary finite amount can flow.

The *value* of a flow f is $|f| = \mathrm{out}(f)(v_s)$, the cumulative flow out of the source vertex. The Max flow problem asks to compute the maximal value of any flow.

> Max flow problem
> Given a capacity constraint $a \in (\mathbb{N} \cup \{\omega\})^{V\times V}$.
> Maximise $|f|$ under constraints in equations (2) and (3).

Due to the presence of edges with unbounded capacities and because every flow must have a finite value, there may not exist flows of maximal value.

Sequential flows. The SEQUENTIAL FLOW PROBLEM is a dynamic variant of this setting in which the maximiser gets to pick a fresh capacity constraint at any unit time. A sequential flow still represents the rate of goods flowing along the edges, but both capacity constraints and flow conservation dynamically reflect maximiser's momentary choice of capacities.

Assume a finite set $A \subseteq (\mathbb{N} \cup \{\omega\})^{V \times V}$ of capacity constraints. A sequence $f = f_1 f_2 \dots f_\ell \in \left(\mathbb{R}^{V \times V}\right)^*$ is a *sequential flow* if for all $0 < i \leq \ell$, both

$$\exists a_i \in A, \forall v, v' \in V, \quad f_i(v, v') \leq a_i(v, v') \quad \text{and} \tag{2'}$$

$$\forall v \in V \setminus \{v_s, v_t\}, \quad \text{in}(f_i)(v) = \text{out}(f_{i+1})(v). \tag{3'}$$

A sequential flow $f = f_1 f_2 \dots f_\ell$ dictates at least one *capacity word* $w = a_1 a_2 \dots a_\ell \in A^*$ that witnesses f satisfying the sequential capacity conditions (2′). We will refer to f as a sequential flow *over* capacity word w. The *value* of f is

$$|f| = \text{in}(f_\ell)(v_t) = \text{out}(f_1)(v_s),$$

the input flow to the target at the latest time ℓ and the output of the source at time 1. We want to optimise the supremum value of any sequential flow.

> SEQUENTIAL FLOW PROBLEM
> Given a finite set $A \subseteq (\mathbb{N} \cup \{\omega\})^{V \times V}$ of capacity constraints.
> Determine the *optimal sequential flow* optSeqFlow $= \sup\{|f| : f \text{ is a sequential flow}\}$.

Note that the SEQUENTIAL FLOW PROBLEM is *not* a linear program because ℓ is not bounded, and hence the number of constraints (2′) and (3′) is not bounded a priori.

There is a natural connection between the SEQUENTIAL FLOW PROBLEM and the MAX FLOW PROBLEM. Every capacity word $a_1 \dots a_\ell$ defines an instance of MAX FLOW PROBLEM in the corresponding pipeline, with source $(v_s, 0)$ and target (v_t, ℓ). This instance has a maximal flow, and optSeqFlow is the supremum of those maximal flows over all capacity words $a_1 \dots a_\ell$. However, there is no simple algorithmic reduction between these two optimisation problems. Indeed, for an instance of the SEQUENTIAL FLOW PROBLEM with capacities $A = \{c\}$, the optimal sequential flow value can be the same, strictly larger, or strictly greater than the maximal flow value if c is interpreted as an instance of the classical MAX FLOW PROBLEM.

▶ **Example 2.** Let $V = \{v_1, v_2, v_3, v_4\}$ and capacities c, d, and e as depicted in Figure 3. If c is considered as an instance of MAX FLOW then only 1 unit of flow can be transported from source $v_s = v_1$ to target $v_t = v_4$, using the edge (v_2, v_3) at its maximal capacity. The values of maximal flows through capacity words $c, cc, ccc, cccc$ are $0, 0, 1, 2$, respectively, and the max flow through any capacity word of the form $c^n, n > 4$ remains 2. The optimal sequential flow given set of capacities $A = \{c\}$ is therefore 2.

Consider now only capacity d. A flow with maximal value $|f| = 2$ from the source $v_s = v_1$ to the target $v_t = v_4$ is $f(v_1, v_3) = f(v_2, v_4) = f(v_1, v_2) = f(v_3, v_4) = 1$. Similarly, the sequential flow ff for capacity word $dd \in A^*$ has value $|ff| = 2$. However, every capacity word d^n of length $n \geq 3$ has maximal flow value 0. The optimal sequential flow given set of capacities $A = \{d\}$ is therefore 2.

Consider now only capacity e. The maximal value of a flow from v_1 to v_4 is 2 whereas the optimal sequential flow is only 1, because for any $n \geq 0$ there is at most one path of length n, and the minimal capacity along these paths is 1. The optimal sequential flow given set of capacities $A = \{e\}$ is therefore 1.

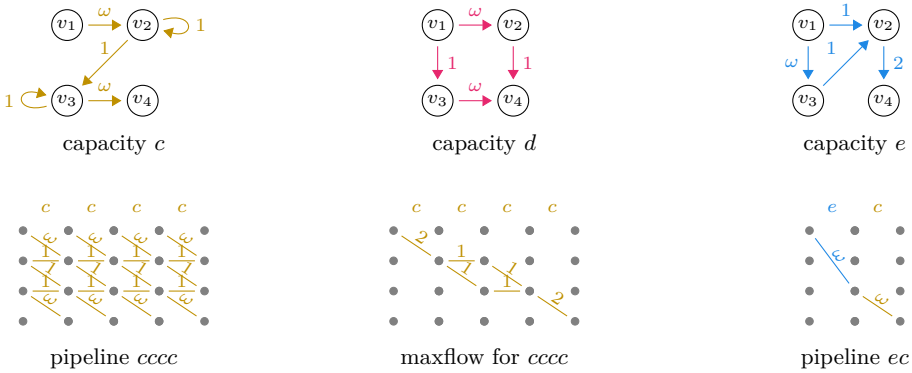
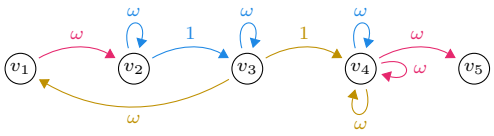


Figure 3 Capacities c, d, e from Example 2, pipeline $cccc$ and its maximal flow, and pipeline ec.

Finally, to demonstrate that combining different capacity letters may be required for the optimal sequential flow, consider the set of capacities $A = \{c, e\}$. The optimal sequential flow value ω can be witnessed by a single capacity word ec and a family $(f_n)_{n\in\mathbb{N}}$ of sequential flows defined as $f_n = f_{n,1}f_{n,2}$ with $f_{n,1}(v_1, v_3) = f_{n,2}(v_3, v_4) = n$ of value $|f_n| = n$.

