The steady-states of splitter networks

Basile Couëtoux, Bastien Gastaldi, *Guyslain Naves*

Aix-Marseille University

G-SCOP 2025

Splitter networks

A *splitter network* consists in conveyor belts joined by splitters.

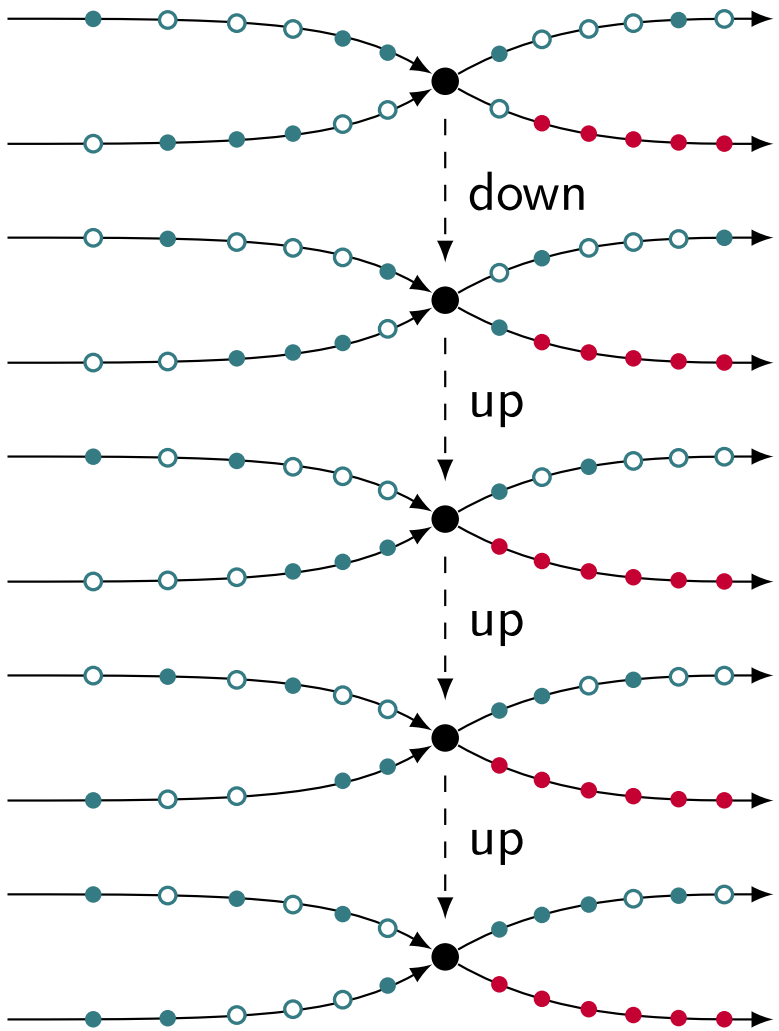
A *conveyor belt* (or *belt*) moves items

- from its tail to its head;
- at a constant speed;
- items may accumulate at its head end: they stop until some item is consumed.

A *splitter* joins one or two incoming belts to one or two outgoing belts

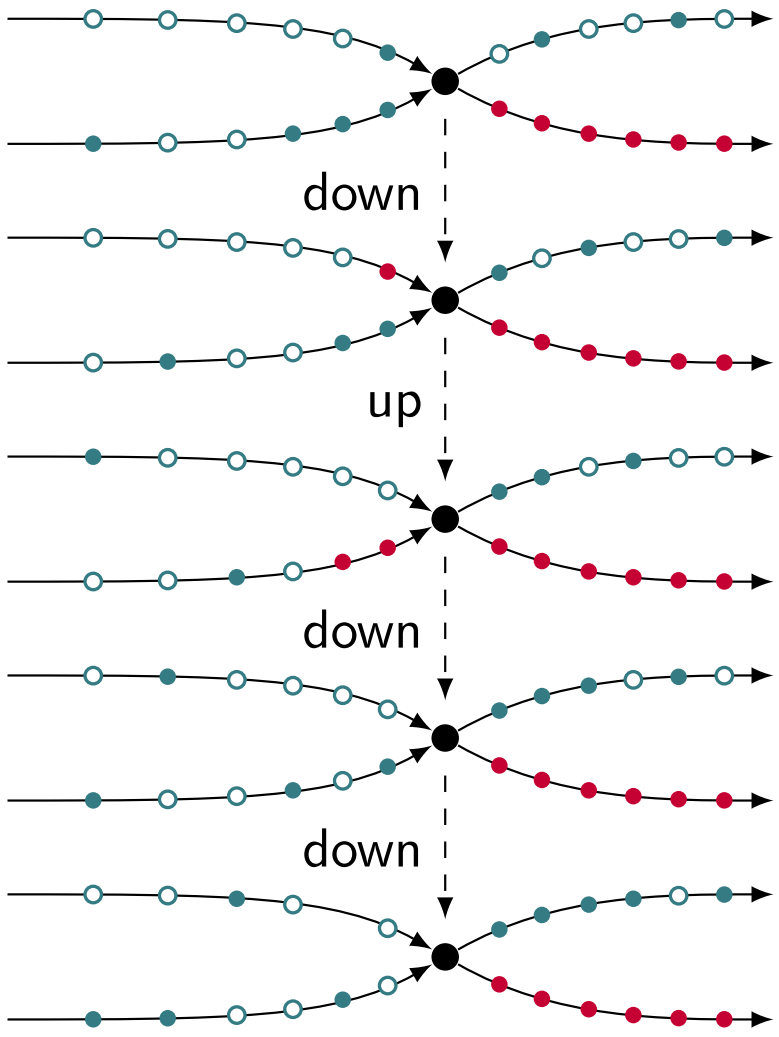
- it takes items from the incoming belts;
- and moves them to the outgoing belts;
- it tries to alternate between the two incoming belts, between the two outgoing belts; it is *fair*.

Splitters are fair (out)



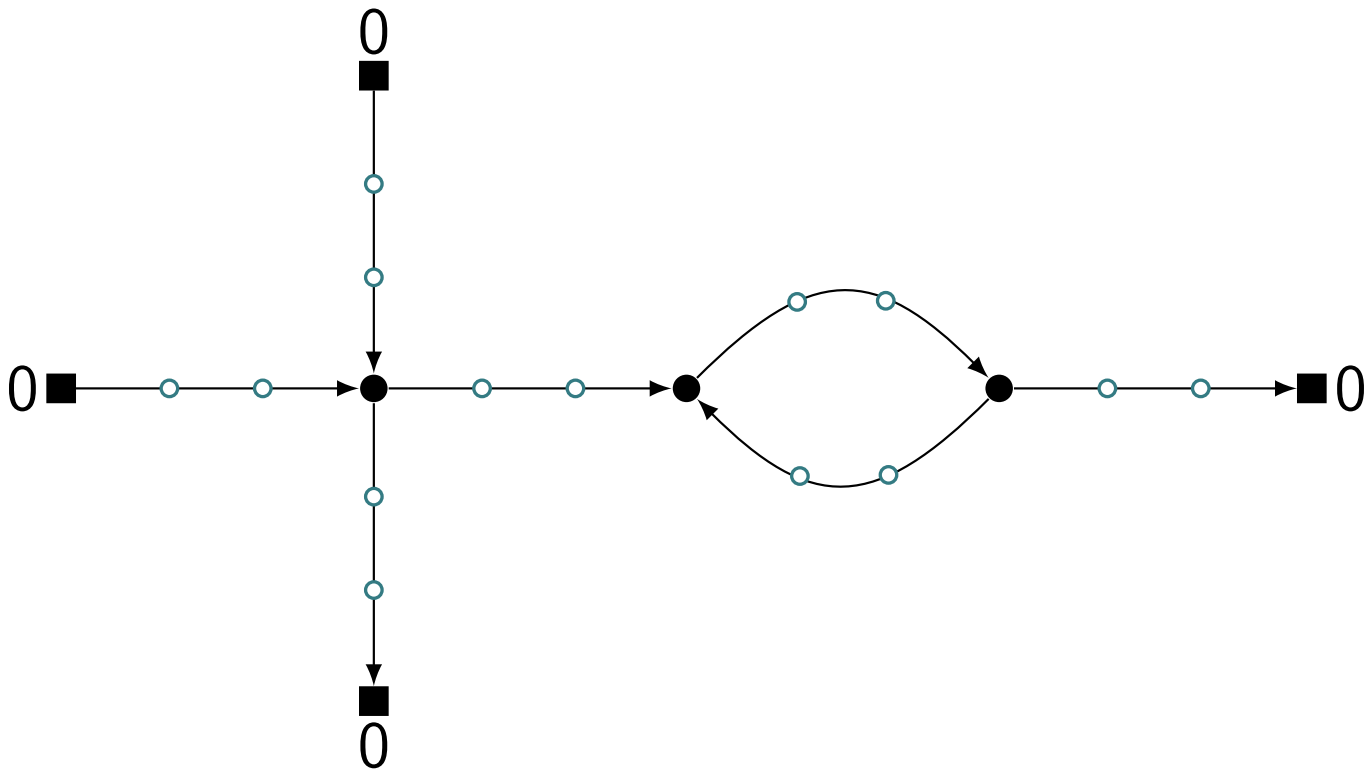
- Splitters alternate pushing on their outgoing arcs;
- except when one of them is completely filled.

Splitters are fair (in)

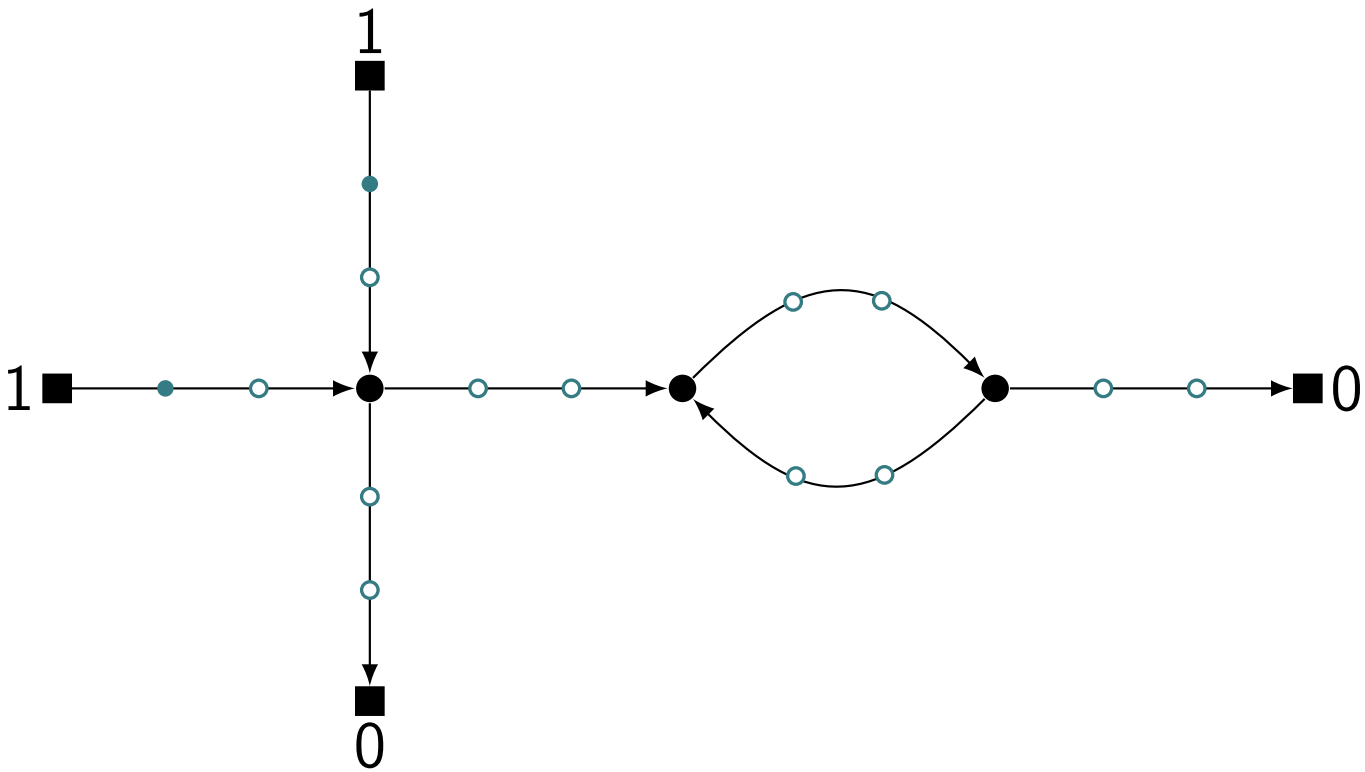


- Splitters alternate pulling from their incoming arcs;
- except when one of them has not available item.

