The steady-states of splitter networks

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

Abstract: We introduce splitter networks, which abstract the behavior 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 [25], a video game published in 2020 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 one or two incoming belts to one or 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

[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.

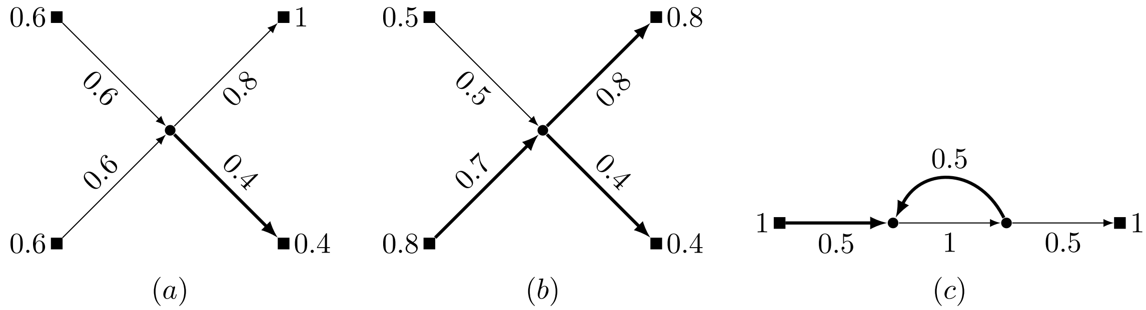

Figure 1: 3 splitter networks given with capacities and associated steady-states. Splitter will be represented by circle vertices, terminals by square vertices. Each terminal is tagged by its capacity, and each arc by its throughput. Saturated arcs are bolder than fluid arcs.

prove that any balancing network on n belts must have $\Omega(n \log n)$ splitters, by exhibiting a relation with 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. A *circulation* on a directed graph is a flow without any excess at any vertex. Given a directed graph (G, A), we denote $\delta^+(v)$ and $\delta^-(v)$ the sets of outgoing and incoming arcs incident to a vertex v. Let $\mathcal{C}^=$ be a partition of A such that for each part $C \in \mathcal{C}^=$, 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 part. 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 [18], even when the partition is exactly the sets of leaving arcs of each vertex [24, 19].

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 [16], 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 developed 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* [21], although it uses non-fair splitters too; our universal balancer only uses fair splitters. They also discovered the relationship with Beneš networks. Factorio-SAT [22] is a project that uses a SAT-solver to find optimal embeddings of splitter networks in the grid. The project *VeriFactory* uses a SAT-solver to check various load-balancing properties of splitter networks [15]. Factorio belts are actually sufficiently complex to be Turing-complete [17]. 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 [23, 20, 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. In Section 6, we define networks that simulate arcs of arbitrary rational capacities. They will be used in Section 7, where we will prove that splitter networks have unique throughputs. In Section 8, we investigate splitter networks when each splitter may receive priorities, prioritizing one incident arc over the others. Using priorities, in Section 9 we will define a balancer with a number of fair splitters closer to our lower bound. Finally in Section 10 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 1.1. 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. For example in Figure 1 (b), as all the outputs have reached their capacities, the splitter cannot accept more flow, even if the bottom input

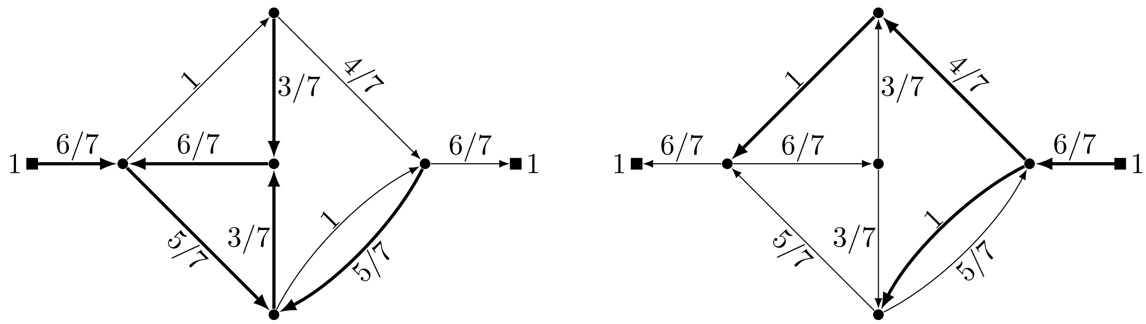

Figure 2: An example of steady-state in a moderately small network, and the reverse network with its steady-state obtained by reversal. Notice that the reversed steady-state satisfies rule R8 but not rule R8S.

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. 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 notions of capacity, throughput and saturations, as well as the behaviour of splitters related to the flow going through the network in a steady state.

Definition 1.2. 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

R1 $t : E \to [0,1]$ is the *throughput* function;
R2 $F \subseteq E$ is the set of *fluid arcs*, $E \setminus F$ is the set of *saturated* arcs;
R3 for each $i \in I$ with $\delta^+(i) = \{e\}$, $t(e) \leq c(i)$ and moreover if $e \in F$ then $t(e) = c(i)$;
R4 for each $o \in O$ with $\delta^-(o) = \{e\}$, $t(e) \leq c(o)$ and moreover if $e \notin F$ then $t(e) = c(o)$;
R5 for each $s \in S$, with $\delta^-(s) = \{e_1, e_2\}$ and $\delta^+(s) = \{e_3, e_4\}$, $t(e_1) + t(e_2) = t(e_3) + t(e_4)$;
R6 for any $e_1, e_2 \in E$ with $\{e_1, e_2\} = \delta^-(s)$ and $e_1 \notin F$, $t(e_1) \geq t(e_2)$;
R7 for any $e_1, e_2 \in E$ with $\{e_1, e_2\} = \delta^+(s)$ and $e_1 \in F$, $t(e_1) \geq t(e_2)$;
R8 for any $uv \in E \setminus F$ and $vw \in F$, $t(uv) = 1$ or $t(vw) = 1$.

Rules R3 and R4 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 R5 imposes the conservation of flow. Rules R6 and R7 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 1.3. Given a 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 R8 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 can 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 not induce any new constraint on the non-dummy arcs as rules R6 and R7 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 R1, R3, R4, R5, R6, R7 can be encoded by linear inequations. Rule R8 is non-convex, but we will later introduce its slight strengthening, rule R8S. 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 R5, 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 relax the rule R3 on input capacities, removing the requirement that an input whose throughput is less than its capacity must have a saturated outgoing arc. This defines the notion of *sub-steady-state*. Given a set F, we solve a linear system to find a sub-steady-state t, and prove once 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 1.4. 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 R1, R2, R4, R5, R6, R7 and the strong maximization rule R8S, 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. Furthermore, rule R8 forces each splitter to send as much flow forward as possible. A non-obvious consequence is that, when increasing the input capacities, throughputs can

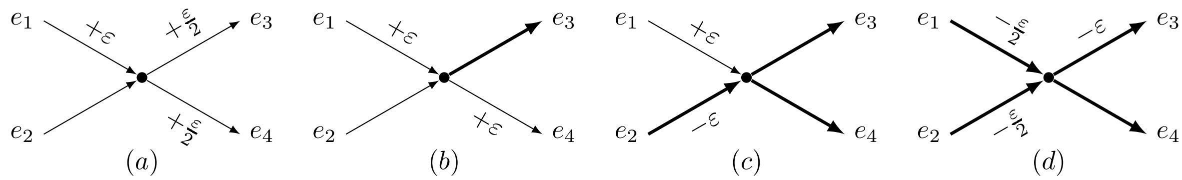

Figure 3: Four examples of throughput changes at a single splitter, depending on which arcs are fluid.

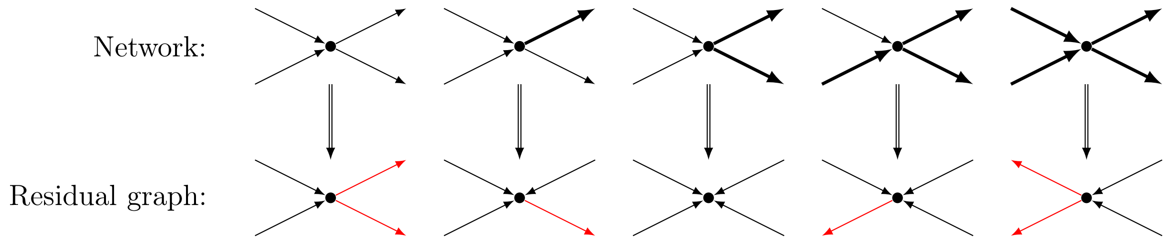

Figure 4: Configurations of splitters and the corresponding vertex in the residual graph. The outgoing arcs from a vertex of the residual graphs are highlighted in red: notice that in a sub-steady-state, the throughputs on these arcs must be equal.

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 R7, 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 $t(e_2)$ decreases while $t(e_1)$ increases, by no more than $(t(e_2) - t(e_1))/2$ because of rule R6. 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 R6, which imposes $t(e_1) \leq t(e_2)$. When $t(e_1) = t(e_2)$, the throughput of e_1 cannot increase, and the throughput of e_2 cannot decrease. We say that e_1 and e_2 are *tight*. In such a case, removing e_1 from F is allowed by rule R6. Fluid arcs e with $t(e) = c(e)$ or saturated arc with $t(e) = 0$ are also *tight*, since we cannot modify their throughput further. Then we define the edge-set of the residual graph to only contain non-tight fluid arcs and reverses of non-tight saturated arcs.

Due to the conservation rule R5, any iterative change to the throughputs of the network must be in accordance with a circulation of the residual graph. Because of rules R6 and R7, 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, obtained from the identification of the inputs and the outputs, 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

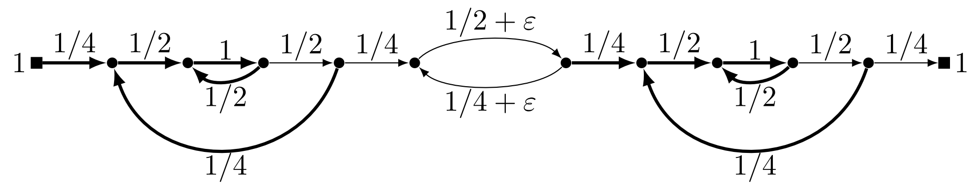

Figure 5: A network having several steady-states. Any value for ε between 0 and $\frac{1}{2}$ gives a steady-state.

