The steady-states of splitter networks

Basile Couëtoux[1], Bastien Gastaldi[2], Guyslain Naves[3]

Abstract: We introduce splitter networks, which abstract the behaviour of conveyor belts found in the video game Factorio. Based on this definition, we show how to compute the steady-state of a splitter network. Then, leveraging insights from the players community, we provide multiple designs of splitter networks capable of load-balancing among several conveyor belts, and prove that any load-balancing network on n belts must have $\Omega(n \log n)$ nodes. Incidentally, we establish connections between splitter networks and various concepts including flow algorithms, flows with equality constraints, Markov chains and the Knuth-Yao theorem about sampling over rational distributions using a fair coin.

1 Introduction

The transportation of materials or data within various networks represents an inexhaustible source of mathematical problems, which has lead to almost as many solutions, theories and algorithms. These advancements have brought about significant improvements across diverse fields including supply chain management, logistics, network optimization. Transportation also serves as a central component in numerous games, as evidenced by the transportation category on BoardGameGeek which lists almost two thousand games [4]. In Factorio [23], a video game published in 2021 by Wube Software, players must mine natural resources to feed a rocket-building factory on an hostile planet. A major part of the gameplay involves the movement of resources within the factory, employing various mechanism: robotic arms, conveyor belts, drones or trains.

In this work, we study the conveyor belts of Factorio. An item placed on a belt will move at a constant speed toward the end of the belt, until it reaches that end, or is blocked by an item preceding it. Belts in Factorio can be combined using a splitter, connecting two incoming belts to two outgoing belts. A splitter takes items from the incoming belts and places them on the outgoing belts, trying to split the flow as fairly as possible between the incident belts, while maximizing the throughput. Given the scale of a typical Factorio game, players frequently encounter the need to balance the loads across multiple belts, and the community has devised numerous efficient networks to address this load-balancing problem.

An intriguing aspect of Factorio is its encouragement for players to construct vast systems of automation, requiring intensive planning and optimization. Ultimately, the limiting factor arises from the CPU load generated by game state updates. Consequently, players are incentivized to prioritize resource efficiency, particularly concerning gameplay elements that entail frequent computations such as splitters. This motivates the minimization of the number of splitters in load-balancing networks.

Our goal is two-fold: first we model the steady-state of a network of splitters. The network of conveyor belts is abstracted as a directed graph, with nodes corresponding to splitters and arcs to belts. A steady-state is a throughput function on the arcs; a circulation with additional constraints to capture the fact that splitters are fair and locally optimizing. We present two polynomial-time algorithms for computing a steady-state in a splitter network. An analogy is made with two classical maximum-flow algorithms: the blocking-flow algorithm [8] and the push-relabel algorithm [10]. In contrast to maximum flows, the primary challenge arises when a belt reaches full capacity, as its supplying splitter may no longer stay both fair and maximizing. In that case, the splitter is allowed to become unfair, but that decision changes the constraints applied to the flow, making the problem fundamentally non-convex. In a second part, we showcase various load-balancing network designs sourced from the Internet, formalizing concepts defined by the players community. Furthermore, we prove that those designs approach optimality. Specifically, we prove that any balancing network on n belts must have $\Omega(n \log n)$ splitters, by exhibiting a relation with

[1]Aix-Marseille Université, CNRS, LIS, Marseille, France, basile.couetoux@univ-amu.fr
[2]Télécom SudParis, Institut Polytechnique de Paris, Evry, France, bastien.gastaldi@telecom-sudparis.eu
[3]Aix-Marseille Université, CNRS, LIS, Marseille, France, guyslain.naves@univ-amu.fr, corrresponding author.

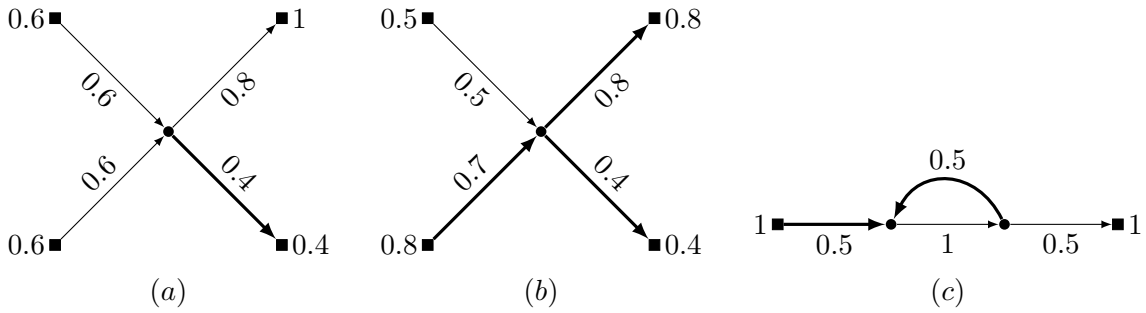


Figure 1: 3 splitter networks given with capacities and associated steady-states. Splitter will be represented by circle vertices, terminals by square vertices. Each terminal is tagged by its capacity, and each arc by its throughput. Saturated arcs are bolder than fluid arcs.

the problem of sampling the uniform distribution over a set of n elements using only a fair coin. The core design is the Beneš network, a circuit-switching network well-known in the field of telecommunication [1, 2].

The blocking-flow-like algorithm relies on finding circulations with equality constraints. Given a directed graph (G, A), let $\mathcal{C}^=$ be a partition of A such that for each equivalence class $C \in \mathcal{C}^=$, there is some vertex v with $C \subseteq \delta^+(v)$. The $\mathcal{C}^=$-circulation problem is to decide whether there is a non-zero circulation f that is constant within each equivalence class? While this problem can easily be solved using linear programming, we require a good characterization of graphs admitting a $\mathcal{C}^=$-circulation, Additionaly a polynomial-time algorithm is needed to either construct a $\mathcal{C}^=$-circulation or identify an obstacle that prevents its existence. The algorithm relies on the computation of a stationary distribution of an auxiliary graph. In contrast, solving maximum integral flow problems with additional equality constraints is known to be NP-hard [16], even when the equivalence classes are exactly the sets of leaving arcs of each vertex [22, 17].

Sorting networks [12] and Beneš networks have topologies similar to splitter networks, with nodes of in-degree and out-degree 2. In microfluidics, mixing graphs are used to produce droplets of specific concentration, using devices that produces two identical droplets from two droplets of any concentration [7]. The concentration values on the arcs are subject to equality constraints similar to those of splitter networks, but without a maximizing constraint. The topology of splitter networks is nonetheless more general than these examples, as splitter networks may have directed cycles, those being necessary in particular to achieve load-balancing with an arbitrary number of outputs.

In an answer to a question on the mathematics section of `stackexchange`, David Ketcheson attempted to model and compute the throughputs of splitter networks [11]. Rather than binary categorizing each belt as full or not, each arc is assigned a density and a velocity. The density will be monotonically increasing, and the velocity monotonically decreasing during the run of the algorithm, until a steady-state is reached. In fact the velocity increases only after the density reaches its maximum at one. Therefore this description is equivalent to our solution, which involves a throughput function and a set of full belts. Unfortunately his algorithm does not always terminate, and its solutions do not satisfy that splitters use their incoming belts fairly. Ketcheson also gave a procedure, albeit non-polynomial, to determine whether a network (not necessarily load-balancing) may limit throughput. Hovewer, this procedure is applicable only to networks without directed cycle. In [14], Leue modeled splitter networks using Petri nets, and uses model checking to check the load-balancing properties of some small networks.

The Factorio community is very active and creative. Players have designed load-balancing networks of various sizes, with efficient embeddings into the grid while respecting the constraints of the game. Additionally, they have developped general methods for constructing arbitrary large load-balancing networks. They introduced the concept of balancing networks, along with the more robust properties of being throughput unlimited or universal, and subsequently designed networks that exhibit these characterics. A

notable example is the universal balancer presented by *pocarski* [19], although it uses non-fair splitters too; our universal balancer only uses fair splitters. They also discovered the relationship with Beneš networks. Factorio-SAT [20] is a project that uses a SAT-solver to find optimal embeddings of splitter networks in the grid. Factorio belts are actually sufficiently complex to be Turing-complete [15]. There are many implementations of various devices inside Factorio, ranging from raytracers to programming language interpreters, using the diverse set of available gameplay mechanisms. Factorio has been the inspiration for several other academic works [21, 18, 5, 6, 9].

The rest of this section presents an overview of the main concepts and results of this work. Then Section 2 is dedicated to the $\mathcal{C}^{=}$-circulation problem, which is a requirement for the next Section 3, that focuses on the two algorithms to compute a steady-state. Section 4 contains the constructions and proofs of the load-balancing designs. Section 5 formalizes the proof for the lower bound on the number of splitters in a load-balancing network. Finally in Section 6 we will present some perspectives.

1.1 Splitter networks and their steady-states

We start by modeling networks of conveyor belts and splitters by directed graphs, where each single belt is an arc, and each splitter is a node (thus abstracting the length of the belts).

Definition 1. A *splitter network* is a directed graph G (with possible loops or parallel arcs) whose vertex set can be partitioned into three sets $V(G) = I \uplus S \uplus O$ where

(i) I is the set of *inputs*, and $d^+(i) = 1$, $d^-(i) = 0$ for any input i;
(ii) O is the set of *outputs*, and $d^-(o) = 1$, $d^+(o) = 0$ for any output o;
(iii) S is the set of *splitters*, and $d^-(s) = d^+(s) = 2$ for any splitter s.

We will use the word *flow* to informally describe the material transported by the network, and *throughput* for the amount of flow going through the arcs. Our work aims to understand the throughputs inside a splitter network at steady state, when some maximum throughputs are forced on its inputs and its outputs, which are respectively the sources and sinks of the flow passing through the network. To this end we will consider capacity functions on the input and output. A capacity c on an input means that the input has an incoming flow of throughput c. The input will try to push that much into the network, but no more. A capacity c on an output means that the output will accept a maximum throughput of c. We consider that the maximum throughput of any arc is 1, with all belts being identical.

A splitter can be described using two operational rules. The first rule, which takes precedence, is to maximize the amount of flow that goes through it. The secondary rule is to be fair. A splitter is fair relatively to its outgoing arcs: it tries to push as much flow onto each of them. It is also fair relatively to its incoming arcs: it tries to pull as much flow from each of them. As the maximization rule takes precedence, it will not be fair when being unfair leads to higher throughput. For instance, consider the network in Figure 1 (a), depicting a network with a single splitter. As one of the output has a lower capacity, it pushes more flow toward the other output, thereby maximizing the total throughput, while still being as fair as possible as it minimizes the difference of throughputs on its outgoing arcs.

The throughput of an arc may reach a limit when its head is an output with a low capacity. For example in Figure 1 (a), an output of capacity 0.4 acts as a bottleneck. In other cases the head of an arc is a splitter, which itself is limited by what its outgoing arcs can accept. As an example, consider the network in Figure 1 (b). As all the outputs have reached their capacities, the splitter cannot accept more flow, even as the top input could provide even more flow. In terms of conveyor belts, some belts will initially receive more items that they can deliver, causing them to fill up. Once full, they can only accept from upstream as much as they deliver downstream, which may in turn limit throughputs upstream. We say that such belts are *saturated*.

The output capacities are not the only factor that limit the total throughput and create bottlenecks. This can be observed in Figure 1 (c). There, the rightmost splitter tries to be fair and send some of the flow back to the left. The leftmost splitter also tries to be fair, thus accept the flow coming from the right.

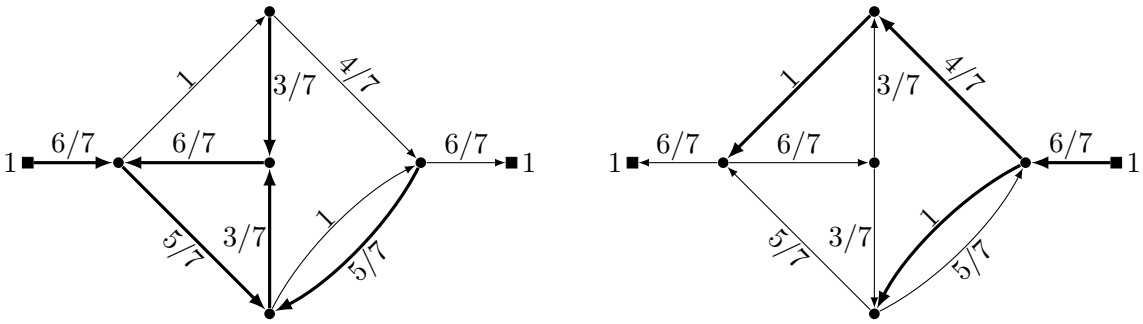


Figure 2: An example of steady-state in a moderately small network, and the reverse network with its steady-state obtained by reversal. Notice that the reversed steady-state satisfies rule E8 but not rule E8S.

This results in the stabilization into the given throughputs. This example illustrates that the throughput is not globally maximized, contrary to the expectation of a total throughput of 1 for this network. Instead, it is only 0.5.

The following definition formalizes the behaviour of splitters related to the flow going through the network in the steady state.