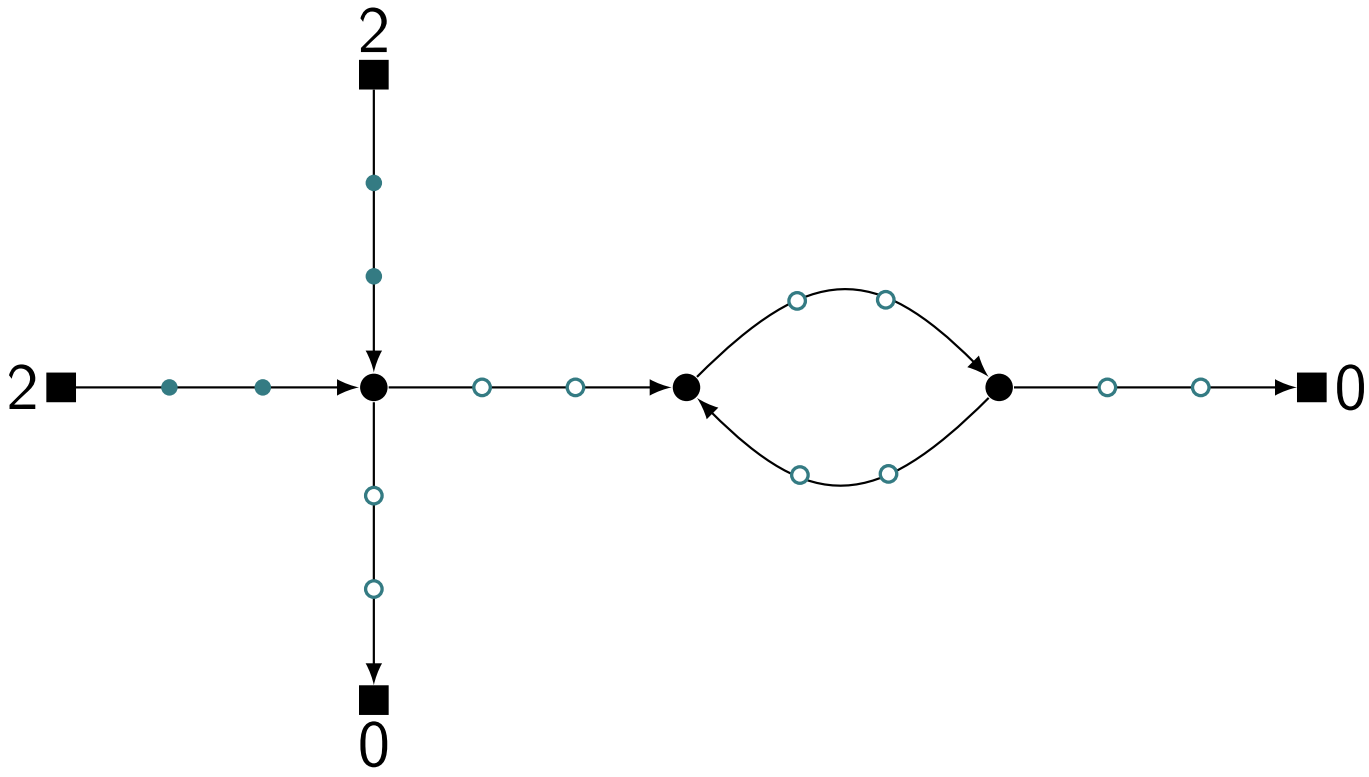
Example of a splitter network dynamics



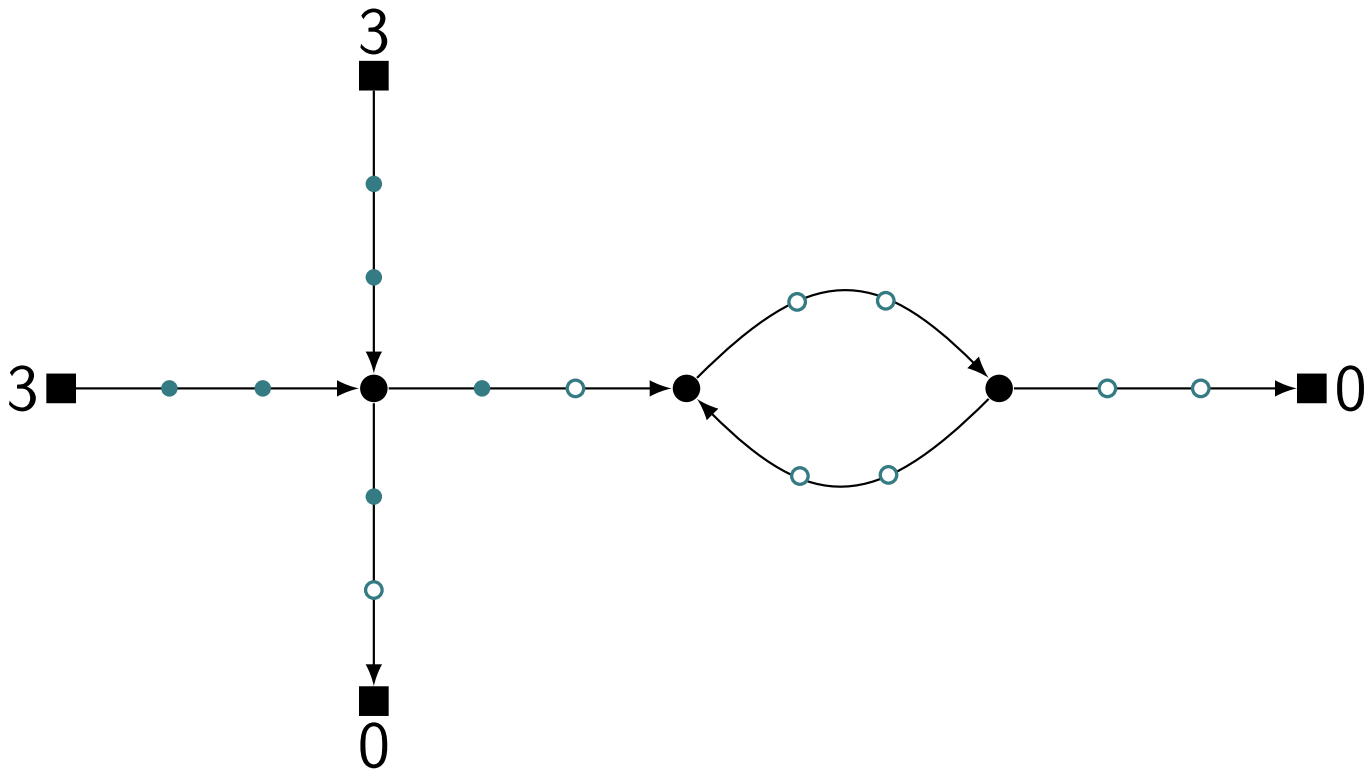
Example of a splitter network dynamics



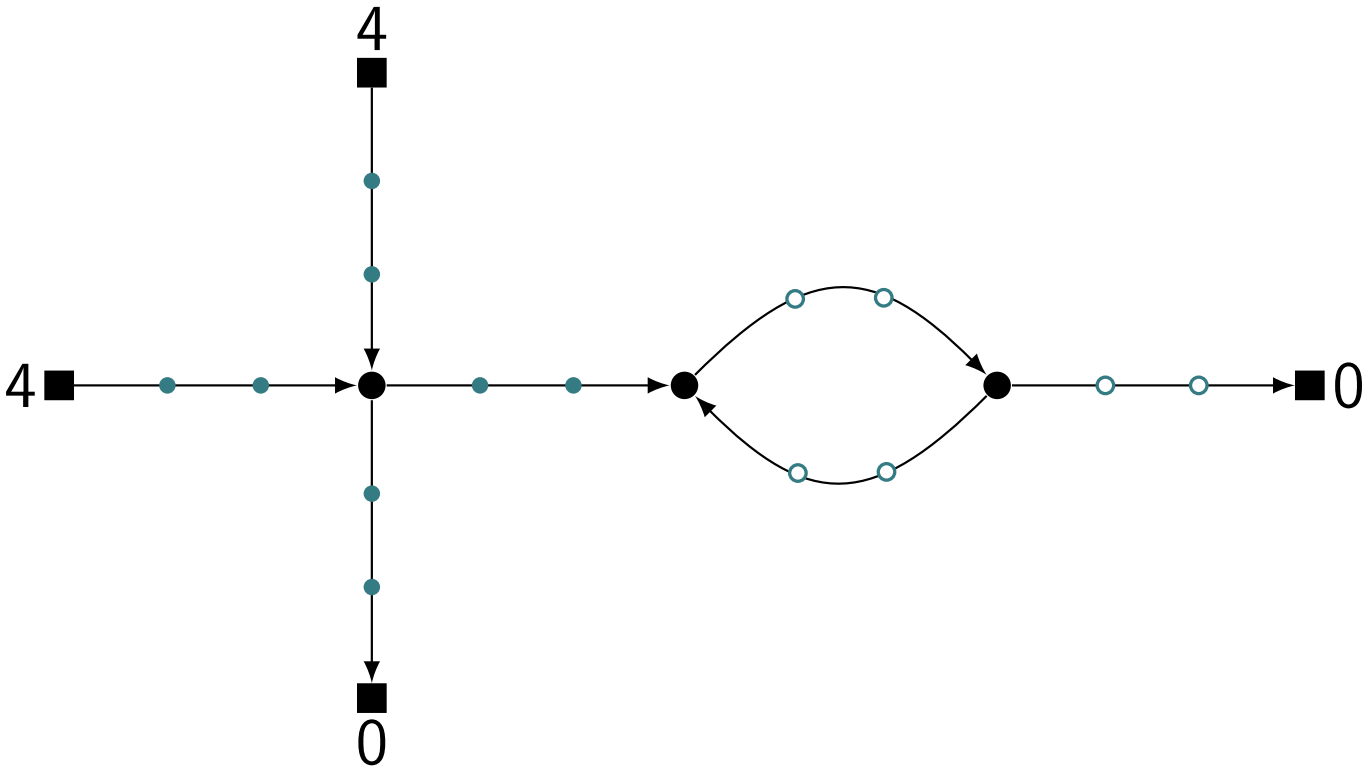
Example of a splitter network dynamics



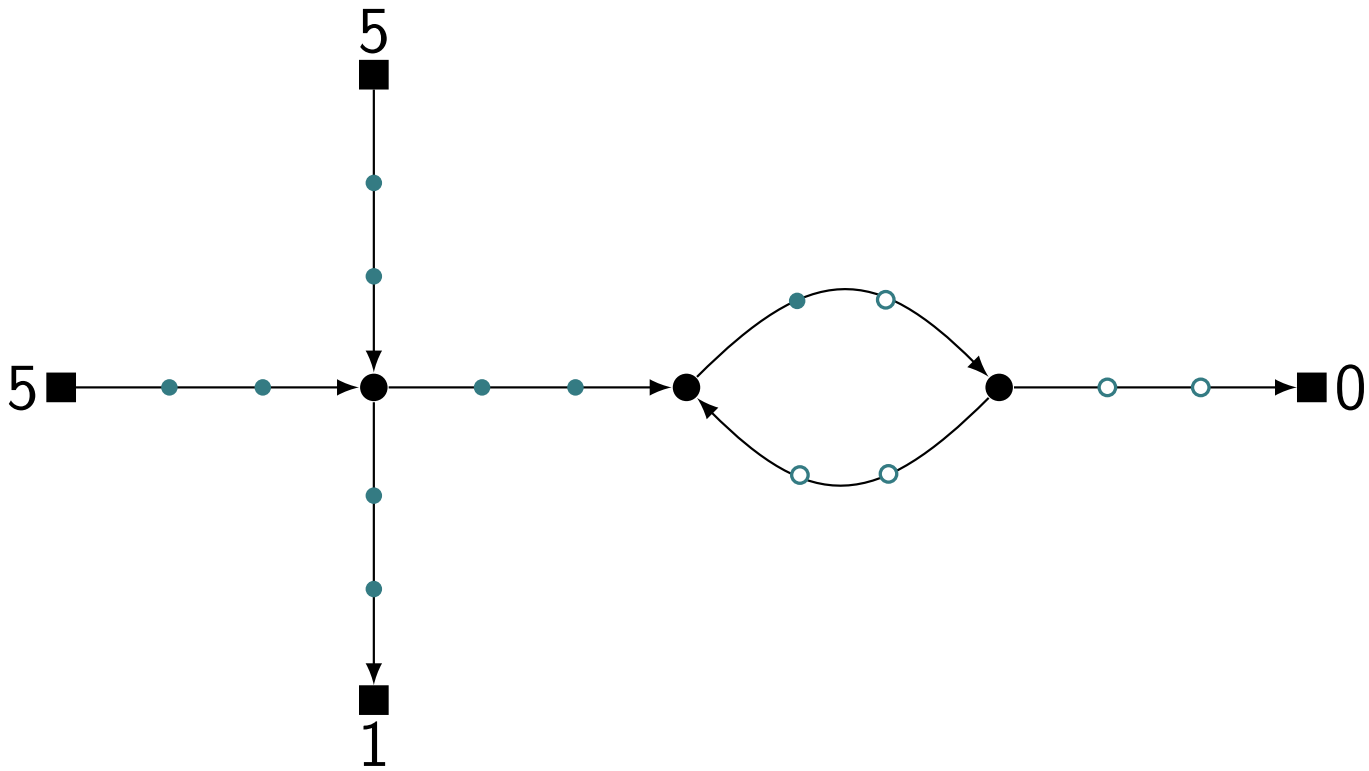
Example of a splitter network dynamics



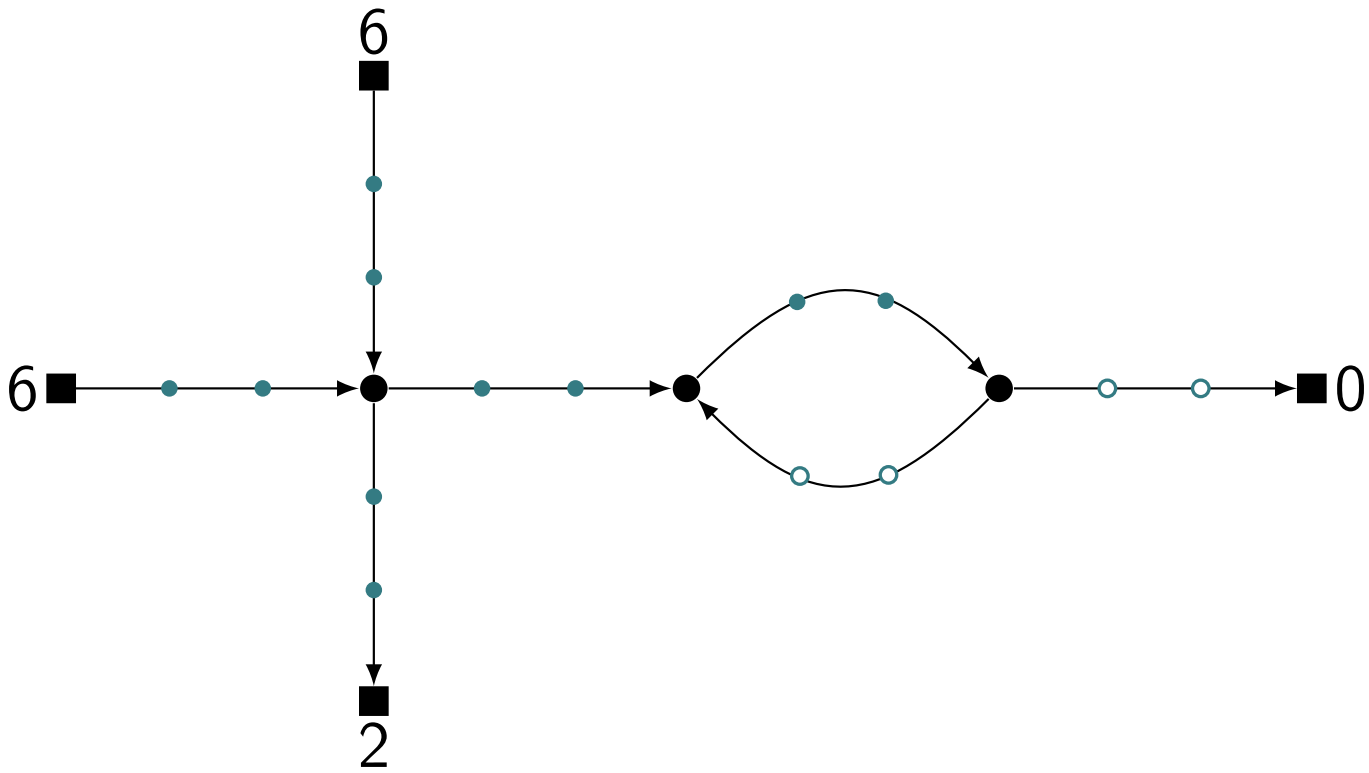
Example of a splitter network dynamics



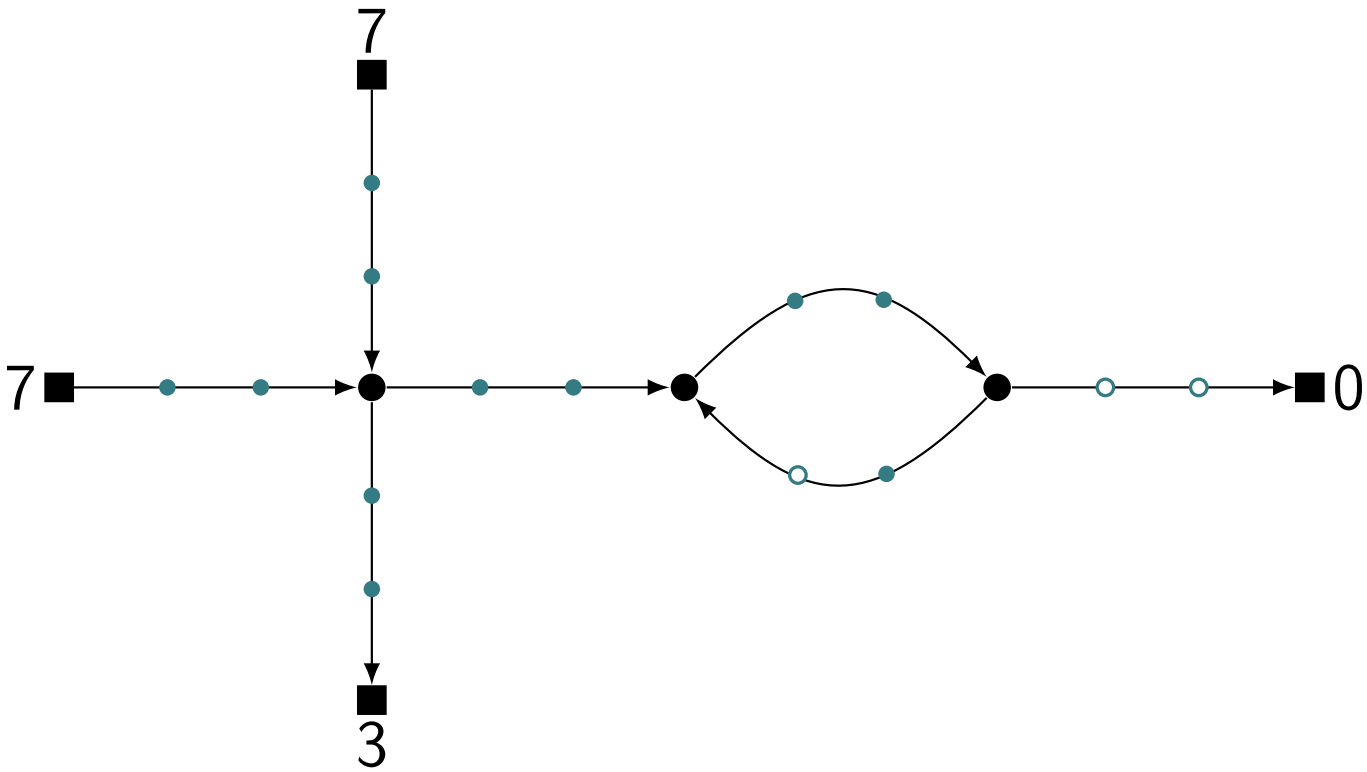
Example of a splitter network dynamics



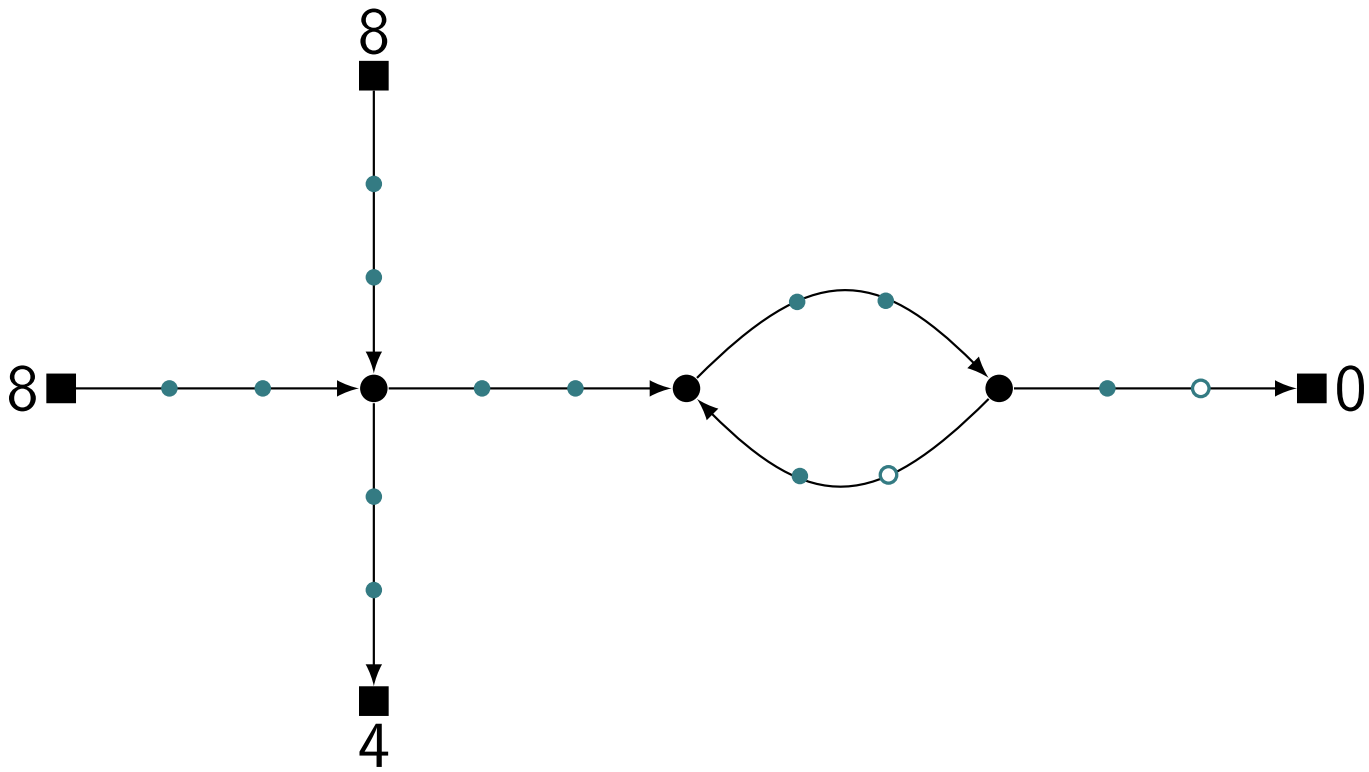
Example of a splitter network dynamics



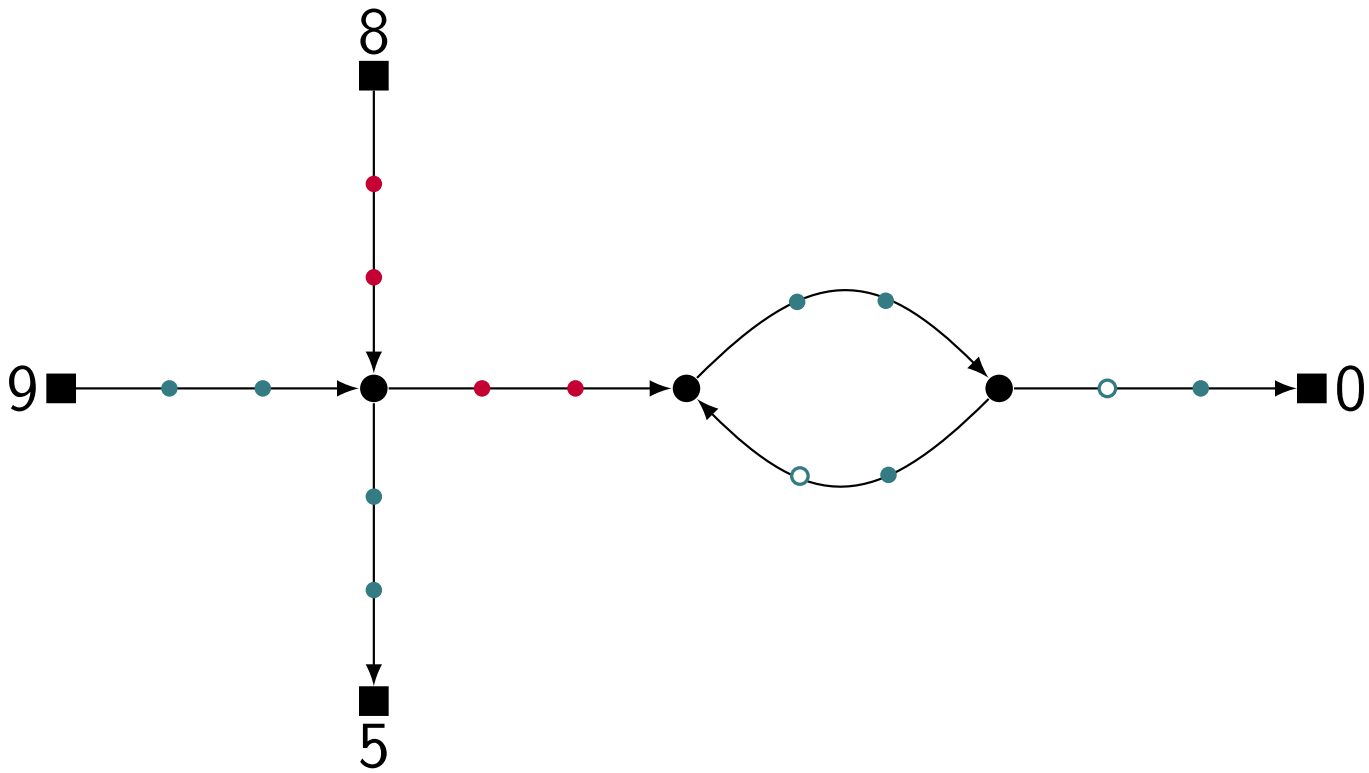
Example of a splitter network dynamics



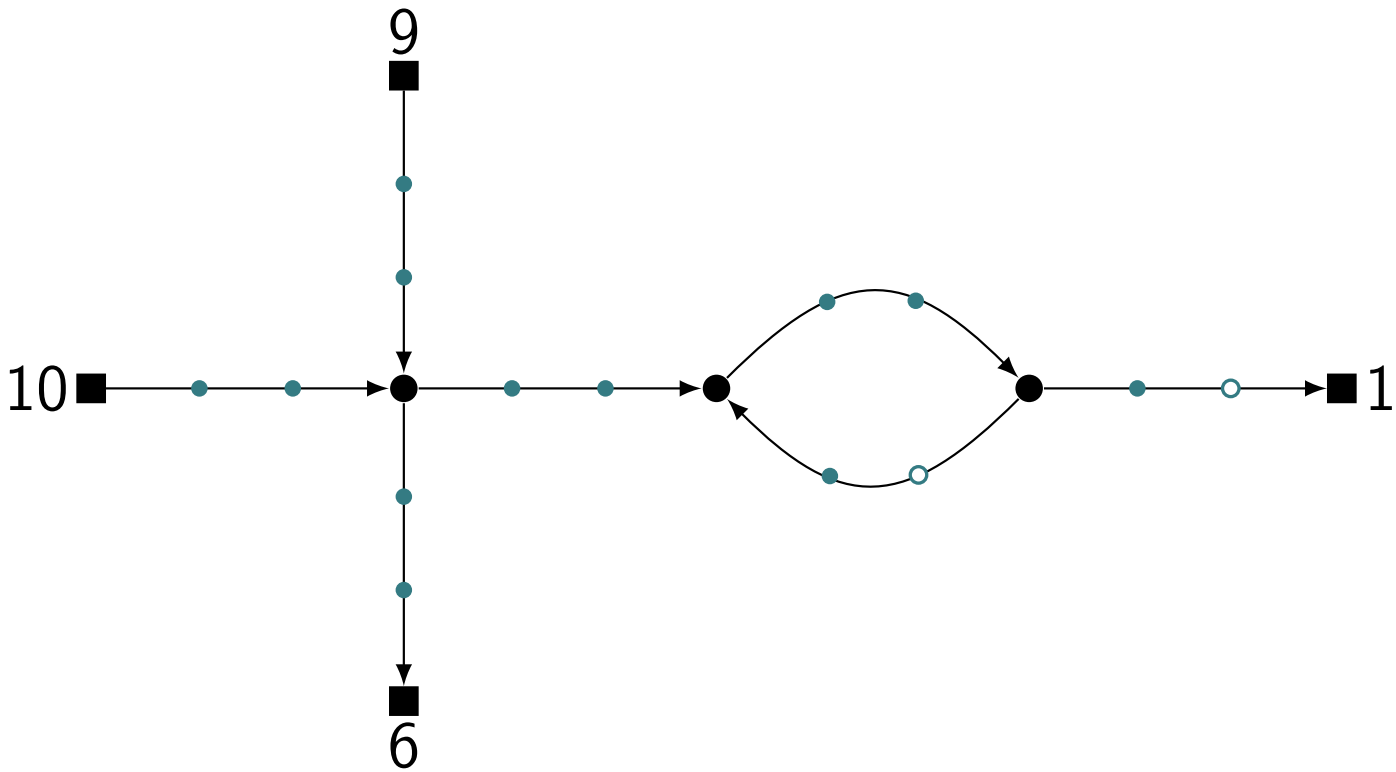
Example of a splitter network dynamics



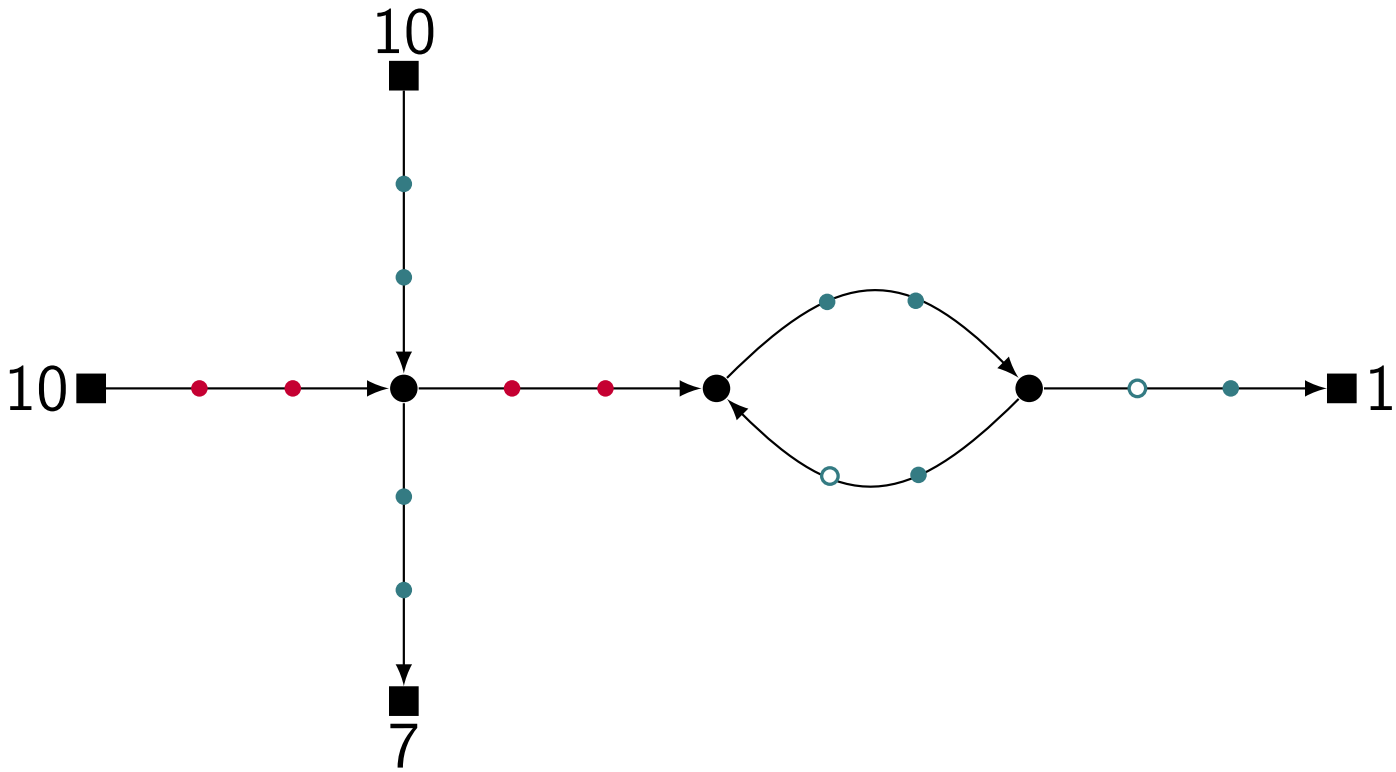
Example of a splitter network dynamics



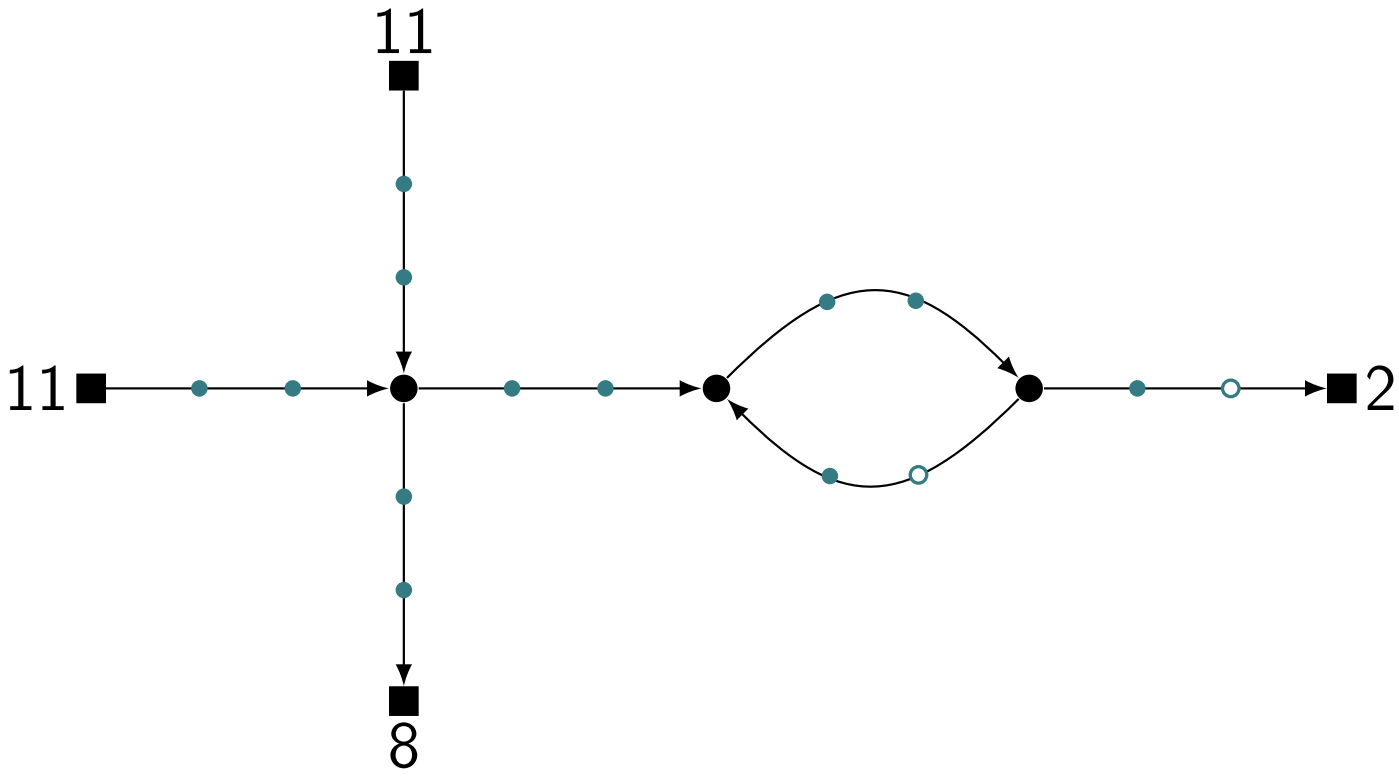
Example of a splitter network dynamics



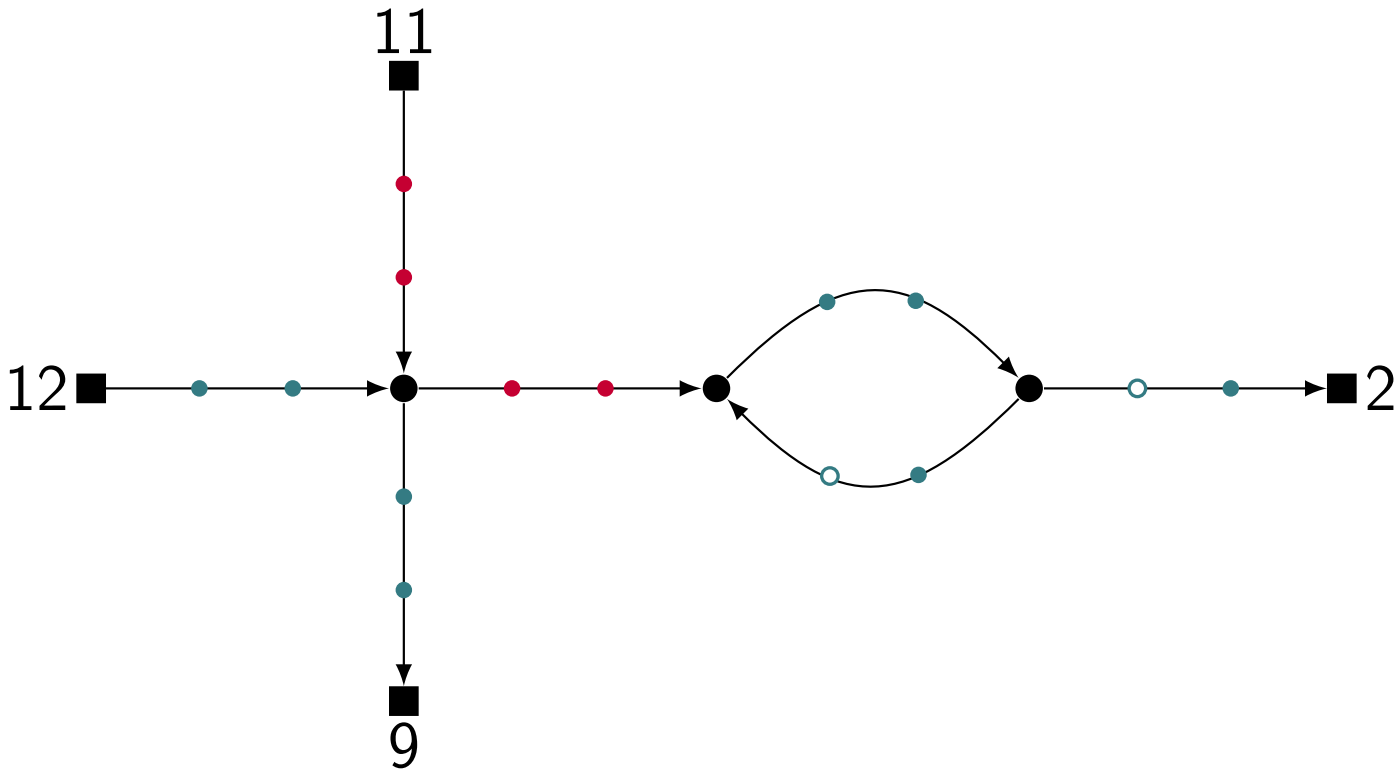
Example of a splitter network dynamics



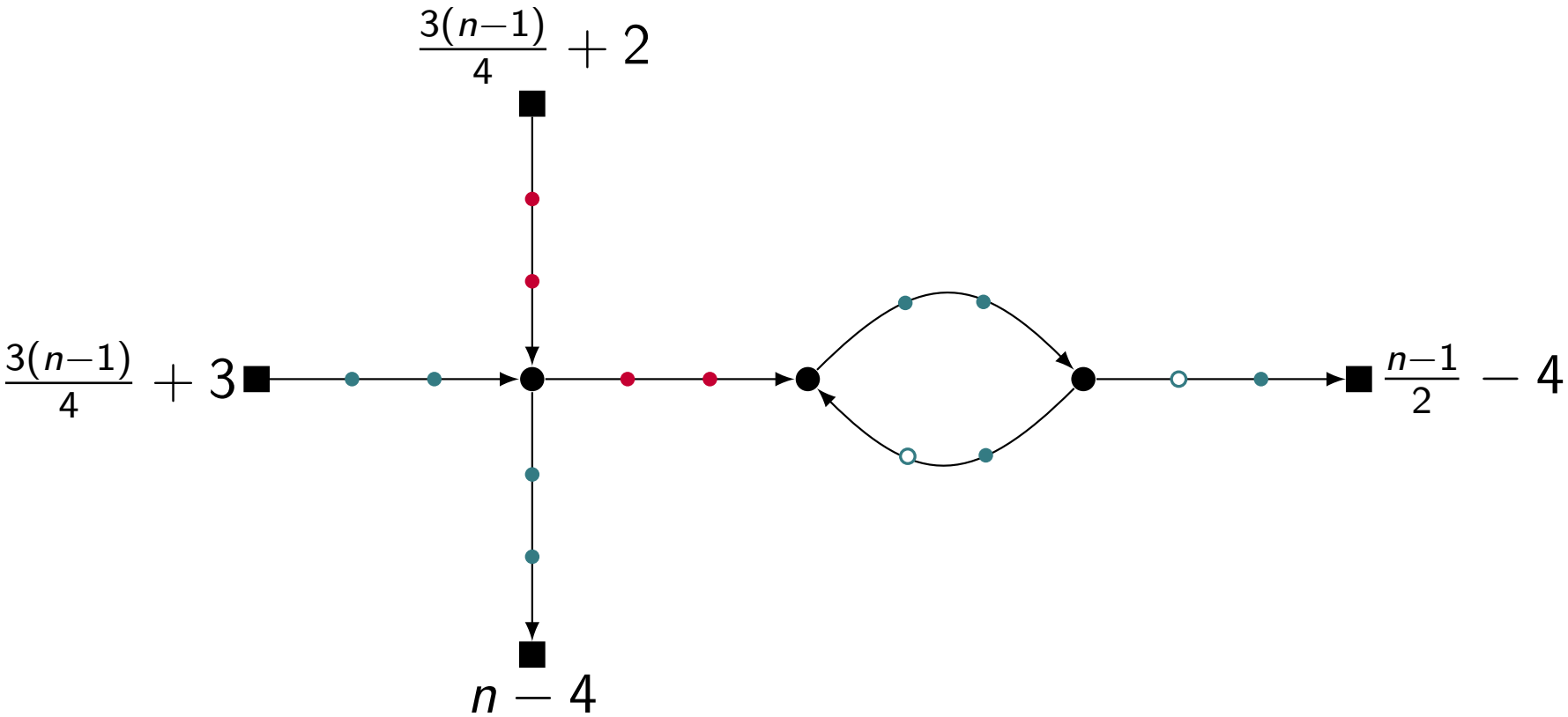
Example of a splitter network dynamics



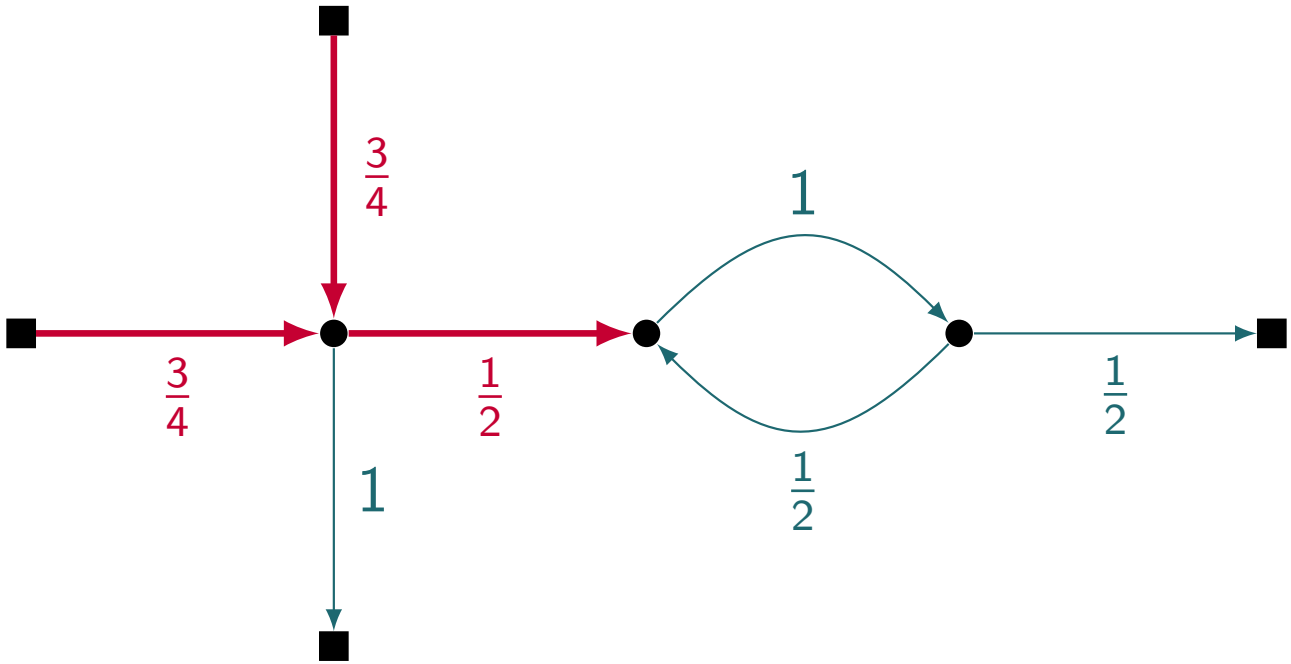
Example of a splitter network dynamics



Example of a splitter network dynamics



The static of a splitter network



A continuous model to study steady-states

- A directed graph $(I \cup S \cup O, E)$;
- I inputs, O outputs with degree 1;
- S splitters with in-degree and out-degree at most 2;
- E belts;
- $c : I \cup O \to [0, 1]$: frequency of item generation (input) or absorption (output);

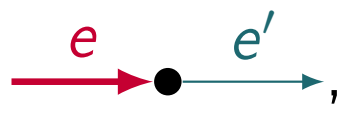
Goal: determine the *throughput* $t : E \to [0, 1]$: frequency of items passing through each arc, in the long run.

Steady-state rules

Steady-state:

- $t : E \to [0, 1]$ throughput function;
- $F \subseteq E$ set of fluid arcs $\longrightarrow$
- $E \setminus F$ set of saturated arcs $\longrightarrow$

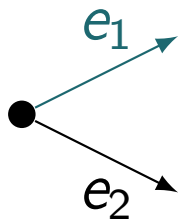
Rules:

- *conservation*: $t(\delta^-(s)) = t(\delta^+(s))$ for any splitter s;
- *maximization*: if 

, then $t(e) = 1$ or $t(e') = 1$;
- ...

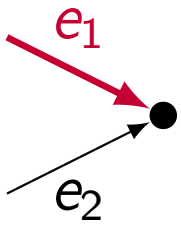
Rules for splitter fairness

A splitter alternates between its outgoing belts, except for full (saturated) belts:

- *out-fairness*: for any 

with e_1 fluid, $t(e_1) \geq t(e_2)$;
- therefore, if the two leaving arcs are fluid, they have equal throughput.

On the incoming belts:

- *in-fairness*: for any 

with e_1 saturated,