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 R3 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 1.5. *There is an algorithm that given a splitter network $G = (I \uplus S \uplus O, E)$ with capacities $c : I \uplus O \to [0, 1]$, finds a steady-state (t, F) in time $O(|S|^2 + |S| \operatorname{sd}(G_z))$, 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 of 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 significantly different steady-states? We will answer this question negatively in Section 7.1, proving that all the steady-states induce the same throughputs on the inputs and outputs of the splitter network.

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 1.6. 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.

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

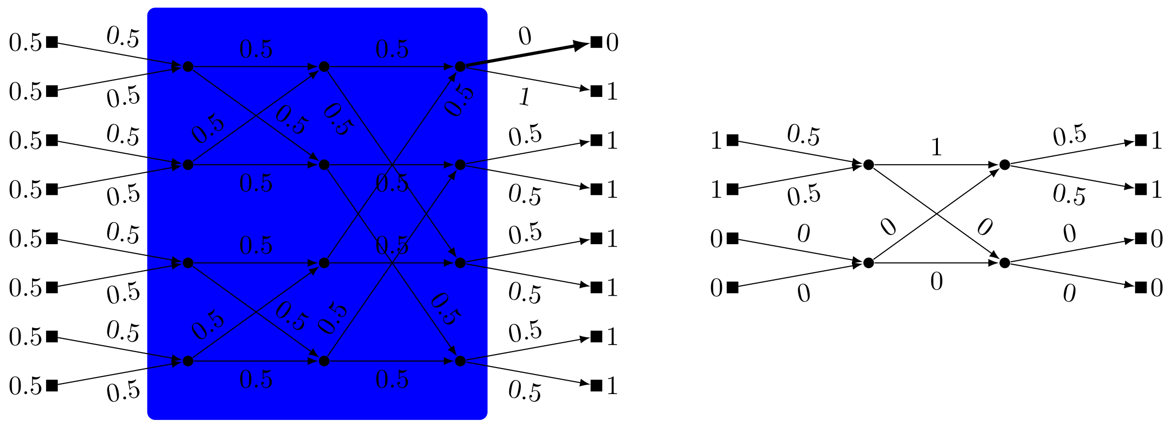

Figure 6: On the left, the simple balancer of order 3, with a steady-state that is not balanced when some output capacity is not 1. The capacity of each input (resp. output) is given at their left (resp. right). On the right, a simple balancer of order 2, with a steady-state with total throughput less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

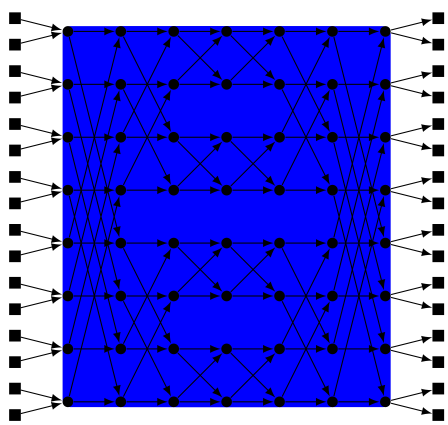

Figure 7: A Beneš network of order 4 with the recursive structure being made explicit.

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 1.7. 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 networks are their own reverses.

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 1.8. 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 1.9. *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 $\{1, \ldots, d\}$, with probabilities given by a distribution $\pi \in [0,1]^d$? First, consider the case when $\pi(i)$ is a rational for each $i \in \{1, \ldots, d\}$, 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 $i \in \{1, \ldots, d\}$. 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 1.10 ([13]). *Let $\pi \in [0,1]^d$ a discrete probability distribution (so $\mathbb{1}\pi = 1$). 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. Because all arcs are still fluid, the sub-steady-state algorithm performs a single iteration. Therefore the increase in throughputs follows 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 every output. This implies that, as we progressively increase each input capacity from 0 to 1, each binary decision tree must uniformly sample 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 1.10. When mapped into the splitter network, for an arc e, the sum of these labels on each

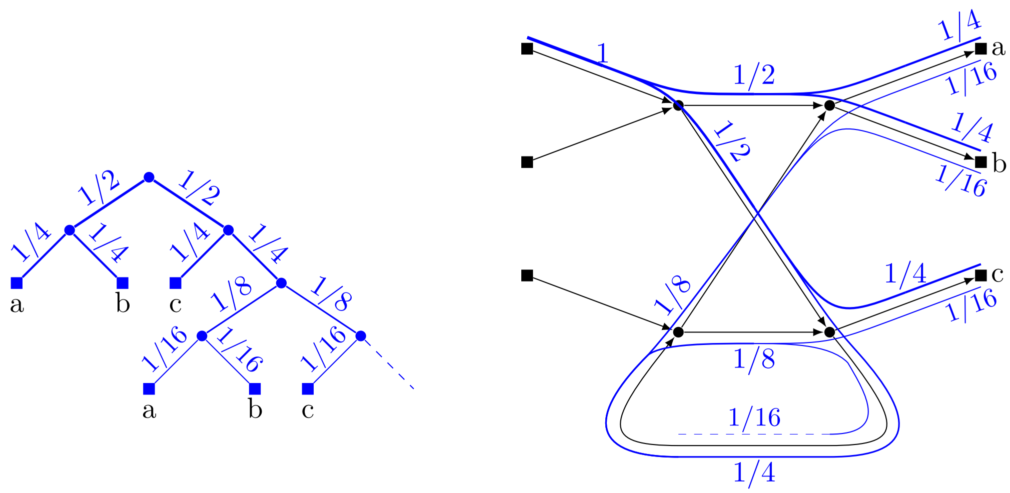

Figure 8: The infinite decision tree (in blue) used to sample uniformly over a three-element set $\{a, b, c\}$ can be embedded from any input into a $(3, 3)$-balancer. Moreover, the sum of the probabilities of the 3 trees, one from each input, will be at most one on any arc, which shows that this network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

edge of the tree mapped to e 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 on balancers, it yields:

Theorem 1.11. *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*

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

For a balancer with $|I| = |O| = 2^k$, since $\sum_{k\in\mathbb{N}} \frac{k}{2^k}\text{binary}_k\left(\frac{1}{|O|}\right) = \frac{k}{2^k}$, 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 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 R8, 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 R6: 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 one 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 3.15, the throughput on each arc increases until it becomes saturated, then decreases. Therefore its throughput varies by at most 2 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 1.12. *Let $G = (I \uplus S \uplus O, E)$ be an (n, p)-balancer. Then*

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

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 $x_e > 0$ satisfying the following linear system:

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

Theorem 2.1. *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 2.2. *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 non-zero circulation, and let $S_0 \cup \ldots \cup S_k$ be the partition given by Theorem 2.1. Then $\delta^+(S_k)$ is empty, indicating that G is not strongly connected. ☐

We can 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 2.3. *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

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

By construction, t is reachable from any vertex in (V, E'). Because $ts \in E'$, s and t are in the same strongly connected component of (V, E'). Let $H = (V_H, E_H)$ be the strongly connected component of (V, E') 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 $\mathbb{1}\pi = 1$. 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 2.4. *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, initially the vertices from which t is not reachable. 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, 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 also 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 2.3 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 2.5. *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 2.3. ☐

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 R8:

R8S for any arcs $uv \in E \setminus F$ and $vw \in F$, $t(vw) = 1$.

Clearly rule R8S implies rule R8. The two rules are actually almost equivalent:

Lemma 3.1. *If (t, F) is a steady-state, then there is a steady-state (t, F') that satisfies rule R8S, with $F \subseteq F'$.*

Proof. By induction on the number of splitters on which rule R8S is not true. Let $uv \notin F$, $vw \in F$ with $t(vw) < 1$, by rule R8 $t(uv) = 1$. 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 R6 is checked on v and rule R7 is checked on u. If $u'v$ is saturated, by rule R6, $t(u'v) = 1$, then $(t, F \cup \{uv, u'v\})$ is a steady-state for similar reasons. Notice that in both cases, the modification preserves rule R8S 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 R5, to define:

Definition 3.2. 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 R1, R2, R3, R4, R6, R7, and R8S, and such that for each splitter $s \in S$,

$$t(\delta^+(s)) \leq t(\delta^-(s)).$$

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

Claim 3.3. *The pair $(t, E(G))$ is a pre-steady-state for (G,c), where*

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

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

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

Conversely, the following lemma is immediate:

Lemma 3.4. *Given $G = (I \uplus S \uplus O, E)$ a 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 3.5. *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 $t^\star(\delta^-(s)) - t^\star(\delta^+(s)) > 0$. Let $\{e_1, e_2\} = \delta^-(s)$ with $t^\star(e_1) \leq t^\star(e_2)$, and $\{e_3, e_4\} = \delta^+(s)$ with $t^\star(e_3) \leq t^\star(e_4)$.

Case 1: there is a fluid leaving arc $e \in \{e_3, e_4\} \cap F$ such that e is in-coupled to an arc $e' \in E \setminus F$ and $t^\star(e) = t^\star(e')$. Then $(t^\star, F \setminus \{e\})$ is a pre-steady-state, as rules R6 and R8S are clearly still satisfied.

Case 2: there is a fluid leaving arc $e \in \{e_3, e_4\} \cap F$ such that $e \in \delta^-(o)$ for some output $o \in O$, and $t^\star(e_3) = c(o)$. Then $(t^\star, F \setminus \{e\})$ is a pre-steady-state, as rules R4 and R7 are clearly still satisfied.

We may now assume that the fluid leaving arcs do not check the conditions for cases 1 and 2. Consequently, increasing the output $t^\star(e)$ to $t^\star(e) + \varepsilon$ for some sufficiently small ε do not break any rule on the destination node of e.

Case 3: e_3 and e_4 are both fluid, with $t^\star(e_3) = t^\star(e_4) < 1$. Then, for some $\varepsilon > 0$ sufficiently small, $(t^\star + \varepsilon\chi_{\{e_3,e_4\}}, F)$ is a pre-steady-state, because we increase the throughput on both arcs uniformly, preserving rule R7. achieving a better objective than $t^\star$. The throughput of this pre-steady-state improves over $t^\star$, a contradiction.

Case 4: exactly one of e_3, e_4 is fluid with throughput strictly less that one, and we may assume it is e_4 by rule R7. Then, for some $\varepsilon > 0$ sufficiently small, $t^\star + \varepsilon\chi_{\{e_4\}}$ is a better solution then $t^\star$, again a contradiction.