As established in both examples so far, there may not exist sequential flows of maximal value because there are in fact sequential flows of arbitrarily high values. In that case we call the optimal sequential flow *unbounded* and write optSeqFlow $= \omega$. This can be witnessed in two ways: either by a single pipeline such as at the end of Example 2, or, as in Example 1, by a family of pipelines of growing length and maximal flow value. In Example 1, the optimal sequential flow value of ω cannot be witnessed by any finite pipeline and is instead witnessed by a family $ab^n a$ of words that iterate the capacity constraint b arbitrarily, but finitely often. Our final example demonstrates that such witnesses may require more complex nested iterations.

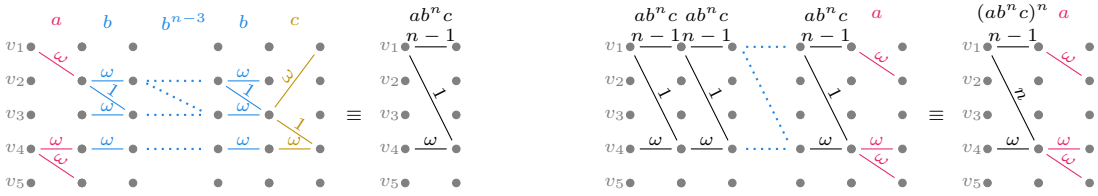
▶ **Example 3.** Take sets $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $A = \{a, b, c\}$ of vertices and capacity constraints as depicted below, where non-zero capacities of a, b, c are shown in red, blue, and yellow.



The optimal sequential flow from source $v_s = v_1$ to target $v_t = v_5$ is ω, yet no finite capacity word witnesses this. To witness a sequential flow of value n, a capacity word must be of the form $(ab^{\geq n}c)^{\geq n}a$. The combined capacities for the word $ab^n c$ is shown in Figure 4 (left). This allows a flow of n from v_1 to v_3 (using ab^n); then to transfer one unit to v_3 (using c, which empties v_3). Iterating this prefix n times allows a flow of n units to v_4, at which point all can flow in one step towards the target v_5 (via a, see the right half of Figure 4).

3 Solving the Sequential flow problem

We present a solution to the Sequential flow problem in two stages. The first stage, described in Section 3, is qualitative: we determine whether the instance is unbounded, i.e., whether there exists sequential flows of arbitrarily high value. The key for that is to abstract



pipeline ab^nc and its combined effect

pipeline for $(ab^nc)^na$

Figure 4 The pipelines from Example 3. The pipeline for ab^nc and its shortened representation (seen on the left) is iterated another n times in the pipeline for $(ab^nc)^na$ (seen on the right).

the exact computation of values by means of a finite algebraic structure called the *flow semigroup*. The elements of this semigroup are enumerated using Algorithm 2 (presented later), which searches for a witness of unboundedness. The second stage is quantitative: it is performed by Algorithm 1, which computes the maximal value of sequential flows, assuming they are bounded, by a simple binary search. Both stages can be carried out in polynomial space. A key component for this upper bound in the second stage is the proof that when sequential flows are bounded the supremum is at most exponential in the number of vertices.

Algorithm 1 Computing the optimal sequential flow.

$M \leftarrow K \cdot |V|^{536 \cdot |V|^{12}} + 1$
$m \leftarrow 0$
repeat
  $C \leftarrow \lceil (m + M)/2 \rceil$
  if there is a sequential flow f such that $|f| = C$ **then**
    $m \leftarrow C$
  else
    $M \leftarrow C$
until $M \leq m + 1$
return m

For the qualitative stage, we will abstract the exact values of capacity constraints and only consider whether those values are 0, finite or ω.

We make use of the *maxmin semiring*

$$\mathbb{M} = (\{0, 1, \omega\}, \max, \min) \text{ with } 0 < 1 < \omega$$

There is a natural structure of semigroup on $\mathbb{M}^{V \times V}$, using the usual matrix product over this semiring. For $x, y \in \mathbb{M}^{V \times V}$, and $v, v' \in V$,

$$(x \cdot y)(v, v') = \max_{v'' \in V} \left(\min \left(x\left(v, v''\right), y\left(v'', v'\right) \right) \right) \quad .$$

Every capacity constraint $a \in \mathbb{N}^{V \times V}$ is naturally abstracted as a matrix $x_a \in \mathbb{M}^{V \times V}$ by losing precision: 0 and ω are preserved while finite positive numbers are mapped to 1.

▶ **Example 4.** In Example 1 there are two capacity constraints a and b. Their abstractions, as well as the product thereof, are as follows:

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

Notice that $a = x_a$ and $b = x_b$ coincide with their abstractions since all finite coordinates are equal to 0 or 1.

The matrix product allows to keep track of which paths have finite or ω-capacity. For every $n \geq 1$, the product $x_a x_b^n x_a$ can be computed easily: since x_b is idempotent, i.e. $x_b^2 = x_b$, we have

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

This simple computation tells us that starting from the source v_s (the first line), any sequential flow through the pipeline $x_a x_b^n x_a$ can carry only a finite flow to the target v_t (the last column), because the top right coefficient is 1. However, this finite representation of the pipeline $x_a x_b^n x_a$ misses an important point: although the flow from v_s to v_t is finite for every $n > 0$, it is actually unbounded and grows with n (cf. Figure 2).

In order to take this phenomenon into account, we introduce an extra operation on idempotent elements, i.e. the elements $e \in \mathbb{M}^{V \times V}$ such that $e = e^2$. This operation computes a new idempotent element $e^\sharp \in \mathbb{M}^{V \times V}$. Intuitively, the element $e^\sharp$ is an abstraction of the sequence of pipelines $(e^n)_{n \in \mathbb{N}}$ and should be thought of as "using e many times".

Before we proceed to define this operation, let us observe that the possible effect that iterating idempotents can have on the maximal flow between any two vertices.

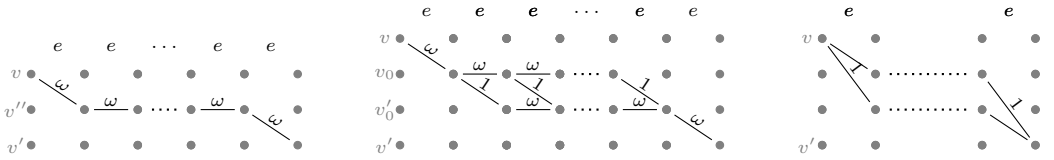


Figure 5 The figure illustrates Lemma 5, which classifies the three possible long-term behaviours of an edge (v, v') of an idempotent e, when $e(v, v') \neq 0$. Case i) is $e(v, v') = \omega$ and case ii) and iii) occur when $e(v, v') = 1$.