  $t(e_1) \geq t(e_2)$;
- therefore, if the two incoming arcs are saturated, they have equal throughput.

Rules for inputs and outputs

An input provides items at a given frequency. Excessive items accumulates, making the outgoing belt saturated.

- *input*: for an input $i \blacksquare\!\xrightarrow{e}$, $t(e) \leq c(i)$, and if $t(e) < c(i)$, e is saturated.

An output consumes items at a given frequency. There is no excess below that frequency.

- *output*: for an output $\xrightarrow{e}\!\blacksquare o$, $t(e) \leq c(o)$, and if $t(e) < c(o)$, e is fluid.

Preliminary remarks

- Splitters networks come from the video game Factorio.
- The definition of steady-states is symmetrical: (t, F) steady-state in $G \Leftrightarrow (t, E \setminus F)$ steady-state in the reverse of G;
- If F is given, the feasible t can be described by a linear system of inequalities, and thus computed efficiently.
- When there is no saturation (only fluid arcs), the discrete model is equivalent to the *rotor-router* model (each vertex routes its passing items cyclically on its outgoing arcs). Rotor-router is a deterministic idealization of a random walk. Stationary distributions will play a role!

Questions

- Existence of steady-states? Algorithms?
- Uniqueness of steady states?
- Design splitter networks with interesting properties. In particular, motivated by Factorio, can we design (minimal) networks with uniform throughput on their output? Can we test these properties?
- Extension to allow belt capacities? Can we make a rate-limiter network for any allowed maximum rate?

Computing a steady-state

Two families of max-flow algorithms

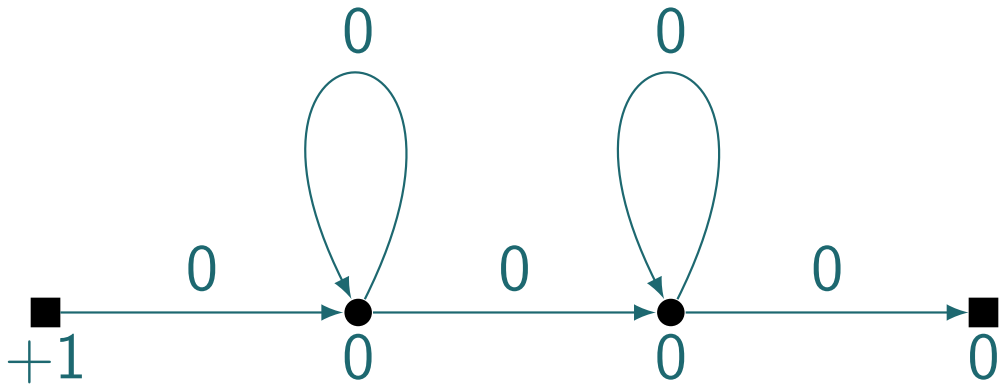
Recall the two classical max-flow algorithms:

- Augmenting paths:
  - ignore the condition of being maximum;
  - improve the flow by increasing along a path or a blocking flow, until reaching maximum.
- Push-relabel:
  - ignore the conservation rule (vertices can have excessive incoming flow);
  - push the excess forward or backward, until there is no excess left.

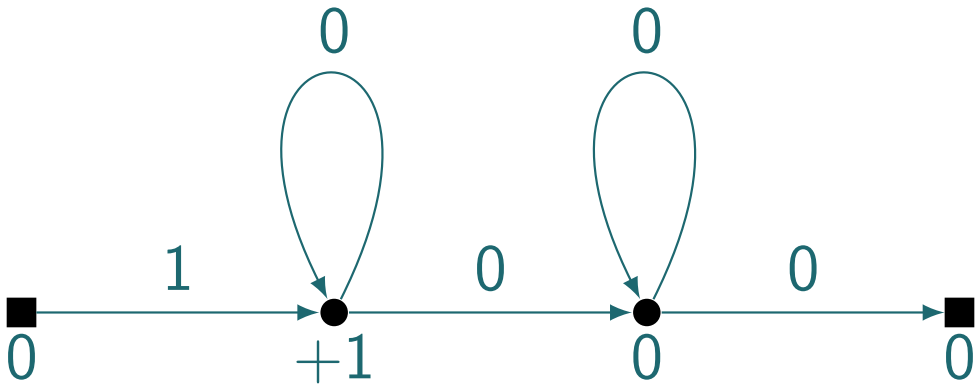
A push-relabel algorithm for steady-state?

- relax the conservation rule (allow flow in excess): *pre-steady-state*;
- push excess forward equally on outgoing fluid arcs;
- when both outgoing arc are full or saturated, make incoming arcs saturated.
- push excess backward equally on incoming saturated arcs;
- repeat until no more excess remain.