Definition 2. Let $G = (I \uplus S \uplus O, E)$ be a splitter network, and let $c : I \cup O \to [0, 1]$ be the maximal capacities of each input and output node. A *steady-state* for (G, c) is a pair (t, F) where

- E1 $t : E \to [0, 1]$ is the throughput function;
- E2 $F \subseteq E$ is the set of *fluid arcs*, $E \setminus F$ is the set of *saturated* arcs;
- E3 for each $i \in I$ with $\delta^+(i) = \{e\}$, $t(e) \leq c(i)$ and moreover if $e \in F$ then $t(e) = c(i)$;
- E4 for each $o \in O$ with $\delta^-(o) = \{e\}$, $t(e) \leq c(o)$ and moreover if $e \notin F$ then $t(e) = c(o)$;
- E5 for each $s \in S$, with $\delta^-(s) = \{e_1, e_2\}$ and $\delta^+(s) = \{e_3, e_4\}$, $t(e_1) + t(e_2) = t(e_3) + t(e_4)$;
- E6 for any $e_1, e_2 \in E$ with $\{e_1, e_2\} = \delta^-(s)$ and $e_1 \notin F$, $t(e_1) \geq t(e_2)$;
- E7 for any $e_1, e_2 \in E$ with $\{e_1, e_2\} = \delta^+(s)$ and $e_1 \in F$, $t(e_1) \geq t(e_2)$;
- E8 for any $uv \in E \setminus F$ and $vw \in F$, $t(uv) = 1$ or $t(vw) = 1$.

Rules E3 and E4 say that the throughputs are limited at each input and each ouput, and moreover, an input pushes as much flow as allowed by its capacity on a fluid arc. Similarly an output absorbs as much flow as allowed by its capacity from a saturated arc. Rule E5 imposes the conservation of flow. Rules E6 and E7 enforce the fairness constraints: a splitter consumes no less flow from a saturated arc than from another incoming arc. A saturated arc represents a belt that is full. Therefore, the splitter is not limited in how much flow it can pull from that arc, and thus cannot pull less than from the other incoming arc. Similarly it produces no less flow in a fluid outgoing arc than in another outgoing arc. In particular, if both incoming arcs are saturated, or if both outgoing arcs are fluid, they must have equal throughput, suggesting the following definition.

Definition 3. Given a continuous splitter network $G = (I \uplus S \uplus O, E)$, and a set $F \subseteq E$ of fluid arcs, we say that two arcs $e, e' \in E$ are

- ▹ *in-coupled* if $e, e' \notin F$ and there is a splitter vertex $v \in S$ with $\delta^-(v) = \{e, e'\}$,
- ▹ *out-coupled* if $e, e' \in F$ and there is a splitter vertex $v \in S$ with $\delta^+(v) = \{e, e'\}$,
- ▹ *coupled* if they are in-coupled or out-coupled.

Finally rule E8 imposes the maximization of the throughput by each splitter. Indeed, a saturated arc can provide more flow, while a fluid arc can absorb more flow. Thus, a steady-state cannot contain a saturated arc followed by a fluid arc. The only exception is when one of them already has a throughput of 1.

We will prove in Section 3.4 that the definitions of splitter networks and steady-states exhibit a remarkable symmetry. By reversing each arc, exchanging the role of inputs and outputs, and complementing the set of fluid arcs, a steady-state is transformed into a steady-state of the reverse graph, as seen in Figure 2.

For convenience, when defining or representing splitter networks, we will allow splitters with in-degree one or out-degree one (see Figure 2 for instance). This is justified by the fact that if a splitter s has in-degree one, we can add a dummy input node i with capacity $c(i) = 0$. An arc from i to s cn then be added, that will always remain fluid. Similarly if s has out-degree one, we can add a dummy output node o with capacity $c(o) = 0$, and an always-saturated arc from s to o. The throughputs on those arcs are forced to be 0. Therefore it does generate any new constraint on the non-dummy arcs as rules E6 and E7 are clearly true for those arcs.

Additionally, for convenience, for any input $i \in I$ with outgoing arc e, we note $t(i) := t(e)$, and similarly for any output $o \in O$ with incoming arc e, $t(o) := t(e)$. We also extend the capacities to arcs by setting $c(e)$ to be either $c(i)$ if $e \in \delta^+(i), i \in I$, or $c(o)$ is $e \in \delta^-(o), o \in O$, or 1 otherwise.

1.2 Existence and computation of steady-states

Let F be a fixed set of fluid arcs. Then the set of possible throughput functions t of a steady-state (T, f) can be described as a polyhedron. Indeed, each of the rules E1, E3, E4, E5, E6, E7 can be encoded by linear inequations. Rule E8 is non-convex, but we will later introduce its slight strengthening, rule E8S. That stronger rule admits an encoding as a family of linear inequations. Thanks to linear programming, finding a steady-state thus reduces to finding a set of fluid arcs that admits a steady-state. Nevertheless, we still need to find F. We propose two algorithms to compute a steady-state, which relates to two families of maximum flow algorithm:

- a push-relabel-like algorithm, where we relax the conservation rule E5, thus defining a *pre-steady-state* by analogy with *pre-flows*. Given a set F, we use a linear program to compute an optimal pre-steady-state (t, F) (for some well-chosen objective), and prove that either (t, F) is a steady-state, or there is an arc $e \in F$ such that $(t, F \setminus e)$ is also a (non-optimal) pre-steady-state. Then after at most $|E|$ steps we get a steady-state;
- a blocking-flow-like algorithm, where we remove the rule E3 on input capacities, thus defining a *sub-steady-state*. Given a set F, we solve a linear system to find a sub-steady-state t, and prove again that either (t, F) is a steady-state or there is an arc $e \in F$ such that $(t, F \setminus e)$ is a sub-steady-state.

The pre-steady-state algorithm is technically simpler but requires an LP-solver. The sub-steady-state only requires an algorithm to compute stationary distributions in directed graphs. We defer a complete presentation and proof of these algorithms to Sections 3.2 and 3.3, and focus for now on explaining the sub-steady-state algorithm.

Definition 4. Given $G = (I \uplus S \uplus O, E)$ a splitter network with capacities $c : I \uplus O \to [0, 1]$, a *sub-steady-state* for (G, c) is a pair (t, F) satisfying E1, E2, E4, E5, E6, E7 and E8S, and for any $i \in I$ and $e \in \delta^+(i)$, $t(e) \leq c(i)$.

The algorithm starts with the trivial sub-steady-state $(t : e \to 0, E)$, and will improve it iteratively until reaching a steady-state. At each iteration of the algorithm, we will be trying to increase the throughputs of the arcs without violating any rule. Unlike in maximum flows, we do not have the choice of which leaving arc to increase the flow on. Rule E8 forces each splitter to send as much flow forward as possible. A non-obvious consequence is that throughputs can only increase on fluid arcs, and can only decrease on saturated arcs. This suggests a definition of the residual graph for the sub-steady-state (t, F). Its vertex set is $\{z\} \cup S$, where z is obtained by identifying all the inputs and outputs into a single node. Its edge set contains some fluid arcs and the reverses of some saturated arcs.

Consider the splitters in Figure 3. We examine what happens when we increase the throughput on edge e_1 by $+\varepsilon$, or in case (d) when we decrease $t(e_3)$ by ε. In case (a), by rule E7, the throughputs on the two leaving arcs must stay equal, hence both increases by $\varepsilon/2$. In case (b), only the throughput of the fluid leaving arc e_4 can increase. In case (c), both leaving arc are saturated, the splitter cannot push more flow downward, hence it is forced to push back flow through its incoming saturated arcs. Thus $t(e_2)$

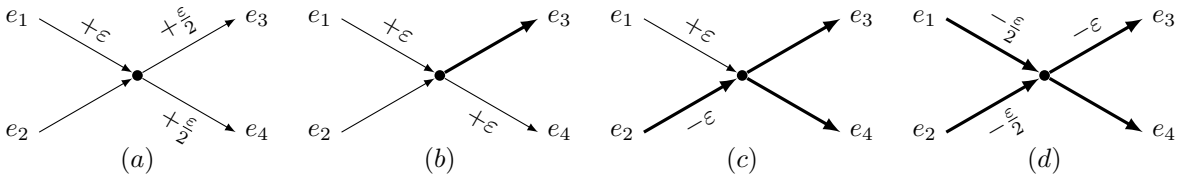


Figure 3: Four examples of throughput changes at a single splitter, depending on which arcs are fluid.

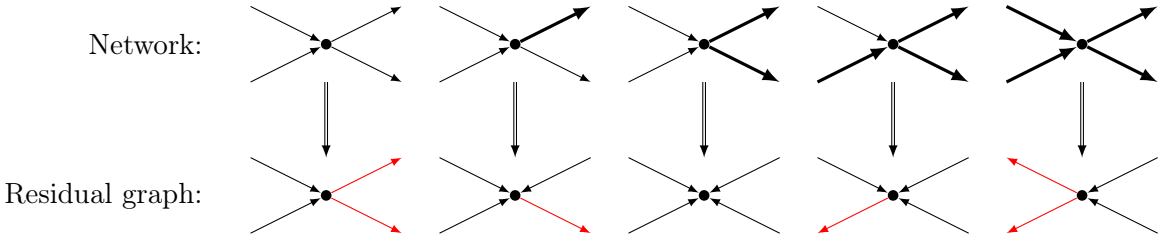


Figure 4: Configurations of splitters and the corresponding vertex in the residual graph. The outgoing arcs from a vertex of the residual graphs are highlighted in red: notice that in a sub-steady-state, the throughputs on these arcs must be equal.

decreases while $t(e_1)$ increases, by no more than $(t(e_2) - t(e_1))/2$ because of rule E6. Finally in case (d), if we decrease $t(e_3)$, then $t(e_1)$ and $t(e_2)$ must decrease by half as much.

Case (c) presents a challenge due to rule E6, which imposes $t(e_1) \leq t(e_2)$. When $t(e_1) = t(e_2)$, the throughput of e_1 cannot increase, while the throughput of e_2 cannot decrease. In such cases, removing e_1 from F would allowed by rule E6. A symmetric case occurs with two outgoing arcs from a splitter, if one is fluid and one is saturated with the same throughput. In both cases, we classify such arcs as *tight*. Fluid arcs e with $t(e) = c(e)$ or saturated arc with $t(e) = 0$ are also *tight*. Then we define the edge-set of the residual graph to only contains non-tight fluid arcs and reverses of non-tight saturated arcs.

Due to the conservation rule E5, any iterative change to the throughputs of the network must be in accordance with a circulation of the residual graph. Because by rules E6 and E7 some arcs are constrained to have the same throughput. Therefore the chosen circulation itself has similar constraints. This is illustrated in Figure 4, where the arcs that have equal throughput are highlighted in the residual graph. As may be readily checked, those constraints are exactly set on the leaving arcs in the residual graph of each vertex corresponding to a splitter. As for the special vertex z, we may non-deterministically select one of its leaving arc. Then we force all other arcs leaving z to have zero flow, by removing those arcs from the residual graph. From the residual graph, we compute a circulation satisfying each equality constraint. First compute a stationary distribution of a random walk on the residual graph. Then assign to each arc the probability of being the next arc in a random walk from that distribution. This results in a so-called *stationary circulation* (see Figure 9). One must be careful if the residual graph is not strongly connected. Then either we can find a strongly connected subgraph induced by the leaving arcs of some subset of vertices, or the residual graph contains a sink (as in Figure 10). In the former case we can still find a circulation, while in the latter case, we will be able to remove some arc from F.

Once a circulation is found, we increase the throughput as much as possible. This process will result in the creation of at least one sink in the updated residual graph. We show that when the residual graph contains a sink, some arc can be safely removed from F and becomes saturated. This bounds the number of steps until the algorithm stops, when z itself becomes a sink. At this point, any arc leaving an input node is either at full capacity or is saturated. Hence rule E3 is satisfied, (t, F) is a steady-state. Some additional details, not covered in this presentation, are provided in Section 3.3. An example of run of the algorithm on a simple network is given in Figures 9 to 14. Summarizing the discussion, we get:

Theorem 1. *There is an algorithm that given a splitter network $G = (I \uplus S \uplus O, E)$ with capacities*

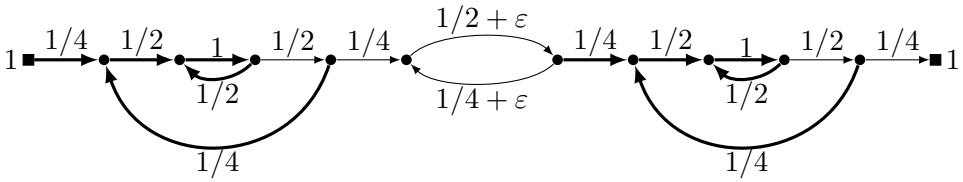


Figure 5: A network having several steady-states. Any value for ε between 0 and $\frac{1}{2}$ gives a steady-state.

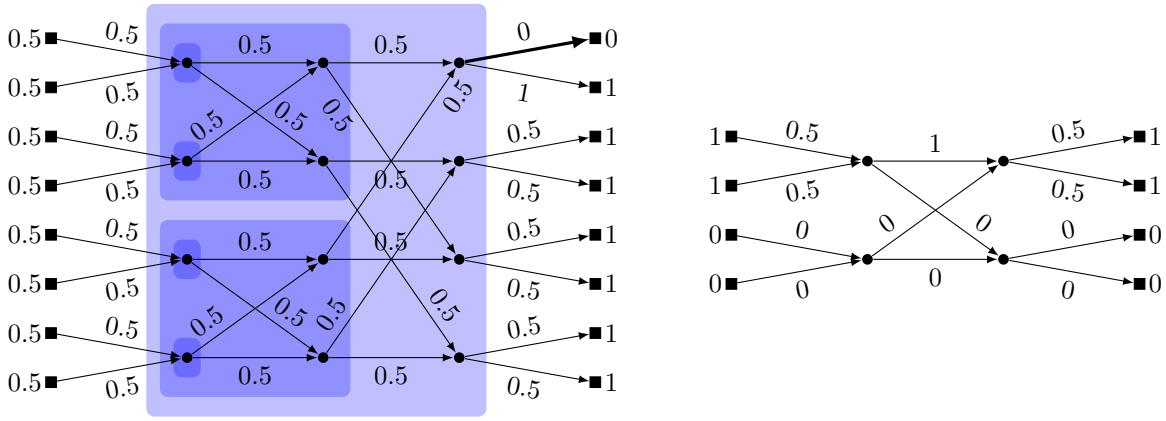


Figure 6: On the left, the simple balancer of order 3, with a steady-state that is not balanced when some output capacity is not 1. The capacity of each input (resp. output) is given at their left (resp. right). On the right, a simple balancer of order 2, with a steady-state with total throughput less than both the total input capacity and the total output capacity.