▶ **Lemma 5** (Flow-carrying Edges of idempotent elements). *Let $e \in \mathbb{M}^{V \times V}$ such that $e = e^2$, and $v, v' \in V$ such that $e(v, v') > 0$. For $n \geq 1$, let K_n denote the optimal flow value from v to v' in the pipeline e^n. Exactly one of the following holds.*

i) *$K_n = \omega$ for all $n \geq 1$. This holds iff $e(v, v') = \omega$ and there exists $v'' \in V$ such that*

$$\omega = e(v, v'') = e(v'', v'') = e(v'', v').$$

ii) *$n - 2 \leq K_n \leq n|V|$ for all $n \geq 1$. This holds iff $e(v, v') = 1$ and there exists $v_0, v_0' \in V$ such that*

$$e(v_0, v_0') = 1 \text{ and } \omega = e(v, v_0) = e(v_0, v_0) = e(v_0', v_0') = e(v_0', v').$$

iii) *$1 \leq K_n \leq 2|V|$ for all $n \geq 1$. This holds iff $e(v, v') = 1$ and for all $v_0, v_0' \in V$,*

$$e(v_0, v_0') \geq 1 \implies (e(v, v_0) \leq 1 \text{ or } e(v_0', v') \leq 1).$$

The following definition of iterations of idempotents explicitly distinguishes the three cases of Lemma 5 to summarise an "ever growing" finite maxflow (case ii) as a new ω.

▶ **Definition 6** (Iteration of an idempotent). *Let $e = e^2$ be an idempotent element of $\mathcal{F}$. A pair $(v, v') \in V^2$ such that $e(v, v') = 1$ is called* unstable *iff there exists $v_0, v_0' \in V$ such that $e(v, v_0) = \omega$, $e(v_0, v_0') = 1$ and $e(v_0', v') = \omega$, and* stable *otherwise.*

Then the iteration *of e, denoted $e^\sharp$, is defined by*

$$e^\sharp(v, v') = \begin{cases} e(v, v') & \text{if } e(v, v') \in \{0, \omega\} \\ 1 & \text{if } e(v, v') = 1 \text{ and } (v, v') \text{ is stable in } e \\ \omega & \text{if } e(v, v') = 1 \text{ and } (v, v') \text{ is unstable in } e. \end{cases}$$

An idempotent e is called unstable *if it has an unstable pair, and* stable *otherwise.*

Note that if an edge is unstable then it does not satisfy condition iii) of Lemma 5, thus it satisfies condition ii) of the same Lemma. Note also that e is unstable if and only if $e \neq e^\sharp$. We also make the following observation.

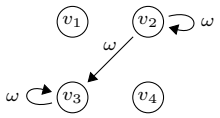
▶ **Lemma 7.** *The iteration of an idempotent e is idempotent and stable, i.e., $(e^\sharp)^\sharp = e^\sharp$.*

Our main algebraic tool is the flow semigroup, a finite structure obtained by application of the product and iteration to the abstract capacities until saturation. This flow semigroup is in fact an example of a *stabilisation monoid* (upon addition of a neutral element), which are ordered monoids with a stabilisation operator [4, Section 3.1].

▶ **Definition 8** (Flow semigroup). *The flow semigroup, denoted $\mathcal{F}$, is the smallest subset of $\mathbb{M}^{V \times V}$ which contains all abstracted capacity constraints $\{x_a \mid a \in A\}$ and is closed under the matrix product and the iteration operation for idempotents.*

▶ **Example 9.** Continue with Example 4. The flow semigroup $\mathcal{F}$ contains x_a and x_b and since $x_b^2 = x_b$, it also contains $x_b^\sharp$. The only capacity-1 edge in x_b is unstable since $x_b(v_2, v_2) = \omega, x_b(v_2, v_3) = 1$ and $x_b(v_3, v_3) = \omega$. Therefore, $\mathcal{F}$ contains the iteration

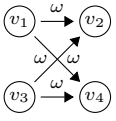
$$x_b^\sharp = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \omega & \omega & 0 \\ 0 & 0 & \omega & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



Intuitively, b allows only one unit of flow from v_2 to v_3, but arbitrarily much flow can remain both in v_2 and v_3. If this is iterated then the sum of those single units of flows from v_2 to v_3 grows, which is represented by a new capacity ω on this edge in the capacity constraint $b^\sharp$.

Finally, $\mathcal{F}$ contains the product

$$x_a x_b^\sharp x_a = \begin{pmatrix} 0 & \omega & 0 & \omega \\ 0 & 0 & 0 & 0 \\ 0 & \omega & 0 & \omega \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



The top right coordinate in $x_a x_b^\sharp x_a$, which corresponds to the edge (v_1, v_4) from the source to the target, is ω. This suggests that an arbitrary amount of flow can be transported from $v_s = v_1$ to $v_4 = v_t$. Such an element is called an *unboundedness witness*.

▶ **Definition 10** (Unboundedness witness). *An* unboundedness witness *is an element of $x \in \mathcal{F}$ such that*

$$x(v_s, v_t) = \omega \ .$$

The existence of such elements in $\mathcal{F}$ is a sufficient and necessary condition for the existence of sequential flows carrying an arbitrary large amount of flow from the source to the target.

▶ **Theorem 11** (Characterization of the unboundedness case). *An instance of the* Sequential flow problem *is unbounded if, and only if, there exists an unboundedness witness in the corresponding flow semigroup.*

We will now present the structure of the proof of this theorem.

To begin with, we observe that since our finite capacities have integer values the integral flow theorem [9] guarantees that every sequence of capacity constraints has a flow with integer coefficients and maximum value. This allows us to work with *token flows*, in which an explicit set of named tokens is fixed, and the trajectory of every token is precisely described.

▶ **Definition 12** (Token flows). *Fix a finite set $\mathcal{T}$ of tokens. A* token flow *of $\mathcal{T}$ of length k over a capacity word w is a mapping $d \in V^{\mathcal{T}\times(0\dots k)}$ which describes the position of every token at every date, and which satisfies capacity constraints. Formally, for every $i \in [1,k]$ and $v, v' \in V$*

$$|\{\tau \in \mathcal{T} \mid d(\tau, i-1) = v \wedge d(\tau, i) = v'\}| \leq a_i(v, v').$$

The outcome of a token flow is described by a *global flow*, which accounts for the number of tokens traveling along every pair of states.

▶ **Definition 13** (Global flows). *With every token flow d of length k is associated a* global flow *denoted $g(d) \in \mathbb{N}^{V\times V}$, which measures the number of tokens moving between every pair of vertices between the dates 0 and k, formally defined as:*

$$g(d)(v, v') = |\{\tau \in \mathcal{T} \mid d(\tau, 0) = v \wedge d(\tau, k) = v'\}|$$

for all $v, v' \in V$.

The proof of Theorem 11 is in two steps: Lemma 14 establishes that the condition is sufficient and Lemma 17 that is it is necessary.

▶ **Lemma 14** (Sufficient condition). *If there exists an unboundedness witness, the answer to the* Sequential flow problem *is ω.*

The proof makes use of the notion of paths.