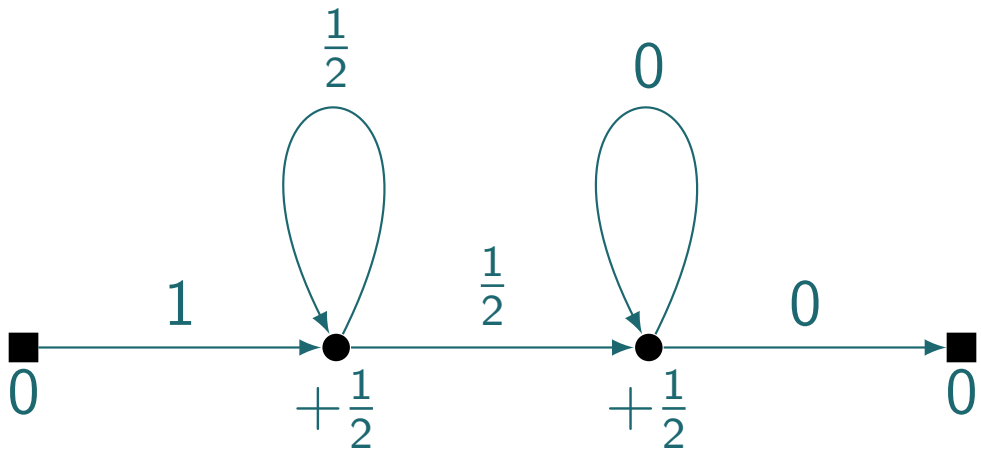
Non-termination of the naive approach



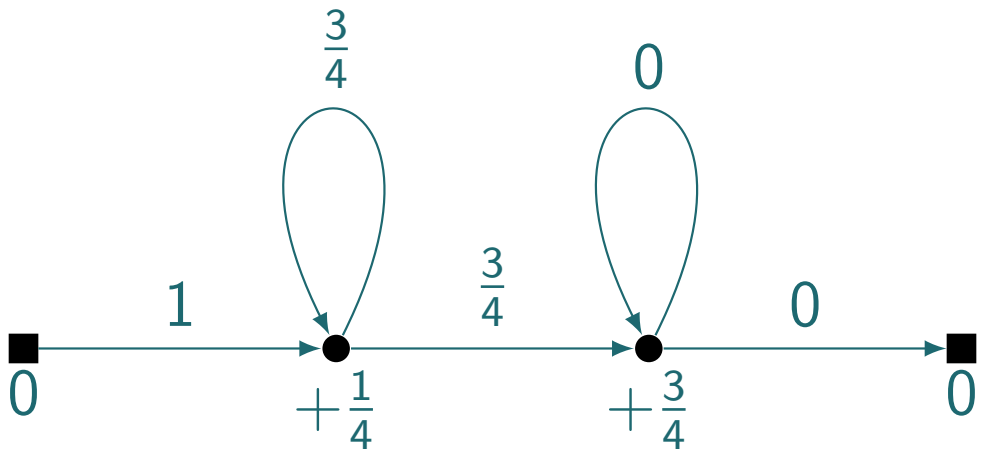
Non-termination of the naive approach



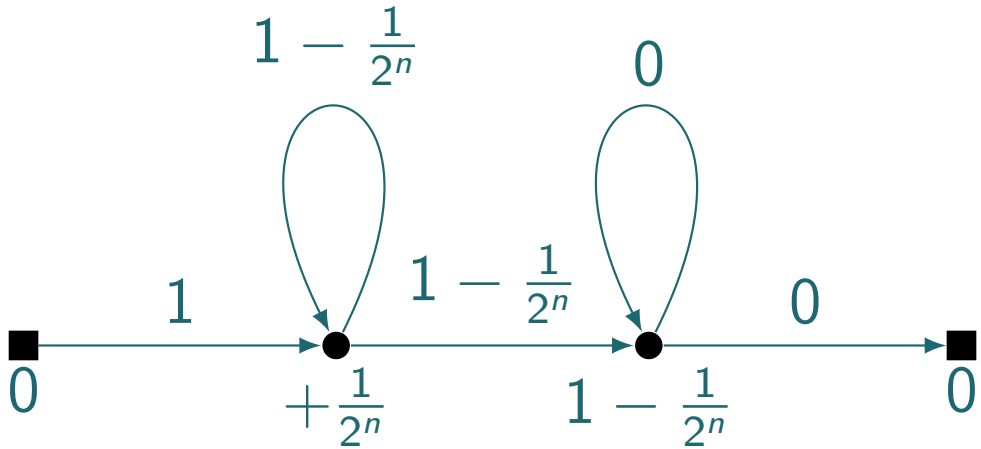
Non-termination of the naive approach



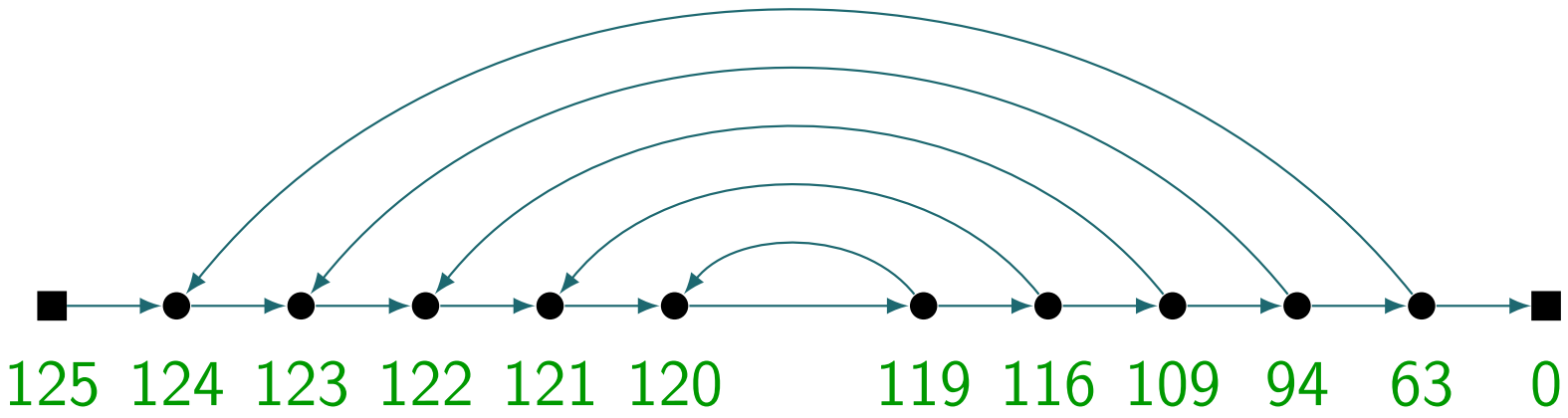
Non-termination of the naive approach



Non-termination of the naive approach



Exponential example



- Let $\phi(u) = 1 + \sum_{v \in N^+(u)} \frac{\phi(v)}{d^+(u)}$ for any $u \in I \cup S$, and $\phi(o) = 0$ for $o \in O$.
- pushing ϵ units of excess flow decreases $\sum_{s \in S} \phi(s)\mathrm{exc}(s)$ only by ϵ.

A terminating push-relabel algorithm.

Solution:

- keep the mecanism to move arcs from fluid to saturated;
- use a linear program to push as much flow as possible. More precisely, F is fixed, and we find t which maximizes

$$t(F) - t(E \setminus F)$$

Then by the optimality property,

- either there is $e \in F$, such that $(t, F \setminus e)$ is a pre-steady-state;
- or (t, F) is a steady-state.

Terminates in at most $|E|$ rounds.

An augmenting path algorithm?

- Relax the input rule (allow fluid input belts below capacity): *sub-steady-state*;
- Define a residual graph: fluid arcs + reversed saturated arcs;
- Find a circulation with equal flow on outgoing arcs of any vertex;
- Increase the sub-steady-state along that circulation.
  - Circulation: conservation is preserved;
  - equal flow on outgoing arcs: fairness is preserved.
- If no circulation exists, find a *sink* in the residual graph, and move some arc from fluid to saturated.
  - Preserves maximality rule.

the $\mathcal{C}^=$-circulation problem

Problem: Given G digraph, $\mathcal{C}^=$ a partition of $E(G)$, find a non-zero circulation f on G, where for each $F \in \mathcal{C}^=$, f is constant on F.

- easily solvable by LP;
- max integer flow variant is NP-hard (Meyers-Schulz).

Special case: $\mathcal{C}^=$ is a refinement of $(\delta^+_G(u))_{u \in V(G)}$.

Good characterization

$$\begin{cases} x(\delta^+(v)) - x(\delta^-(v)) = 0 & (v \in V) \\ x_e - x_{e'} = 0 & (e, e' \in C \in \mathcal{C}^=) \\ x \geq 0 \end{cases} \quad (1)$$

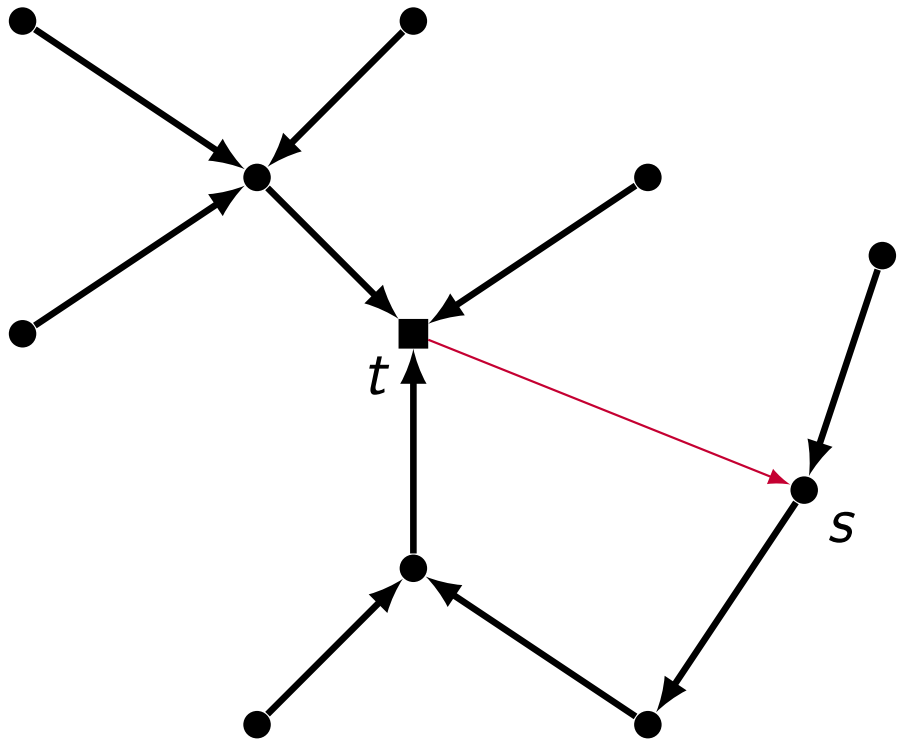
By Farkas lemma:

Theorem

$(G = (V, E), ts, \mathcal{C}^=)$ has a $\mathcal{C}^=$-circulation x with $x_{ts} > 0$ iff there is no set $S \subseteq V \setminus \{t\}$ with a partition $S = S_0 \uplus S_1 \ldots \uplus S_k$ where

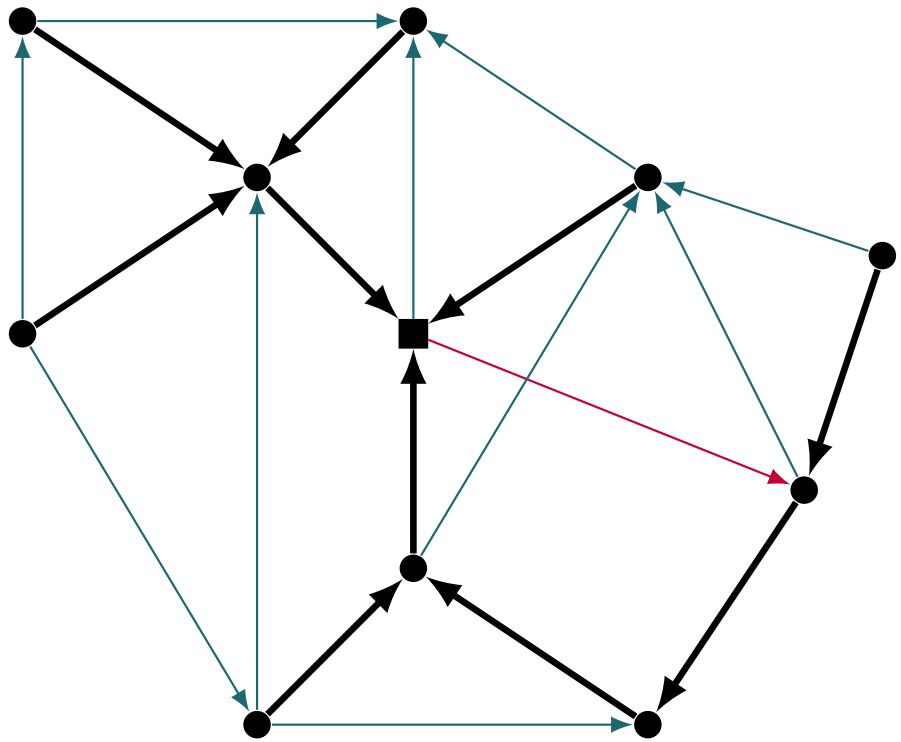
- *$s \in S_0$;*
- *for any $C \in \mathcal{C}^=$, $e \in C$, there is $e' \in C$ (possibly $e = e'$) such that $e' \in E[S_i, S_j]$ and $i < j$.*

Solving in strongly connected graph



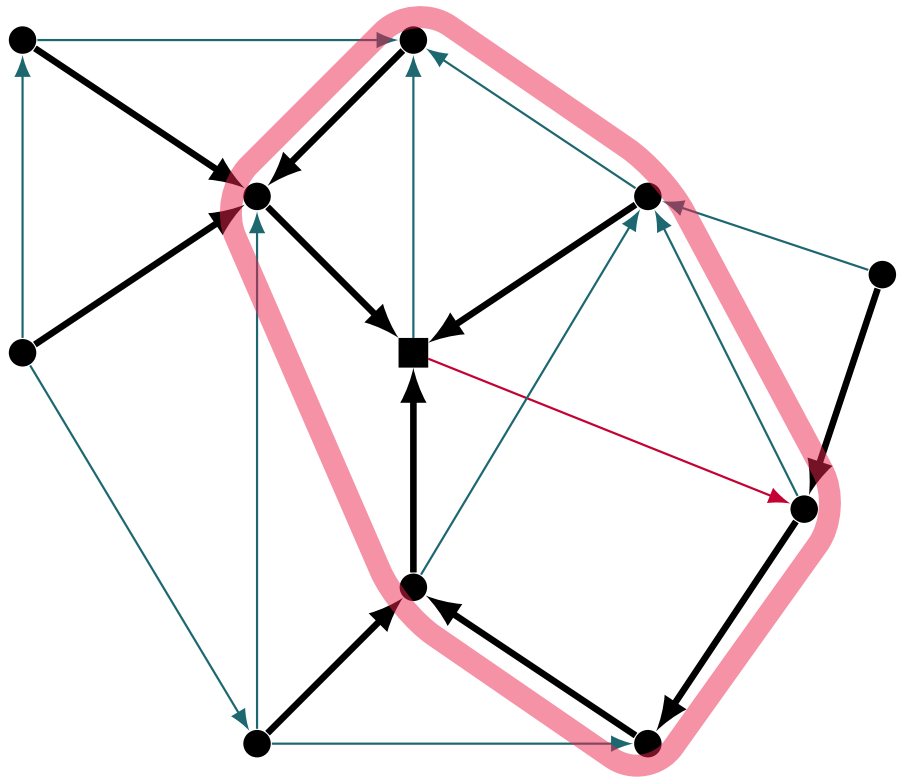
- T in-arborescence;

Solving in strongly connected graph



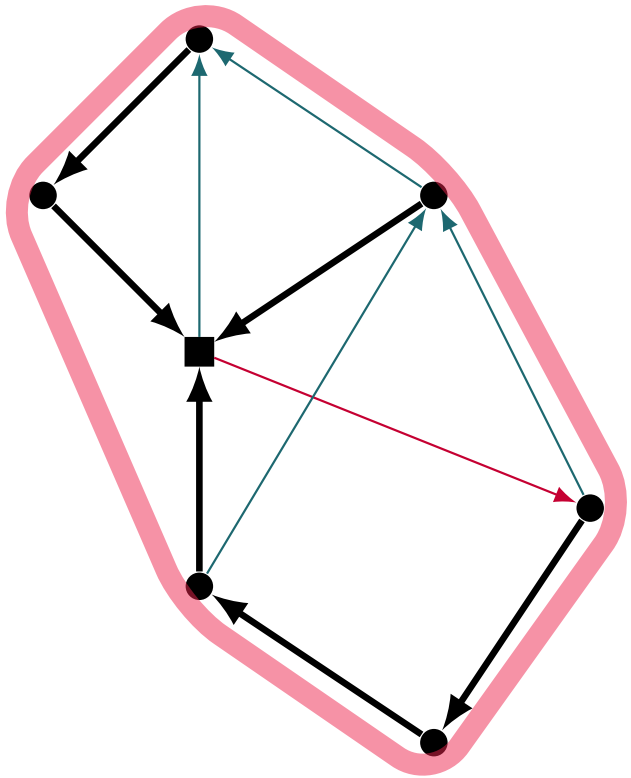
- T in-arborescence;
- $\bigcup_{e \in T} C_e$;

Solving in strongly connected graph



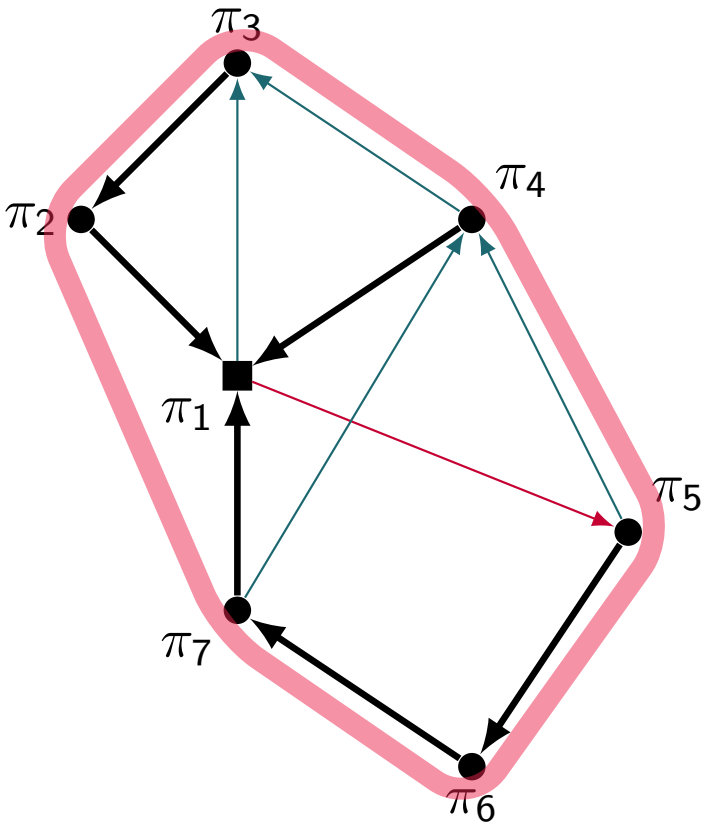
- T in-arborescence;
- $\bigcup_{e \in T} C_e$;
- H SCC with $ts \in H$;

Solving in strongly connected graph



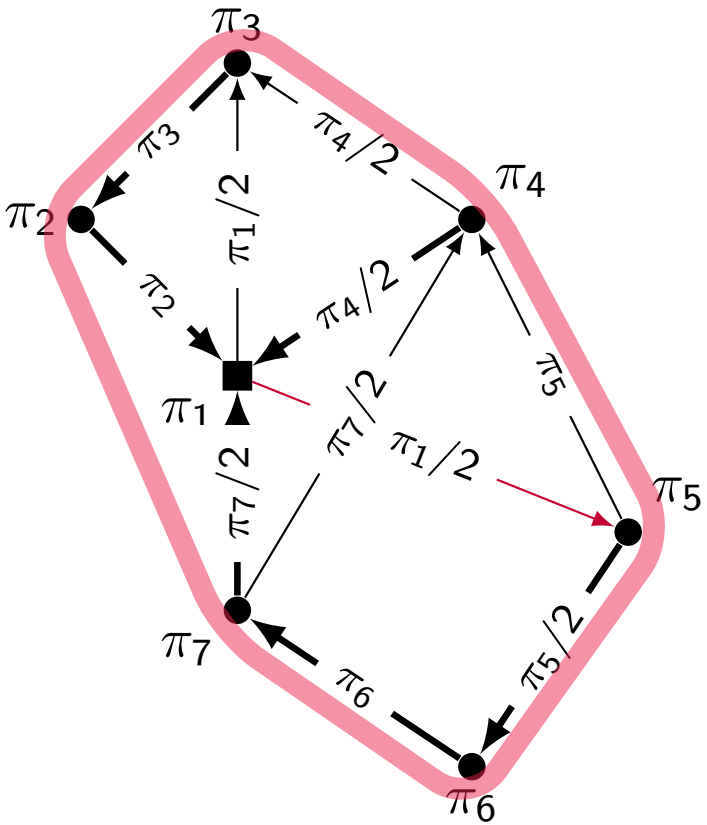
- T in-arborescence;
- $\bigcup_{e \in T} C_e$;
- H SCC with $ts \in H$;

Solving in strongly connected graph



- T in-arborescence;
- $\bigcup_{e \in T} C_e$;
- H SCC with $ts \in H$;
- π stationary distribution;

Solving in strongly connected graph



- T in-arborescence;
- $\bigcup_{e \in T} C_e$;
- H SCC with $ts \in H$;
- π stationary distribution;
- $f(uv) = \frac{\pi(u)}{d^+(u)}$.

Solving in general graphs

Lemma

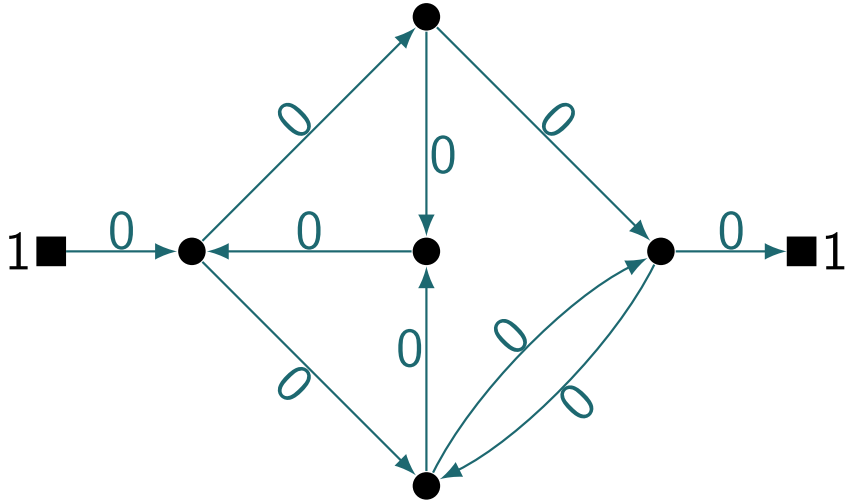
Algorithm in time $O(|E| \log^4 |V| + \mathrm{sd}(G))$ *that*

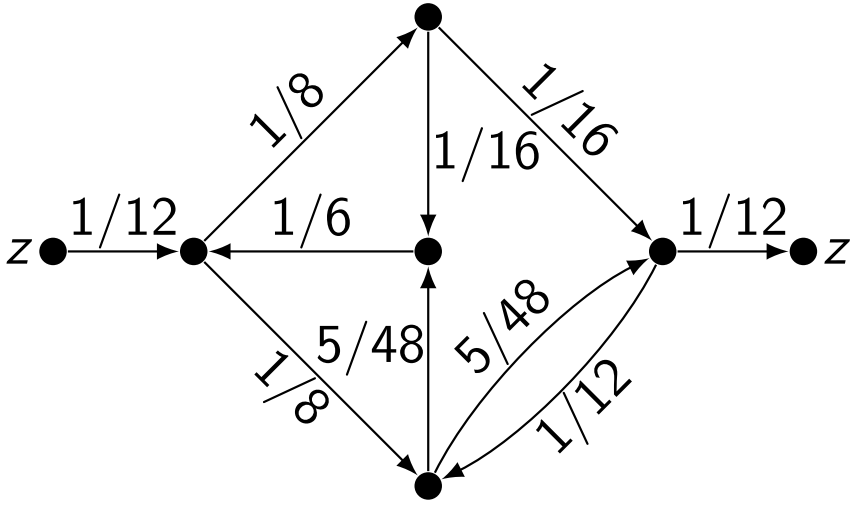
- *either find a feasible solution* x *with* $x_{ts} > 0$*;*
- *or correctly assert that none exists.*

Lemma

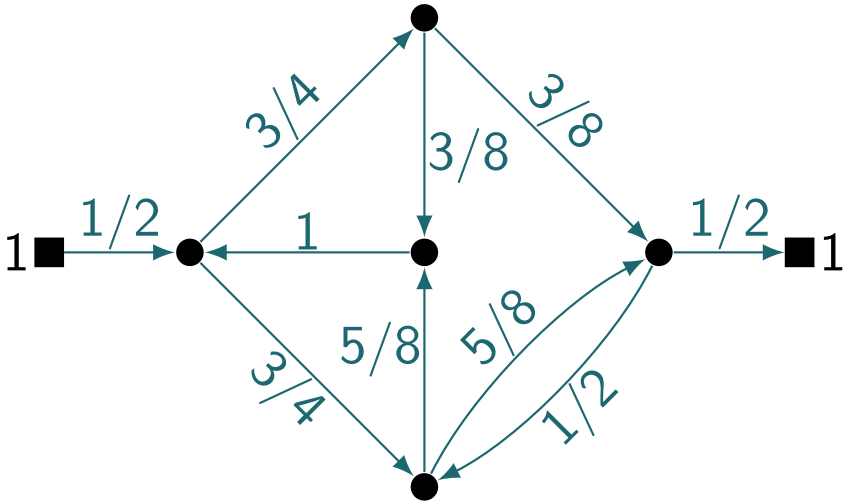
- *Either there is a non-zero* $\mathcal{C}^=$*-circulation in* G*;*
- *or there is a vertex* $v \in V$ *with* $\delta^+(v) = \emptyset$*.*