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 5: $e_2 \in F$. Then $(t^\star, F \setminus \{e_2\})$ is a pre-steady-state, as rules R6 and R8S are both satisfied.

We may now assume that $e_2 \in E \setminus F$.

Case 6: $e_1 \in F$ and $t^\star(e_1) = t^\star(e_2)$. Then $(t^\star, F \setminus \{e_1\})$ is a pre-steady-state, as rules R6 and R8S are satisfied.

Case 7: $e_1 \in E \setminus F$. Then $t^\star - \varepsilon\chi_{\delta^-(s)}$ is a better solution than $t^\star$ for some $\varepsilon > 0$ sufficiently small, which leads to a contradiction.

Case 8: $t^\star(e_1) < t^\star(e_2)$, $e_1 \in F$, $e_2 \notin F$. Then, for some $\varepsilon > 0$ sufficiently small, $t^\star - \varepsilon\chi_{\{}e_2\}$ is a better solution to (PSS) than $t^\star$, again a contradiction. ☐

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 programs of polynomial size in the worst case.

Notice that the proof still holds with additional constraints $t(e) \geq l_e$ for any fluid arc e, and $t(e) \leq u_e$ for any saturated arc e. This is because the algorithm tries to increase the throughput on fluid arcs, and decrease it on saturated arcs. Therefore, as long as the algorithm can be initialized with a feasible solution, we can find a steady-state that also checks these constraints. The feasible solution is a pre-steady-state that already satisfies those additional constraints. This implies the following corollary.

Corollary 3.6. *Given $G = (I \uplus S \uplus O, E)$ a splitter network, $c : I \uplus O \to [0, 1]$ a capacity function and (t_0, F_0) a pre-steady-state for G, c. Then there exists steady-state (t, F) for G, c such that*

(i) $F \subseteq F_0$;

(ii) for each arc $e \in F$, $t(e) \geq t_0(e)$;

(iii) for each arc $e \in E \setminus F_0$, $t(e) \leq t_o(e)$.

Proof. Find a steady-state with the additional constraints $t(e) \leq t_0(e)$ for each $e \in E \setminus F_0$, and $t(e) \geq t_0(E)$ as long as e is a fluid arc. ☐

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

Let (t, F) be a sub-steady-state for a splitter network $G = (I \uplus S \uplus O, E)$ with capacities $c : I \uplus O \to [0, 1]$. Similar to the blocking-flow algorithm, we employ a notion of residual graph. We notice that throughput values may be bounded by rule R6, which governs the throughput of arcs entering a splitter, and R7, which bounds the throughput leaving a splitter. 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 R6. We say that e is *upper-tight* and e' is *lower-tight*. For any output $o \in O$, with an incoming fluid arc $e \in \delta^-(o)$, if $t(e) = c(o)$, e is also *upper-tight*. Furthermore, rules R6 and R7 imposes some arcs to have equal throughput: out-coupled fluid arcs, or in-coupled saturated arcs. Any modification to such an arc imposes a modification to its coupled arc. Therefore, we say that a fluid arc is *loose* if none of its out-coupled fluid arcs is upper-tight. A saturated arc is *loose* if none of its in-coupled saturated arcs is lower-tight.

Definition 3.7. 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 fluid arc $uv \in F$ with $t(uv) < c(e)$, $\phi(u)\phi(v) \in E_H$,
- ▹ for each saturated arc $uv \in E \setminus F$ with $t(uv) > 0$, $\phi(v)\phi(u) \in E_H$,

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

We denote ρ the natural bijection from the arcs in E to E_H. For each $uv \in E$, let $C_{uv} := \{\rho(u'v') \,:\, u'v' \in E \text{ coupled with } uv\}$ and $\mathcal{C}^= := \{C_{uv} \,:\, uv \in E\}$. 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. Therefore $\mathcal{C}^=$ is the partition of the arcs into the sets of coupled arcs. We first ensure that all arcs of the residual graph correspond to loose arcs.

Lemma 3.8. *Let (t, F) be a sub-steady-state of a capacitated splitter network (G, c), and let H be its residual graph. If there is a non-loose fluid arc e in E_H, or a non-loose saturated arc whose reverse is in E_H, then there is an arc $e' \in F$ such that $(t, F \setminus \{e'\})$ is a sub-steady-state.*

Proof. Let e be a fluid edge with $t(e) < c(e)$ such that e is not loose. By definition of loose, e is out-coupled to a fluid arc e' (possibly e itself) that is upper-tight. If $e' \in \delta^- - (o)$ for some output $o \in O$ and $t(e') = c(o)$, then $(t, F \setminus \{e'\})$ is a sub-steady-state. Otherwise, e' is in-coupled to a saturated arc $e'' \notin F$ with $t(e') = t(e'')$. In that case, $(t, F \setminus \{e'\})$ is a sub-steady-state (checking rules R6 and R8S). Similarly, for a saturated edge e with $t(e) > 0$ that is not loose, e is in-coupled to a lower-tight arc e'. e' is in-coupled to a fluid arc e'' with $t(e') = t(e'')$. In that case, $(t, F \setminus \{e''\})$ is a sub-steady-state. □

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

Definition 3.9. For a sub-steady-state (t, F), we define $\psi(t, F)$ by

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

Lemma 3.10. *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), whose support contains only loose arcs. Then there is a value $\lambda > 0$ such that $(\bar{t}, F)$ is a sub-steady-state, where*

$$\bar{t}(e) := \begin{cases} t(e) + \lambda x_{\rho(e)} & \text{if } e \in F, \\ t(e) - \lambda x_{\rho(e)} & \text{if } e \in E \setminus F. \end{cases}$$

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 $\bar{t}$, 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 reaches its bound in $(\bar{t}, F)$, that is:

- ▹ either there is an arc $e \in F$, $t(e) < \bar{t}(e) = c(e)$, 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' become tight). By rule R8S, arcs in $\delta^+(v)$ either are saturated or have throughput 1 in t and in $\bar{t}$. Then $(\bar{t}, F \setminus \{e\})$ is a sub-steady-state as rules R6 and R8S are satisfied, and $\psi(\bar{t}, F \setminus \{e\}) > \psi(t, F)$, as the contribution of e increases. □

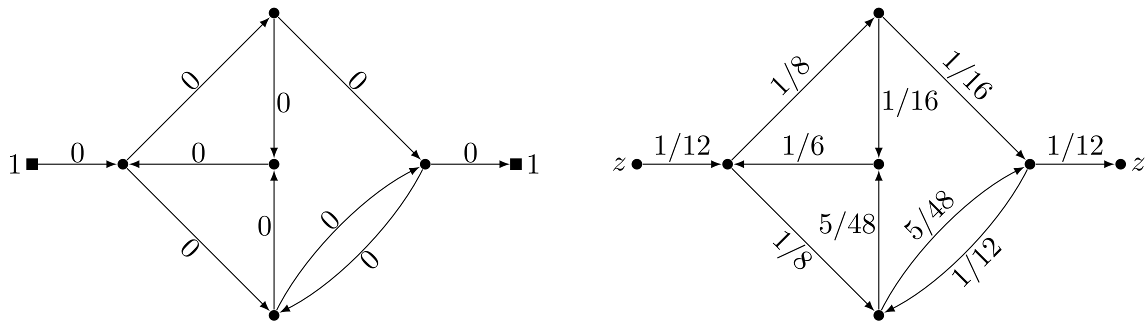

Figure 9: Starting from a trivial sub-steady-state, we compute a residual graph and a stationary circulation in this graph (the two vertices marked z should be identified). Then we increase the throughputs accordingly, as much as possible without violating a sub-steady-state rule, by adding $\lambda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

Lemma 3.11. *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. If H contains a non-loose arc, it suffices to apply Lemma 3.8. Assume that all residual arcs are loose.

Consider a sink v in H. We may now assume 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 maximum. If $e \in F$, then $(t, F \setminus \{e\})$ is a sub-steady-state, as rules R6 and R8S are satisfied. If $e \notin F$, then $t(e) = 0$, and by rule R6 $0 = \sum_{e \in \delta^-_G(v)} t(e) = \sum_{e \in \delta^+_G(v)} t(e)$, hence $\delta^+_G(v) \subseteq E \setminus F$. Because v is not isolated, the arc e' out-coupled to e must be fluid with $t(e') = 0$, and $(t, F \setminus \{e'\})$ is a sub-steady-state, as rules R6 and R8S are satisfied on v and rule R7 is satisfied on the origin of e'. ☐

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 3.12. *There is an algorithm that, given a 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 3.8 until the residual graph H contains only loose arcs, then we apply Lemma 2.5 to the main component of H.

Case 1: there is a non-zero circulation. By Lemma 3.10, 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 tight, which implies that rule R3 is checked, hence (t, F) is a steady-state. If $v \in S$, by Lemma 3.11 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 1.5. The algorithm repeatedly applies Lemma 3.12 from the initial sub-steady-state $(t : e \to 0, F)$. Each iteration takes time $O(|E| + \mathrm{sd}(G))$ and increases ψ by at least one. As ψ is initially $-|E|$, and is at most $|E|$, this bounds the number of iterations by $2|E|$, from which we get the claimed complexity. ☐

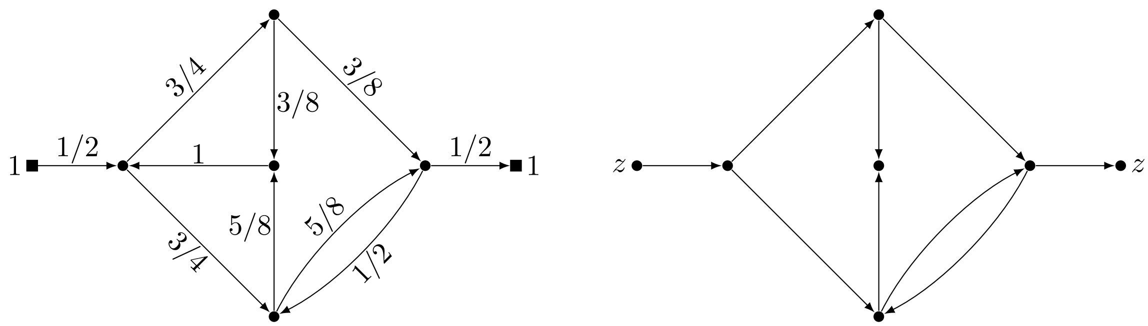

Figure 10: We compute a new residual graph, which does not contain the arc with throughput 1, since this arc cannot increase. Then the existence of a sink prevents us to find a stationary circulation in this residual graph (the two vertices marked z should be identified). We remove from F the incoming arc to the sink with highest throughput, and go to the next iteration.