▶ **Definition 15** (Paths). *A* path *in a capacity word $w = a_1 \dots a_k \in A^*$ is a sequence of vertices $\pi : \{0, \dots, k\} \to V$ that may be followed by a token, i.e., such that $a_i(\pi(i-1), \pi(i)) \geq 1$ for all $i \in \{1, \dots, k\}$.*

Proof sketch of Lemma 14. The key idea is to establish that the following property of element x in the flow semigroup is invariant by product and iteration, and satisfied by the abstract capacities. We prove the following property on elements of the flow semigroup $\mathcal{F}$:

($\diamondsuit$) For all $x \in \mathcal{F}$, for all $N \in \mathbb{N}$, there exist a capacity word w and a token flow d over w such that for all $v, v' \in V$, the following two conditions are satisfied:

1. $x(v, v') = \omega \Rightarrow g(d)(v, v') \geq N$
2. $x(v, v') \geq 1 \Rightarrow$ there is a path in w from v to v' (in the sense of Definition 15).

The proof is by induction on the flow semigroup. For abstract capacity constraints x_a, the capacity word is simply a and we have N tokens following each edge (v, v') with $x_a(v, v') = \omega$.

For a product $x \cdot y$ the token flow $d(x \cdot y)$ for N is obtained by considering two token flows $d(x)$ and $d(y)$ for $|V|^2 N$ tokens, renaming and deleting some of the tokens and concatenating the resulting sequential flows.

The last case is the iteration $f = e^\sharp$. We start by showing that we can decompose $e^\sharp$ as a product of e with so-called simple unstable idempotents, where all non-zero entries are self-loops, except for a single unstable pair. This simple structure is used to craft token flows carrying arbitrarily large amount of tokens along unstable pairs. ◀

For the other direction, we show that when evaluating capacity words in the flow semigroup, the flows between pairs of vertices mapped to 0 or 1 are uniformly bounded.

▶ **Theorem 16.** *Let K be the largest finite constant appearing in a capacity constraint of A. For every token flow d, there exists an element $x \in \mathcal{F}$ such that, for every $v, v' \in V$,*

$$x(v, v') = 0 \implies g(d)(v, v') = 0 \qquad \text{and}$$
$$x(v, v') = 1 \implies g(d)(v, v') \leq K \cdot |V|^{536 \cdot |V|^{12}}.$$

This result relies on a subtle and careful analysis of the flow semigroup, performed in Section 4. The first step is a general result about finite semigroups, showing that every element of a semigroup $\mathcal{S}$ has a finite representation as a binary tree whose height is polynomial in a parameter called the regular $\mathcal{J}$-length of the semigroup (Theorem 26). The second step is specific to the flow semigroup $\mathcal{F}$, and shows that for $\mathcal{F}$ this parameter is polynomial in $|V|$ (Theorem 33). We then extend these trees to incorporate the $\sharp$ operator, and prove that the resulting trees still have polynomial height. That leads to Theorem 16. By contraposition, a consequence of Theorem 16 is the following.

▶ **Lemma 17** (Necessary condition)**.** *If the answer to the* Sequential flow problem *is ω then there is an unboundedness witness in $\mathcal{F}$.*

Proof. Since we have sequential flows of unbounded values between v_s and v_t, there exists a capacity word w with a sequential flow of value greater than $K \cdot |V|^{536 \cdot |V|^{12}}$. By the integral flow theorem [9], we also have a token flow d over w such that $g(d)(v_s, v_t) > K \cdot |V|^{536 \cdot |V|^{12}}$. We apply Theorem 16 to d. By case inspection, the only possible value of $x(v, v')$ is ω, thus x is an unboundedness witness. ◀

In order to test unboundedness for the Sequential flow problem we can compute the entire flow semigroup $\mathcal{F}$, starting with the capacity abstractions $\{x_a \mid a \in A\}$ and closing it by product and $\sharp$, and then check if it contains an unboundedness witness. The correctness of this algorithm follows directly from the definition of $\mathcal{F}$, as well as Lemma 14 and Lemma 17. The resulting algorithm runs in exponential time and space, essentially bounded by the size of $\mathcal{F}$. This can be improved to polynomial space: Instead of explicitly enumerating elements of the flow group we can enumerate so-called $\sharp$-expressions, which represent elements of $\mathcal{F}$.

▶ **Definition 18.** *A $\sharp$-expression of an element $x \in \mathcal{F}$ is a finite $\mathcal{F}$-labeled ordered tree such that the root node is labeled by x, and every node ν is of one of three possible types:*

- *either a leaf node labeled by an abstract capacities $x_a, a \in A$;*
- *it has a single child labeled by e, in which case $e = e^2$ is idempotent and ν is labeled by $e^\sharp$,*
- *it has two children labeled by x_1 and x_2, in which case ν is labeled by $x_1 \cdot x_2$.*

Algorithm 2 Check if x has a $\sharp$-expression of height at most h.

```
1: function IsIn-Rec(x, h)
2:     if x ∈ {x_a | a ∈ A} then return true
3:     if h=0 then return false
4:     for every y, z ∈ {0,1,ω}^{V×V} with y · z = x do
5:         if IsIn-Rec(y, h − 1) and IsIn-Rec(z, h − 1) then return true
6:     for every e ∈ {0,1,ω}^{V×V} with e · e = e and e^♯ = x do
7:         if IsIn-Rec(e, h − 1) then return true
8:     return false
```

The recursive nature of this definition dictates a recursive algorithm to determine if a given element $x \in \{0,1,\omega\}^{V\times V}$ has a $\sharp$-expression of at most a given height h. By convention, the height of a single leaf node is 0.

▶ **Lemma 19.** *Given capacities A, element $x \in \{0,1,\omega\}^{V\times V}$, and $h \geq 0$, Algorithm 2 returns* true *iff x has a $\sharp$-expression of height at most h.*

Notice that Algorithm 2 still runs in (deterministic) exponential time due to the enumerations in lines 4 and 6. However, it only requires space polynomial in $|V|$ and h due to the explicit bound on the recursion depth.

The central argument for showing that unboundedness can be tested in polynomial space is a polynomial bound on the number of nested applications of the $\sharp$ operator necessary to produce an unboundedness witness, provided by the following theorem.

▶ **Theorem 20.** *Every element of the flow semigroup $\mathcal{F}$ is generated by a $\sharp$-expression of height at most $2n^4$.*

We make use of proof techniques used for designing polynomial-space algorithms in other contexts, in particular for checking limitedness of desert automata [17] and the value 1 problem of probabilistic automata [8, 7]. According to Theorem 20, every element of $\mathcal{F}$ has a $\sharp$-expression of height at most $2n^4$. We can therefore check in polynomial space if a given element x is in the flow semigroup $\mathcal{F}$ and whether there exist unboundedness witnesses.