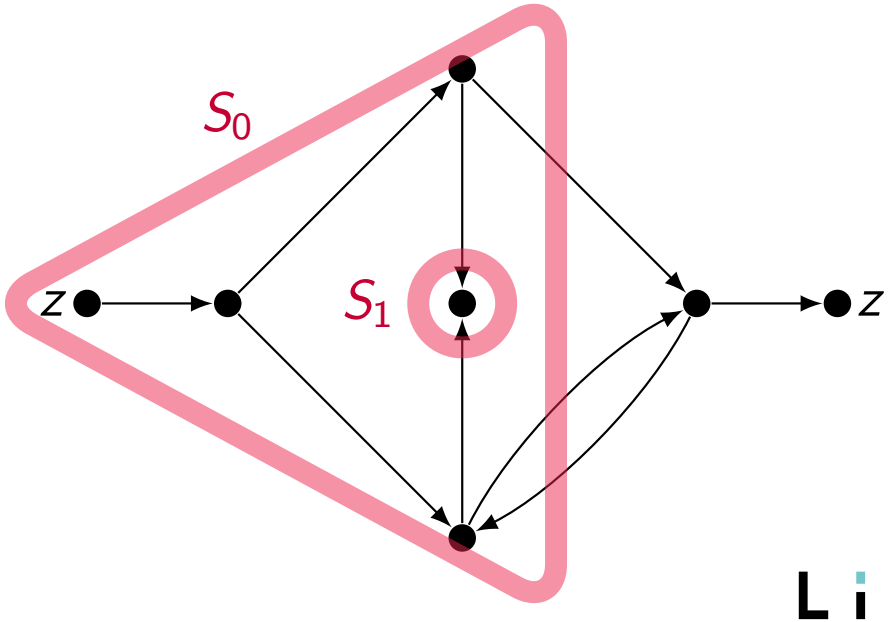
Example of the sub-steady-state algorithm



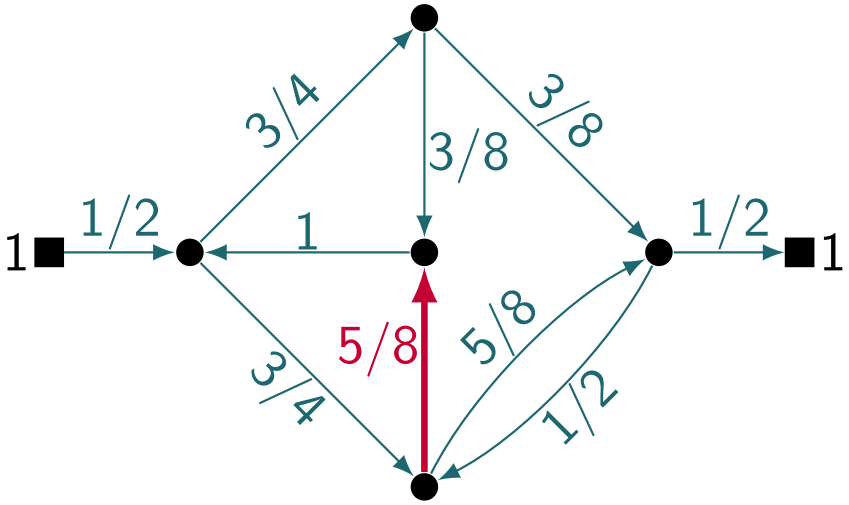


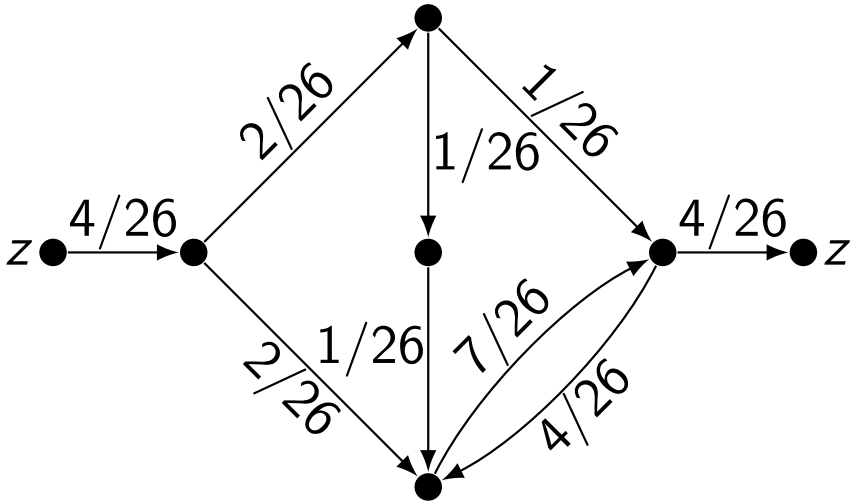
Example of the sub-steady-state algorithm



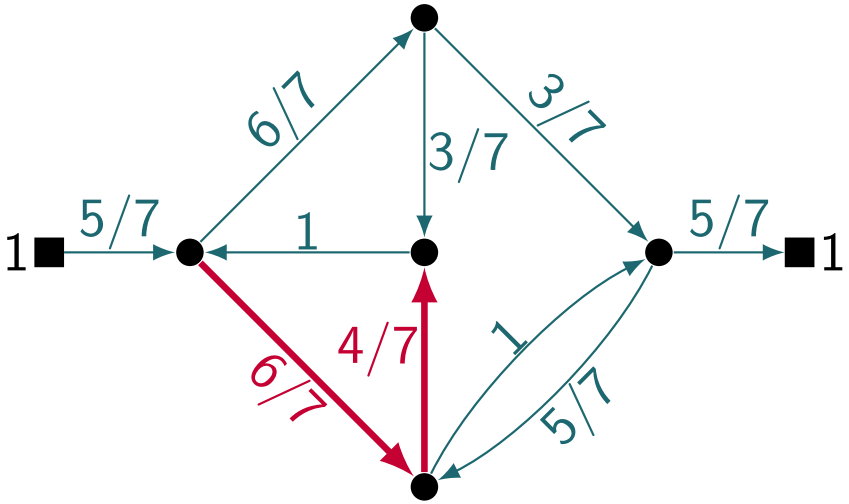


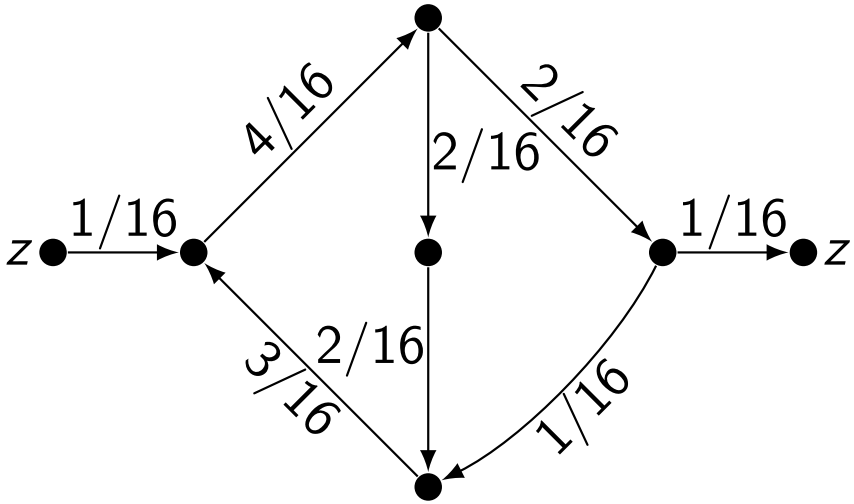
Example of the sub-steady-state algorithm



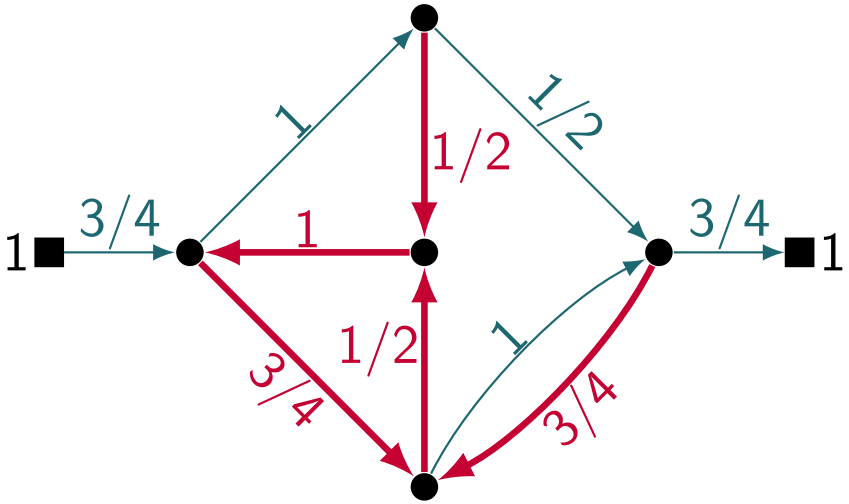


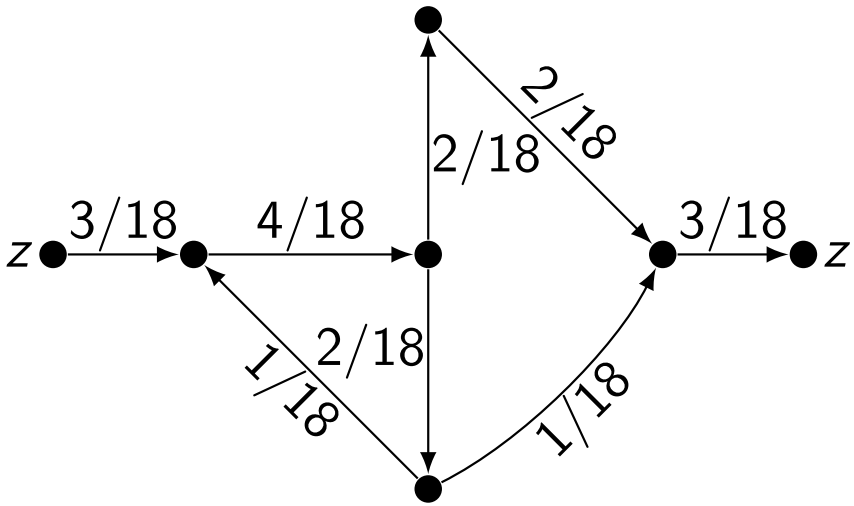
Example of the sub-steady-state algorithm



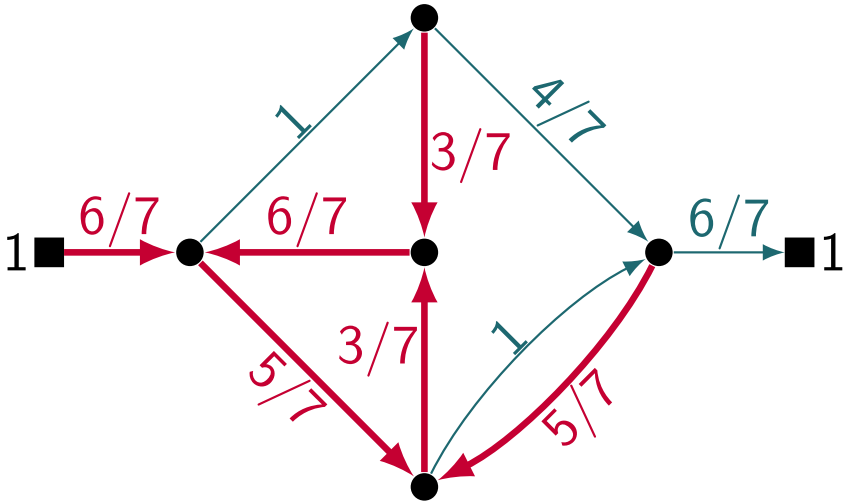


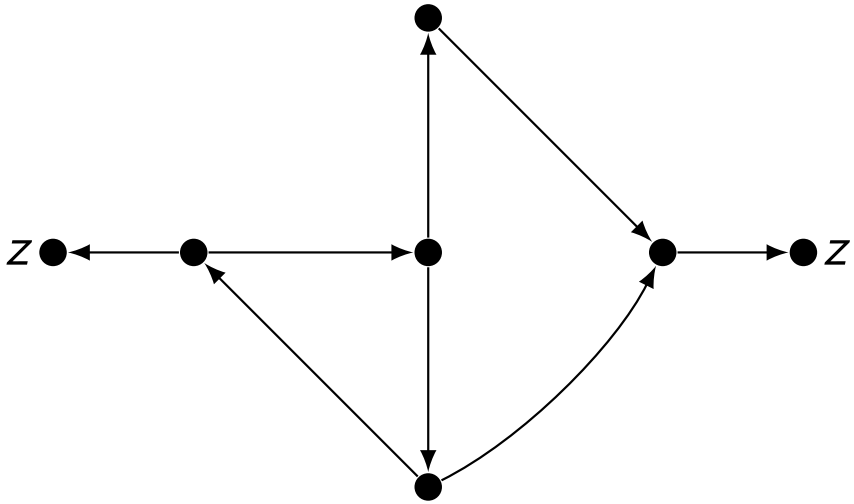
Example of the sub-steady-state algorithm





Example of the sub-steady-state algorithm





The sub-steady-state algorithm

- computes a steady-state;
- by augmenting along *stationary circulation*;
- in $O(m)$ rounds, and at most as many computations of a stationary distribution in the residual graph.

Some consequences

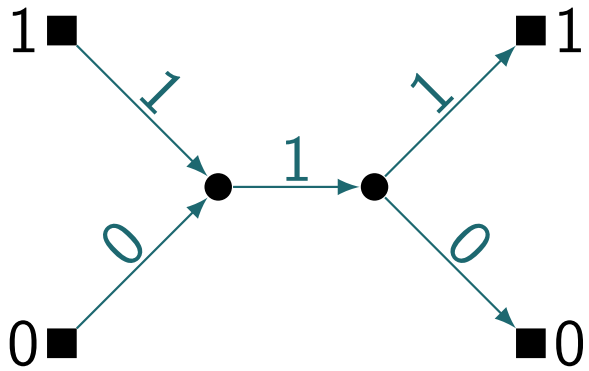
Increasing the input capacities:

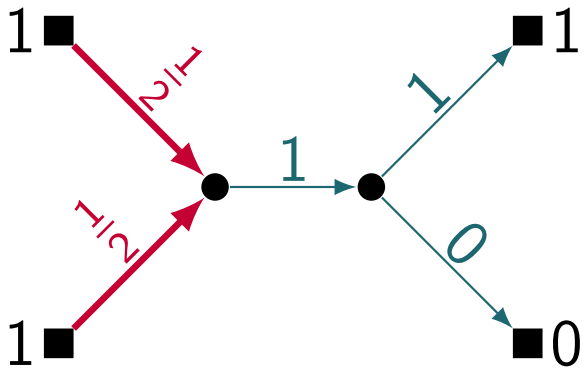
- increases the throughput on fluid arcs, and decreases it on saturated arcs;
- cannot decrease the throughput on any ouput;
- may decrease the throughput on some input!

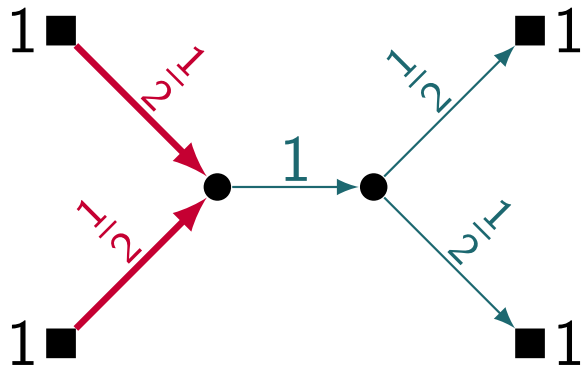
Increasing the output capacities:

- decreases the throughput on fluid arcs, and increases the throughput on saturated arcs;
- cannot decrease the throughput on any input;
- may decrease the throughput on some output!

Example of non-monotonicity

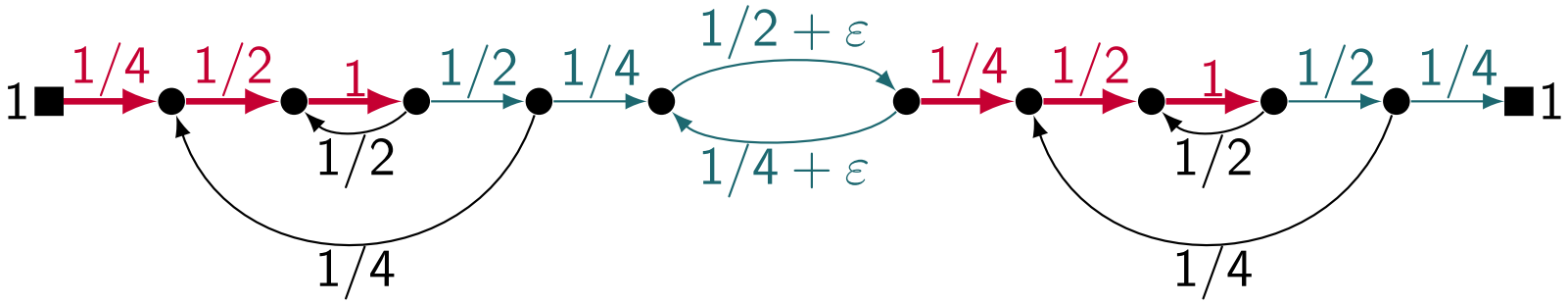






Uniqueness

Steady-states are not unique.



Unique throughputs

Theorem: Steady-states have the same throughputs on their inputs and outputs.

Definition: *Uniform sub-steady states* = steady-states where all fluid inputs have equal throughput γ.

$$\text{For any } is \in \delta^+(I), \text{ if } is \in F, \text{ then } t(is) = \max\{c(i), \gamma\}.$$

Recall the algorithm

The sub-steady-state algorithm iterates 3 operations:

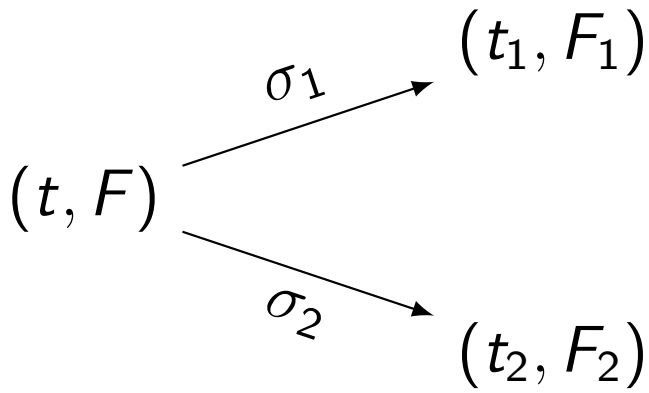
- augment(f): augment along a $\mathcal{C}^=$-circulation f that improves the total throughput;
- move(f): augment along a $\mathcal{C}^=$-circulation f that does not improve the throughput;
- saturate(e): remove an arc e from F.

Performs a sequence of these three operations:

$$(0, E) \xrightarrow{\sigma_1} (t_1, F_1) \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_k} (t^\star, F^\star)$$

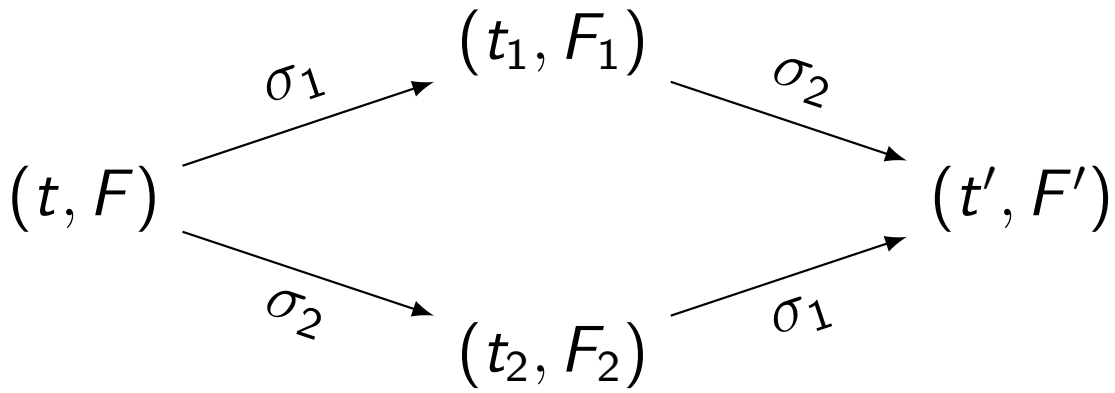
Confluence of operations

Proposition: these operations are locally confluent.



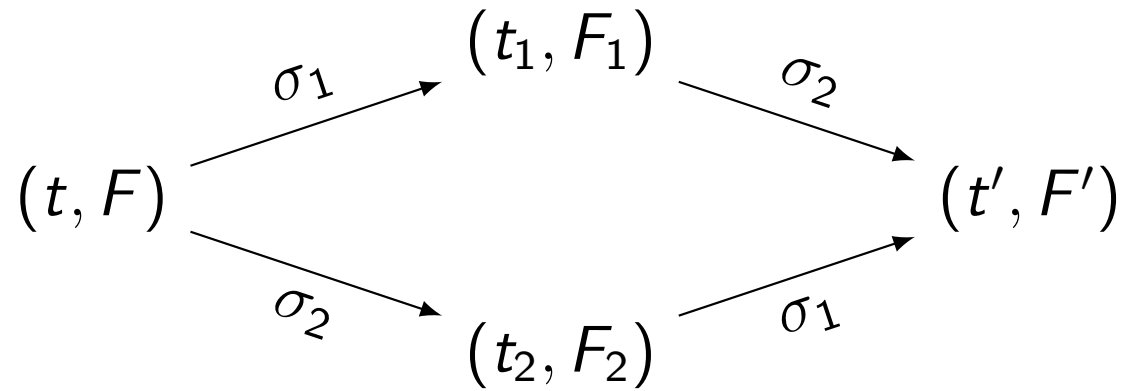
Confluence of operations

Proposition: these operations are locally confluent.

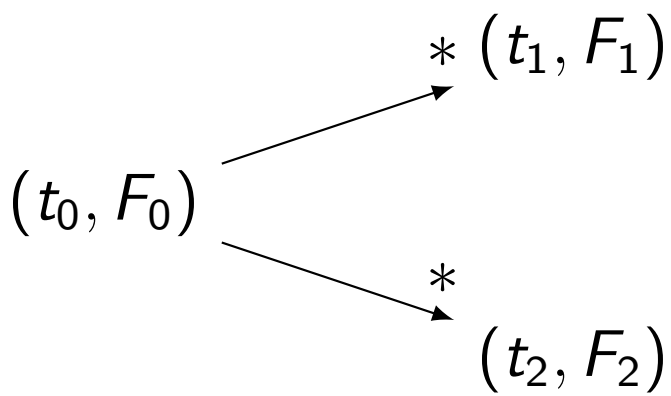


Confluence of operations

Proposition: these operations are locally confluent.