$c : I \uplus O \to [0, 1]$, find a steady-state (t, F) in time $O(n^2 + n \operatorname{sd}(G_z))$, where G_z is the graph obtained by identifying $I \cup O$ into a single vertex z, and $\operatorname{sd}(G_z)$ denotes the time to compute a stationary distribution on any orientation of a subgraph G_z.

Steady-states are not unique: a directed cycle with no input or output can have any constant throughput on all its arcs. Figure 5 showcases a more interesting network, having one input, one output, and many possible steady-states. However, in this example, all steady-states have the same throughputs on the inputs and outputs. Is there a network with two steady-states having different throughputs on their inputs and outputs. We conjecture that this cannot happen: steady-states are unique up to minor modifications, as in Figure 5.

1.3 Balancers

We now define load-balancing networks and their properties. The goal of a load-balancing network is to divide some input flow evenly between several output belts. In the simplest case, the output belts can receive an arbitrarily large flow (up to the capacity of the belt). In more general cases, some outputs may be restricted but we still want the flow to be divided as evenly as possible, without limiting the total throughput available. We distinguish three properties of load-balancing networks. The first of these properties considers networks where the output capacities are not constrained.

Definition 5. A splitter network $G = (I \uplus S \uplus O, E)$ is a *balancer* if for any $c : I \uplus O \to [0, 1]$ such that for each output $o \in O$, $c(o) = 1$, there is a steady-state (t, F) for (G, c) with t constant on $\delta^-(O)$. An (n, p)-balancer is defined as a balancer with $|I| = n$ inputs and $|O| = p$ outputs.

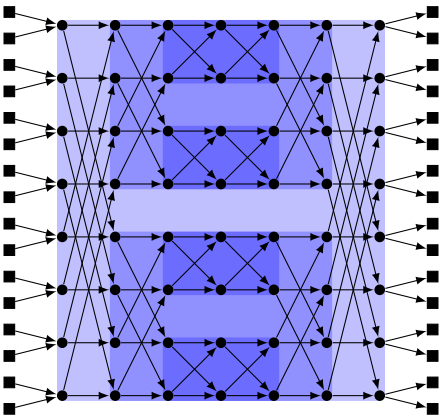


Figure 7: A Beneš network of order 4 with the recursive structure being made explicit.

When $|I| = |O| = 2^k$, the *simple balancer of order k* is a balancer network. It can be defined recursively: a simple balancer of order $k+1$ is made from two simple balancers of order k in parallel. We identify each pair of outputs with equal index from the two balancers, creating a new splitter whose leaving arcs go to new output nodes. The recursive process is highlighted by blue boxes in Figure 6. A drawback of the simple balancer occurs when the output capacities are not uniformly 1. Then the balancing property is lost, as can be seen on the network in the left side of Figure 6.

Another limitation of simple balancers is that the total throughput at steady-state is not as much as we could expect. A simple upper bound on the total throughput is $\min\{c(I), c(O)\}$. It is reasonable to expect from a load-balancing network to always reach that bound. However, simple balancers do not have this property, as shown by the example on the right side of Figure 6. Improving over the definition of simple balancer, the concept of throughput-unlimited balancer imposes a maximized global throughput.

Definition 6. A balancer $G = (I \uplus S \uplus O, E)$ is *throughput-unlimited* if for any $c : I \uplus O \to [0, 1]$, there is a steady-state (t, F) for (G, c) such that total throughput $t(I) = t(O)$ is maximized at $\min\{c(I), c(O)\}$.

Notice that it has to be balancing only when the output capacities are uniformly 1. Beneš networks are throughput-unlimited networks with $|O| = |I| = 2^k$. They can be described as gluing two simple balancers, where the second balancer is reversed, see Figure 7. Observe that Beneš network are their own reverse.

On the negative side, Beneš network are still not balancing when output capacities are not uniformly 1, for instance one could extend the steady-state in the network on the left side of Figure 6 to a steady-state in a Beneš network with the same throughputs. This calls for a stronger property, that a network should be load-balancing and throughput-unlimited for any capacity function. This is the notion of universal balancer.

Definition 7. A splitter network $G = (I \uplus S \uplus O, E)$ is *universally balancing* if for each capacity $c : I \uplus O \to [0, 1]$, there is a steady-state (t, F) and $\alpha, \beta \in \mathbb{R}_{\geq 0}$ such that

(i) for each input i, $t(\delta^+(i)) = \min\{c(i), \alpha\}$,
(ii) for each output o, $t(\delta^-(o)) = \min\{c(o), \beta\}$.
(iii) the total throughput $T := t(\delta^+(I))$ equals $\min\{c(I), c(O)\}$.

In Section 4, we will show how to build a universal balancer with $|I| = |O| = 2^k$. From such a universal balancer, by ignoring any set of inputs and outputs (setting their capacities to 0), we can make balancers with arbitrary numbers of inputs and outputs. We will also prove that every balancer presented here contains $\Theta(n \log n)$ splitters where n is the number of inputs and outputs.

Proposition 1. *The number of splitters in the simple balancer, Beneš network and universal network of order k are respectively* $S(k) = k \cdot 2^{k-1}$, $B(k) = (2k-1) \cdot 2^{k-1}$, *and* $U(k) = (k+1)2^{k+2}$.

1.4 Lower bounds on the number of splitters

.

Our next goal is to provide an $\Omega((n+p)\log(n+p))$ lower bound on the number of splitters in a (n,p)-balancer. We begin with what may seem as an unrelated problem: sampling in a discrete probability distribution. Given a fair coin that can be tossed arbitrarily often, how to choose an outcome in $\{1,\dots,d\}$, with probabilities given by a distribution $\pi \in [0,1]^d$? First, consider the case when $\pi(i)$ is a rational for each $i \in \{1,\dots,d\}$, say $\pi(i) = p_i/q$ where q is a common denominator. Then a sequence of coin tossing can be described as a (possibly infinite) binary decision tree, with each leaf labeled with a sampled value. Here we present a construction of such a tree. Start from a single vertex, which serves as the root. Grow the tree in repeated iterations. At each iteration, add two children to every unlabelled leaf. As soon as the deepest level of the tree contains at least q leaves, label p_i of these leaves with i, for each $i \in \{1,\dots,d\}$. Once labeled, each leaf becomes definitive and will not grow anymore. The process goes on by once again growing the unlabelled leaves, as long (possibly infinitely) as some unlabelled leaf exists. After the tree is completed, the tree can be optimized using a simple trick repeated multiple times. If at any depth d, two leaves share a common label, move them under a common parent, then replace these two leaves with a single leaf at depth $d-1$ bearing the same label. This process can be generalized to irrational probabilities, and gives a sampling algorithm that minimizes the number of coins tossed:

Theorem 2 ([13])**.** *Let $\pi \in [0,1]^d$ a discrete probability distribution (so $\mathbf{1}\pi = 1$). Then the minimum expected number of coin tosses necessary to sample an element with probability distribution π is $\sum_{i=1}^{d}\sum_{k\in\mathbb{N}}\frac{k}{2^k}\mathrm{binary}_k(\pi_i)$. This minimum is achieved by a binary decision tree where at each depth k and for each $i \in [\![1,d]\!]$, the number of leaves with label i is $\mathrm{binary}_k(\pi_i)$.*

Consider a splitter network, and think of the flow as discrete, arbitrarily small items. An item enters the network from some input, then meets splitters repeatedly until reaching an output. When an item arrives at a splitter with both outgoing arcs being fluid, it will continue on any of the two outgoing arcs, without preference for one over the other because the splitter is fair. It implies that, from the perspective of this single item, the splitter network behaves like a coin-tossing network, with each splitter corresponding to a coin toss. If the network is a balancer, the sampled distribution is the uniform distribution on O.

Formally, when all the arcs remains fluid, increasing a single input capacity from 0 to 1 results in a non-decreasing throughput on each arc. Becasue all arcs are still fluid, the sub-steady-state algorithm performed a single iteration. Therefore the increase in throughputs is a single stationary circulation. As illustrated on Figure 8, it is obtained from the embedding of a binary decision tree T onto the splitter network. The increase in throughput on an arc e is the sum of probabilities of the edges mapped to e. Furthermore, in a balancer network, the increase of throughput is the same on eavery output. This implies that, as we progressively increase each input capacity from 0 to 1, each binary decision tree must uniformly sampling from O.

In each binary decision tree, label each edge e with the probability of its usage during sampling. The sum of these labels represents the expected number of tosses, and can be bounded as shown in Theorem 2. When mapped into the splitter network, the sum of these labels for each edge mapped to an arc e of the splitter network is the additional throughput on e. By summing over all the binary decision tree, we get that the sum of all labels is at most the number of outgoing arcs of all splitters, that is $2|S|$. Applied this to balancers yields:

Theorem 3. *Let $G = (I \uplus S \uplus O, E)$ be an (n,p)-balancer, such that when all input capacities are 1, the steady-state has no saturated arc. Then*

$$|S| \geq \frac{1}{2}|I||O|\sum_{k\in\mathbb{N}}\frac{k}{2^k}\mathrm{binary}_k\left(\frac{1}{|O|}\right)$$

For a balancer with $|I| = |O| = 2^k$, since $\sum_{k\in\mathbb{N}}\frac{k}{2^k}\mathrm{binary}_k\left(\frac{1}{|O|}\right) = \frac{k}{2^k}$, we get a lower bound of $k2^{k-1}$ splitters, matching the value of $S(k)$. Therefore the simple balancer of order k is optimal among

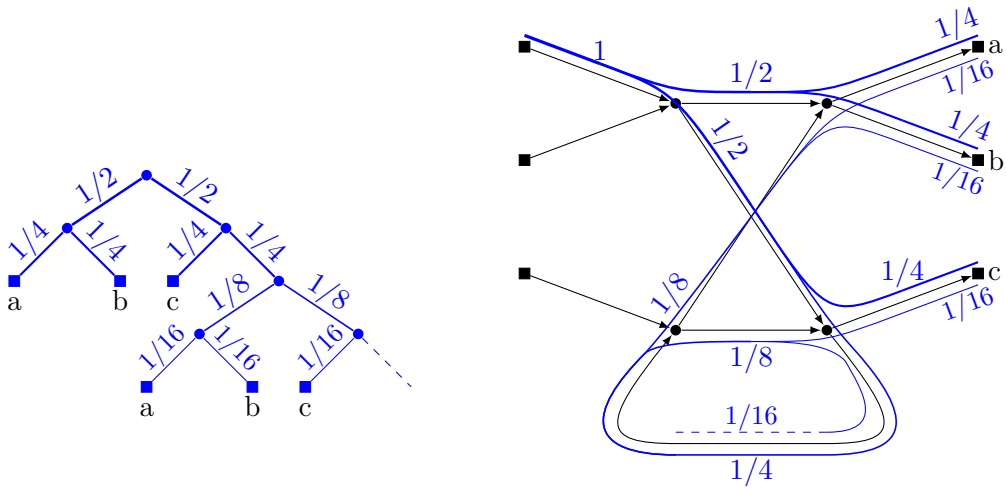


Figure 8: The infinite decision tree (in blue) used to sample uniformly over a three-element set $\{a, b, c\}$ can be embedded from any input into a $(3, 3)$-balancer. Moreover, the sum of the probabilities of the 3 trees, one from each input, will be at most one on any arc, which shows that this network is indeed a simple balancer.

all balancer networks without any saturated arcs in their steady-states. By extending this argument to steady-states with saturated arcs, we can remove that restriction, albeit at the cost of halving the lower bound.