▶ **Theorem 21** (Checking unboundedness)**.** *Checking whether the optimal sequential flow is unbounded is decidable in polynomial space.*

Proof. By Theorem 20, any positive instance admits an unboundedness witnessed x that has $\sharp$-expressions of height at most $h = 2n^4$. It therefore suffices to enumerate (in polynomial space, using Algorithm 2) all $\sharp$-expressions of such bounded height and for each check if the represented element $x \in \mathcal{F}$ constitutes an unboundedness witness, i.e., that $x(v_s, v_t) = \omega$. ◀

If the maximal finite sequential flow value exists, then we can compute it in polynomial space thanks to the exponential bound K established in Theorem 16. Indeed, if there is a flow of value above K then there are flows of unbounded values. It therefore suffices to check whether there is a flow of value $K + 1$. If so then flows can have unbounded values, if not, then we find the optimal value between 0 and K by dichotomic search. Hence we only need to be able to check whether there is a flow of a given, at most exponential, value. This is done in polynomial space via classic graph exploration.

▶ **Theorem 22.** *Given capacities A, the optimal sequential flow can be computed in* PSPACE.

4 Diving into the flow semigroup

This section is dedicated to the proof of two bounds which are crucial to show that the Sequential flow problem can be solved in PSPACE. The first one is Theorem 20, which gives a polynomial upper-bound on the maximal depth of a $\sharp$-expression generating elements of $\mathcal{F}$. The second one is Theorem 16, which establishes a polynomial upper bound on the optimal sequential flow, in case it is finite.

The central tool for these proofs are summaries and $\sharp$-summaries, which provide a bounded-size representation of words of arbitrary length.

4.1 A general factorization theorem for finite semigroups

We rely on a form of factorization of words in a finite semigroup, which we call *summaries*. Let $(\mathcal{S}, \cdot)$ be a finite semigroup[3]. An element $e \in \mathcal{S}$ is called *idempotent* if $e \cdot e = e$. The set of idempotent elements of $\mathcal{S}$ is denoted $E(\mathcal{S})$. We write $\mathcal{S}^{\mathbf{1}}$ for the monoid obtained by extending $\mathcal{S}$ with a neutral element $\mathbf{1}$.

We define a morphism $\varphi_{\mathcal{S}} : \mathcal{S}^* \to \mathcal{S}$ that evaluates sequences of elements of $\mathcal{S}$ by applying the semigroup (product) operation: $\varphi_{\mathcal{S}}(x_0 \dots x_k) = x_0 \cdot x_1 \cdots x_k$.

We recall the Green relations, introduced in [12], $\mathcal{J}, \mathcal{L}, \mathcal{R}, \mathcal{H}$ on $\mathcal{S}$, starting with the following partial orders:

- $x \leq_{\mathcal{J}} y$ if there exist $a, b \in \mathcal{S}^{\mathbf{1}}$ such that $x = ayb$
- $x \leq_{\mathcal{L}} y$ if there exist $a \in \mathcal{S}^{\mathbf{1}}$ such that $x = ay$
- $x \leq_{\mathcal{R}} y$ if there exist $b \in \mathcal{S}^{\mathbf{1}}$ such that $x = yb$
- $x \leq_{\mathcal{H}} y$ if $x \leq_{\mathcal{L}} y$ and $x \leq_{\mathcal{R}} y$

These partial orders can be thought of as reachability relations on $\mathcal{S}$, and the corresponding equivalence classes as strongly connected components.

We use well-established notations for the relations $\mathcal{J}, \mathcal{L}, \mathcal{R}, \mathcal{H}$, the equivalence relations induced by those partial orders. Formally, for each $\mathcal{X} \in \{\mathcal{J}, \mathcal{L}, \mathcal{R}, \mathcal{H}\}$ we define the relations $\mathcal{X} = \leq_{\mathcal{X}} \cap \geq_{\mathcal{X}}$ and $<_{\mathcal{X}} = \leq_{\mathcal{X}} \setminus \geq_{\mathcal{X}}$. Note that $\mathcal{J}$ is coarser than $\mathcal{L}$ and $\mathcal{R}$, themselves coarser than $\mathcal{H}$.

The *regular $\mathcal{J}$-length* [4] of a semigroup $\mathcal{S}$, denoted $L(\mathcal{S})$, is defined as

$$\sup\{k \in \mathbb{N} \mid \exists e_1 <_{\mathcal{J}} \cdots <_{\mathcal{J}} e_k \in E(\mathcal{S})\}.$$

The *Ramsey function* of $\mathcal{S}$ is the function $R_{\mathcal{S}} : \mathbb{N} \to \mathbb{N}$ where $R_{\mathcal{S}}(k)$ is the minimal number n such that every word $w \in \mathcal{S}^*$ of length n contains an infix of the form $u_1 \cdots u_k$ with $\varphi(u_1) = \cdots = \varphi(u_k) = e$ for some $e \in E(\mathcal{S})$. The existence of such a number n for all k can be inferred from Ramsey's theorem, but the following theorem gives much more precise bounds.

A core element of the construction is the following result, which guarantees the existence of consecutive idempotent factors in sufficiently long words over $\mathcal{S}$. Note that the bound is only exponential in the regular $\mathcal{J}$-length of the semigroup, not its size.

---

[3] We choose to work with semigroups in this paper since they are slightly more general than monoids. Note that [13] formulates everything for finite monoids, but the existence of a neutral element is never used in the paper. Every statement from that paper holds for finite semigroups as well.

[4] Called regular $\mathcal{D}$-length in [13]. For finite semigroups, $\mathcal{D} = \mathcal{J}$, and we use $\mathcal{J}$ here since it is more common. Note that the paper introduces it with a different definition, but the two are proven equivalent in the extended version [14, Appendix B].

▶ **Theorem 23** ([13, Theorem 1]). *For all* $k \in \mathbb{N}$,

$k^{L(\mathcal{S})} \leq R_{\mathcal{S}}(k) \leq (k|\mathcal{S}|^4)^{L(\mathcal{S})}$.

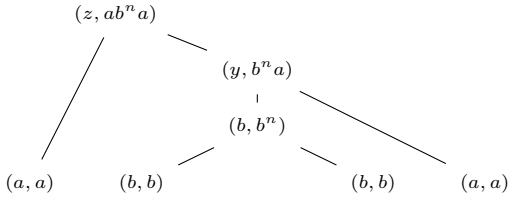
We can now define the central object of our proofs. The theorem gives a way to summarize a word with respect to a finite semigroup. A summary abstracts sequences of idempotent infixes by only keeping the first and last ones. We do so in a way that ensures that the remaining idempotent factors are "short", so that the number of letters we keep from the initial word is polynomial in the size of the semigroup and $R_{\mathcal{S}}(3)$.