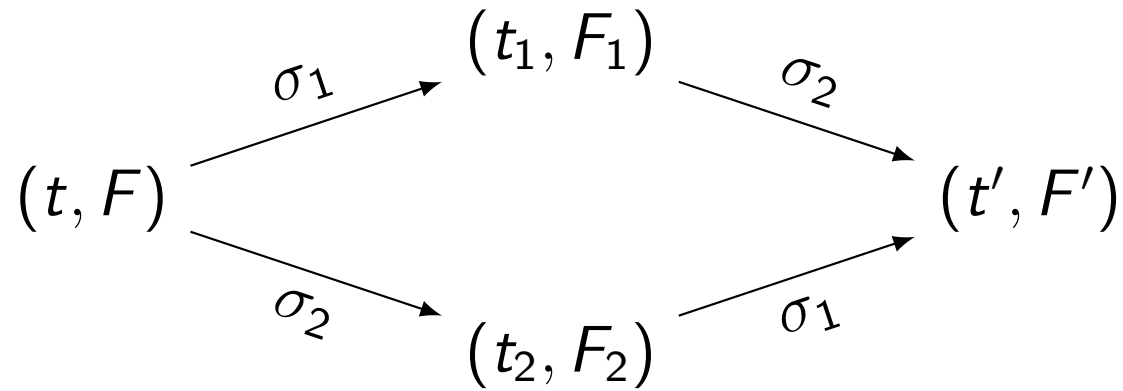


Consequence:

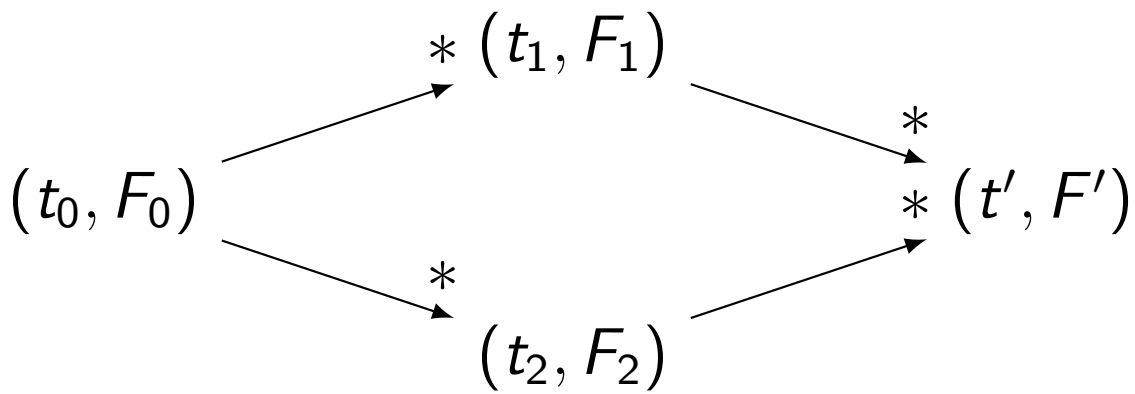


Confluence of operations

Proposition: these operations are locally confluent.



Consequence:



Any steady-state is reachable

Proposition: Any steady-state is reachable by a sequence of operations of the uniform sub-steady-state algorithm.

+ Confluence: steady-states have unique throughputs on $I \cup O$.

Proposition: The valid sequences of operations from $(0, E)$ is an antimatroid. The uniform sub-steady-states form a semimodular lattice.

The join and meet operations

$(t_1, F_1) \wedge (t_2, F_2) = (t_0, F_1 \cup F_2)$ and
$(t_1, F_1) \vee (t_2, F_2) = (t', F_1 \cap F_2)$ with

$$t_0(e) = \begin{cases} \max\{t_1(e), t_2(e)\} & \text{if } e \in F_1 \cap F_2 \\ t_1(e) & \text{if } e \in F_1 \setminus F_2 \\ t_2(e) & \text{if } e \in F_2 \setminus F_1 \\ \min\{t_1(e), t_2(e)\} & \text{if } e \notin F_1 \cup F_2 \end{cases}$$

$$t'(e) = \begin{cases} \min\{t_1(e), t_2(e)\} & \text{if } e \in F_1 \cap F_2 \\ t_2(e) & \text{if } e \in F_1 \setminus F_2 \\ t_1(e) & \text{if } e \in F_2 \setminus F_1 \\ \max\{t_1(e), t_2(e)\} & \text{if } e \notin F_1 \cup F_2 \end{cases}$$

Balancer networks (definitions and upper bounds)

Designing balancers

Factorio players need a family of splitter networks:

- arbitrary many inputs;
- arbitrary many outputs;
- for any input capacities, the output throughputs are all equals.
- Motivation 1: Balances the throughput between multiple belts;
- Motivation 2: reduce or increase the number of belts, for instance going from 5 input belts to 3 output belts.

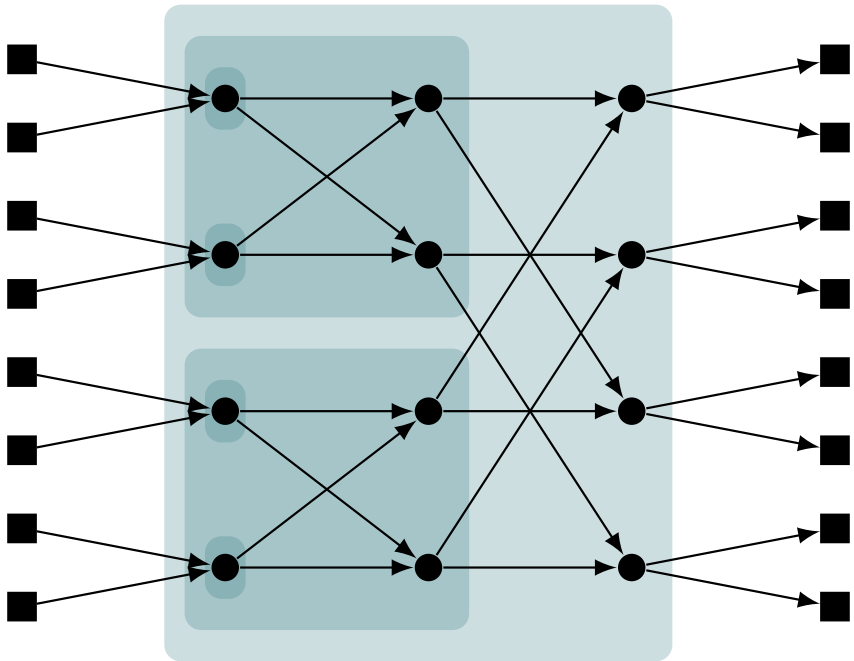
Such networks are informally called *balancers*.

- Minimize CPU cost $\Leftrightarrow$ minimize the number of splitters in a balancer.

The simple balancer

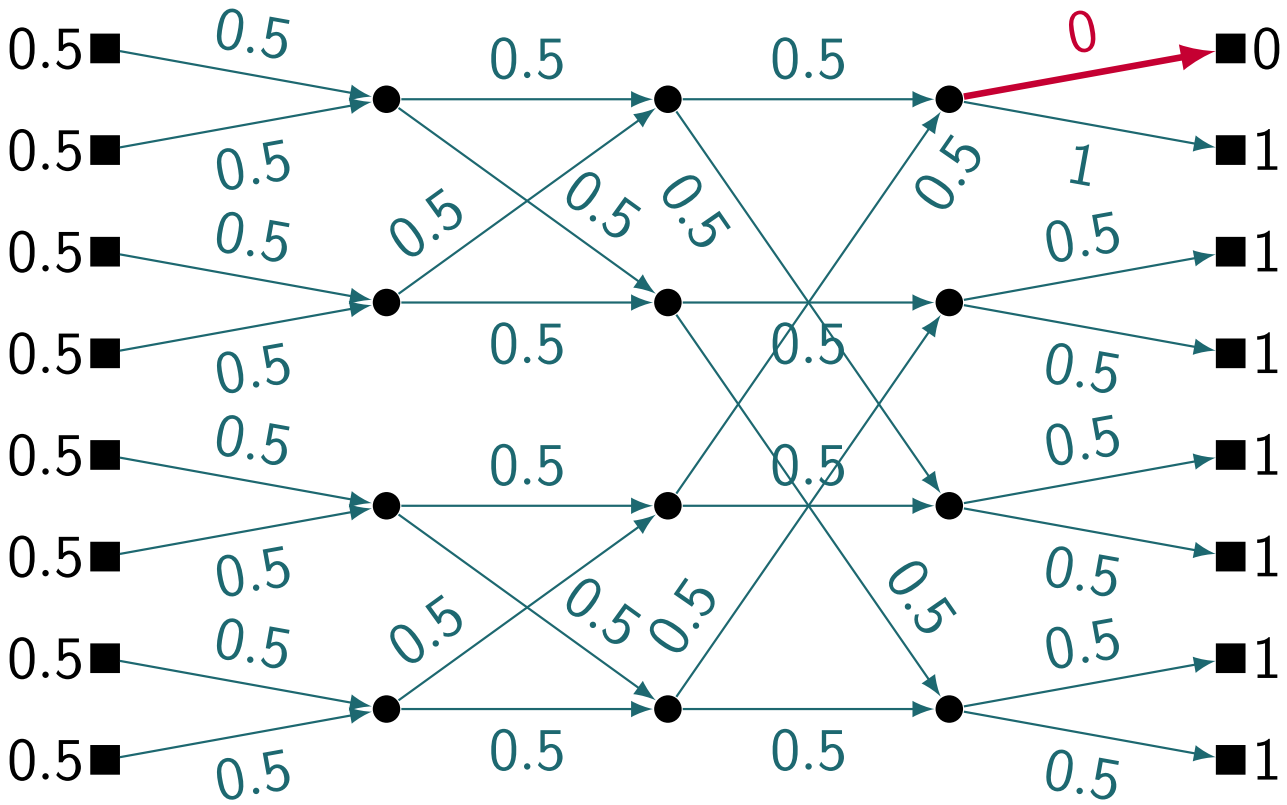
Balancing property: all outputs have capacity one, for any choice of input capacities, the output throughputs are equal.

The simple balancer, a recursive construction:



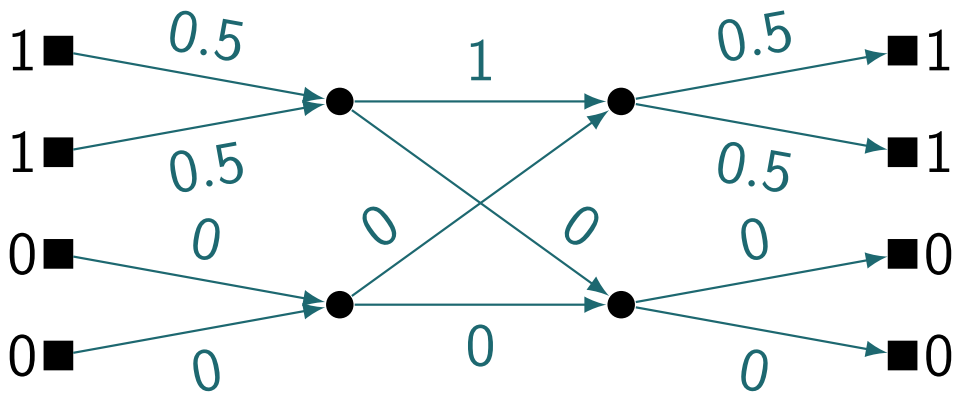
Limitations of the simple balancer (1)

Not balancing when output capacities are not uniform.



Limitations of the simple balancer (2)

Limiting the total throughput when some output capacities are less than 1.

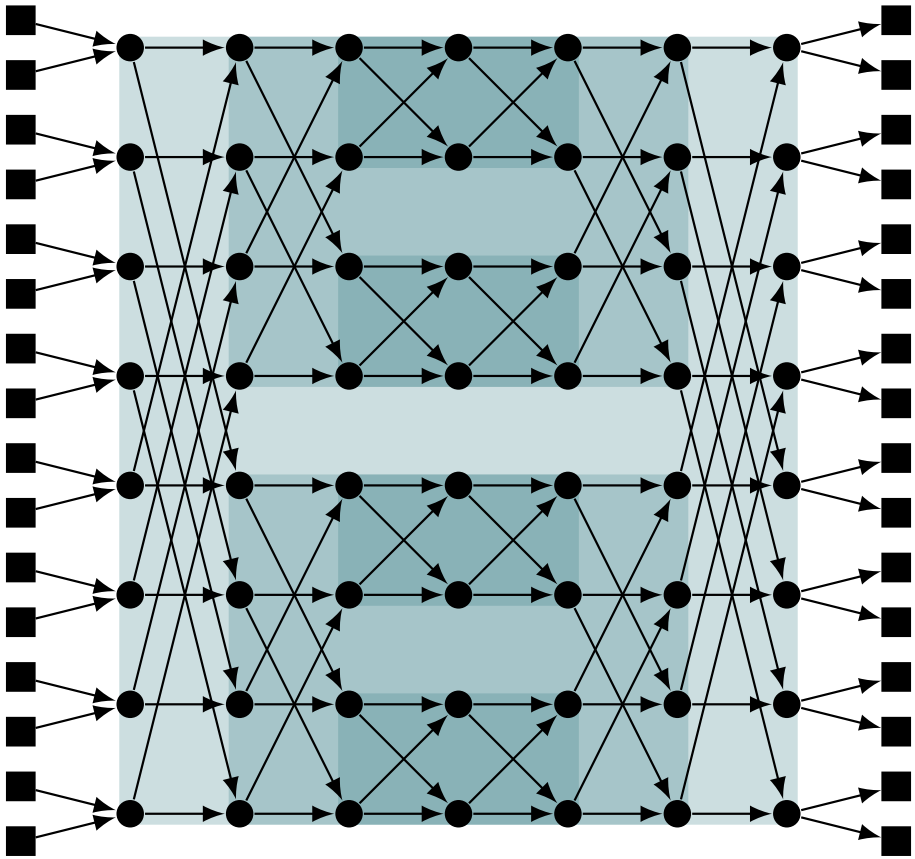


The throughput-unlimited balancer

Throughput-unlimited (TU) property: The total throughput is always the minimum of the input capacities and the output capacities (the network is never a bottleneck).

Construction: glue a simple balancer with the reverse of a simple balancer. Known as the Beneš network, both balancing and TU.

The Beneš network

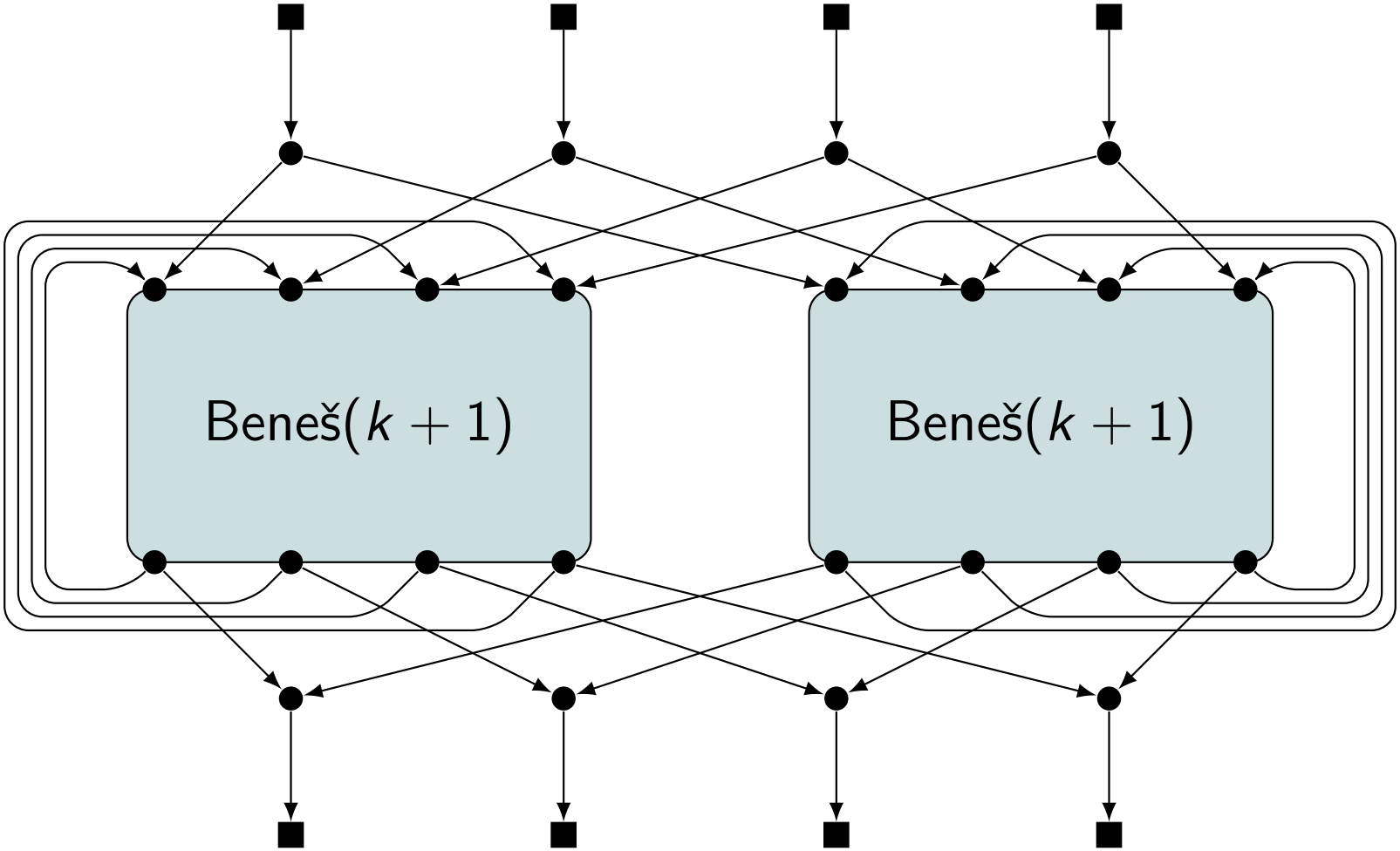


Universal property

- Balancer properties assumes output capacities are 1;
- Even Beneš network is not balancing when output capacities are not uniform;
- Can we design a splitter network whose output throughputs are equals on fluid outputs (recall that saturated outputs have throughput equal to capacity)?

Call this *the universal property!*

A universal network



Remarks on balancer designs

- These designs were proposed by players (with the universal network slightly modified);
- we proved that those designs are correct;
- for n inputs and outputs, $\Theta(n \log n)$ splitters;
- from the universal network, we can ignore some inputs and outputs: universal network with arbitrary number of inputs and outputs;
- another technique used by players is to loop back superfluous outputs to superfluous inputs (beware that it may break the balancing property if not done carefully).

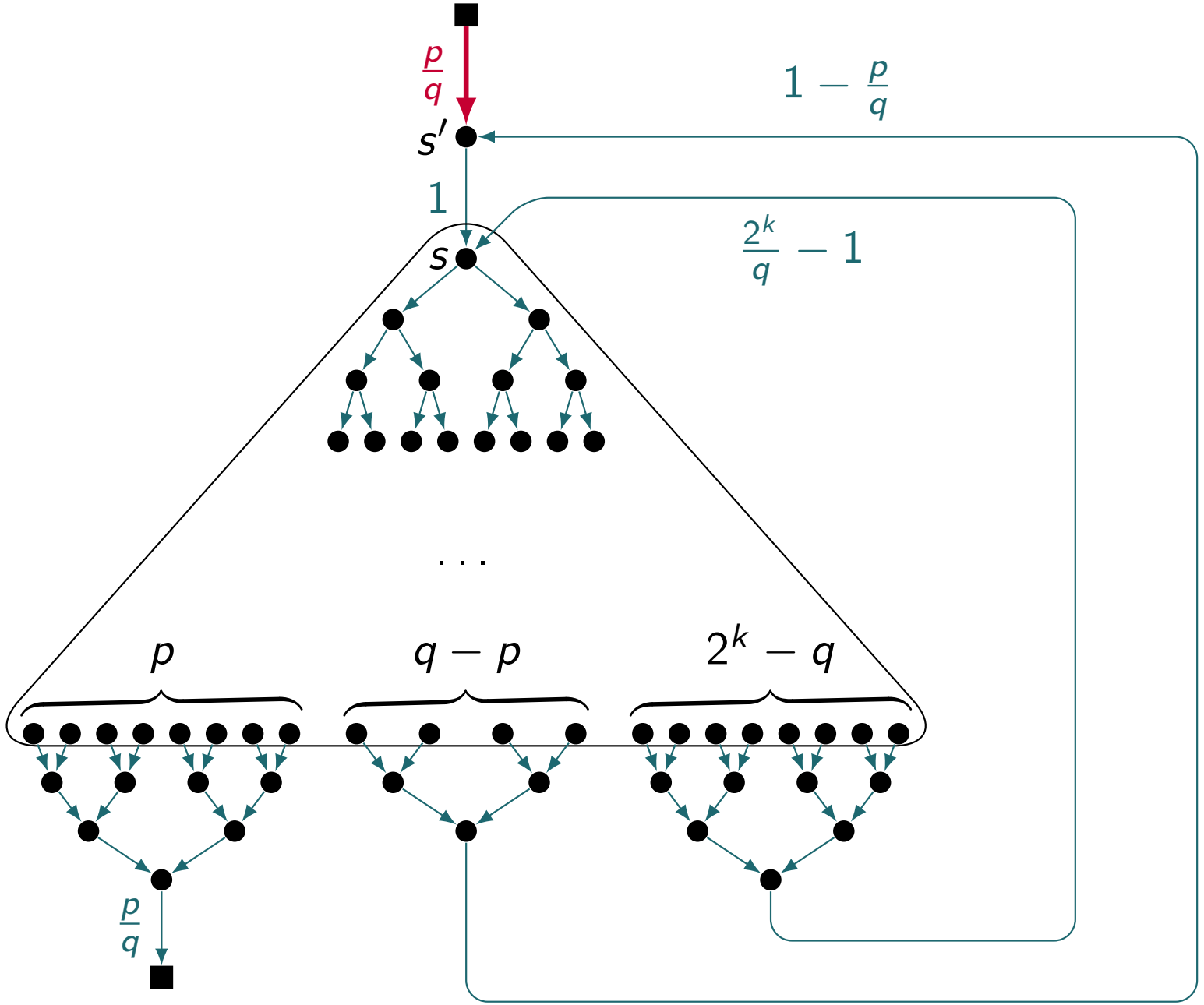
Rate-limiters

Rate-limiters

Another design, used to limit the throughput on a single belt.

- For a rational r, design a single-input, single-output splitter network;
- there is a steady-state with throughput $\min\{c(i), r, c(o)\}$;
- with a minimum number of splitters;
- Motivation: reduce the maximum throughput in a single belt;
- Not possible for irrational r (throughputs are solutions of rational linear systems).