Consider the various configurations of fluid and saturated arcs incident to a splitter, illustrated in Figure 3. If a splitter has two fluid outgoing arcs, any additional flow is evenly distributed between the two outputs, akin to the probabilities of a coin toss. If a splitter has two incoming saturated arcs, by rule E8, its outgoing arcs are saturated or at full capacity. In an augmenting circulation, the throughput on those arcs may only decrease by the same quantity by rule E6: the splitter still acts as a coin toss, but on the flow that is pushed back. Otherwise, a positive change in throughput on an incoming arc will be followed by an increase on a single outgoing fluid arc or a decrease on a single incoming saturated arc. Similarly a negative change of throughput on an outgoing saturated arc will impact only on other arc. Any additional unit of flow entering the splitter would be routed deterministically. Therefore, in the embedding of a binary decision tree into the splitter network, a node cannot be mapped to such a vertex, and no coin toss occurs here. Thus any splitter, depending on which of its incident arcs are fluid, acts as either a coin toss or a deterministic router. Thus, even in the presence of saturated arcs, we can embed a binary decision tree, by mapping each edge to a directed path in the residual graph. The inner nodes of any such path are *deterministic* splitters, while its extremities are *tossing* splitters. By Corollary 2, the throughput on each arc increases until it becomes saturate, then decreases. Therefore its throughput varies by at most during the whole algorithm. This limits the extent to which an arc can be utilized by the embeddings of binary decision trees, leading us to the following conclusion:

Theorem 4. *Let $G = (I \uplus S \uplus O, E)$ be an (n, p)-balancer. Then*

$$|S| \geq \frac{1}{4}|I||O| \sum_{k \in \mathbb{N}} \frac{k}{2^k} \text{binary}_k \left(\frac{1}{|O|} \right)$$

We will formalize this discussion and prove the theorem in Section 5

2 The $\mathcal{C}^=$-circulation problem

Let $G = (V, E)$ be a directed graph. We consider a partition $\mathcal{C}^= \subseteq 2^E$ of the arcs, such that each $C \in \mathcal{C}^=$ is a subset of the outgoing arcs $C \subseteq \delta^+(v)$ of some vertex $v \in V$. Two arcs $e', e'' \in E$ are considered $\mathcal{C}^=$-*coupled* (or just *coupled* when no ambiguity arises) if there is $C \in \mathcal{C}^=$ such that e', e'' belongs to C (an arc is coupled to itself). An arc e' is *single* if $\{e'\} \in \mathcal{C}^=$, indicating that it is not coupled to another arc. The tuple $(G, e, \mathcal{C}^=)$, where $e \in E$, is then an instance of the $\mathcal{C}^=$-*circulation* problem: find a circulation in G that is non-zero on e and is constant on any set of $\mathcal{C}^=$. It can be expressed as finding a vector x with $x_{ts} > 0$ satisfying the following linear system:

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

Theorem 5. *An instance $(G = (V, E), ts, \mathcal{C}^=)$ of the $\mathcal{C}^=$-circulation problem has a solution x with $x_{ts} > 0$ if and only if there is no set $S \subseteq V \setminus \{t\}$ with a partition $S = S_0 \uplus S_1 \ldots \uplus S_k$ where $s \in S_0$ and for any arc $e \in \delta^+(S_i)$, there is an arc e' coupled to e (possibly $e = e'$) such that $e' \in E[S_i, S_j]$ and $i < j$.*

Proof. Suppose $S = S_0 \uplus S_1 \ldots \uplus S_k$ exist and let x denote a solution to (1). We prove by induction from k to 0 that for all $i \in [\![0, k]\!]$, $x(\delta(S_i)) = 0$. This will imply that $x_{ts} = 0$ as $ts \in \delta^-(S_0)$. By assumption, $\delta^+(S_k) = \emptyset$, and because x is a circulation, $x(\delta^-(S_k)) = 0$, proving the base case. Let $i \in [\![1, k-1]\!]$ and suppose that we have for all $j \in [\![i+1, k]\!]$, $x(\delta(S_j)) = 0$. Let $uv \in \delta^+(S_i)$, then there is an arc $uw \in E[S_i, S_j]$ with $j > i$, where uw is either equal to or coupled with uv, hence by the induction hypothesis $x_{uv} = x_{uw} = 0$. Thus $x(\delta^+(S_i)) = 0$. Again, by x being a circulation, we get that $x(\delta^-(S_i)) = 0$.

Now suppose that there is no $x \in \mathbb{R}^E_{\geq 0}$ satisfying (1) and $x_{ts} > 0$. By Farkas lemma, this implies that the following linear system has a solution $y \in \mathbb{R}^V$:

$$\begin{aligned} y_s - y_t &\geq 1 \\ \sum_{uv \in C} y_v - y_u &\geq 0 \qquad (C \in \mathcal{C}^=) \end{aligned}$$

We may partition V into $S_{-l}, S_{-l+1}, \ldots, S_k$, the equivalence classes defined by the relation $u \sim v$ if $y_u = y_v$, ordered based on increasing y-values, and such that $s \in S_0$. Because $y_t < y_s$, $t \notin S_0 \cup S_1 \ldots \cup S_k$.

Let $i \in [\![0, k]\!]$ and let $uv \in \delta^+(S_i)$. Let $C \in \mathcal{C}^=$ with $uv \in C$. Then $\sum_{uw \in C} y_w - y_u \geq 0$ implies that either $y_w = y_u$ for all $uw \in C$, which is not the case as $uv \in \delta^+(S_i)$ hence $y_u \neq y_v$, or there is some $uw \in C$ with $y_w > y_u$. But then $w \in S_j$ with $j > i$, proving that the partition S satisfies the desired property. □

Corollary 1. *An instance $(G = (V, E), ts, \mathcal{C}^=)$ of the $\mathcal{C}^=$-circulation problem where G is strongly connected has a solution x with $x_{ts} > 0$.*

Proof. By contraposition, consider an instance lacking a positive partition, and let $S_0 \cup \ldots \cup S_k$ be the partition given by Theorem 5. Then $\delta^+(S_k)$ is empty, indicating that G is not strongly connected. □

We may compute efficiently a $\mathcal{C}^=$-circulation in the strongly connected case. We do so by restricting the problem to a spanning subgraph in which the classes in $\mathcal{C}^=$ coincides with the leaving arcs of each vertex.

Lemma 1. *There is an algorithm that, given an instance $(G = (V, E), ts, \mathcal{C}^=)$ of the $\mathcal{C}^=$-circulation problem with G strongly connected, find a feasible solution x with $x_{ts} > 0$ in time $O(|E| + \mathrm{sd}(G))$, where $\mathrm{sd}(G)$ is the complexity of computing a stationary distribution on any subgraph of G.*

Proof. Let T denote a maximal in-arborescence of G rooted at t. Let

$$E' := \{ts\} \cup \{e \in E \ : \ e \text{ is coupled to some arc in } T\}.$$

By construction, t is reachable from any vertex in (V, E'), and s and t are in the same strongly connected component of (V, E'). Let $H = (V_H, E_H)$ be the strongly connected component containing s and t. Because any vertex in T has out-degree at most 1, and by our choice of E', for any $v \in V$, $\delta^+(v) \cap E' \in \mathcal{C}^=$. Let $\mathcal{C}^=_H := \{C \in \mathcal{C}^= : C \subseteq E_H\}$.

We can rewrite the problem of finding a $\mathcal{C}^=_H$-circulation on H, by setting $y_v := x_e$ for $e \in \delta^+(v)$:

$$\sum_{uv \in \delta^-(v)} y_u = d^+(v) y_v,$$
$$y \geq 0,$$

and then setting $\pi_v = d^+(v) y_v$:

$$\sum_{uv \in \delta^-(v)} \frac{\pi_u}{d^+(u)} = \pi_v,$$
$$\pi \geq 0.$$

We recognize the last system as the one defining a stationary distribution of a Markov chain over H, provided that we add the constraint $\mathbf{1}\pi = 1$. As H is strongly connected, the stationary distribution exists and is nowhere zero. Thus we obtain a non-zero $\mathcal{C}^=_H$-circulation for H. We extend it into a non-zero $\mathcal{C}^=$-circulation for G by setting $x_e = 0$ for any edge not in E_H. □

The strongly connected case serves as a basis for an algorithm solving the general problem.

Lemma 2. *There is an algorithm that, given an instance $(G = (V, E), ts, \mathcal{C}^=)$ of the $\mathcal{C}^=$-circulation problem, in time $O(|E| \log^4 |V| + \mathrm{sd}(G))$, either find a feasible solution x with $x_{ts} > 0$ or correctly assert that none exists. Here $\mathrm{sd}(G)$ is the complexity of computing a stationary distribution on any subgraph of G.*

Proof. We start by computing a strongly connected subgraph H of G containing ts and ensuring that for any $C \in \mathcal{C}^=$, either $C \subseteq E(H)$ or $C \cap E(H) = \emptyset$. If no such H exists, then the $\mathcal{C}^=$-circulation instance does not admit a non-zero solution. If H exists, we reduce the problem of finding x to computing a stationary distribution in H.

To compute H, we use a dynamic decremental single-sink reachability algorithm with sink t. If ever t becomes unreachable from s, then we conclude that the instance does not admit a non-zero solution. The algorithm keeps a queue of vertices to delete. While the queue is non-empty, it takes a vertex from it and delete its incident arcs and all the arcs coupled to them. As soon as it detects that some vertex v cannot reach t (in particular at the start of the algorithm), it queues that vertex v for deletion. When the queue is empty, remove all the vertices not reachable from s and return the remaining subgraph H.

First, suppose that this algorithm fails to build H because t becomes unreachable from s. Each time a subset of vertices becomes disconnected from t, we label it S_i (with decreasing index i, adjusting the value of the starting index at the end of the algorithm). When such a component appears, all its leaving arcs must have been removed, which means they are coupled to an arc entering some already removed component. Hence we get the sequence $S_0, S_1 \dots S_k$ proving that a non-zero $\mathcal{C}^=$-circulation cannot exist.

Suppose now that the algorithm returns a subgraph $H = (V_H, E_H)$. By construction, for any $C \in \mathcal{C}^=$, either $C \subseteq E_H$ or $C \cap E_H = \emptyset$. Indeed, if any arc of C is removed during the main loop, then C is removed in the same iteration. If some arc of C is removed during the final phase, when vertices unreachable from s are removed, then the vertex v for which $C \subseteq \delta^+(v)$ must not be reachable from s and thus C was removed.

Next we claim that H is strongly connected. By construction, for any vertex v, there is a path from t to v starting with ts. Moreover, notice that before removing the vertices unreachable from s, there was a path from v to t. Thus any vertex on this path is reachable from s and is kept during the last phase. Hence t is reachable from v, proving that H is strongly connected. We then apply Lemma 1 to compute x, and extend it to G by setting $x_e = 0$ for any arc not in H. The complexity follows by using the decremental single-sink reachability algorithm from [3]. □

The next lemma gives a sufficient condition for the existence of a non-zero $\mathcal{C}^=$-circulation, when we may choose any arc to be positive instead of the specific arc ts as above.

Lemma 3. *Given a connected directed graph $G = (V, E)$ and a partition $\mathcal{C}^=$ of E that is a refinement of the out-incidencies of G, then*

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

and we can find one or the other in time $O(|E| + \mathrm{sd}(G))$.

Proof. One can find in time $O(|E|)$ a strongly connected component X of G that is a sink component: $\delta^+(X) = \emptyset$. If $|X| = 1$ we are done, because $v \in X$ is a sink. Now, assume that $|X| > 1$.

Because for any $C \in \mathcal{C}^=$, there is a vertex v with $C \subseteq \delta^+(v)$, for any arc $e \in E[X]$, any arc coupled to e is also in $E[X]$. Hence, a $\mathcal{C}^=$-circulation x of $G[X]$ can be extended to a $\mathcal{C}^=$-circulation on G by setting $x_e = 0$ for each $e \notin E[X]$. Such a $\mathcal{C}^=$-circulation on $G[X]$ exists and can be computed in time $O(|E| + \mathrm{sd}(G))$ by Lemma 1. □

3 Computing a steady-state

In this section, we give two algorithms to compute a steady-state in a splitter network.

3.1 A stronger maximization rule

The algorithms use a slightly stronger property than rule E8:

E8S for any arcs $uv \in E \setminus F$ and $vw \in F$, $t(vw) = 1$.

Clearly rule E8S implies rule E8. The two rules are actually almost equivalent:

Lemma 4. *If (t, F) is a steady-state, then there is a steady-state (t, F') that satisfies rule E8S, with $F \subseteq F'$.*

Proof. By induction on the number of splitters on which rule E8S is not true. Let $uv \notin F$, $vw \in F$ with $t(vw) < 1$, by rule E8 $t(uv) = 1$. Let $u'v$ be the arc in-coupled to uv. If $u'v$ is fluid, then $(t, F \cup \{uv\})$ is a steady-state as rule E6 is checked on v and rule E7 is checked on u. If $u'v$ is saturated, by rule E6, $t(u'v) = 1$, then $(t, F \cup \{uv, u'v\})$ is a steady-state for similar reasons. Notice that in both cases, the modification preserves rule E8S on any splitter for which it held. Hence the number of those splitters increases by one. □

3.2 An push-relabel-like algorithm to compute steady-states

In analogy with a pre-flow in a push-relabel max-flow algorithm, we relax the conservation rule E5, to define:

Definition 8. Given a splitter network $G = (I \uplus S \uplus O, E)$ with capacities $c : I \uplus O \to [0, 1]$, a *pre-steady-state* for (G, c) is a pair (t, F) satisfying E1, E2, E3, E4, E6, E7, and E8S, and such that for each splitter $s \in S$,

$$\sum_{e \in \delta^+(s)} t(e) \leq \sum_{e \in \delta^-(s)} t(e).$$

The next claim which can be readily checked, asserts the existence of a simple pre-steady-state.