▶ **Definition 24.** *A* summary *of a word* $w \in \mathcal{S}^*$ *is a* $\mathcal{S} \times \mathcal{S}^*$*-labeled ordered binary tree with three types of nodes:*

- *a leaf has no children and a label* (x, x) *for some* $x \in \mathcal{S}$
- *a product node labeled* (x, w) *has two children labeled* (y_1, u_1) *and* (y_2, u_2) *such that* $y_1 \cdot y_2 = x$ *and* $u_1 u_2 = w$.
- *an idempotent node labeled* (e, w) *has two children labeled* (e, u_1) *and* (e, u_2) *such that we have* $w = u_1 w' u_2$, $e \in E(\mathcal{S})$ *and* $\varphi_{\mathcal{S}}(u_1) = \varphi_{\mathcal{S}}(u_2) = \varphi_{\mathcal{S}}(w') = e$.

The root is labeled by (x, w)*, for some* $x \in F$*, called the* result *of the summary.*

▶ **Example 25.** Consider the flow semigroup $\mathcal{F}$ for Example 1. Remember that $a = x_a$ and $b = x_b$: capacities match their abstractions because finite constants are 0 or 1. Let $z, y \in \mathcal{F}$ be the products $y = b \cdot a$ and $z = a \cdot b \cdot a$. Since $b = b^2$, the following tree is a summary of $ab^n a$, for any $n > 2$.



Independently of their length, all words $(ab^n a)_{n \geq 2}$ have this summary of height 4 and size 7. The existence of summaries of constant depth is true in general, as shown in Theorem 26.

This definition resembles the one of Simon's factorization forests [18]. However, an important difference is that Simon's trees are meant to factorize the entire word, while ours omit a lot of information by skipping intermediate idempotents. This lets us obtain better bounds on the height of the tree, since Simon's trees have linear height in the size of the semigroup, not just its regular $\mathcal{J}$-length. Those bounds are essential to obtain singly-exponential bounds, and then a polynomial-space algorithm, in the quantitative setting.

▶ **Theorem 26.** *For all* $w \in \mathcal{S}^*$ *there exists a summary whose result is* $\varphi_{\mathcal{S}}(w)$ *and of height at most* $L(\mathcal{S})(\log_2(|\mathcal{S}|) + 2\log_2(R_{\mathcal{S}}(3)) + 4)$.

Proof sketch. We first define the regular $\mathcal{J}$-length of an element of the semigroup, as the one of the subsemigroup of elements $\leq_{\mathcal{J}}$-below it. The proof goes through an induction on the regular $\mathcal{J}$-length of elements of the semigroup. We start by cutting the word in minimal blocks of maximal regular $\mathcal{J}$-length. We factorize these blocks as a single letter and a block of smaller regular $\mathcal{J}$-length, for which we get a factorization by induction. Then we consider the word obtained by replacing each block with its value in the semigroup.

We cut this word into infixes, each long enough to guarantee that it contains idempotents. We describe an operation that lets us merge some blocks where the same idempotent appears. We then use properties of the Green relations to bound the number of blocks obtained this way. ◀

4.2 Application to the flow semigroup and iterations

This subsection is dedicated to the proof of two bounds which are crucial to show that the Sequential flow problem can be solved in polynomial space. The first one is Theorem 20, which gives a polynomial upper-bound on the maximal depth of a $\sharp$-expression generating elements of $\mathcal{F}$. The second one is Theorem 16, which establishes a polynomial upper bound on the optimal sequential flow, in case it is finite.

Let $n = |V|$, and let $\mathcal{B} = \mathbb{B}^{n\times n}$ be the finite semigroup of Boolean matrices of dimension n, equipped with the $(\vee, \wedge)$ matrix product. The following result bounds its regular $\mathcal{J}$-length by a polynomial in its dimension.

▶ **Theorem 27** ([13, Theorem 2]). *The regular $\mathcal{J}$-length of $\mathcal{B}$ is at most $(n^2+n+2)/2$.*

Let us now take a look at the flow semigroup, $\mathcal{F} \subseteq \mathbb{M}^{n\times n}$ with $\mathbb{M} = (\{0, 1, \omega\}, \max, \min)$.

▶ **Lemma 28.** *$\mathcal{F}$ is isomorphic to a sub-semigroup of $\mathcal{B}^2$.*

Proof. Define $\psi : \mathcal{F} \to \mathcal{B}^2$ such that for all matrix $x \in \mathcal{F}$, $\psi(x) = (\mu_1, \mu_\omega)$ with, for all $i, j \in [1, n]$,

$$\mu_1(i,j) = \begin{cases} \top \text{ if } x(i,j) \geq 1 \\ \bot \text{ otherwise} \end{cases} \qquad \text{and} \qquad \mu_\omega(i,j) = \begin{cases} \top \text{ if } x(i,j) = \omega \\ \bot \text{ otherwise.} \end{cases}$$

This is an injective function, and it is easily verified that this is a morphism. ◀

▶ **Theorem 29.** *Every word $w \in \mathcal{F}^*$ admits a summary of height at most $536 \cdot n^{10}$.*

Proof. As observed above, $\mathcal{F}$ is isomorphic to a subsemigroup of $\mathcal{B}^2$, which has regular $\mathcal{J}$-length bounded by $(n^2+n+2)/2$. The regular $\mathcal{J}$-length of $\mathcal{F}$ is then at most $(n^2+n+2)^2/4$, while its size is 3^{n^2}. By Theorem 23, we obtain $R_{\mathcal{F}}(3) \leq 3^{(n^2+1)(n^2+n+2)^2} \leq 3^{32n^6}$ for $n \geq 1$. Then, by Theorem 26 we have that every word over $\mathcal{F}$ has a summary of height at most

$$\begin{aligned} & L(\mathcal{F})(\log_2(|\mathcal{F}|) + 2\log_2(R_{\mathcal{F}}(3)) + 4) \\ \leq\ & \frac{(n^2+n+2)^2}{4}(n^2\log_2(3) + 2(32n^6\log_2(3)) + 4) \\ \leq\ & 4n^4(2n^2 + 128n^6 + 4) \\ \leq\ & 536n^{10} \end{aligned}$$

◀

We now need to integrate the $\sharp$ operator in this construction. We do this by following a proof of Simon [19, Theorem 9] on a different semigroup.

▶ **Lemma 30.** *Let $m \geq n^2$ and $e_1, \ldots, e_m \in E(\mathcal{F})$ idempotents of $\mathcal{F}$ such that $e_{i+1} \leq_{\mathcal{J}} e_i^\sharp$ for all i. Then there exists i such that $e_i^\sharp = e_i$.*

Observe that Example 3 can be generalized to obtain instances where we need a linear number of nested $\sharp$ in order to obtain some elements of $\mathcal{F}$.