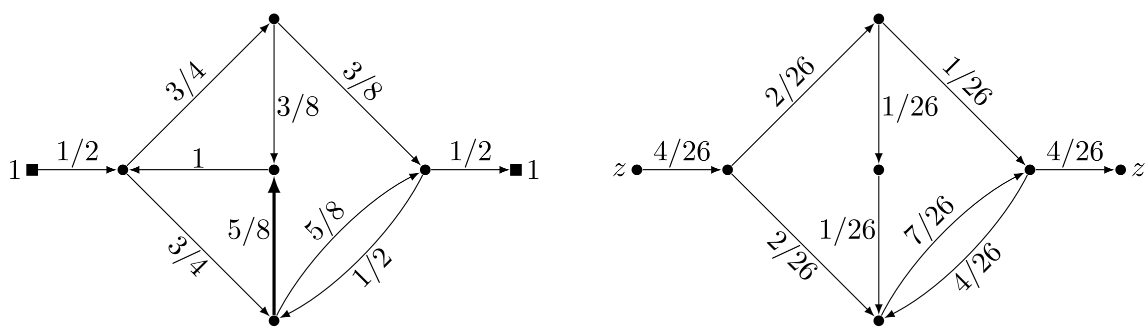

Figure 11: We compute a new residual graph, including the reverse of the newly saturated arc. Then we compute a stationary circulation in this residual graph. This leads to an improved sub-steady-state (taking $\lambda = \frac{3 \cdot 26}{4 \cdot 14}$, when another edge reaches throughput 1, see Figure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

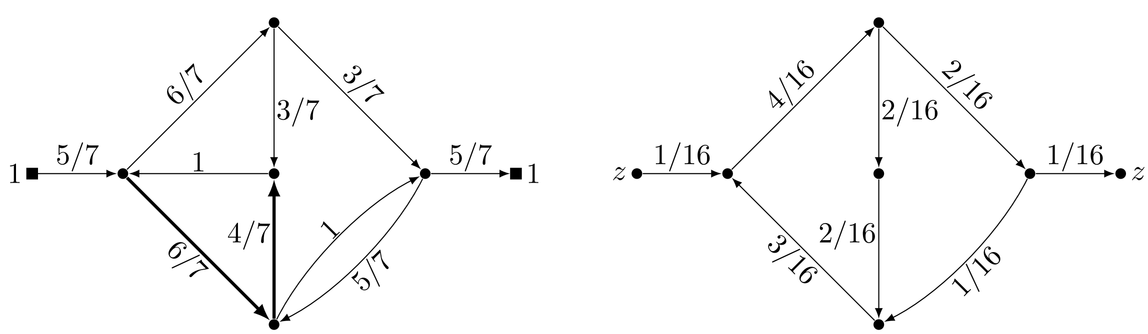

Figure 12: The next residual graph contains a sink, thus we remove one more arc from F. This yields the residual graph on the right. We compute the stationary circulation in this residual graph. Then we imporve the sub-steady-state according to this circulation (here $\lambda = \frac{4}{7}$, when the incoming arcs of the central vertex become tight, see Figure 13)

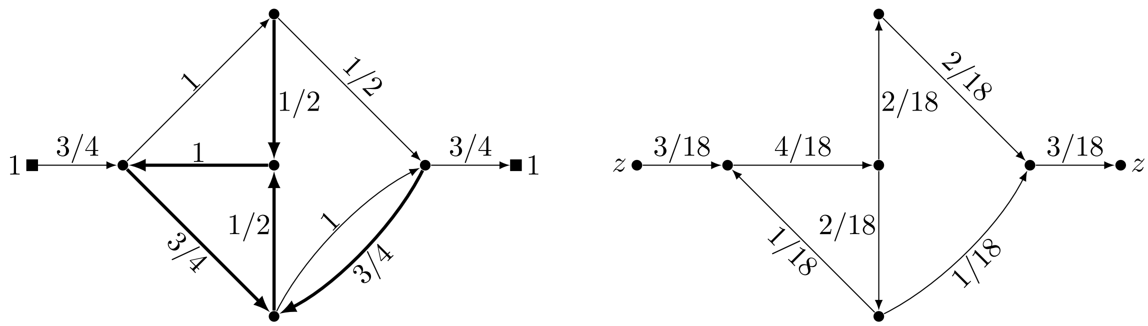

Figure 13: Again the residual graphs contains sinks, that we remove by making some arcs saturated. This yields the residual graph on the right, free of any sink. We compute a stationary circulation in this residual graph, and use it to improve the sub-steady-state (taking $\lambda = \frac{9}{14}$, when the incoming arcs of the leftmost splitter become tight, see Figure 14)

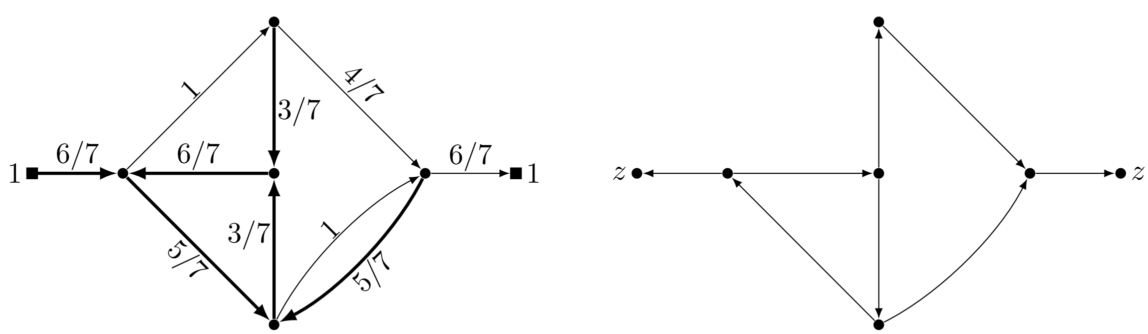

Figure 14: After removing one more arc from F, z becomes a sink in the residual graph. It implies that (t, F) is a steady-state. The algorithm stops.

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 3.13. 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 $(O \uplus S \uplus I, \overleftarrow{E})$, where the inputs and outputs are interchanged.

Lemma 3.14. *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 R1, R2, R5 and R8 can be readily checked. Rules R3 and R4 translates into each other, as do rules R6 and R7. $\square$

Observe that the strong maximization rule R8S does not translate well into $\overleftarrow{G}$, hence the reverse of a steady-state does not necessarily check rule R8S. But by Lemma 3.1, there is a steady-state $(t, \overleftarrow{E} \setminus \overleftarrow{F'})$ of $(\overleftarrow{G}, c)$ with $F' \subseteq F$ that satisfies rule R8S.

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 3.15. *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 $e \in E \setminus F$, $\overline{t}(e) \leq t(e)$.*

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 3.16. *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. (t, F) is a sub-steady-state, which we can improve using the algorithm from Theorem 1.5. Let $(\overline{t}, \overline{F})$ be the resulting steady-state. Notice that if an arc $e \in \delta^-(O)$ is saturated, by rule R4, $t(e) = c(e)$, hence e is not loose. Thus the throughputs of saturated arcs incident to outputs cannot decrease. This proves (i). Then (ii) follows by taking the reverse network and applying (i) to the reverse steady-state, thanks to Lemma 3.14. $\square$

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 $k = 1$, the networks without splitter and with a single splitter respectively are universal balancers. We denote $[\![l, u]\!]$ the set of integers i with $l \leq i \leq u$.

Definition 4.1. For any integer $k \geq 2$, the *simple balancer of order* k is the network splitter $G = (I \uplus S \uplus O, E)$ where:

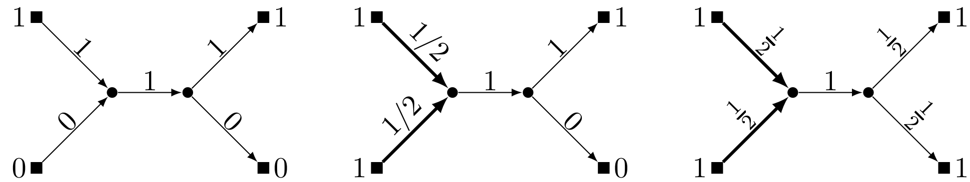

Figure 15: Increasing the input capacities may result in reduced effective throughputs on some inputs, as if they are in competition. Similarly, increasing the output capacities may also result in reduced effective throughputs on some outputs.

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

where $\oplus$ is the bitwise exclusive or.

Proposition 4.2. *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. $\square$

4.1 Throughput-unlimited balancers

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

Definition 4.3. A *Beneš network* of order k is a splitter network $G = (I \uplus S \uplus O, E)$ where

- $\triangleright$ $I := \{i_j : j \in \llbracket 0, 2^k - 1 \rrbracket\}$ and $O := \{o_j : j \in \llbracket 0, 2^k - 1 \rrbracket\}$,
- $\triangleright$ $S := \{s_{(l,j)} : l \in \llbracket 0, 2k-2 \rrbracket, j \in \llbracket 0, 2^{k-1} - 1 \rrbracket\}$,
- $\triangleright$ $E := \{i_j s_{(0,j/2)}, s_{(2k-2,j/2)} o_j : j \in \llbracket 0, 2^k - 1 \rrbracket\} \cup \bigcup_{l \in \llbracket 0, 2k-3 \rrbracket} E_l$,
- $\triangleright$ $E_l := \{(s_{(l,j)} s_{(l+1,j)}, s_{(l,j)} s_{(l+1,j \oplus 2^{l'})} : j \in \llbracket 0, 2^{k-1} - 1 \rrbracket\}$ where $l' = \max\{k-2-l, l-k+1\}$.

The arcs between levels l and $l+1$ represent swapping a bit of weight w, where w varies from $k-2$ to 0 then to $k-2$ 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.

Proposition 4.4. *Beneš networks are balancer.*

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

Proposition 4.5. *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 $c(I) \geq c(O)$ by the reversibility property of splitter networks Lemma 3.14.

We define a new splitter network $G_L = (V_L, E_L)$, by removing O and all vertices $s_{(l,j)}$ with $l \geq k$, and adding 2^k new outputs O', with arcs $s_{(k,j/2)}o'_j$ for each $j \in [\![0, 2^k - 1]\!]$. G_L is a simple balancer of order k between I and O'. Symmetrically we define $G_R = (V_R, E_R)$, a simple balancer of order k, by removing I and all vertices $s_{(l,j)}$ with $l < k-1$, and adding a set I' of new inputs.