Claim 1. *The pair $(t, E(G))$ is a pre-steady-state for (G, c), where*

$$t : uv \to \begin{cases} c(u) & \text{if } u \in I, \\ 0 & \text{otherwise} \end{cases}$$

We proceed by iteratively improving upon this initial pre-steady-state. Let $G = (I \uplus S \uplus O, E)$ be a continuous splitter network with capacities $c : I \uplus O \to [0, 1]$. Let $F \subseteq E$ be a set of fluid arcs and (t, F) is a pre-steady-state, then t is a feasible solution to the following linear program:

$$\begin{array}{|lll|lll|} \max t(\delta^-(O)) & \text{subject to} & & & & \\ 0 \le t \le 1 & (e \in E) & & t(\delta^+(s)) \le t(\delta^-(s)) & (s \in S) & \\ t(is) \le c(i) & (is \in \delta^+(I)) & & t(is) = c(i) & (is \in \delta^+(I) \cap F) & \\ t(so) \le c(o) & (so \in \delta^-(O)) & & t(so) = c(o) & (so \in \delta^-(O) \setminus F) & \\ t(wv) \le t(uv) & (uv \in E \setminus F, wv \in E) & & t(uw) \le t(uv) & (uv \in F, uw \in E) & \\ t(vw) = 1 & (uv \in E \setminus F, vw \in F) & & & & \end{array} \tag{PSS}$$

Conversely, the following lemma is immediate:

Lemma 5. *Given $G = (I \uplus S \uplus O, E)$ a continuous splitter network, $c : I \uplus O \to [0, 1]$ a capacity function and $F \subseteq E$ a set of fluid arcs, for any feasible solution t of (PSS), (t, F) is a pre-steady-state of (G, c).*

Our algorithm is based on the next lemma.

Lemma 6. *Let $t^\star$ be an optimal solution to (PSS) such that $\sum_{e \in F} t(e) - \sum_{e \in E \setminus F} t(e)$ is maximized. Then either $(t^\star, F)$ is a steady-state, or there is an arc $e \in F$ such that $(t^\star, F \setminus \{e\})$ is a pre-steady-state.*

Proof. Assume that $(t^\star, F)$ is not a steady-state: there is a vertex $s \in S$ with $\text{excess}_{t^\star}(s) > 0$. Let $\{e_1, e_2\} = \delta^-(s)$ with $t^\star(e_1) \le t^\star(e_2)$, and $\{e_3, e_4\} = \delta^+(s)$ with $t^\star(e_3) \le t^\star(e_4)$.

Case 1: $e_3 \in F$ and $t^\star(e_3) < 1$. Then, for some $\epsilon > 0$, $t^\star + \epsilon\chi_{\{e_3\}}$ is a better solution to (PSS) than $t^\star$ unless:

- ▹ either rule E6 breaks at some vertex $w \in S$, that is: there is an arc e_3' with $\{e_3, e_3'\} = \delta^-(w)$, $e_3' \in E \setminus F$ and $t^\star(e_3) = t^\star(e_3')$. Therefore $(t^\star, F \setminus \{e_3\})$ is a pre-steady-state, as rules E6 and E8S are clearly still satisfied.
- ▹ or rule E4 breaks at some vertex $o \in O$, that is: $e_3 = so$ and $t^\star(e_3) = c(o)$. Therefore $(t^\star, F \setminus \{e_3\})$ is a pre-steady-state.

Case 2: $e_4 \in F$ and $t^\star(e_4) < 1$. Then the same argument shows that $(t^\star, F \setminus \{e_4\})$ is a pre-steady-state.

We may now assume that $e_3 \notin F$ or $t^\star(e_3) = 1$, and that $e_4 \notin F$ or $t^\star(e_4) = 1$.

Case 3: $e_2 \in F$. Then $(t^\star, F \setminus \{e_2\})$ is a pre-steady-state, as rules E6 and E8S are both satisfied.

We may now assume that $e_2 \notin F$.

Case 4: $e_1 \in F$ and $t^\star(e_1) = t^\star(e_2)$. Then $(t^\star, F \setminus \{e_1\})$ is a pre-steady-state, as rules E6 and E8S are satisfied.

Case 5: $e_1 \notin F$. Then $t^\star - \epsilon\chi_{\delta^-(s)}$ is a better solution than $t^\star$ for some $\epsilon > 0$, which leads to a contradiction.

Case 6: $t^\star(e_1) < t^\star(e_2)$, $e_1 \in F$, $e_2 \notin F$. Contradiction, as for some $\epsilon > 0$, $t^\star - \epsilon\chi_{\{}e_2\}$ is a better solution to (PSS) than $t^\star$. □

This leads to the iterative algorithm of repeatedly solving the LP and removing an arc from F, that stops after at most $|E| + 1$ iterations, and solves $|E| + 1$ linear program of polynomial size in the worst case.

3.3 A blocking-flow-like algorithm to compute steady-states

Given a sub-steady-state (t, F) for a splitter network $G = (I \uplus S \uplus O, E)$ with capacities $c : I \uplus O \to [0, 1]$. Similar to the blocking-flow algorithm, we employ a notion of residual graph. We notice that throughput values may be bounded by rules E6 and E7. Let s be a splitter, with $\delta^-(s) = \{e, e'\}$, where $e \in F$ and $e' \in E \setminus F$, and $t(e) = t(e')$, then $t(e)$ is lower-bounded and $t(e')$ is upper-bounded by rule E6. We say that e is *upper-tight* and e' is *lower-tight*. A fluid arc $e \in F$ is *loose* when $t(e) < c(e)$ and e is not upper-tight, and a saturated arc $e \in E \setminus F$ is *loose* when $t(e) > 0$ and e is not lower-tight. Thus, an arc is loose when the capacities of all arcs coupled to it may be uniformly increased for fluid arcs, or uniformly decreased for saturated arcs, without violating any steady-state rule except conservation.

Definition 9. Let $G = (I \uplus S \uplus O, E)$ be a splitter network, with capacities $c : I \uplus O \to [0, 1]$ and a sub-steady-state (t, F). The *residual graph* for (t, F) is the pair $(H, \mathcal{C}^=)$ where $H = (V_H, E_H)$ is a graph defined by

- $V_H := \{z\} \cup S$ where z is a new vertex.
- for each loose fluid arc $uv \in F$, $\phi(u)\phi(v) \in E_H$,
- for each loose saturated arc $uv \in E \setminus F$, $\phi(v)\phi(u) \in E_H$,

where $\phi(u) = z$ if $u \in I \cup O$, $\phi(u) = u$ otherwise.

We denote ρ the natural bijection from the loose arcs in E to E_H. For each loose $uv \in E$, let $C_{uv} := \{\rho(u'v') \ : \ u'v' \in E \text{ loose and coupled with } uv\}$ and $\mathcal{C}^= := \{C_{uv} \ : \ uv \in E \text{ loose}\}$. Notice that $\mathcal{C}^=$ forms a partition of E_H, and for all $C \in \mathcal{C}^=$, there is a vertex $v \in V_H$ with $C \subseteq \delta^+_H(v)$. The connected component of H containing z will be called its *main component*.

Coupled arcs must have the same throughput, thus their throughput can only be changed by an equal amount. We define $\mathcal{C}^=$ to be the partition of the arcs into sets of coupled arcs.

Our next result is that a $\mathcal{C}^=$-circulation of the residual graph fills the role of an augmenting flow for the sub-steady-state.

Definition 10. For a sub-steady-state (t, F), we define $\psi(t, F)$ by

$$\psi(t, F) = |\{e \in E \setminus F \ : \ t(e) = 0\}| - |\{e \in F \ : \ t(e) < 1\}| \tag{3}$$

Lemma 7. *Let $G = (I \uplus S \uplus O, E)$ be a splitter network with capacities $c : I \uplus O \to [0, 1]$. Let x be a non-zero circulation on the residual graph H of a sub-steady-state (t, F). Then there is a value $\lambda > 0$ such that $(\bar{t}, F)$ is a sub-steady-state, where*

$$\bar{t}(e) := \begin{cases} t(e) & \text{if } e \in E \text{ is not loose,} \\ t(e) + \lambda x_{\rho(e)} & \text{if } e \in F \text{ is loose,} \\ t(e) - \lambda x_{\rho(e)} & \text{if } e \in E \setminus F \text{ is loose.} \end{cases}$$

Moreover, either $\psi(\bar{t}, F) > \psi(t, F)$, or there is an arc $e \in F$ such that $(\bar{t}, \overline{F} := F \setminus \{e\})$ is a sub-steady-state with $\psi(\bar{t}, \overline{F}) > \psi(t, F)$.

Proof. We compute $\lambda > 0$ to be the maximum value such that $(\bar{t} := t + \lambda x, F)$ is a sub-steady-state. A positive λ exists because the support of x is the set of loose arcs of G, thus choosing λ small enough will not break any sub-steady-state rule. Notice that by definition of $\overline{T}$, the contribution of any arc e to ψ cannot decrease, thus we need only find one arc whose contribution increases. By the maximality of λ, there is a loose arc in (t, F) that is no longer loose in $(\bar{t}, F)$, that is:

- either there is an arc $e \in F$, $t(e) < \bar{t}(e) = 1$, hence $\psi(\bar{t}, F) > \psi(t, F)$;
- or there is an arc $e \notin F$, $t(e) > \bar{t}(e) = 0$, hence $\psi(\bar{t}, F) > \psi(t, F)$;
- or there is a vertex $v \in S$ with $\delta^-(v) = \{e, e'\}$, $e \in F$, $e' \in E \setminus F$ with $t(e') > t(e)$ and $\bar{t}(e') = \bar{t}(e)$ (e and e' stop being loose). By rule E8S, arcs in $\delta^+(v)$ are either saturated or have throughput 1 in t and in $\bar{t}$. Then $(\bar{t}, F \setminus \{e\})$ is a sub-steady-state as rules E6 and E8S are satisfied, and $\psi(\bar{t}, F \setminus \{e\}) > \psi(t, F)$, as the contribution of e increases. □

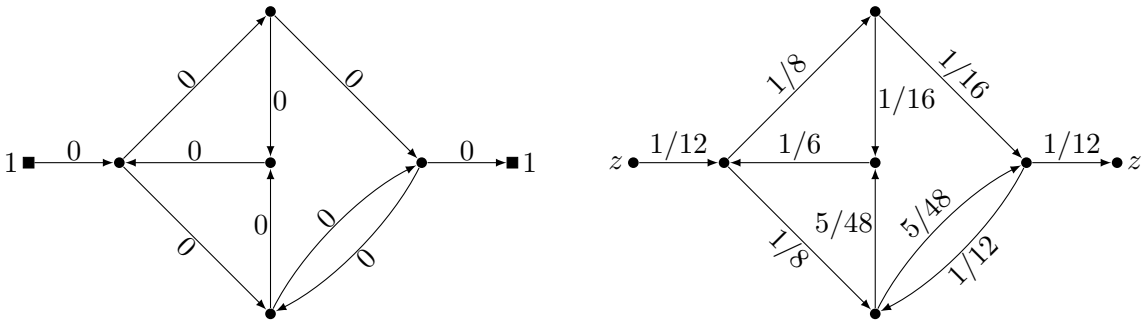


Figure 9: Starting from a trivial sub-steady-state, we compute a residual graph and a stationary circulation in this graph (the two vertices marked z should be identified). Then we increase the throughputs accordingly, as much as possible without violating a sub-steady-state rule, by adding $\lambda = 6$ times the circulation at which point some edge reaches its capacity (see Figure 10).

Lemma 8. *Let (t, F) be a sub-steady-state for a splitter network $G = (I \uplus S \uplus O, E)$ with capacities $c : I \uplus O \to [0, 1]$, and let H be its residual graph. If the main connected component of H contains a sink $v \in S$, then there is an arc $e \in \delta_G(v)$ such that $(t, F \setminus \{e\})$ is a sub-steady-state.*

Proof. First, if there is an edge $e = vw \in \delta^+(v) \cap F$ with $t(e) < c(e)$ but e is not in H, then e is upper-tight, which means that there is a saturated arc $e' = uw \notin F$ with $t(e) = t(e')$. In that case, $(t, F \setminus \{e\})$ is a substeady-state (checking rules E6 and E8S). Similarly, if $e = wv \in \delta^-(v) \setminus F$ is such that $t(e) > 0$ but the reverse of e is not in H, then e is lower-tight: there is a fluid arc $e = uw \in F$ with $t(e) = t(e')$. In that case, $(t, F \setminus \{e'\})$ is a substeady-state.

Now, considering thate v is a sink in H, we observe that for any $e \in \delta^+_G(v)$, either $t(e) = c(e)$ or $e \notin F$, and for any $e \in \delta^-_G(v)$, $t(e) = 0$ or $e \in F$. Let $e \in \delta^-_G(v)$ such that $t(e)$ is minimum. If $e \in F$, then $(t, F \setminus \{e\})$ is a sub-steady-state, as rules E6 and E8S are satisfied. If $e \notin F$ and $t(e) = 0$, then by rule E6 $0 = \sum_{e \in \delta^-_G(v)} t(e) = \sum_{e \in \delta^+_G(v)} t(e)$, hence $\delta^+_G(v) \subseteq E \setminus F$. But v is not isolated, hence there is a fluid arc $e' \in \delta^-_G(v)$ with $t(e') = 0$, and $(t, F \setminus \{e'\})$ is a sub-steady-state, as rules E6 and E8S are satisfied. □

As in an augmenting-path algorithm, we will repeatedly build the residual graph, and deduce either that the current pair (t, F) is a steady-state, or find a better solution.

Lemma 9. *There is an algorithm that, given a continuous splitter network $G = (I \uplus S \uplus O, E)$ with capacities $c : I \uplus O \to [0, 1]$, and a sub-steady-state (t, F), either checks that (t, F) is a steady-state, or find another sub-steady-state $(\bar{t}, \overline{F})$ such that $\psi(\bar{t}, \overline{F}) > \psi(t, F)$ and that runs in time $O(n + \mathrm{sd}(G_z))$, where $\mathrm{sd}(G_z)$ is the complexity of finding a stationary distribution on any residual graph for G.*

Proof. We apply Lemma 3 to the main component of the residual graph H, from which we get one of two cases.

Case 1: there is a non-zero circulation. By Lemma 7, the sub-steady-state can be improved by this circulation into a new sub-steady-state $(\bar{t}, \overline{F})$ with $\psi(\bar{t}, \overline{F}) > \psi(t, F)$.