That result has two interesting consequences, which are the keys to obtain PSPACE algorithms for the Sequential flow problem. The first consequence is that small "$\sharp$-expressions" (Definition 18) are enough to generate all elements in the flow semigroup, as stated in Theorem 20. The idea is as follows: take a $\sharp$-expression generating an element of $\mathcal{F}$. First use Lemma 30 to eliminate redundant idempotent nodes and reduce its $\sharp$-height below n^2. Then reorganise its product nodes to obtain balanced subtrees of product nodes. Since the monoid has exponential size in n, trees of polynomial height suffice to obtain everything we can with products.

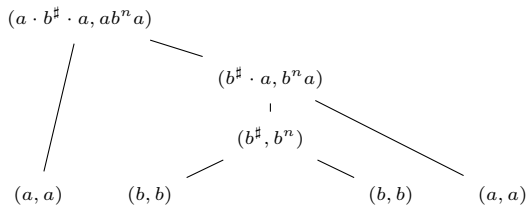
The second consequence of Lemma 30 is that every capacity word can be represented as a small $\sharp$-summary: We define $\sharp$-*summaries*, where idempotent nodes for $e \in E(\mathcal{S})$ are labeled with $e^\sharp$ instead of e.

▶ **Definition 31.** *A $\sharp$-summary of a word $w \in \mathcal{F}^*$ is a $\mathcal{F} \times \mathcal{F}^*$-labeled ordered binary tree, with three types of nodes:*

- *A leaf is labeled by (x, x) for some $x \in \mathcal{F}$,*
- *A product node has two children. If their labels are (x_1, u_1) and (x_2, u_2) then its label is $(x_1 \cdot x_2, u_1 u_2)$*
- *An idempotent node has two children. If their labels are (x_1, u_1) and (x_2, u_2) then $x_1 = x_2 = e$ is an idempotent of $\mathcal{F}$ and the label of the node is $(e^\sharp, u_1 w_1 \dots w_m u_2)$ for some $w_1, \dots, w_m \in \mathcal{F}^*$ such that all w_i have a $\sharp$-summary whose root is labeled (e, w_i). In the case that $e = e^\sharp$ we say that the node is a stable idempotent node, and an unstable idempotent node otherwise.*

Moreover, the root is labeled by (x, w) for some $x \in \mathcal{F}$, which is called the result of the $\sharp$-summary.

▶ **Example 32.** In Example 1, since $b = b^2$, the following tree is a $\sharp$-summary of $ab^n a$.



This $\sharp$-summary bears some similarity with the summary of Example 25. However, there is a crucial difference: the root of the $\sharp$-summary is labelled by $a \cdot b^\sharp \cdot a$, which is an unboundedness witness (see details before Definition 10). This is not the case in the summary of Example 25: the root is labelled by $z = aba$ which is not an unboundedness witness since $z(v_1, v_4) = 1$.

Independently of their length, all words $(ab^n a)_{n \geq 2}$ have this $\sharp$-summary of height 4 and size 7. The existence of $\sharp$-summaries of constant height (and size), whatever the size of the word, is true in general, as shown in Theorem 33.

Our next step is to show that all words have a $\sharp$-summary of polynomial height in the number of vertices $n = |V|$. We take inspiration from two existing proofs: First, one by Kirsten [17] to show that the number of unstable nodes along a branch of a $\sharp$-summary (or a $\sharp$-expression) is bounded by a polynomial in n, the number of vertices[5]. Second, one by Simon [18] to show that every word has a $\sharp$-summary where the distance between consecutive unstable nodes along a branch is bounded by another polynomial in n. Adapting and combining those two arguments yields the result.

▶ **Theorem 33.** *For all $w \in \mathcal{F}^*$ there is a $\sharp$-summary of height at most $536 \cdot n^{12}$.*

We make use of Theorem 33 to prove Theorem 16. By the max flow-min cut theorem [10], to prove a bound on the maximal flow of capacity words it suffices to find for each one a cut of cost at most this bound. We construct this cut by induction on the height of a $\sharp$-summary for the capacity word. A key part of the argument is the trichotomy from Lemma 5, particularly

[5] A polynomial bound in $\mathcal{O}(n^4)$ can be obtained by proving that $e^\sharp <_{\mathcal{J}} e$ for all $e \in \mathcal{F}$, and using the bound on the regular $\mathcal{J}$-length. We prefer to use Lemma 30, which gives $\mathcal{O}(n^2)$.

case (ii). When dealing with an idempotent node we show that for all v, v', either iterating the corresponding idempotent gives us unbounded flows from v to v', or, in any iteration of the idempotent, we can find a cut of bounded cost between v and v' within the first and last iterations. This justifies the abstraction of $\sharp$-summaries: in a sequence of iterations of an idempotent we keep only the first and last. The value of the constructed cut is exponential in the height of the $\sharp$-summary, which yields the result by Theorem 33.

5 Extensions

Our approach to solve the Sequential flow problem can be extended to further generalisations that consider sequential flows between sets of source and target vertices and under regular constraints on the witnessing capacity words.

5.1 Fair flows along multiple edges / out of multiple sources

The previous results and algorithms can be adapted to solve a more general problem.

Instead of a single source-target pair (v_s, v_t), the problem comes with a collection $E \subseteq V \times V$ of edges, and one wishes to carry as much flow as possible along those edges.

That is, the objective is *not* to maximize the total amount of flow through the edges (this can easily be reformulated as an instance of the Sequential flow problem, solutions to which then may lead to one of the edges being unused). Instead, we ask to maximize the minimal global flow among all given edges.

Using the notation from Definition 12 of token flow d and its associated global flow $g(d)$), the *Fair sequential flow problem* asks to compute the value $\sup_{\text{token flow } d} |d|$, where

$$|d| = \min\{g(d)(v_s, v_t) \mid (v_s, v_t) \in E\}.$$

This problem can be solved similarly to the Sequential flow problem, except one looks for a different kind of witnesses in the flow semigroup $\mathcal{F}$.

▶ **Definition 34.** *A* fair unboundedness witness *is an element of $x \in \mathcal{F}$ such that*

$$\forall (v_s, v_t) \in E, x(v_s, v_t) = \omega.$$

▶ **Theorem 35.** *The Fair sequential flow problem can be solved in polynomial space.*

Proof. The unboundedness of the Fair sequential flow problem is equivalent to the existence of a fair unboundedness witness in the flow semigroup $\mathcal{F}$, the proof is a straightforward adaptation of the proofs of Lemma 14 and 17. Such a witness can be looked for in polynomial space using Algorithm 2. If no such witness exists then the value $\sup_d |d|$ is finite and even bounded by $B = K \cdot |V|^{536 \cdot |V|^{12}}$ according to Theorem 16. Then a variant of Algorithm 1 can be used in order to optimise $|d| = \min\{g(d)(v_s, v_t) \mid (v_s, v_t) \in E\}$ by looking for a token flow d moving exactly k tokens along every edge in E, where k is optimized by dichotomic search in the interval $[0, B]$. ◀

▶ Remark 36. If, instead of simultaneous flow across a subset of edges, one is interested in checking simultaneous flows out of several sources into the target, the resulting problem easily reduces to the Fair sequential flow problem. Indeed, it suffices to introduce a new target vertex t and a new final capacity constraint that transfers tokens from all target states to t. Then the multi-source flow problem is an instance of the Fair sequential flow problem where we ask to maximise the simultaneous flow from each source to t.