Consider $c'_L : I \cup O' \to [0,1]$ defined by $c'_L(I) = c(I)$ and $c'_L(O) = \mathbb{1}$. By Proposition 4.2, there exists a steady-state (t'_L, F'_L) for G_L, c_L, such that for each $e \in \delta^-(O)$, $t'(e) = c(I)/2^k$, and $F'_L = E_L$.

Next consider $c_R : I' \cup O \to [0,1]$ defined by $c_R(I') = 1$ and $c_R(O) = c(O)$. The reverse of G_R is a simple balancer of order k. Therefore by Proposition 4.2, there exists a steady-state $(\overleftarrow{t_R}, \overleftarrow{F_R})$ for $\overleftarrow{G_R}, c_R$, such that for each $\overleftarrow{e} \in \delta^-(I')$ of the reverse graph, $t(\overleftarrow{e}) = c(O)/2^k$ and $\overleftarrow{e} \in \overleftarrow{F_R}$. Reversing the network and the steady-state, by Lemma 3.14, $(t_R : e \to \overleftarrow{t_R}(\overleftarrow{e}), F_R = E_R \setminus \overleftarrow{F_R})$ is a steady-state in G_R. Furthermore, for each arc $e \in \delta^+(I')$, $t_R(e) = c(O)/2^k$ and $e \in E_R \setminus F_R$ is saturated.

We plan to build a steady-state using t'_L and t_R. However when $c(I) > c(O)$, this would lead to positive excess on splitters $s_{(k-1,j)}$. To avoid this, we decrease the throughputs on G_L with the following procedure. Let $c_L : I \uplus O' \to [0,1]$ denote a capacity function with $c_L(I) = c(I)$ and $c_L(o) = c(O)/2^k$ for each $o \in O'$. Then define a throughput function $\bar{t}_L$, where $\bar{t}_L(e) = c_L(o)$ for each $o \in O'$ and $\{e\} = \delta^-(o)$, and $\bar{t}_L(e) = t'_L(e)$ for any other arc $e \notin \delta^-(O')$. Then (t'_L, E_L) is a pre-steady-state for (G_L, c_L), with positive excess on splitters $s_{(k-1,j)}$ when $c(I) > c(O)$. Using the pre-steady-state algorithm and Lemma 3.5, there exists a steady-state (t_L, F_L) for G_L, c_L. Furthermore, let $e \in \delta^-(O')$. If e is fluid, then by Corollary 3.6, $c(e) = c(O)/2^k$. If e is saturated, then by rule R4, $c(e) = c(O)/2^k$ also.

Finally, we define $t(e) = t_L(e)$ if $e \in E \cap E_L$, and $t(e) = t_R(e)$ if $e \in E \cap E_R$. Let $s = s_{(k,j)}$ for some $j \in [\![0, 2^k - 1]\!]$. Then $t(\delta^-(s)) = c(O)/2^{k-1} \geq t'(\delta^+(s)) = c(O)/2^{k-1}$. Let $F = F_L \cup F_R$. Then the arcs in $\delta^+(s)$ are saturated. Because rules R6, R7 and R8 hold on s and on each splitter, (t, F) is a steady-state, with t uniform on $\delta^-(O)$. $\square$

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 4.6. 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. The half-universal network is its own reverse. Its design includes loopback connections from each output back to each input. When the throughput of an output reaches its capacity, any excess throughput can be redirected back through one of these loopback arc, to the entry of the network. From there, it flows again to different outputs. Hence, the loopback arcs prevent the occurence of saturated arcs inside 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 4.7. *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*

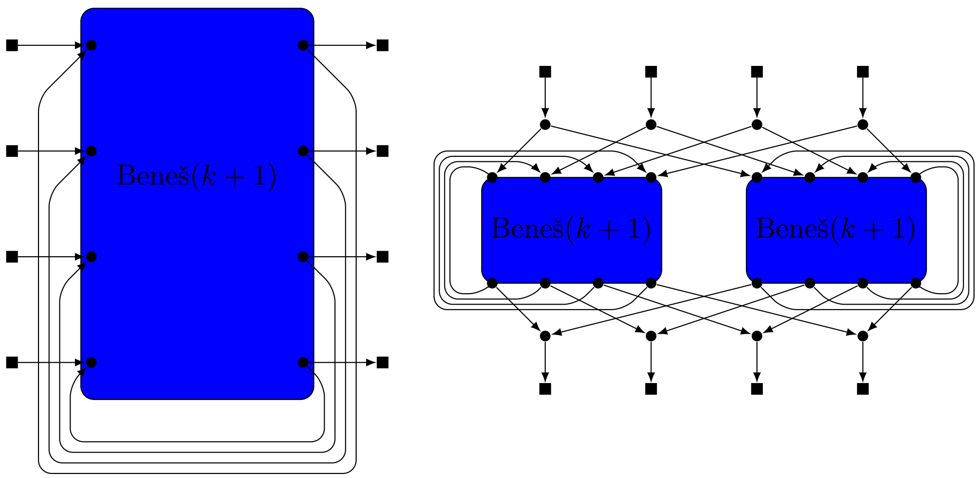

Figure 16: A schematic representation of a half-universal network on the left, and of a universal network on the right.

(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 Lemma 3.14, because the half-universal network is its own reverse. 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 loopback 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 Beněs 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 a steady-state. Indeed, by Proposition 4.5, 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 4.4, the flow entering any node $s_{(2k,j)}$ must be 2β, which equals the leaving flow. Moreover, Rule R7 is clearly satisfied on node $s_{(2k,j)}$ as $t(f_j) \geq t(\delta^-(o_j))$, concluding the proof. ☐

Definition 4.8. The *universal network of order k* is the splitter network built from two disjoint copies of the half universal network of order k, by pairwise identifying the inputs and outputs from both copies, and adding 2^k input nodes and 2^k output nodes, each with one arc linking it to one of the identified vertices.

The right side of Figure 16 illustrates the construction of a half-universal network. Again, universal networks are their own reverse.

Theorem 4.9. *The universal network of order k is universally balancing.*

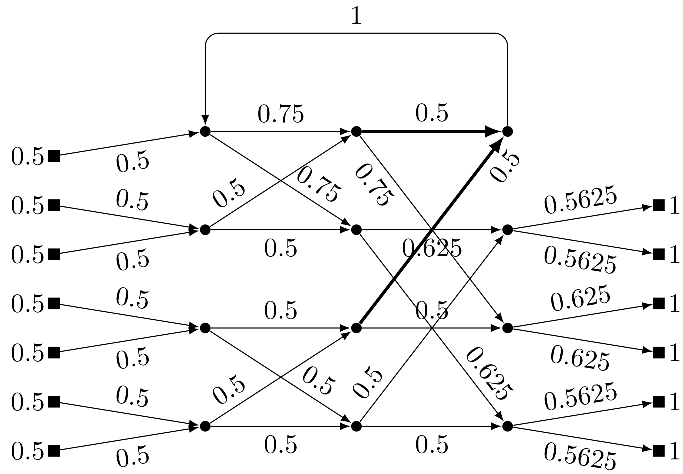

Figure 17: Adding loops on a simple balancer to reduce the number of inputs and outputs does not necessarily preserve the balancing property.

Proof. Let $c : I \uplus O \to [0, 1]$ be a capacity function on the universal balancer G of order k. Consider a half-universal balancer $H = (I_H \uplus S_H \uplus O_H, E_H)$ of order $k - 1$, let $c_H : I_H \uplus O_H \to [\![0, \frac{1}{2}]\!]$ be a capacity function defined by $c_H(i_j) = \frac{1}{2}c(i_j)$ and $c_H(o_j) = \frac{1}{2}c(o_j)$. Let (t_H, F_H) a steady-state for (H, c_H) given by Proposition 4.7. Then one can define a steady-state (t, F) for (G, c) by using t_H on both sides of G and completing t to the arcs incident to output and input nodes, and easily check that it has the expected properties. $\square$

4.3 Balancers of all sizes

We extend the construction of balancers to general (n_i, n_o)-balancers. Given n_i and n_o, two positive integers representing respectively the number of inputs and outputs, let $k \in \mathbb{N}$ such that $m := \max\{n_i, n_o\} \leq 2^k$. Let $G = (I \uplus S \uplus O, E)$ be the universal network with 2^k inputs and 2^k outputs. Remove any surplus inputs or outputs to achieve a network with n_i inputs and n_o outputs. This can be accomplished by setting their capacities to 0. We obtain a universally balancing network with n_i inputs and n_0 outputs.

Another method to decrease the number of outputs and inputs would be to add loopback arcs from the surplus output to the surplus input. This would avoid the usage of dummy terminals. Loopback arcs are also useful to reduce the number of inputs and outputs in non-universal balancer. However, preserving the balancer property when adding loopback arcs requires some care. Indeed, in a simple balancer, balancing is achieved when the capacities of the outputs are uniformly 1. Therefore it is necessary to ensure that the loopback arcs remain fluid. A careless implementation of this process may result in a network failing to be a balancer, as illustrated by Figure 17.

5 Lower bounds for the size of balancers

We begin this section by establishing Proposition 1.9, followed by the derivation of a lower bound on the number of splitters in a balancer.

Proof of Proposition 1.9. 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

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

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 1.12. 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 $f_1, \ldots, f_m$ 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 $e_k \in \delta^+(I)$. The support of f_k is defined as the subgraph of all arcs e such that $f_k(e) > 0$. We will assume that the support of each circulation intersects $\delta^+(I)$ (and thus increases the global throughput), as other circulations will not contribute to the proof and thus can be ignored.

Consider the support of f_k on G. Reverse all saturated arcs, to ensure that f_k is a flow on this graph. Then contract each arc $uv \neq e_k$ 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 i_k to O. Let δ_k^+ and δ_k^- denote the incidence functions of H_k. The following properties hold:

Claim 5.1. *(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) follow from the fact that each vertex other than i_k in the residual graph has out-degree at most 2 and its outgoing arcs are coupled. This last fact 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 the difference of two throughput functions that are both uniform on $\delta^-(O)$. Therefore f_k is itself uniform on fluid arcs of $\delta^-(O)$, and so is g_k, proving (iii). ☐

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 from 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, E_T) 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 $\phi(\text{parent})$ to $\phi(\text{child})$. 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 $p : E_T \to [0, 1]$, 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 $\Lambda(s) = \sum_{w' \in W_s} p(ww')$.