Case 2: there is a sink v in the main component of H. If $v \notin S$, v is the special vertex z obtained by identifying the vertices of $I \cup O$, then every arc in $\delta^+_G(I)$ is saturated or not loose, which implies that rule E3 is checked, hence (t, F) is a steady-state. If $v \in S$, by Lemma 8 there is a fluid arc $e \in \delta^-_G(v) \cap F$ such that $(t, F \setminus \{e\})$ is a sub-steady-state. □

Proof of Theorem 1. The algorithm repeatedly applies Lemma 9 from the initial substeady-state $(t : e \to 0, F)$. Each iteration takes time $O(n + \mathrm{sd}(G))$ and increases ψ by one. As ψ is initially $-m$, and is at most m, this bounds the number of iterations by $2m$, from which we get the claimed complexity. □

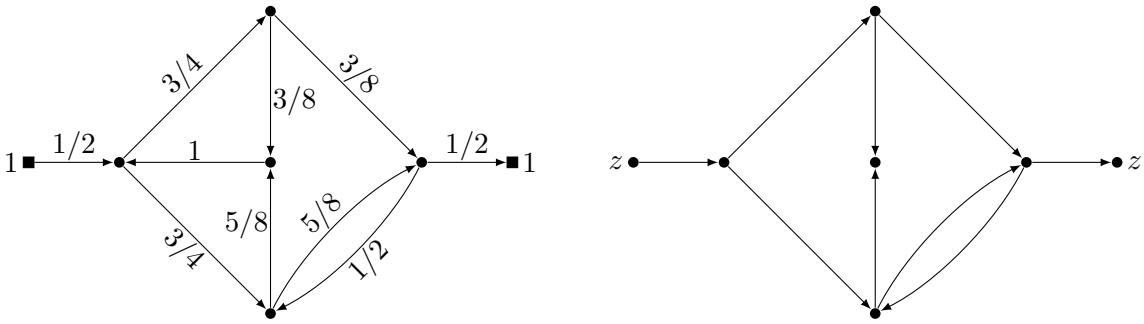


Figure 10: We compute a new residual graph, which does not contain the arc with throughput 1 as this arc cannot increase. Then we fail to compute a stationary circulation in this residual graph (the two vertices marked z should be identified), because there is a sink. We remove from F the incoming arc to the sink with highest throughput from F, and go to the next iteration.

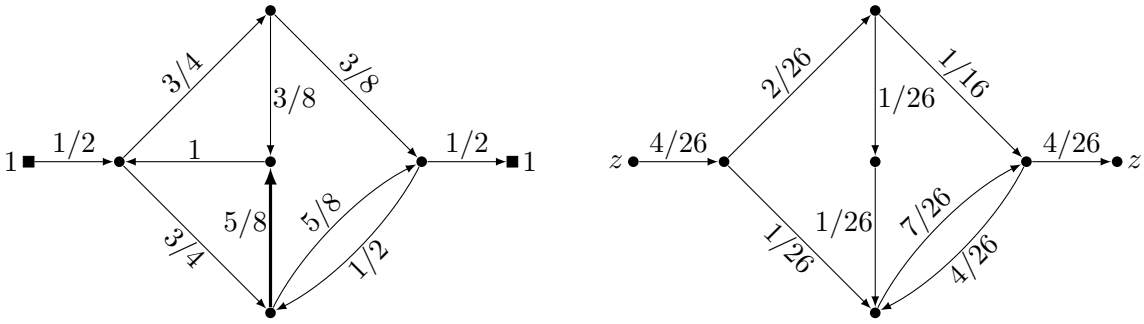


Figure 11: We compute a new residual graph, which contains the reverse of the saturated arc. Then we compute a stationary circulation in this residual graph, which will lead to a better sub-steady-state (taking $\lambda = \frac{3 \cdot 26}{4 \cdot 14}$, another edge reaches throughput 1, see Figure 12)

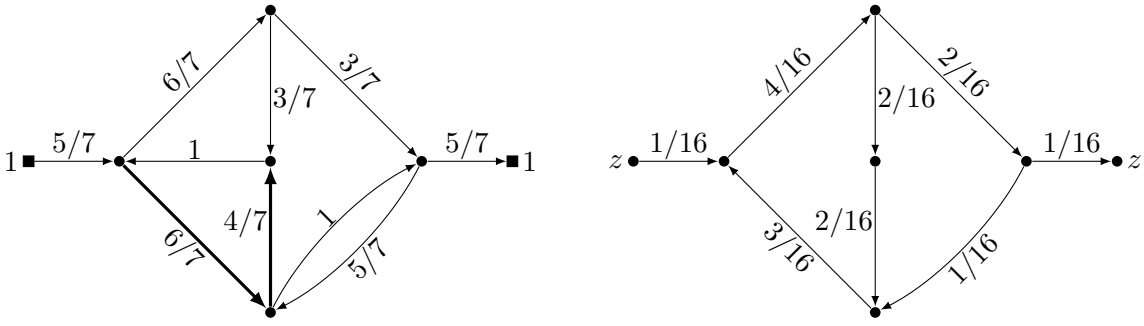


Figure 12: The next residual graph have a sink, thus we remove one more arc from F, and compute the residual graph on the right. We compute one more stationary circulation in this residual graph, which will lead to a better sub-steady-state (here $\lambda = \frac{4}{7}$, as the incoming arcs of the central vertex become tight, see Figure 13)

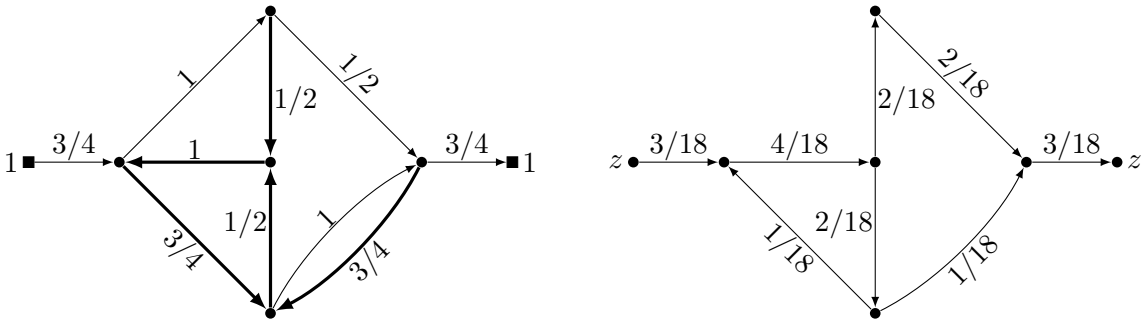


Figure 13: The next residual graphs have sinks, thus we remove more arcs from F, until we get the residual graph on the right. We compute a stationary circulation in this residual graph, which will lead to a better sub-steady-state (taking $\lambda = \frac{9}{14}$, when the incoming arcs of the leftmost splitter become tight, see Figure 14)

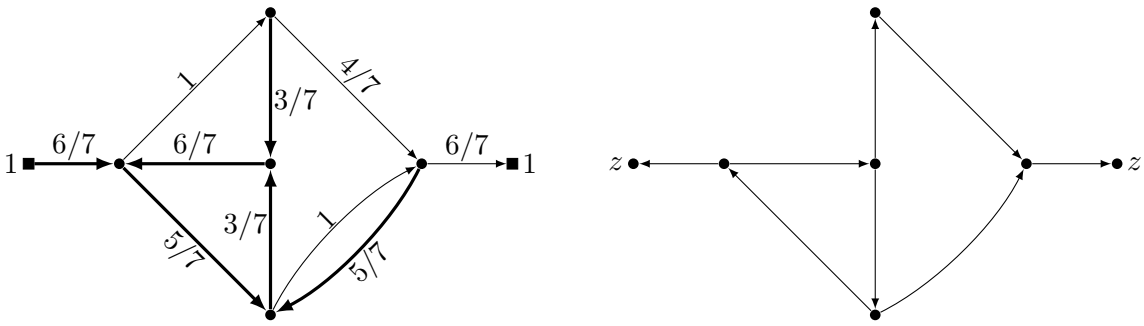


Figure 14: After removing one more arc from F, we get that z is a sink in the residual graph, which means that (t, F) is an steady-state. The algorithm stops.

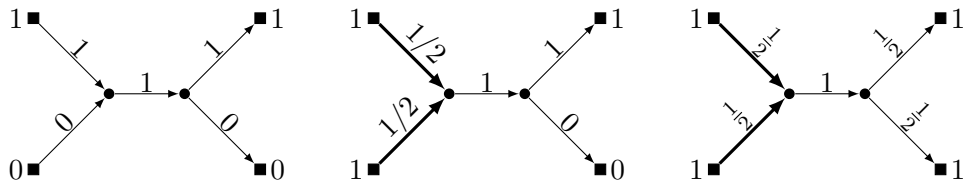


Figure 15: Increasing the input capacities may decrease the throughput of some inputs. Increasing the output capacities may decrease the throughput of some outputs.

3.4 Reverse splitter networks and consequences

We present some consequences of the algorithms and of the following fact that splitter networks can be reversed. Because splitters are fair on their input as well as on their output, splitter networks and steady-states display a useful symmetry. For an arc set X, we denote $\overleftarrow{X} := \{vu \ : \ uv \in X\}$ the set of reverse arcs of X.

Definition 11. Let $G = (I \uplus S \uplus O, E)$ be a splitter network, we denote $\overleftarrow{G}$ the *reverse* of G, defined as the splitter network $((V, \overleftarrow{E}, O, I))$, where the inputs and outputs are interchanged.

Lemma 10. *Let $G = (I \uplus S \uplus O, E)$ be a splitter network with capacities, $c : I \uplus O \to [0, 1]$, and (t, F) a steady-state for (G, c). Then $(t, \overleftarrow{E} \setminus \overleftarrow{F})$ is a steady-state for $(\overleftarrow{G}, c)$.*

Proof. We check that each steady-state-defining rule is satisfied. Rules E1, E2, E5 and E8 can be readily checked. Rules E3 and E4 translates into each other, as do rules E6 and E7. □

Observe that the strong maximization rule E8S does not translate well into $\overleftarrow{G}$, hence the reverse of a steady-state does not necessarily check rule E8S. But by Lemma 4, there is a steady-state $(t, \overleftarrow{E} \setminus \overleftarrow{F'})$ of $(\overleftarrow{G}, c)$ with $F' \subseteq F$ that satisfies rule E8S.

A careful look at the sub-steady-state algorithm shows that, during its resolution, each edge $t(e)$ starts in F with $t(e) = 0$, then $t(e)$ increases, before possibly being removed from F, then $t(e)$ decreases.

Corollary 2. *If (t, F) is a sub-steady-state for (G, c), then there exists an steady-state $(\overline{t}, \overline{F})$ with $\overline{F} \subseteq F$, and for each $e \in E$, if $e \in \overline{F}$ then $\overline{t}(e) \geq t(e)$, and if $e \in F$, $\overline{t}(e) \leq t(e)$.*

If we increase the input capacities to $\overline{c}$, a steady-state (t, F) for (G, c) becomes a sub-steady-state for $G, \overline{C})$. Hence,

Corollary 3. *If (t, F) is a steady-state for (G, c), and $\overline{c}$ be a capacity function with $\overline{c}(u) \geq c(u)$ for each input or output $u \in I \cup O$. There exists a steady-state $(\overline{t}, \overline{F})$ for $(G, \overline{c})$ such that:*

(i) if $\overline{c}(o) = c(o)$ for each output $o \in O$, then $\overline{t}(e) \geq t(e)$ for each $e \in \delta^-(O)$;

(ii) if $\overline{c}(i) = c(i)$ for each input $i \in I$, then $\overline{t}(e) \geq t(e)$ for each $e \in \delta^+(I)$.

Proof. Using the algorithm from Theorem 1, (t, F) is a sub-steady-state, which we can improve into a steady-state $(\overline{t}, \overline{F})$. Notice that if an arc $e \in \delta^-(O)$ is saturated, by rule E4, $t(e) = c(e)$, hence e is not loose. Thus the throughputs of saturated arcs incident to outputs cannot decrease proving (i). Then (ii) follows by taking the reverse network and applying (i) to the reverse steady-state, thanks to Lemma 10. □

Thus increasing the input capacities will not decrease the output throughputs. Notice that increasing the input capacities may lead to the decrease in the throughput of some of the inputs, as can be seen in the example of Figure 15.

4 Design of balancer networks

We first define formally the simple balancer of order k, for $k \geq 0$, whose recursive definition was suggested in Section 1.3, then prove that it is a balancer. For $k = 0$ and $k = 1$, we can use a network without splitter and with a single splitter respectively.

Definition 12. For any integer $k \geq 1$, the *simple balancer of order* k is the network splitter $G = (I \uplus S \uplus O, E)$ where:

- $I := \{i_j : j \in [\![0, 2^k - 1]\!]\}$ and $O := \{o_j : j \in [\![0, 2^k - 1]\!]\}$,
- $S := \{s_{(l,j)} : (l, j) \in [\![0, k]\!] \times [\![0, 2^{k-1} - 1]\!]\}$,
- $E := \{i_j s_{(0,j/2)}, s_{(k,j/2)} o_j : j \in [\![0, 2^k - 1]\!]\} \cup \bigcup_{l \in [\![0,k-1]\!]} E_l$,
- $E_l := \left\{(s_{(l,j)} s_{(l+1,j)}, s_{(l,j)} s_{(l+1,j \oplus 2^l)} : j \in [\![0, 2^{k-1} - 1]\!]\right\}$.

where $\oplus$ is the bitwise exclusive or.