Rate-limiter: naive idea

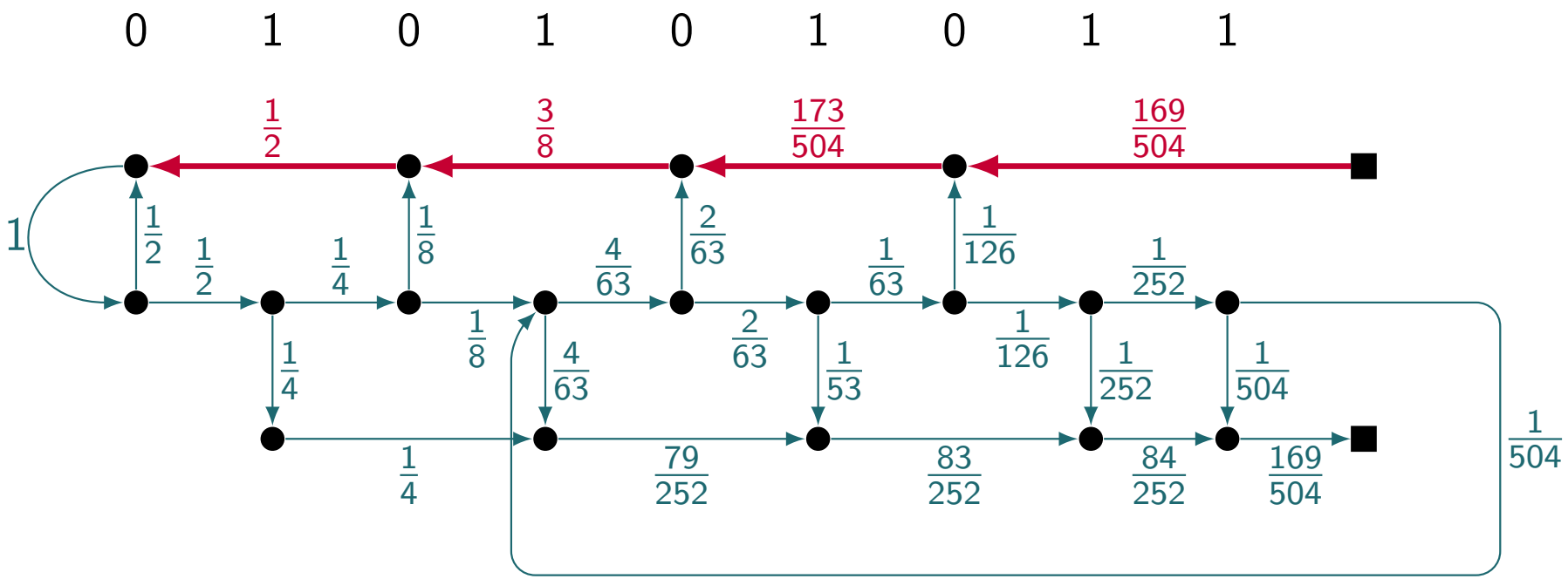


Improving over the binary tree

- Naive construction as $O(q)$ splitters;
- can contract some subtrees into single nodes, to get $O(\log q)$ splitters;
- works for $\frac{p}{q} \geq \frac{1}{2}$, can be modified to work with $\frac{p}{q} < \frac{1}{2}$;
- Another construction with number of splitters logarithmic in the binary representation of $r = \frac{p}{q}$.

A binary-based construction.

Construction for $r = \frac{169}{504} = 0.010(101011)^\omega$



Balancers (lower bounds)

Minimize the number of splitters

Number of splitters in balancers on 2^k inputs and outputs:

- Simple balancer: $S(k) = k \cdot 2^{k-1}$;
- Benes network: $B(k) = (2k-1)2^{k-1}$;
- Universal balancer: $U(k) = (k+1)2^{k+1}$.

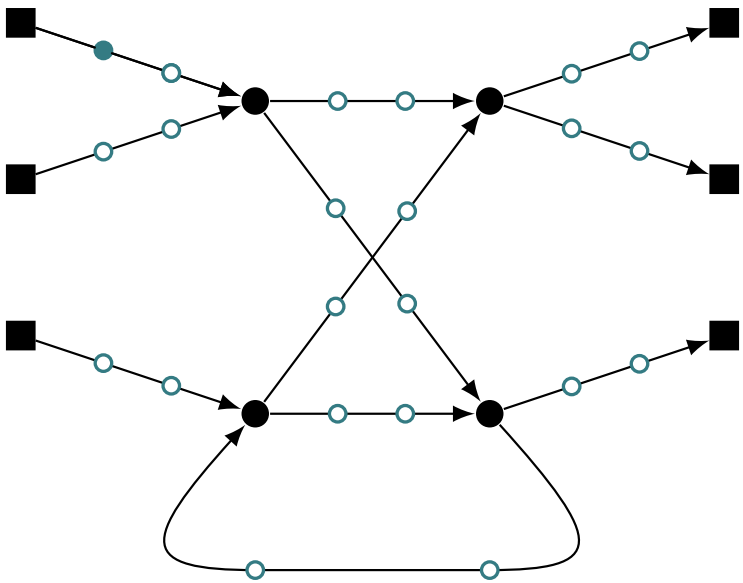
Goal: an $\Omega(k2^k)$ lower bound would prove that these designs are almost optimal.

Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

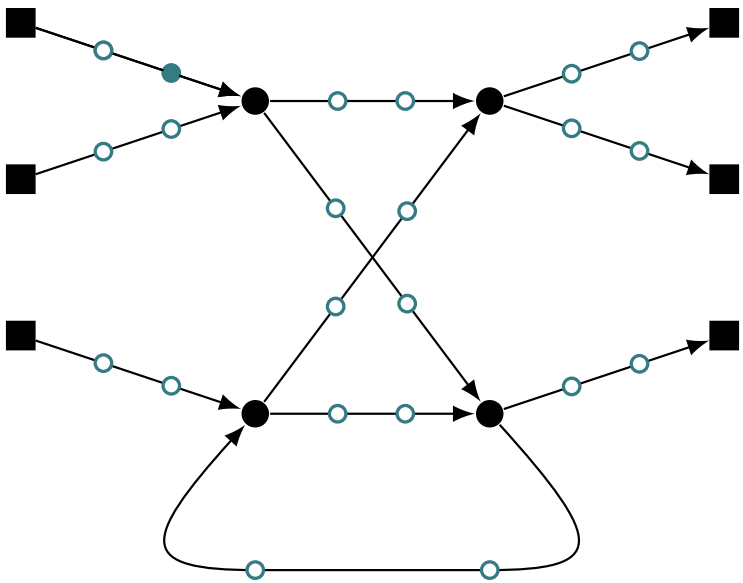


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

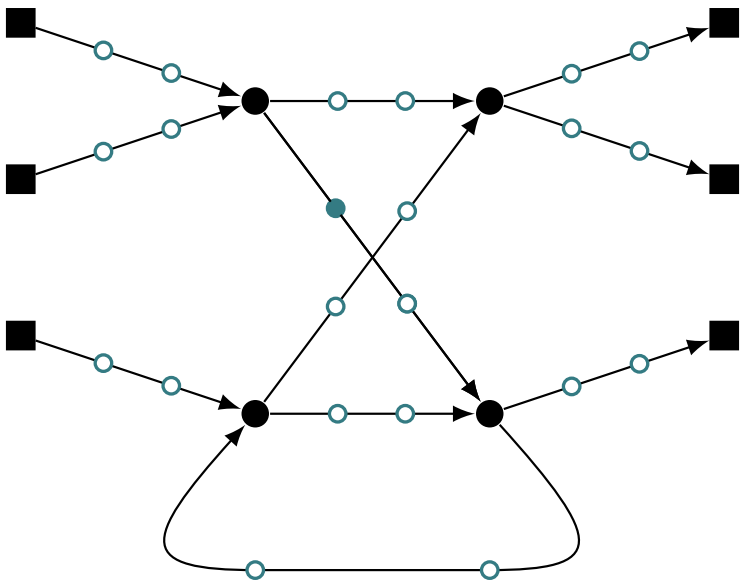


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

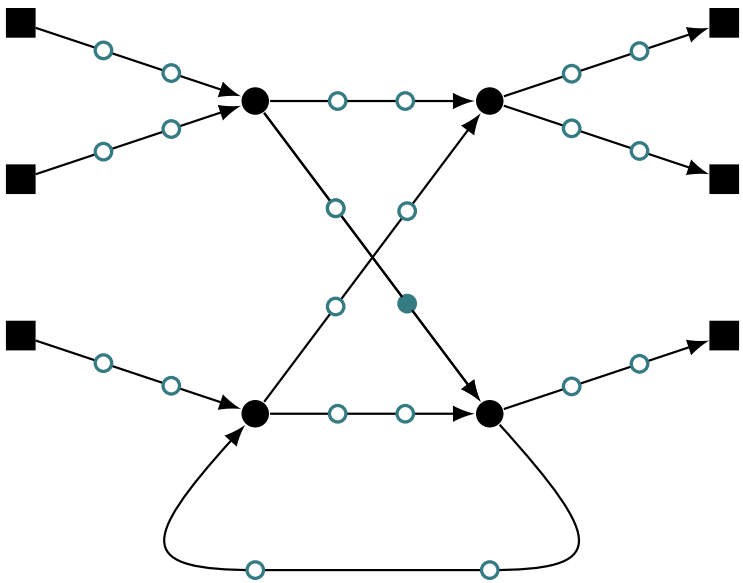


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

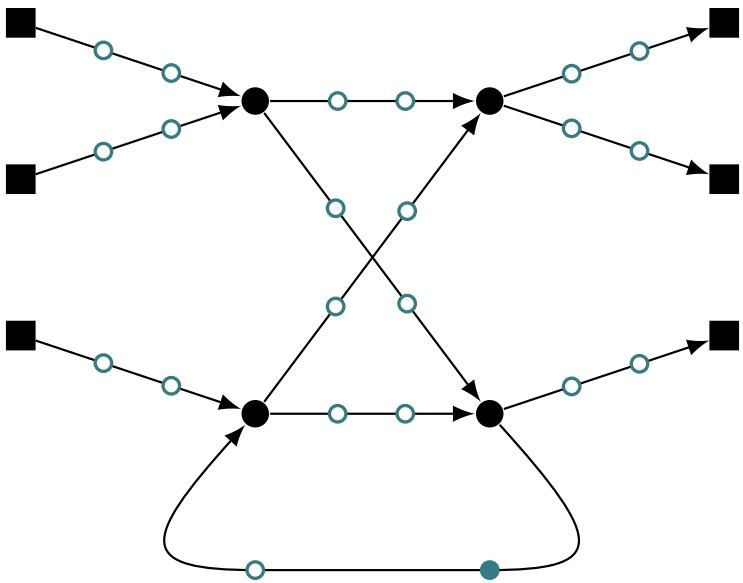


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

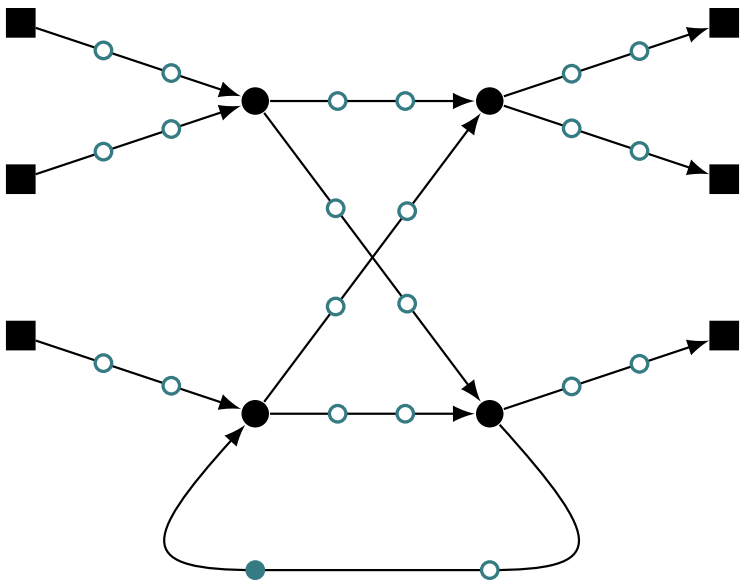


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

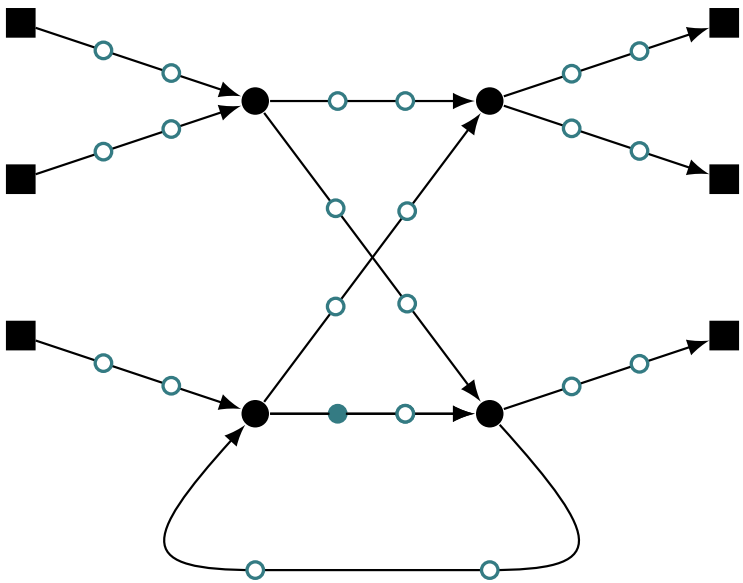


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

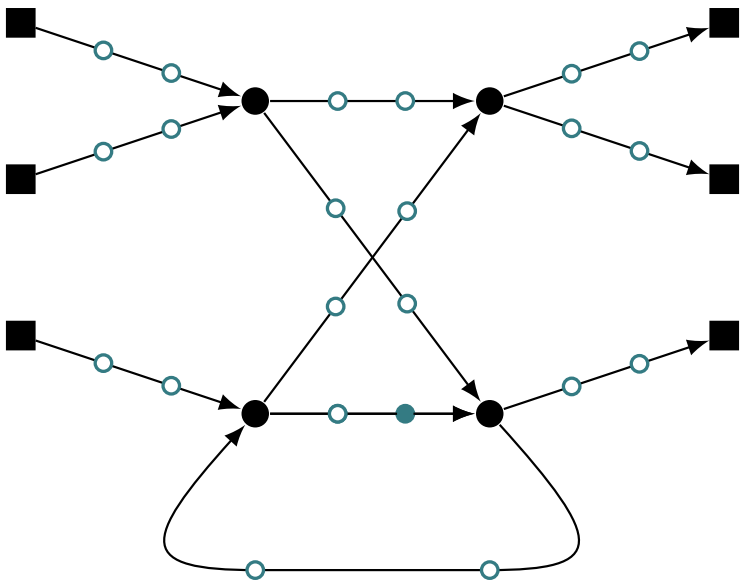


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

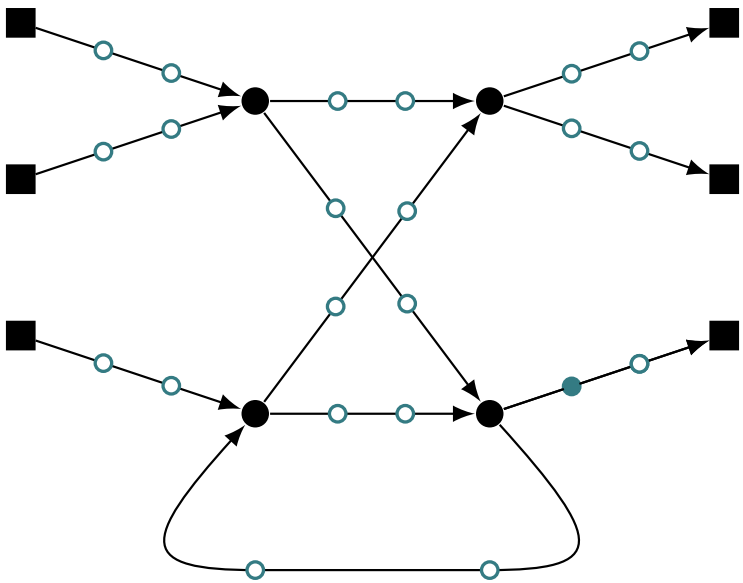


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.

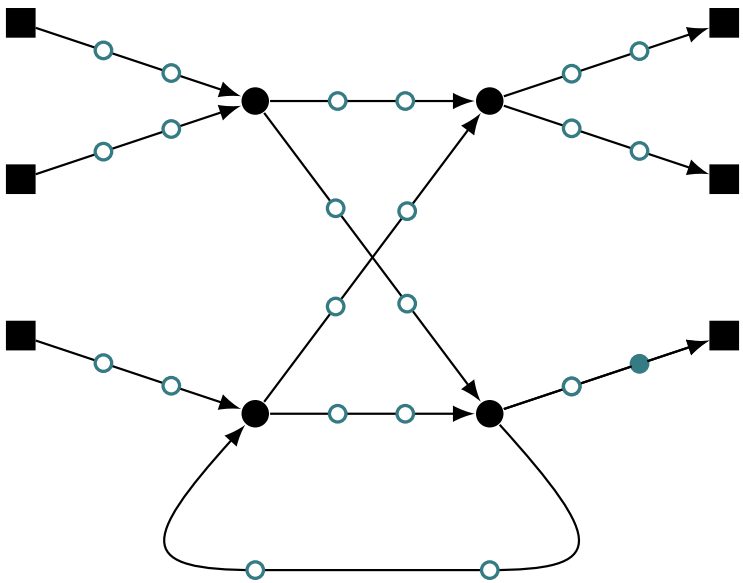


Splitter networks are coin-tossing network

Point of view of a single item, going through a splitter:

- probability 0.5 to go left;
- probability 0.5 to go right.

A splitter acts as a coin toss.



Balancers are uniformly-distributing

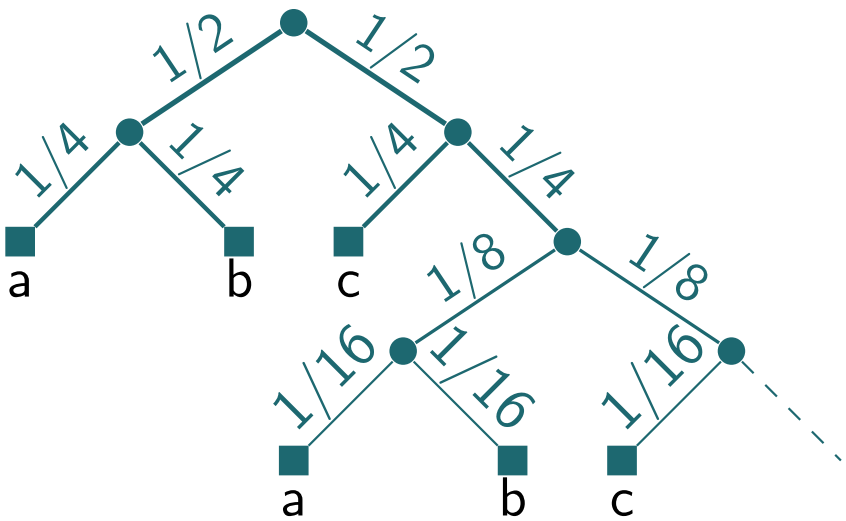
In an arbitrary balancer network:

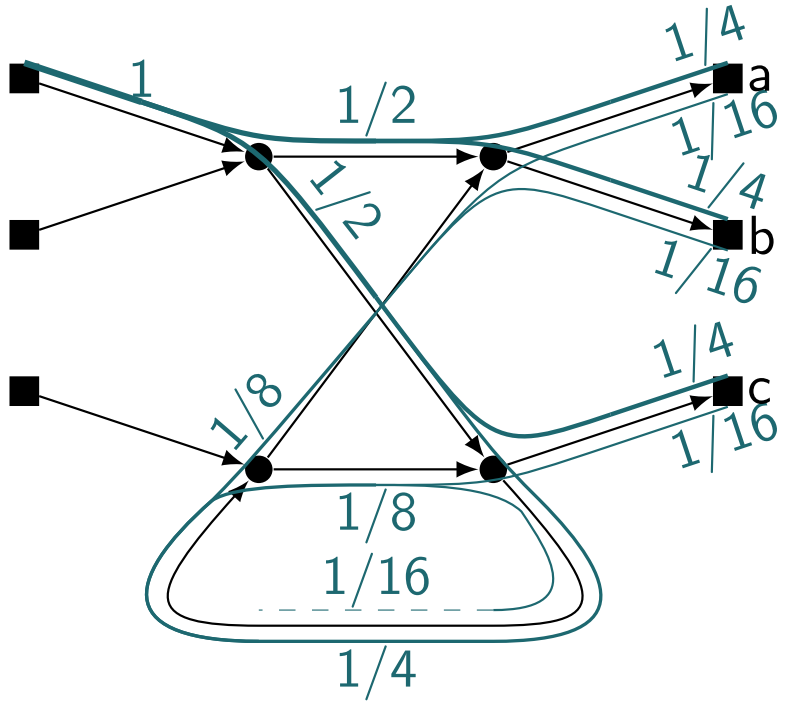
- set all input capacities to 0 except one;
- set all output capacities to 1;
- set the last input capacity sufficiently small to avoid saturated arcs.

Then items from the single active input are uniformly distributed over the outputs.

Balancer networks are uniformly-distributing coin-tossing network.

Binary decision tree maps into balancer





Coin-tossing potential

Sub-steady-state algorithm: augment along stationary distribution.

Random walks induce that stationary distribution.

Taking into account the capacities and the saturated arcs:

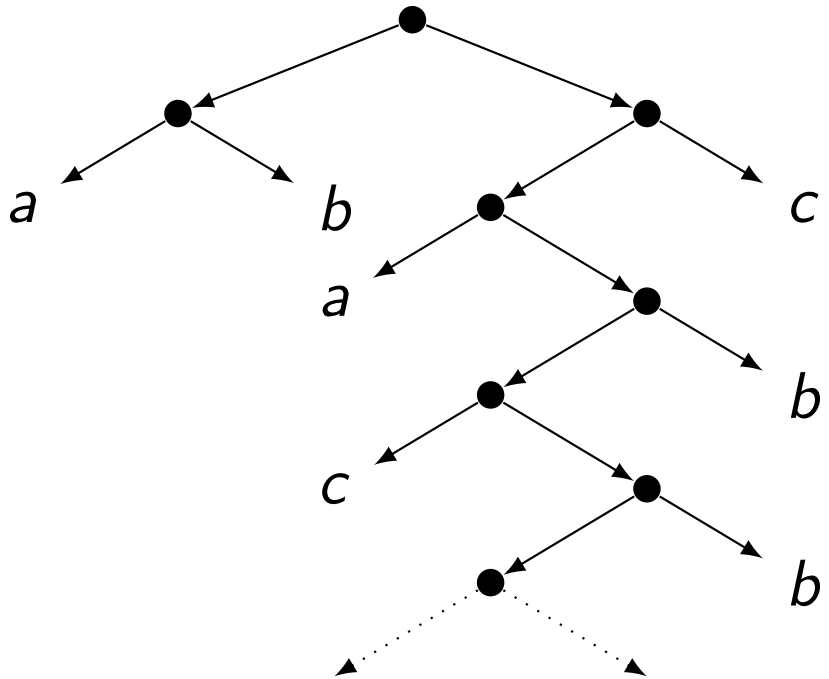
- each splitter acts as a coin-tossing device for its fluid outgoing belts, up to 2 units of flow;
- each splitter acts as a coin-tossing device for its saturated incoming belts, up to 2 units of flow.