Using p, we construct a flow on the graph H_k. For each arc a, let $\phi^{-1}(a) \subseteq E_T$ 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, E_T) is the binary decision tree of a sampling process over the uniform distribution on O. By Theorem 1.10, in the residual graph,

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

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

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

We conclude the proof by analysing the contribution of each arc to the left-hand side of the inequality. Because of Corollary 3.15, 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 at most $4|S|$, yielding:

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

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 1.11:

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

6 Simulating an arbitrary capacity

We present a way to build, for any rational $r \in [0,1]\cap\mathbb{Q}$, a continuous splitter network with one input vertex i and one output vertex o, such that, when $c(i) = 1$, then the throughput of any equilibrium of this network is $\min\{r, c(o)\}$. Those networks can be used as gadgets to simulate arcs with rational capacities in larger networks, thanks to the additional property that the arc leaving i may be set as saturated if and only if its throughput is r or $c(o)$.

Let $r = \frac{p}{q} \in \mathbb{Q}$ be a positive rational with $0 < p < q$. Let $k \in \mathbb{N}$ be such that $2^{k-1} < q \leq 2^k$, $k >= 1$.

We construct a splitter network from a complete binary out-arborescence of depth k from a source splitter s. The throughput entering s is fairly split among the 2^k leaves. Then we partition the leaves into three sets P, Q, R, with $|P| = p$, $|Q| = q - p$ and $|R| = 2^k - q$. Route all the flow from R back to the root of the tree, by adding an in-arborescence from these leaves to s. Then a random walk from s ends in P with probability $\frac{p}{q}$, and in Q with probability $1 - \frac{p}{q}$. If we send all the flow from P to an output, and all the flow from Q back to the input, as illustrated in Figure 18, we get a splitter network with maximal throughput r.

Informally, the arc $s's$ entering the root of the main arborsecence serves as the bottleneck of that network. Its throughput, when capped at 1, equals the sum of the input throughput and the throughputs from $q-p$ leaves. By the rule of conservation R5, the input throughput equals the output throughput, which is the sum of throughputs of p leaves. Therefore the throughput x at each leaf satisfies $px+(q-p)x = t(ss')$, that is $x = \frac{t(ss')}{q}$. It follows that, when $t(ss')$ is capped at 1, the input throughput is $\frac{p}{q} = r$. We do not prove precisely this result because this construction has two main shortcomings:

- it only holds as long as $\frac{p}{q} \geq \frac{1}{2}$. Otherwise, the tree collecting the flow back from the $p - q$ leaves to the input would become saturated, while we would rather have that the arc is' is saturated.

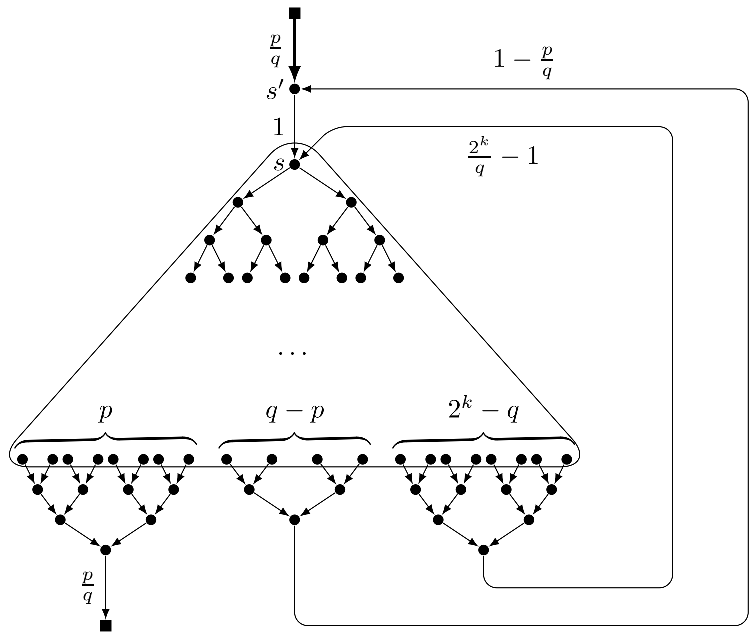

Figure 18: An inefficient network to simulate a rational capacity of $\frac{p}{q}$, where $2^{k-1} < q \leq 2^k$. This only works when $\frac{p}{q} \geq 1 - \frac{p}{q}$, because we expect the arc leaving the input node to saturate as soon as the throughput reaches $\frac{p}{q}$.

▹ the size of this construction is linear in q (hence exponential in the size of the encoding of the capacities).

Optimizing over that solution would overcome these shortcomings, but we will rather directly define a correct network with an optimal number of splitters (up to a multiplicative constant). Before introducing an efficient design, we consider another construction based on the binary expansion of r. Because $r = \frac{p}{q}$ is rational, its binary expansion is ultimately periodic: there exist binary words $x, y \in \{0,1\}^\star$ with $\text{binary}(r) = 0.xy^\omega$. Let $l = |x| + |y|$. We define the splitter network $G'_r := (V := I \uplus S \uplus O, E := E_0 \cup E_r \cup E_{1-r})$ with

▹ $I := \{i\}$, $O := \{o\}$,

▹ $S := \bigcup_{j=1}^{l} \{u_j, v_j, w_j\}$,

▹ $E_0 := \{iv_l, v_1u_1, w_lo, u_lu_{|y|+1}\} \cup \bigcup_{j=1}^{l-1} \{u_ju_{j+1}, v_{j+1}v_j, w_jw_{j+1}\}$,

▹ $E_r := \{u_iv_i \ : \text{ for each } i \in [\![1, l]\!] \text{ with } (xy)_i = 0\}$,

▹ $E_{1-r} := \{u_iw_i \ : \text{ for each } i \in [\![1, l]\!] \text{ with } (xy)_i = 1\}$.

Then G_r is obtained by contracting any splitter s with $d^+(s) = d^-(s) = 1$ into a single arc. Figure 19 shows an example of network G_r with $r = \frac{169}{504}$. One way to understand this design is to unfold the loop made by $u_{l-1}u_{|y|+1}$, making the u, v and w lines extend infinitely to the right; this gives a network that simply split the flow following the binary expansion of r, each vertex u_i distributing 2^{-i} unit of flow. This construction seems more compact than the construction based on trees, but can still have a large number of splitters, as l can be as large as $q-1$. Also, it still needs to be adapted for some values of r to make sure that the only saturated arcs are the arcs $v_{i+1}v_i$. Namely, it is sufficient to choose a periodic representation where the period starts with a 1.

Our best construction relies on combining the main ideas of the two previous constructions:

- split the flow into 2^k equal chunks, and make three groups of size p, $q-p$, 2^k-q, so that we get three flows of size $\lambda p 2^{-k}$, $\lambda(q-p)2^{-k}$ and $1 - q2^{-k}$. One flow is output, one loops back to the head of the bottleneck arc, and one loops back to the tail of the bottleneck arc.
- use the binary representations of each group size to create the three flows.

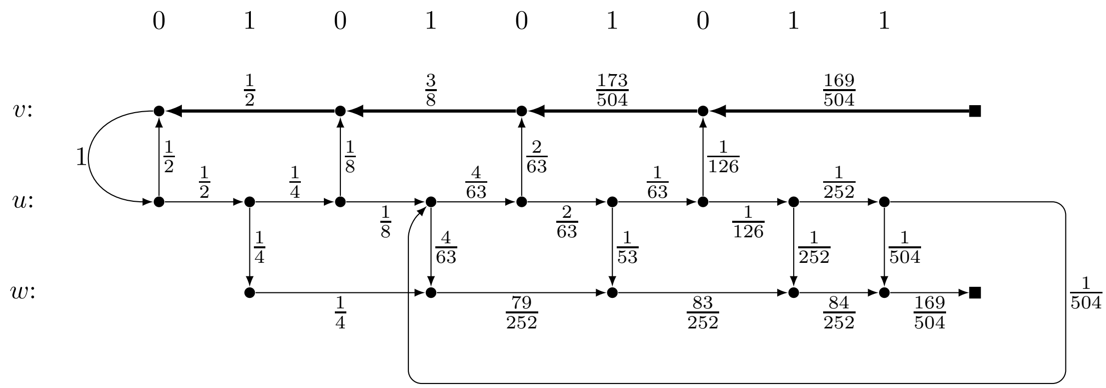

Figure 19: A network with maximal throughput $r = \frac{169}{504} = \frac{169}{8\cdot 63}$, built from the binary representation of r: $0.010(101011)^\omega$.

We first give the construction for $r = \frac{p}{q} \geq \frac{1}{2}$, and we will later show how to extend it to any rational value.

For an integer $n \in \mathbb{N}$, denote binary (n) the binary representation of n, and $\text{binary}_i(n)$ the value of the bit of weight 2^i in that representation.

Definition 6.1. Let $p_1, p_2, \ldots, p_l, k \in \mathbb{N}_{>0}$ be integers with $\sum_{i=1}^{l} p_i = 2^k$. Let $T_{p_1,p_2,\ldots,p_l,k}$ be a binary out-arborescence where each leaf is labeled by an integer in $[\![1,l]\!]$, such that for any label $i \in [\![1,l]\!]$ and any depth $d \in [\![1,k]\!]$, the number of leaves with label i at depth d is $\text{binary}_{k-d}(p_i)$.

Notice that $T_{p_1,p_2,\ldots,p_l,k}$ always exists, but is not necessarily unique. We prove its existence by showing that the number n_d of inner nodes at depth $d \in [\![0,k]\!]$ is

$$n_d = 2^d - \sum_{i=1}^{l} \left\lfloor \frac{p_i}{2^{k-d}} \right\rfloor.$$

The proof is by induction on d, where $n_0 = 1$. Let $d \in [\![1,k]\!]$, and assume that $n_{d-1} = 2^{d-1} - \sum_{i=1}^{l} \left\lfloor \frac{p_i}{2^{k-d+1}} \right\rfloor$. Then the number of nodes at depth d is twice n_{d-1}, to which we subtract the leaves at depth d. This yields

$$\begin{aligned}
n_d &= 2n^{d-1} - \sum_{i=1}^{l} \text{binary}_{k-d}(p_i) \\
&= 2 \cdot \left(2^{d-1} - \sum_{i=1}^{l} \left\lfloor \frac{p_i}{2^{n-d+1}} \right\rfloor\right) - \sum_{i=1}^{l} \text{binary}_{k-d}(p_i) \\
&= 2^d - \sum_{i=1}^{l} 2 \left\lfloor \frac{p_i}{2^{n-d+1}} \right\rfloor + \text{binary}_{k-d}(p_i) \\
&= 2^d - \sum_{i=1}^{l} \left\lfloor \frac{p_i}{2^{k-d}} \right\rfloor.
\end{aligned}$$

As $\sum_{i=1}^{l} p_i \leq 2^k$, n_d is non-negative, hence $T_{p_1,p_2,\ldots,p_l,k}$ exists.