Proposition 2. *For any $k \geq 1$, the simple balancer of order k is a balancer.*

Proof. Let $G = (I \uplus S \uplus O, E)$ be the simple balancer of order k, $c : I \uplus O \to [0, 1]$ with $c(o) = 1$ for each output $o \in O$. For each arc $e = s_{(l,j)} s_{(l+1,j')}$, we define $J_{(l,j)}$ to be the set of inputs i such that there is a directed path from i to e. Clearly, we have that $J_{(l,j)} = J_{(l-1,j)} \uplus J_{(l-1,j \oplus 2^{l-1})}$ and $J_{(l,j)}$ is the set of integers j' such that the binary representations of j and $j'/2$ coincides on the $k - 1 - l$ heavier bits. Then, set

$$t(s_{(l,j)}) := \frac{1}{2^{l+1}} \sum_{\alpha \in J_{(l,j)}} c(i_\alpha)$$

and $F := E$. It can be easily checked that (t, F) is a steady-state where each output has the same throughput. □

4.1 Throughput unlimited balancers

We now define formally the Beneš network, and prove its property of being throughput-unlimited.

Definition 13. A *Beneš network* of order k is a continuous splitter network $G = (I \uplus S \uplus O, E)$ where

- $I := \{i_j : j \in [\![0, 2^k - 1]\!]\}$ and $O := \{o_j : j \in [\![0, 2^k - 1]\!]\}$,
- $S := \{s_{(l,j)} : l \in [\![0, 2k - 1]\!], j \in 0, 2^{k-1} - 1\}$,
- $E := \{i_j s_{(0,j/2)}, s_{(k,j/2)} o_j : j \in [\![0, 2^k - 1]\!]\} \cup \bigcup_{l \in [\![0,2k-1]\!]} E_l$,
- $E_l := \{(s_{(l,j)} s_{(l+1,j)}, s_{(l,j)} s_{(l+1,j \oplus 2^{l'})} : j \in [\![0, 2^{k-1} - 1]\!]\}$ where $l' = \min\{l, l - (2k - 2)\}$.

The arcs between levels l and $l + 1$ represent swapping a bit of weight w, where w varies from k to 1 then to k again. Alternatively, a recursive definition of a Beneš network of order $k + 1$ consists in placing two Beneš networks of order k in parallel. Then identify the inputs from both networks, index by index, into a splitter, and proceed similarly for the outputs (see Figure 7).

Because the latter half of a Beneš network is a simple balancer, it is itself a simple balancer. Beneš networks are known for allowing any possible permutations from inputs to outputs to be realized by arc-disjoint paths, which means that they can be used as crossbar switches in telecommunication networks. We use the same property to show that a Beneš network is actually an unlimited-throughput balancer.

Proposition 3. *Beneš networks are balancer.*

Proof. This follows from the fact that the subgraph of splitters $s_{(l,j)}$ for $l \in [\![k - 1, 2k - 1]\!]$ with their incident arcs is a simple balancer. □

The following lemma is due to Beneš [1], [2]. We include a proof for completeness.

Lemma 11. *Let $\sigma \in S_{2^k}$ be any permutation on $[\![0, 2^k - 1]\!]$. There is a collection of arc-disjoint paths $P_0, \ldots, P_{2^k-1}$ in the Beneš network of order k where each P_j is from i_j to $o_{\sigma(j)}$.*

Proof. We consider the recursive construction of Beneš networks and prove it by induction on the order. It is clear when $k = 1$. Let $k > 1$, then for each $j \in [\![0, 2^k - 1]\!]$, P_j must be routed by either of the Beneš subnetworks A and B of order $k - 1$. For any $2j \in [\![0, 2^k - 1]\!]$, P_{2j} and P_{2j+1} go through $s_{(0,j)}$ and then must each use a distinct subnetwork among A and B, we say that P_{2j} and P_{2j+1} are in *input-conflicts*. Similarly, for $j \in [\![0, 2^k - 1]\!]$, let $j' \in [\![0, 2^k - 1]\!] \setminus \{j\}$ such that $h := \lfloor \sigma(j)/2 \rfloor = \lfloor \sigma(j')/2 \rfloor$, then P_j and P'_j go through $s_{(2k-2,\lfloor \sigma(j)/2 \rfloor)}$ and comes from distinct subnetworks. In that case we say that P_j and $P_{j'}$ are in *output-conflicts*. Consider the graph with vertices $\{P_j : j \in [\![0, 2^k - 1]\!]\}$ and P_j and $P_{j'}$ are adjacent when they are in input-conflicts or output-conflicts. By definition, this is a 2-regular bipartite graph, hence a union of even cycles. Choose every other vertex in each cycle to go through A, and let $B := [\![0, 2^k - 1]\!] \setminus A$, thus deciding the two first and two last arcs of each path. Those are disjoint because paths selected to go through A are not in conflict, hence do not use the same splitters on the first and last layers. By induction on each of the two Beneš subnetworks A and B, we can complete all the paths as desired. □

Proposition 4. *Beneš networks are throughput-unlimited.*

Proof. Let $G = (I \uplus S \uplus O, E)$ be the Beneš network of order k, and $c : I \uplus O \to [0, 1]$. We may assume that $c(I) \leq c(O)$ by the reversibility property of splitter networks Lemma 10. Let (t, F) be a steady-state for (G, c), and suppose that the total throughput $T = t(\delta^+(I))$ is strictly less than $c(I)$.

Let $X \subseteq V$ be the subset of vertices from which O is reachable only through fluid arcs e with $t(e) < c(e)$. Hence, for any edge $e \in \delta^-(X)$, $t(e) = c(e)$ or $e \notin F$. In the latter case, by rule E8, $t(e) = c(e)$ anyway. Because (t, F) is a steady-state, by rule E3, $I \cap X = \emptyset$. Due to the acyclicity of the Beneš network, we must have $t(\delta^-(X)) = T$. Let $I' := \{i \in I : \delta^+(i) \in \delta^-(X)\}$ and $O' := \{o \in O : \delta^-(i) \in \delta^-(X)\}$ and $S_X := \{s_1 s_2 \in \delta^-(X) : s_1, s_2 \in S\}$. Then

$$t(\delta^-(X)) = c(\delta^+(I')) + c(\delta^-(O')) + |S_X|.$$

Observe that $|S_X| \geq \min\{|I \setminus I'|, |O \setminus O'|\}$, because by Lemma 11, we can route that many paths from $I \setminus I'$ to $O \setminus O'$. If $|I \setminus I'| \leq |O \setminus O'|$, then $t(\delta^-(X)) \geq c(\delta^+(I')) + |I \setminus I'| \geq c(I)$. Otherwise, $t(\delta^-(X)) \geq c(\delta^-(O')) + |O \setminus O'| \geq c(O) \geq c(I)$. In both cases we get $T \geq c(I)$. □

4.2 Universal balancer

Before giving a design for universal balancer, we first define a network that behaves like a universal balancer as long as the input and output capacities are at most 1/2.

Definition 14. The *half-universal network of order k* is a continous splitter network $G = (I \uplus S \uplus O, E)$ where

- $I := \{i_j : j \in [\![0, 2^k - 1]\!]\}$ and $O := \{o_j : j \in [\![0, 2^k - 1]\!]\}$;
- $S := S_B$;
- $E := \{uv \in E_B : u, v \in S_B\} \cup F \cup \{i_j s_{(0,j)}, s_{(2k,j)} o_j : j \in [\![0, 2^k - 1]\!]\}$;
- $F := \{s_{(2k,j)} s_{(0,j)} : j \in [\![0, 2^k - 1]\!]\}$;
- $(I_B \uplus S_B \uplus O_B, E_B)$ is the Beneš network of order $k + 1$, and $S_B = \{s_{(l,j)} : l \in [\![0, 2k]\!], j \in [\![0, 2^{k+1} - 1]\!]\}$.

See the left side of Figure 16. It is noteworthy that the half-universal network is its own reverse. By adding a loopback connection from each output back to each input, we ensure that when the throughput of an output reaches its capacity, any excess throughput can be redirected back to the entry of the network, and thus send again to different outputs. These loopback arcs prevent the occurence of saturated arcs inside

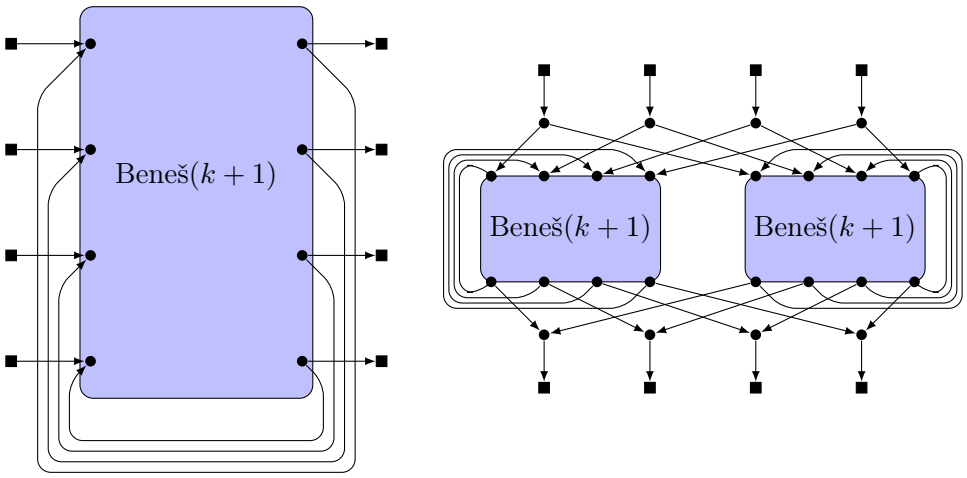


Figure 16: A schematic representation of a half-universal network on the left, and of a universal network on the right.

the Beneš network, as long as at least one output has not reached its maximum capacity. Consequently, the balancing property of the Beneš network is also preserved. This is done at the cost of reducing by half the throughput, as we send back half of the flow. This limitation justifies the name half-universal, and our universal balancer will use two half-universal networks in parallel.

Proposition 5. *Let $G = (I \uplus S \uplus O, E)$ be the half-universal splitter network, and let $c : I \uplus O \to [0, 1/2]$. Then there is a steady-state (t, F) for (G, c) and $\alpha, \beta \in \mathbb{R}_{\geq 0}$ such that*

(i) for each input i, $t(\delta^+(i)) = \min\{c(i), \alpha\}$,
(ii) for each output o, $t(\delta^-(o)) = \min\{c(o), \beta\}$.
(iii) the total throughput $T := t(\delta^+(I))$ equals $\min\{c(I), c(O)\}$.

Proof. We may assume that $c(I) \leq c(O)$ by the reversibility property of splitter networks Lemma 10. We build a steady-state with the stated properties. Let $\beta \geq 0$ be such that $\sum_{o \in O} \min\{c(o), \beta\} = c(I)$. Observe that $\beta \leq \max_{o \in O} c(o) \leq \frac{1}{2}$. Let f_j be the arc $s_{(2k,j)}s_{(0,j)}$ and $J' := \{j \in [\![0, 2^k - 1]\!] : c(o_j) < \beta\}$.

For $j \in [\![0, 2^k - 1]\!]$, set

$$\begin{aligned} t(\delta^-(o_j)) &:= \min\{c(o_j), \beta\}, \\ t(\delta^+(i_j)) &:= c(i_j), \\ t(f_j) &:= 2\beta - \min\{c(o_j), \beta\} \leq 1. \end{aligned}$$

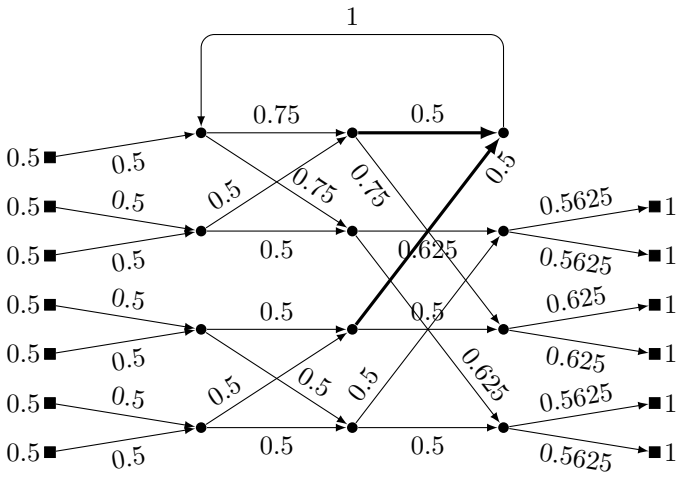
Let $(t^\star, E_B)$ be the steady-state of the Benĕs network of order k, when the input capacities are given by the values of $t(\delta^+(i_j))$ and $t(f_j)$ (for $j \in [\![0, 2^k - 1]\!]$) and the output capacities are uniformly 1. For any arc e in the Beneš subnetwork, set $t(e) := t^\star(e)$.

We claim that $(t, E \setminus \{\delta^-(o_j) : j \in J'\})$ is an steady-state. Indeed, by Proposition 4, the flow going through the Beneš subnetwork is

$$t(I) + t(F) = c(I) + \sum_{o \in O} 2\beta - \min\{c(o), \beta\} = \sum_{o \in O} 2\beta.$$

By Proposition 3, the flow entering any node $s_{(2k,j)}$ must be 2β, which equals the leaving flow. Moreover, Rule E7 is clearly satisfied on node $s_{(2k,j)}$ as $t(f_j) \geq t(\delta^-(o_j))$, concluding the proof. □

Definition 15. The *universal network of order k* is the continuous splitter network built from two disjoint copies of the half universal network of order k, by pairwise identifying the inputs and outputs from both, and adding 2^k input nodes and 2^k output nodes, each with one arc linking it to one of the identified vertices.