Total coin-tossing potential is $4|S|$.

Knuth-Yao sampling algorithm

Sampling in $(a, \frac{3}{8}), (b, \frac{1}{3}), (c, \frac{7}{24})$:

a	b	c
0	0	0
1	1	1
1	0	0
0	1	0
0	0	1
0	1	0
⋮	⋮	⋮



Knuth-Yao bound

Minimum expected number of coins tosses to sample a distribution π over all possible algorithm is

$$E = \sum_{i=1}^{d} \sum_{k \in \mathbb{N}} \frac{k}{2^k} \mathrm{binary}_k(\pi_i)$$

We need $\frac{E}{4}$ splitters to get the same throughput outputs.

Lower-bound for balancers on 2^k belts: $k2^{k-2}$

Lower-bound analysis

- Simple balancers have twice as many splitters as the lower bound;
- lower-bound based on a potential of 4 per splitter, that is using saturation;
- if we want an all-fluid balancer, then simple balancers are optimal!
- Design balancers using saturation?
- Cost of throughput-unlimitedness (factor 4)? of universality (factor 16)?

Priority splitters

Priority splitters

In Factorio, splitters may be set with priorities:

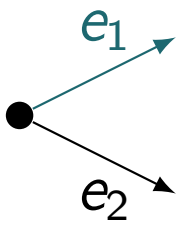
- out-priority: the splitter send as many items as possible to some outgoing belt. What is left goes on the other outgoing belt.
- in-priority: the splitter takes as many items as possible from some incoming belt. If possible it also takes from the other incoming belt.
- choose out-priority or out-fair, in-priority or in-fair independently.

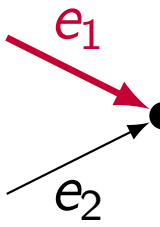
New questions:

- Steady-state and algorithms?
- Better designs? Lower bounds?
- Complexity: choose priorities to achieve some goal.

Steady-states with priorities

Rules for priority splitters with priorities $(p^+, p^-) : S \to E \cup \{\perp\}$:

- out-priority: if 

with e_1 fluid,
  - if $p^+(s) = e_1$ then $t(e_1) = 1$ or $t(e_2) = 0$;
  - if $p^+(s) \neq e_2$ then $t(e_1) \geq t(e_2)$;
- in-priority: if 

with e_1 saturated,
  - if $p^-(s) = e_1$ then $t(e_1) = 1$ or $t(e_2) = 0$;
  - if $p^-(s) \neq e_2$ then $t(e_1) \geq t(e_2)$;

The algorithms can easily be adapted.

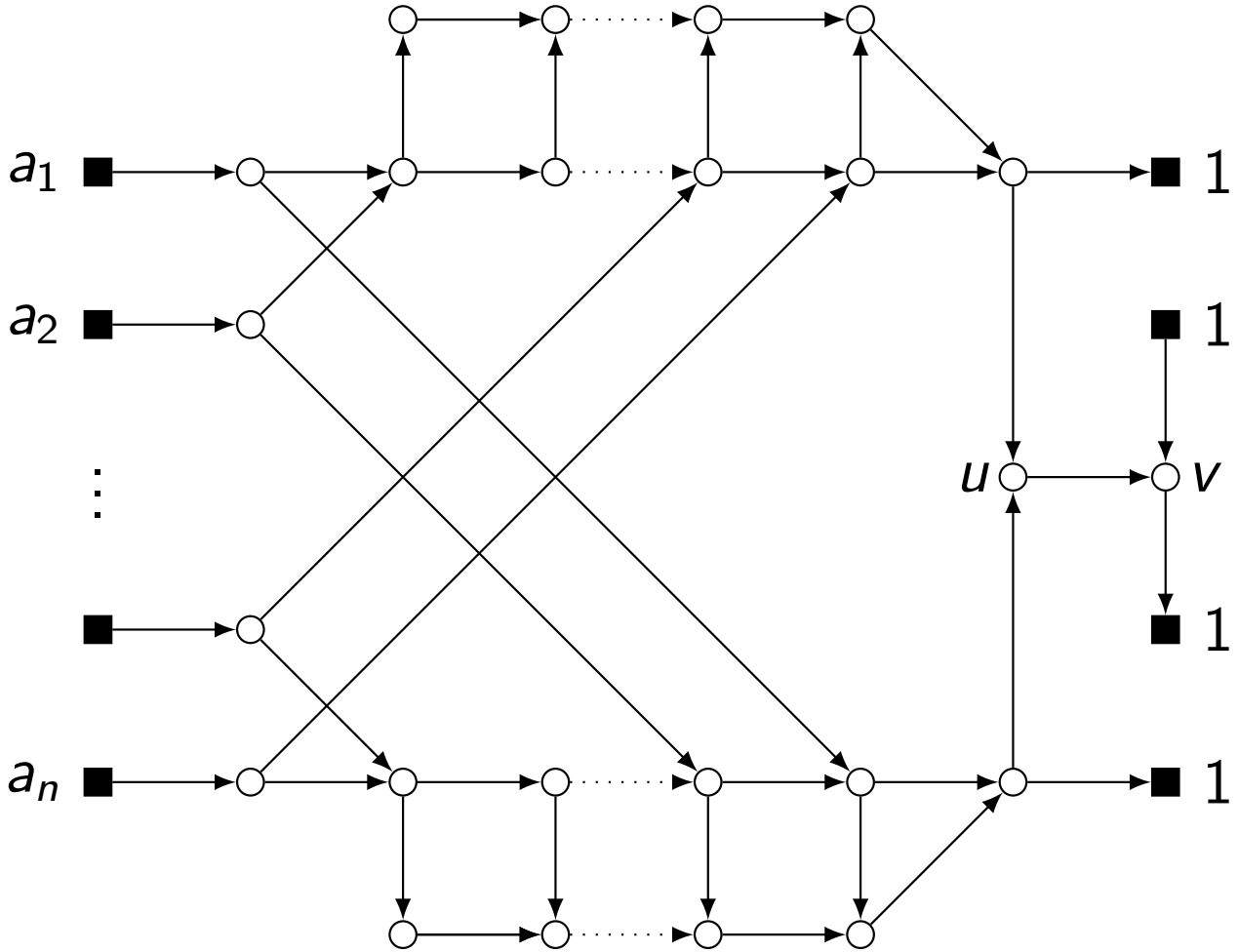
Maximizing the total throughput

Problem: given a splitter network with capacities, find priorities to maximize the sum of throughputs on the outputs.

- (uniform choose all) polynomial-time algorithm when all capacities are 1;
- (uniform choose out) polynomial-time algorithm when all capacities are 1 and all splitter are in-fair;
- (force-choose out) NP-hard to choose output priorities, when output capacities are 1, all splitters are in-fair and must not be out-fair;
- (uniform restricted choose out) NP-hard to choose output priorities for a subset of splitters, when all capacities are 1, all splitters are in-fair.

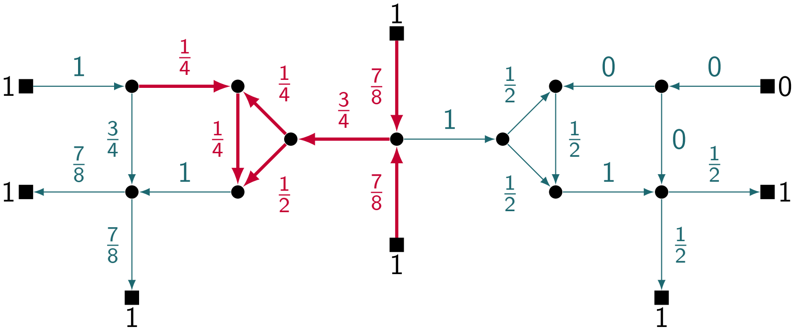
Force choose-out

Reduction from PARTITION of $\{a_1, a_2, \ldots, a_n\}$ with sum 2.



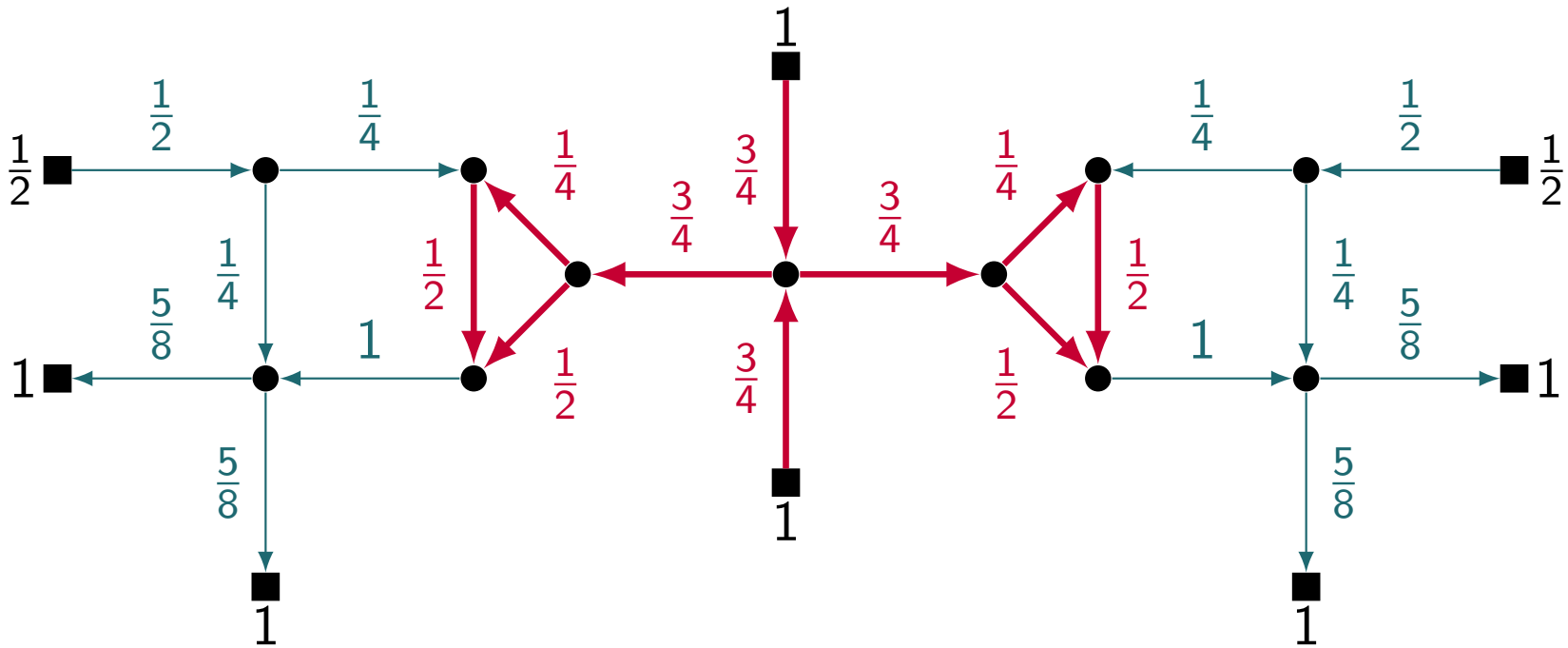
Uniform restricted choose-out

Reduction from 3-Satisfiability. Variable gadget:



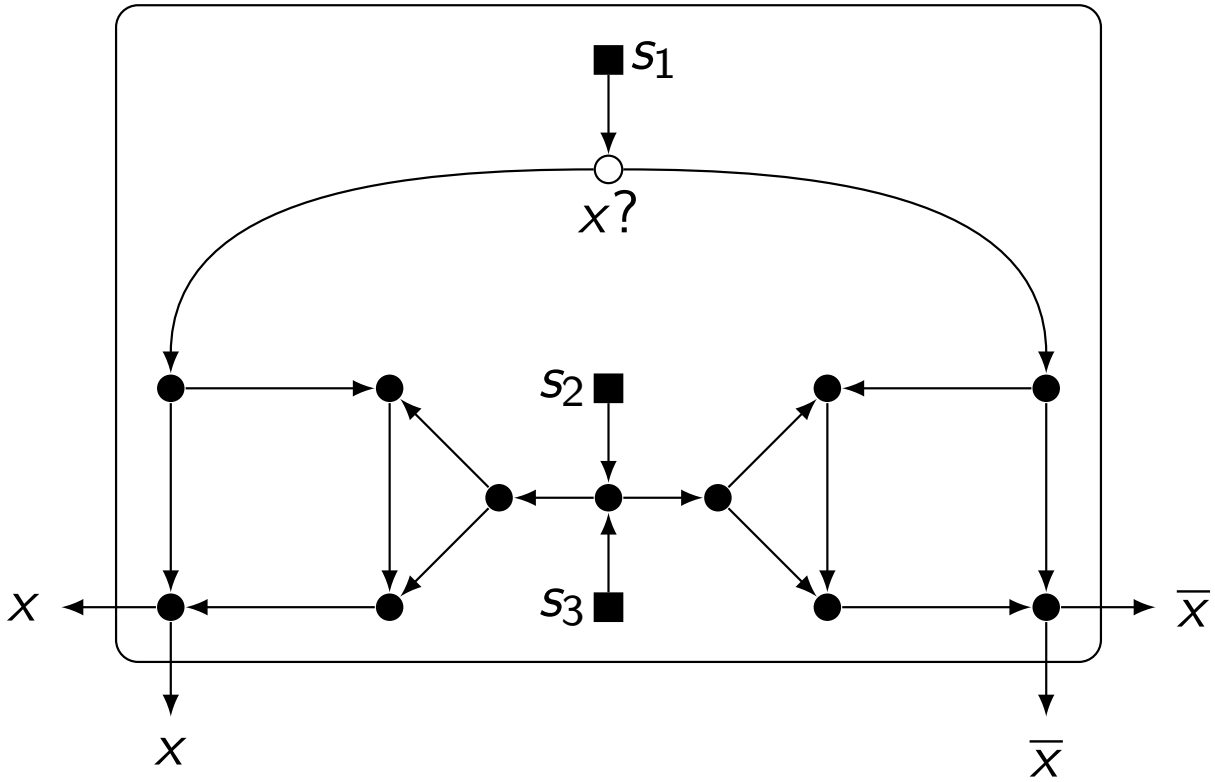
Uniform restricted choose-out

Reduction from 3-SATISFIABILITY. Variable gadget:



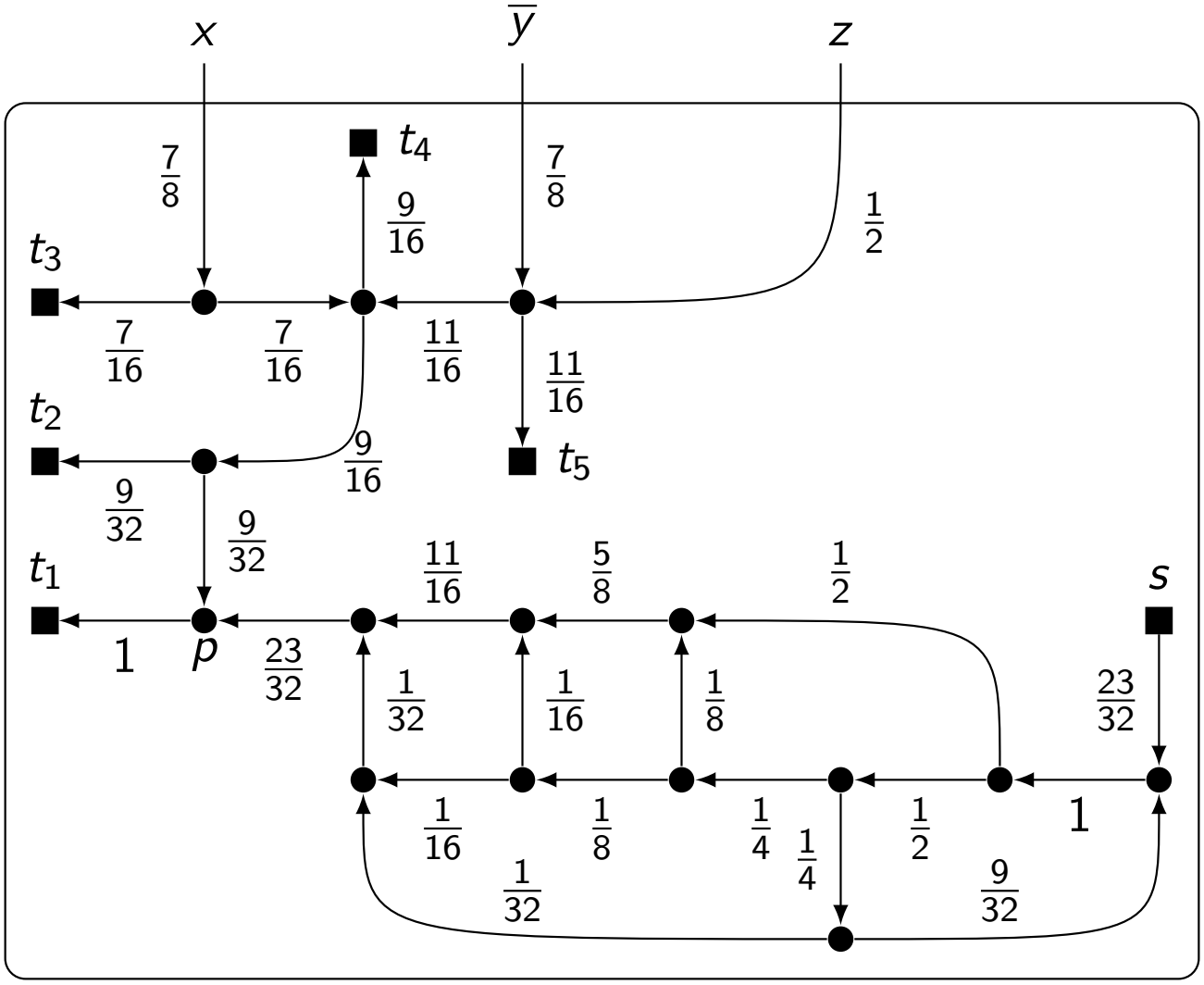
Uniform restricted choose-out

Reduction from 3-SATISFIABILITY. Variable gadget:



Uniform restricted choose-out

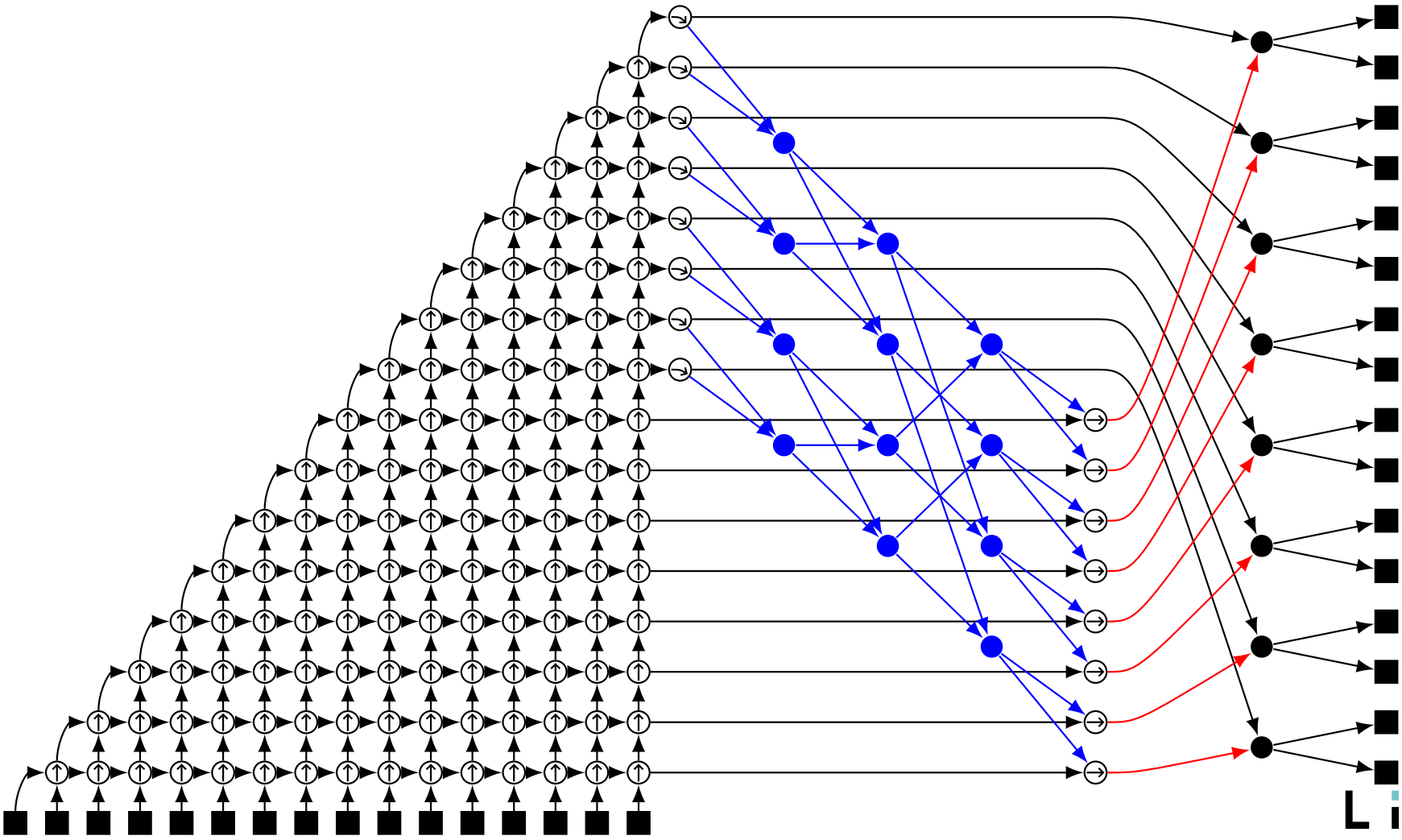
Clause gadget:



Designing balancers with priorities

- Lower bound still applies to bound the number of out-fair and in-fair splitters;
- may use priority splitters to reduce the number of splitters;
- universal balancer splitter invented by players has 4 times less splitters, with a small number of priority splitters;
- there is a splitter network with almost optimal $(k+2)2^{k-2}$ fair splitters, using (too many) priority splitters, leveraging saturation to balance.

A saturating balancer



Conclusion

Perspectives

- Purely combinatorial algorithm (without Laplacian solver)?
- Stronger lower bounds, better designs for balancers.
- Complexity for the non-uniform choose-all throughput maximization problem.
- How to check the balancing property? OK for non-saturating balancers.