Definition 6.2. Given $r = \frac{p}{q} \in \mathbb{Q}$, with $\frac{1}{2} \leq \frac{p}{q} < 1$, $\gcd(p,q) = 1$, and $k \in \mathbb{N}_{>0}$ such that $2^{k-1} < q \leq 2^k$. Let $T = T_{p,q-p,2^k-q,k}$ with root r. We define the splitter network $G_{p/q} := (V := I \uplus S \uplus O, E)$ where

- ▹ $I := \{i\}$, $O = \{o\}$ and $S = \{r'\} \uplus V(T)$,
- ▹ $E := E(T) \uplus E(P_1) \uplus E(P_2) \uplus E(P_3) \uplus \{ir', r'r, t_1 o, t_2 r', t_3 r\}$,

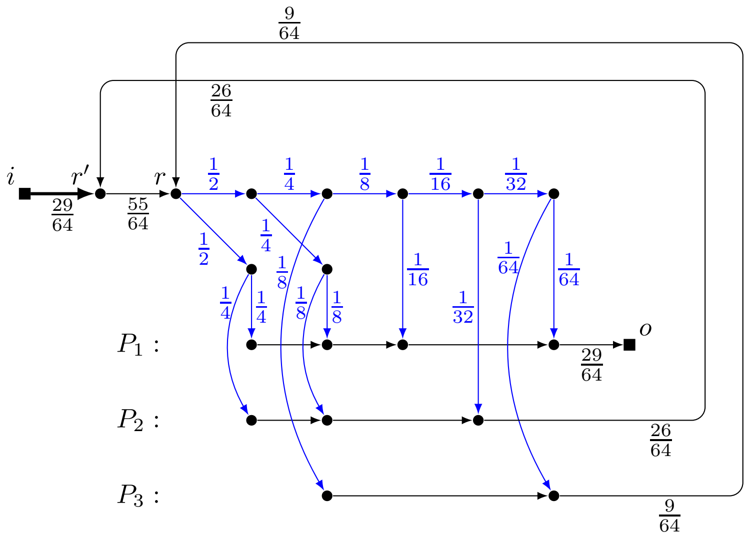

Figure 20: A network of size $O(\log q)$ with maximum throughput $\frac{29}{55}$. The throughput values shown on the figure should be scaled by a factor $\frac{64}{55}$ to get the value at equilibrium. The lines P_1, P_2, P_3 are given by the binary encoding of $p = 29 = 011101$, $q = 26 = 011010$ and $2^k - q = 9 = 001001$. Figure 21 (a) gives another representation of the same network.

- ▹ for each label $\gamma \in \{1, 2, 3\}$, P_γ is a path connecting all the leaves with label γ in increasing depth order, and t_γ is the end vertex of P_γ.

The construction is illustrated in Figure 20 and Figure 21 (a).

Lemma 6.3. *$G_{p/q}$ has a steady-state (t, F) with throughput $t^\star = \min\left\{c(i), c(o), \frac{p}{q}\right\}$. Moreover*
(i) if $t^\star = \frac{p}{q}$, we may take $F = E(G_{p/q}) \setminus \{ii'\}$;
(ii) if $t^\star = c(i)$, we may take $F = E(G_{p/q})$;
(iii) if $t^\star = c(o)$, we may take $\{ii', o'o\} \subseteq F$.

Proof. Let t be the following throughput function:

$$t(e) := \begin{cases} \frac{2^{k-d}}{q} & \text{if } e \in T \text{ is from depth } d-1 \text{ to depth } d, \\ \frac{2^{k+d}}{q}\left\lfloor \frac{v(\gamma)}{2^d} \right\rfloor & \text{if } e \in P_\gamma,\ \gamma \in \{1,2,3\}, \text{ is from depth } d \text{ to depth } d' > d, \\ \frac{p}{q} & \text{if } e \in \{ir', t_1 o\}, \\ 1 - \frac{p}{q} & \text{if } e = t_2 r' \\ \frac{2^k}{q} - 1 & \text{if } e = t_3 r \\ 1 & \text{if } e = r'r \end{cases}$$

where $v(1) = p, v(2) = q - p, v(3) = 2^k - q$. We check that $(t, E \setminus \{ir'\})$ is a steady-state when $c(i) = c(o) = 1$. As $2^{k-1} < q \leq 2^k$, the throughputs are between 0 and 1. The conservation rule R5 is easily checked on every vertex, except vertices of P_γ. Let $u \in V(P_\gamma)$ at depth d. let wu the incoming arc on P_γ, with w at depth d'. Then $\text{binary}_{d'}(v(\gamma)) = \text{binary}_d(v(\gamma)) = 1$, and $\text{binary}_\delta(v(\gamma)) = 0$ for each $\delta \in [\![d'+1, d-1]\!]$. This implies

$$2^{d'-d} \cdot 2^{d'} \left\lfloor \frac{v(\gamma)}{2^{d'}} \right\rfloor + 1 = \frac{1}{2^d} \left\lfloor \frac{v(\gamma)}{2^d} \right\rfloor.$$

Scaling this equation by a factor $\frac{2^k}{q}$ yields the conservation rule for u. Rule R6 at r' requires that $\frac{p}{q} \geq 1 - \frac{p}{q}$, which holds by the assumption that $\frac{p}{q} \geq \frac{1}{2}$. The other rules can be readily checked. This steady-state remains valid as long as $\min\{c(i), c(o)\} \geq \frac{p}{q}$.

Suppose now that $\min\{c(i), c(o)\} \leq \frac{p}{q}$. When $c(i) \geq c(o)$, we build a pre-steady-state by decreasing $t(t_1 o)$ to $c(o)$, generating some excess on t_1, and removing $t_1 o$ from the fluid arcs. Then using the pre-steady-state algorithm, the flow will be push back to i through ir', when ir' is saturated. Therefore there is a steady-state with total throughput $c(o)$ and $ir', t_1 o$ are saturated. When $c(i) < c(o)$, we start from $(t, E \setminus \{ir'\})$, take the reverse steady-state in the reverse network, then decrease its output capacity on output i to $c(i)$, saturating $r'i$ and inducing some excess at r'. Then we apply the pre-steady-state algorithm to find a steady-state. The first step of the algorithm will be to make rr' saturated, in order to push back the excess at r'. From there all arcs are saturated. Running the algorithm to its termination, we obtain a steady-state for the reverse network, that we reverse into a steady-state for $G_{p/q}$ with only fluid arcs. ☐

When $\frac{p}{q} < \frac{1}{2}$, we construct $G_{p/q}$ from $G_{2p/q}$. This adds at most $2 \log q$ additional splitters, thus the number of splitters of $G_{p/q}$ remains $O(\log q)$.

Definition 6.4. Let $\frac{p}{q} \in \mathbb{Q}_{>0}$ with $\frac{p}{q} < \frac{1}{2}$, we define the splitter network $G_{p/q} := (V := I \uplus S \uplus O, E)$ where

- $I = \{i\}$, $O = \{o\}$, $S = V(H)$,
- $E = \{ii', o'o, o'i'\} \uplus E(H)$,

where H is a copy of $G_{2p/q}$ with input i' and output o'.

We extend the proof of Lemma 6.3 to every rational $\frac{p}{q}$.

Proof. We prove it by induction on the number of recursive steps in the definition of $G_{p/q}$. The base case is given by the first proof of the lemma, when $\frac{p}{q} \geq \frac{1}{2}$. Let $\frac{p}{q} < \frac{1}{2}$, and suppose that $G_{2p/q}$ behaves as expected.

Case 1: $\frac{p}{q} \leq \min\{c(i), c(o)\}$. Let (t', F') be a steady-state on $G_{2p/q}$ when $c'(i') = 1 = c'(o')$, obtained by induction hypothesis. Then $\delta^+(i') \cap F' = \emptyset$. Let $t(ii') = t(o'o) = t(o'i') = \frac{p}{q}$ and $t(e) = t'(e)$ for any other arc e. Then, checking all the rules on i, o, i', and o', we get that $(t, F' \cup \{ii', oi'\}$ is a steady-state, as expected.

Case 2: $c(i) \leq c(o)$ and $c(i) < \frac{p}{q}$. Let (t', F') be a steady-state for $G_{2p/q}$ and $c'(i') = c'(o') = 2c(i) < \frac{1p}{q}$, obtained by induction hypothesis. Then $F' = E(G_{2p/q})$, all arcs are fluid. We extend this steady-state, by setting $t(ii') = t(o'o) = t(o'i') = c(i)$ and $t(e) = t'(e)$ for any other arcs. Then $(t, E(G_{2p/q}))$ is a steady-state, with only fluid arcs.

Case 3: $c(o) < \min\{c(i), \frac{p}{q}\}$. Let (t', F') be a steady-state on $G_{2p/q}$ when $c'(i') = c'(o') = 2c(o) \leq \frac{2p}{q}$, obtained by induction hypothesis. Then $\delta^-(o')$ and $\delta^+(i')$ are saturated in this steady-state. We extend it by setting $t(ii') = t(o'o) = t(o'i') = c(o)$, $t(e) = t'(e)$ for any other arc e, and $F = F'$. Then (t, F) is a steady-state for (G, c), with ii' and $o'o$ saturated, as expected. ☐

The next theorem summarizes the previous results.

Theorem 6.5. *For any rational $r = \frac{p}{q} \in \mathbb{Q}$ with $r \in [0, 1]$, there is a splitter network with maximum throughput $\frac{p}{q}$. This network may be used as a gadget to simulate an arc with capacity r. The size of this network is linear in the size of r when r is encoded either as a ratio or by its binary expansion (prefix and period).*

The network $G_{p/q}$ exhibits a curious asymmetry. It operates by splitting the input flow carefully until reaching the exact throughput $\frac{p}{q}$. Intuitively $G_{p/q}$ acts on the input. The reversed network $\overleftarrow{G_{p/q}}$ is also a capacity-simulating network, with the same maximum throughput. Intuitively $\overleftarrow{G_{p/q}}$ acts on the output, by carefully grabbing amount of flows until reaching throughput $\frac{p}{q}$. $G_{p/q}$ and $\overleftarrow{G_{p/q}}$ share the same general behaviour, with two distinct strategies. Figure 21 shows an example of $G_{p/q}$ together with its reverse networks and the throughputs at maximum capacity.