1
0.75
0.5
0.5
0.5

Proof of Proposition 1. This follows directly from the definitions for $S(k)$ and $B(k)$. A universal network of order k consists in its two Beneš subnetworks of order $k+1$, plus $2 \cdot 2^k$ additional splitters to connect the inputs and outputs to each half-universal network. In total, we get

$$U(k) = 2 \cdot (2k+1)2^k + 2 \cdot 2^k = (k+1) \cdot 2^{k+2}$$ □

As a consequence, the number of splitters of any balancer discussed in Section 4 is $O(n \log n)$ where $n = \max |I|, |O|$. We proceed to establish a corresponding lower bound.

Proof of Theorem 4. Let $G = (I \uplus S \uplus O, E)$ be a balancer network. Let $c : I \uplus O \to [0,1]$ be a capacity function. We initialize a capacity c' with $c'(i) = 0$ for each $i \in I$ and $c'(o) = c(o)$ for each $o \in O$. We then incrementally increase the capacity of each input i, from 0 to $c(i)$, recalculating a new steady-state at each step. Increasing an input capacity transforms a steady-state into a sub-steady-state, therefore each iteration performs a series of augmentations. Denote by $f_1, \ldots, f_m$ be the sequence of augmenting circulations computed until reaching the final steady-state.

As G is a balancer, and each intermediary sub-steady-state is a steady-state for some choice of input capacities, each augmenting circulation f_k is uniform on $\delta^-(O)$. By construction f_k is non-zero on at most one input arc $e_k \in \delta^+(I)$. We will assume that the support of each circulation intersects $\delta^+(I)$, as other circulations will not contribute to the proof and thus can be ignored.

Consider the support of f_k on G, that is, the subgraph with arcs e with $f_k(e) > 0$. Reverse all saturated arcs, to ensure that f_k is a flow on this graph. Then contract each arc uv with $|\delta^+(u)| = 1$, by identifying u and v and removing uv. This action eliminates all the vertices with out-degree one, except for one input i_k. The resulting graph is denoted H_k, and the flow g_k, the restriction of f_k on H_k, represents a flow from f_k to O. Let δ_k^+ and δ_k^- denote the incidence functions of H_k. The following properties hold:

Claim 2. *(i) each remaining splitter s in H_k has out-degree 2;*
(ii) for each vertex v in H_k, g_k is constant on $\delta_k^+(v)$;
(iii) g_k is constant on $\delta_k^-(O) \cap F$.

Proof. (i) and (ii) follows from the fact that each vertex other than z in the residual graph has out-degree at most 2 and its outgoing arcs are coupled, which itself follows by a simple case analysis on which incident arcs are saturated.

By construction, f_k is an augmenting circulation between two steady-states of a balancer. Consequently, f_k is uniform on the fluid incoming arcs of O. Thus all fluid output arcs increase uniformly at each step of the sub-steady-state algorithm on G, and the total throughputs of those arcs remains uniform. □

We proceed by constructing a directed arborescence rooted at i_k, which may be infinite. This is achieved by establishing a parent-child relation among the walks originating i_k in H. A walk w is a parent of a walk w' if the length of w' is one plus the length of w, and w is a prefix of w'. This defines an arborescence (T, F) whose root r is the empty walk and each node is a walk. Additionally, there is a natural morphism ϕ from each walk $w \in T$ to its end vertex, mapping each parent-child pair to a directed path in H from ϕ(parent) to ϕ(child). Observe that nodes in the arborescence, whose end vertices are splitters, possess two children. In contrast, nodes whose end vertices are outputs have no children. Let us introduce the function $p : F \to [0,1]$, where $p(w, w')$ denotes the probability of a random walk originating from i_k having w' as a prefix. Specifically, because the tree is binary, $p(w, w') = 2^{-\text{depth}(w)}$. For a splitter s, let W_s be the set of all walks whose end vertex is s. The expected number of occurences of s in a random walk starting from i_k, is $E(s) = \sum_{w' \in W_s} p(ww')$.

Using p, we construct a flow on the graph H_k. For each arc a, let $\phi^{-1}(a) \subseteq F$ be the set of edges e in the arborescence such that $a \in \phi(e)$. Then then flow value $f'_k(a)$ on an arc a is defined by $f'_k(a) = \sum_{e \in \phi^{-1}(e)} p(e)$. Due to its construction, f'_k satisfies the conservation rules on each splitter, hence is a flow from i_k to O. Observe that the augmenting flows are defined by a linear system. For each node $v \in S \cup \{i_k\}$

of H_k, this linear system contains one variable, representing the amount of flow on each outgoing arc, and one constraint, representing the conservation rule. The constraints are linearly dependant, as their sum is zero, but have rank one less than the number of variables. Hence the solution space has dimension 1. Therefore f_k and f'_k are identical up to a scaling factor: $f_k = f_k(e_k) \cdot f'_k$. Because f_k is uniform on $\delta^-(O)$, so is f'_k. This implies that (T, F) is the binary decision tree of a sampling process over the uniform distribution on O. By Theorem 2, in the residual graph,

$$\sum_{s\in S} \sum_{e\in\delta_k^-(s)} f_k(e) = f_k(e_k) \sum_{s\in S} E(s) \geq f_h(e_k)|O| \sum_{k\in\mathbb{N}} \frac{k}{2^k} \text{binary}_k(|O|)\,.$$

Summing over all augmenting circulation $f_1, \ldots, f_m$, we obtain:

$$\sum_{k=1}^{m} \sum_{s\in S} \sum_{e\in\delta_k^-(s)} f_k(e) \geq |I||O| \sum_{k\in\mathbb{N}} \frac{k}{2^k} \text{binary}_k(|O|)\,.$$

We conclude the proof by analysing the contribution of each arc to the left-hand side of the inequality. Because of Corollary 2, a fluid arc entering a splitter contributes up to a value of 1, until it reaches a throughput of 1. Then it may contribute as a saturated arc to the splitter from which it as a leaving arc, again by a total amount of at most 1, until it reaches a throughput of 0. Each leaving arc of a splitter contributes at most 1 when saturated, and each entering arc of a splitter contributes at most 1 when fluid. Therefore, the total contribution is $4|S|$, yielding:

$$|S| \geq \frac{1}{4}|I||O| \sum_{k\in\mathbb{N}} \frac{k}{2^k} \text{binary}_k(|O|)\,.$$

In the case when the final steady-state contains only fluid arcs, we do not count the contribution of saturated arcs. This leads to Theorem 3:

$$|S| \geq \frac{1}{2}|I||O| \sum_{k\in\mathbb{N}} \frac{k}{2^k} \text{binary}_k(|O|)\,. \qquad \square$$

6 Perspectives

We formalized splitter networks and their steady-states, and presented various load-balancing designs. The ability to design universal balancers enables the simulation of networks with integral capacities: each arc is replicated according to its capacity, and each splitter is replaced by a universal balancer. A universal balancer is fair by the balancing property, and maximizing by the unlimited-throughput property, effectively generalizing splitters. Our definition of splitter network can also be extended to support arc capacities natively, with most of the proofs requiring only minor modifications. In an extended version of this paper, we will demonstrate how to simulate any rational capacity. Given an arbitrary rational value between 0 and 1, we will design a splitter network, with a single input and a single output, and achieving this value as maximal throughput. However, simulating irrational capacities is not feasible, as the steady-state throughput is a solution to a linear system of inequations.

Although our continuous model is convenient for modeling the expected throughput of a splitter networks, Factorio's belt systems operates discretely. Therefore, the observed throughputs in Factorio splitter networks are only approximations of those theorized by our model. Further investigation into the disparities between the discrete and continuous splitter networks is necessary to accurately apply our findings to Factorio.

We have let several questions unanswered. The most fundamental remains regarding the uniqueness of the steady-state throughput. While it is possible for a single splitter network to admit multiple steady-states, we have yet to encounter a network with two steady-states that yield different throughputs on their outputs.

Our lower bounds for the number of splitters in balancers have a constant multiplicative gap across all designs, indicating they are not tight. For simple balancers of order k, this gap is closed when we forbid saturated arcs in the steady-state of the balancer. Consequently, leveraging saturation is necessary to further reduce the number of splitters in load-balancing networks. Furthermore, it is worth investigating stronger lower bounds in the context of universal balancers.

Factorio allows splitters to be configured to prioritize either an outgoing arc, or an incoming arc. Utilizing this feature, the universal network described in [19] achieves a significantly smaller size compared to our design. Our technique still establishes a lower bound on the number of fair splitters. In general, what is the minimum size achievable for networks utilizing these more general splitters? It is straightforward to extend the definition of steady-state to accomodate unfair splitters. Additionally, in the extended version of this paper, we will provide complexity results for maximizing throughput, by determining which arcs to prioritize in each splitter or subset of those splitters.

Consider a network whose steady-state, when all inputs and outputs have capacity 1, has no saturated arcs. If the augmenting flow from any single input is uniformly distributed across the outputs, then the network is a balancer. This provides a polynomial-time procedure for determining whether a network is a balancer, subject to the absence of saturation. Is it feasible to devise a general procedure to decide whether a splitter network is balancing, throughput unlimited or universal?

References

[1] Václav E Beneš. "On rearrangeable three-stage connecting networks". In: *The Bell System Technical Journal* 41.5 (1962), pp. 1481–1492.

[2] Václav E Beneš. "Permutation groups, complexes, and rearrangeable connecting networks". In: *Bell System Technical Journal* 43.4 (1964), pp. 1619–1640.

[3] Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. "Decremental strongly-connected components and single-source reachability in near-linear time". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. 2019, pp. 365–376.

[4] BoardGameGeek. *Boardgame category: transportation*. https://boardgamegeek.com/boardgamecategory/1011/transportation.

[5] Bonnie S Boardman and Caroline C Krejci. "Simulation of Production and Inventory Control using the Computer Game Factorio". In: *ASEE 2021 Gulf-Southwest Annual Conference*. 2021.

[6] Chase Covello, Hyunjang Jung, and Bryan C Watson. "Using graph theory to investigate the role of expertise on infrastructure evolution: A case study examining the game Factorio". In: (2023).

[7] Miguel Coviello Gonzalez and Marek Chrobak. "Towards a theory of mixing graphs: A characterization of perfect mixability". In: *Theoretical Computer Science* 845 (2020), pp. 98–121. ISSN: 0304-3975.

[8] Yefim Abramovich Dinits. "The method of scaling and transportation problems". In: *Studies in Discrete Mathematics, Moscow* (1973), pp. 46–57.

[9] Shivam Duhan, Chengming Zhang, Wenyu Jing, and Mingqi Li. "Factory optimization using deep reinforcement learning AI". In: (2019).

[10] Andrew V Goldberg and Robert E Tarjan. "A new approach to the maximum-flow problem". In: *Journal of the ACM (JACM)* 35.4 (1988), pp. 921–940.

[11] Ketcheson David. *Mathematics Stackexchange: Belt Balancer problem (Factorio)*. https://math.stackexchange.com/questions/1775378/belt-balancer-problem-factorio.

[12] D.E. Knuth. *The Art of Computer Programming: Sorting and Searching, Volume 3*. Pearson Education, 1998. ISBN: 9780321635785.

[13] Donald Ervin Knuth and AC Yao. *The complexity of nonuniform random number generation. Algorithms and Complexity: New Directions and Recent Results, ed. JF Traub*. 1976.

[14] Andre Leue. "Verification of Factorio Belt Balancers using Petri Nets". PhD thesis. Bachelorarbeit, Darmstadt, Technische Universität Darmstadt, 2021.

[15] MatthaeusHarris. *Factorio belts are Turing-complete*. https://www.reddit.com/r/factorio/comments/lc25cx/factorio_belts_are_turing_complete/.

[16] Carol A. Meyers and Andreas S. Schulz. "Integer equal flows". In: *Operations Research Letters* 37.4 (2009), pp. 245–249. ISSN: 0167-6377.

[17] Amandeep Parmar. "Integer programming approaches for equal-split network flow problems". PhD thesis. Georgia Institute of Technology, 2007.

[18] Sean Patterson, Joan Espasa, Mun See Chang, and Ruth Hoffmann. "Towards Automatic Design of Factorio Blueprints". In: *arXiv preprint arXiv:2310.01505* (2023).

[19] pocarski. *Universal 8-8: Perfectly Balanced, as All Things Should Be*. https://web.archive.org/web/20230922022806/https://alt-f4.blog/ALTF4-27/.

[20] R-O-C-K-E-T. *factorio-SAT: Enhancing the Factorio experience with SAT solvers*. https://github.com/R-O-C-K-E-T/Factorio-SAT.

[21] Kenneth N Reid, Iliya Miralavy, Stephen Kelly, Wolfgang Banzhaf, and Cedric Gondro. "The factory must grow: automation in Factorio". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2021, pp. 243–244.

[22] K Srinathan, Pranava R Goundan, MVN Ashwin Kumar, R Nandakumar, and C Pandu Rangan. "Theory of equal-flows in networks". In: *Computing and Combinatorics: 8th Annual International Conference, COCOON 2002 Singapore, August 15–17, 2002 Proceedings 8*. Springer. 2002, pp. 514–524.

[23] Wube Software. *Factorio*. https://www.factorio.com/.