5.2 Regular constraints

The *Fair sequential flow problem with regular constraints* generalises the Fair sequential flow problem by requiring that we only consider capacity words within a given regular language. Formally, consider a finite set $A \subseteq (\mathbb{N} \cup \{\omega\})^{V \times V}$ of capacities, a set $E \subseteq V \times V$ and a regular language $L \subseteq A^*$ recognized by a non-deterministic automaton with m states. Let $n = |V|$ and K be the largest finite capacity in any element of A. The problem is to compute the optimal fair token flow over capacity words in L, i.e.,

$$\sup_{w \in L} \sup_{\substack{\text{token flow } d \\ \text{over } w}} |d| \ .$$

▶ **Theorem 37.** *The Fair sequential flow problem with regular constraints can be solved in polynomial space. Furthermore, if the answer is bounded then it is at most* $K(2|V|)^{(170 \log_2(m)+835)|V|^{12}}$.

This can be shown with the technique discussed in Sections 3 and 5.1 with a slightly extended definition of the flow semigroup. We describe the necessary adjustments below.

Fix a non-deterministic finite automaton $\mathcal{A}$ with Q the set of m control states, $I, F \subseteq Q$ sets of initial and final states, and $\Delta \subseteq Q \times A \times Q$ the transitions over the alphabet A. We define the semigroup $\mathcal{B}_\mathcal{A}$ made of the set of triples in $Q \times \{0, 1, \omega\}^{V \times V} \times Q$, plus an element $\bot$, and where the product $\star$ is defined as follows:

$$(q_1, x, q_1') \star (q_2, y, q_2') = \begin{cases} (q_1, x \cdot y, q_2') \text{ if } q_1' = q_2 \\ \bot \text{ otherwise.} \end{cases}$$

We set $\bot \star \bot = \bot$ and $(q, x, q') \star \bot = \bot \star (q, x, q') = \bot$ for all $(q, x, q') \in Q \times \{0, 1, \omega\}^{V \times V} \times Q$.

To lift the iteration operation, notice that, except for $\bot$, an idempotent of $\mathcal{B}_\mathcal{A}$ is of the form (q, e, q) with an idempotent matrix e. Define $(q, e, q)^\sharp = (q, e^\sharp, q)$ for all $q \in Q$ and idempotent e, and let $\bot^\sharp = \bot$.

The *labeled flow semigroup* $\mathcal{F}_\mathcal{A}$ is defined as the smallest sub-semigroup of $\mathcal{B}_\mathcal{A}$ which contains $\{(q, x_a, q') \mid (q, a, q') \in \Delta\}$ and is stable under $\star$ and $\sharp$.

It then suffices to go through the same steps as in Sections 3 and 4, with some minor changes to accommodate the state constraints.

6 Conclusion

We provide a new algebraic technique to solve the Sequential flow problem in PSPACE. We mention two promising directions to utilize the results shown here. First, we aim to adapt our techniques to graphs generated by graph grammars. Second, we plan to extend our framework to settings with asynchronous flows, with applications to asynchronous distributed computing.

References

1 Eleni C. Akrida, Jurek Czyzowicz, Leszek Gąsieniec, Łukasz Kuszner, and Paul G. Spirakis. Temporal flows in temporal networks. *Journal of Computer and System Sciences*, 103:46–60, 2019. doi:10.1016/j.jcss.2019.02.003.

2 Jay E. Aronson. A survey of dynamic network flows. *Annals of Operations Research*, 20(1):1–66, December 1989. doi:10.1007/BF02216922.

3 Achim Blumensath, Thomas Colcombet, and Pawel Parys. On a fragment of AMSO and tiling systems. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.STACS.2016.19.

4 Thomas Colcombet. Regular cost functions, part I: logic and algebra over words. *Log. Methods Comput. Sci.*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

5 Thomas Colcombet, Nathanaël Fijalkow, and Pierre Ohlmann. Controlling a random population. In *International Conference on Foundations of Software Science and Computational Structures (FoSSaCS)*, volume 12077 of *Lecture Notes in Computer Science*, pages 119–135. Springer, 2020. doi:10.1007/978-3-030-45231-5_7.

6 Thomas Colcombet, Nathanaël Fijalkow, and Pierre Ohlmann. Controlling a random population. *Logical Methods in Computer Science*, 17(4), 2021. doi:10.46298/LMCS-17(4:12)2021.

7 Nathanaël Fijalkow, Hugo Gimbert, Edon Kelmendi, and Denis Kuperberg. Stamina: Stabilisation monoids in automata theory. In *International Conference on Implementation and Application of Automata*, volume 10329 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2017. doi:10.1007/978-3-319-60134-2_9.

8 Nathanaël Fijalkow, Hugo Gimbert, Edon Kelmendi, and Youssouf Oualhadj. Deciding the value 1 problem for probabilistic leaktight automata. *Log. Methods Comput. Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:12)2015.

9 L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6(3):419–433, 1958. doi:10.1287/opre.6.3.419.

10 Lester Randolph Ford and Delbert Ray Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

11 Hugo Gimbert, Corto Mascle, and Patrick Totzke. Optimally controlling a random population. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis, editors, *53rd International Colloquium on Automata, Languages, and Programming (ICALP 2026)*, volume 374 of *LIPIcs*, pages 179:1–179:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi:10.4230/LIPIcs.ICALP.2026.179.

12 James Alexander Green. On the structure of semigroups. *Annals of Mathematics*, 54(1):163–172, 1951. doi:10.2307/1969317.

13 Ismaël Jecker. A Ramsey theorem for finite monoids. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 187 of *LIPIcs*, pages 44:1–44:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.44.

14 Ismaël Jecker. A ramsey theorem for finite monoids. *CoRR*, abs/2101.05895, 2021. arXiv:2101.05895.

15 David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. In *Symposium on Theory of Computing (STOC)*, STOC '00, pages 504–513. Association for Computing Machinery, 2000. doi:10.1145/335305.335364.

16 David Kempe, Jon M. Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences*, 64(4):820–842, 2002. doi:10.1006/JCSS.2002.1829.

17 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO Theor. Informatics Appl.*, 39(3):455–509, 2005. doi:10.1051/ITA:2005027.

18 Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72(1):65–94, 1990. doi:10.1016/0304-3975(90)90047-L.

19 Imre Simon. On semigroups of matrices over the tropical semiring. *Informatique Théorique et Applications*, 28(3-4):277–294, 1994. doi:10.1051/ITA/1994283-402771.