We cannot extend these constructions to build network with an irrational maximum throughput.

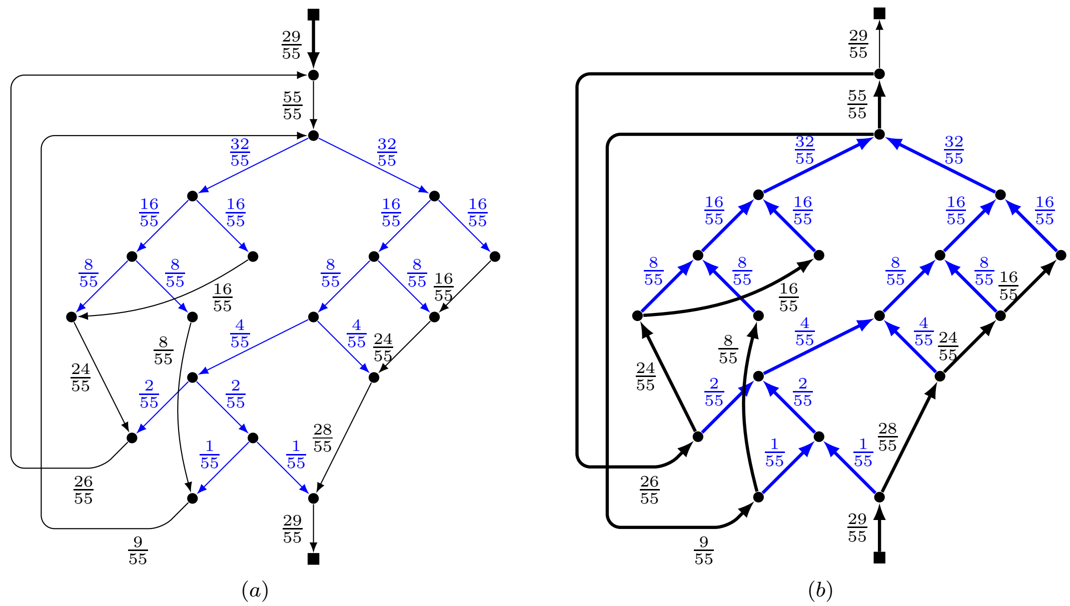

Figure 21: Two networks with maximum throughput $\frac{29}{55}$, reverse from each other.

Lemma 6.6. *For any irrational $x \in [0,1] \setminus \mathbb{Q}$, there is no splitter network with a single input i and a single output o, such that $c(i) = c(o) = 1$, the total throughput of a steady-state is x.*

Proof. The maximal throughput t of a network under some capacities is an optimal solution to the linear program (PSS) for some set F of fluid arcs, hence t must be a convex combination of rational vectors. By maximality of t, all those rational vectors have the same total throughput, which is a rational. $\square$

We conclude this section, by mentioning that the construction of a capacity-simulating network can be extended to design networks that distribute their incoming flow over their outputs following an arbitrary rational distribution. Considering a rational vector $(p_1, \ldots, p_l)$, and $q \in \mathbb{N}$, where l is the required number of outputs and $\frac{p_o}{q}$ is the expected throughput for output o. Choose k with $2^{k-1} < q \leq 2^k$. We replace the single edge bottleneck by a bottleneck cut of size $s = \left\lceil \frac{p_1 + \ldots + p_l}{q} \right\rceil$. Then we define $\frac{r_i}{q}$ for $i \in [\![1, s]\!]$, the amount of flow that loop back after the bottleneck, and $\frac{q'}{q}$ the amount of flow that loop back before the bottleneck. We start from a binary out-arborescence $T = T_{p_1,\ldots,p_l,q',r_1,\ldots,r_s}$. As the maximum throughput may be larger than 1, we replicate the highest nodes of T, until each layer contains at least s nodes. For each of the s replicated roots, we add to incoming arc, one is in the bottleneck cut, the other is a loopback arc with throughput $\frac{r_i}{q}$. Figure 22 illustrates an example of distribution network. From there, it is possible to glue a simple balancer over the inputs to make the network throughput unlimited, and then to add even more loopback arcs, to make a *half-universal distribution network*, and duplicate it to define ultimately a *universal distribution network*. In such a network, the throughputs on fluid outputs arcs stay proportional to each other, following the required distribution.

7 Structure of the steady-states of a splitter network

7.1 Uniqueness of steady-states

In this section, we investigate the structure of the set of sub-steady-states, and prove that, in a splitter network with input and output capacities, all the steady-states induces the same throughput of its inputs and outputs. Our strategy relies on restricting sub-steady-states to uniform sub-steady-states, where all inputs contribute with an equal throughput as much as possible. Then we study the two operations:

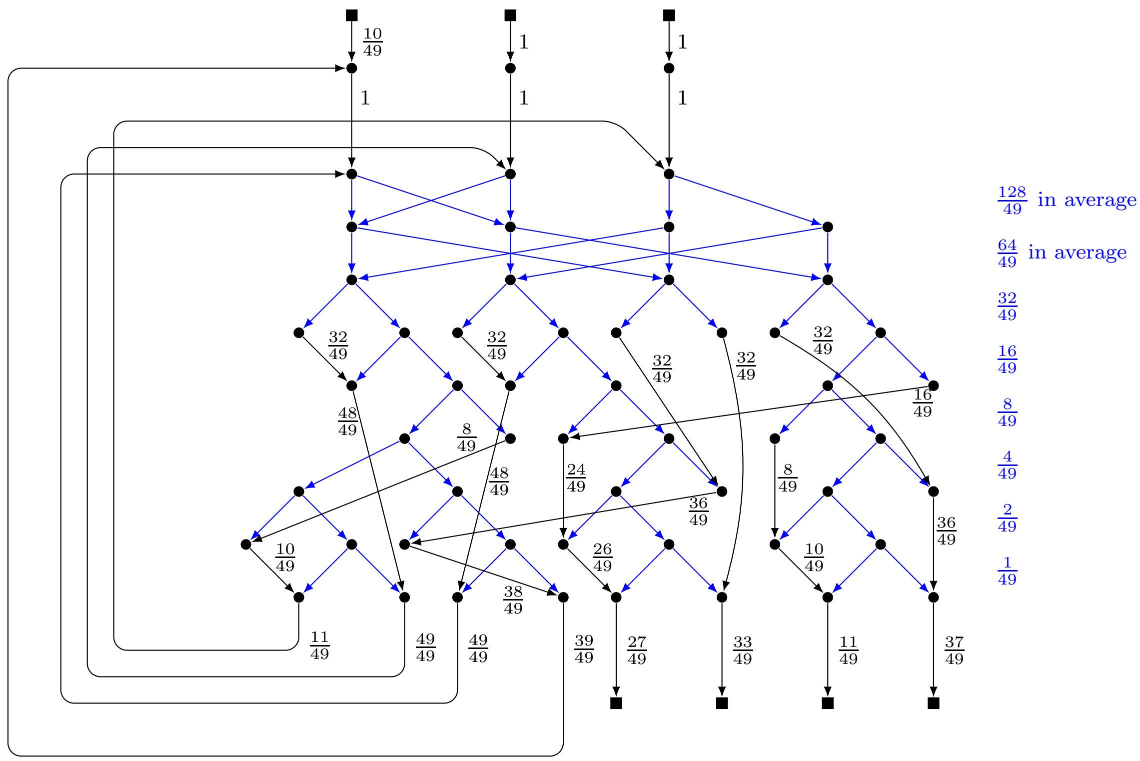

Figure 22: An splitter network defined from a rational vector $(\frac{27}{49}, \frac{33}{49}, \frac{11}{49}, \frac{37}{49})$, with $(r_1, r_2, r_3, r_4) = (49, 49, 11)$. The acyclic subgraph H, colored in blue, is obtained by replicating the highest layers of a tree. All blue arcs (in H) between two consecutive levels have equal throughput indicated on the right, except for the first two levels that appeared by replicating the top of the tree and form a simple balancer.

making some arc saturated, and improving the throughput along a circulation. We show that they are confluent, therefore for any order of operations processed by the sub-steady-state algorithm, we obtain the same final throughput. The proof will be concluded by showing that any steady-state is reachable by the sub-steady-state algorithm.

We will impose that all the inputs with a fluid outgoing arc must be augmented at the same rate. This reduces the non-determinism in the sub-steady-state algorithm and will make the proof of confluence easier. To this end, we introduce the following rule:

R9 there exists $\gamma \geq 0$ such that for any input $i \in I$ with $\delta^+(i) = \{e\}$, if $e \in F$, then $t(e) = \max\{c(i), \gamma\}$.

A sub-steady-state that satisfies rule R9 will be called *uniform sub-steady-state*. To simplify the proof further, we will assume in this section that the capacities of inputs and outputs are always 1. There is no loss of generality in this assumption: if an arbitrary network (G, c) has several steady-states, then its derived network $(G', \mathbb{1})$, obtained by replacing each input and output by a rate-limiting network simulating its capacity will also have several steady-states. Therefore, in this context of all-one capacities, rule R9 becomes: there exists $\gamma \geq 0$ such that for each $e \in \delta^+(I) \cap F$, $t(e) = \gamma$.

Because of the restriction to uniform sub-steady-state, we make two modifications to the residual graph of a uniform sub-steady-state. First, $\delta^+(I) \cap F$ is a single class in $\mathcal{C}^=$. Second, we remove from $\delta^+(z)$ the saturated arcs $\delta^-(O) \setminus F$. Therefore, $\delta^+(z)$ corresponds to $\delta^+(I)$, and consequently any $\mathcal{C}^=$-circulation will be a stationary circulation. In particular it will be uniform on $\delta^+(I) \cap F$, and be able to augment the current uniform sub-steady-state to another uniform sub-steady-state in Lemma 3.10. The removal of $\delta^-(O) \setminus F$ has no consequence as the throughput on these arcs cannot change anymore.

The sub-steady-state algorithm computes a steady state by a sequence of three *elementary operations*: increasing along a stationary circulation of the residual graph whose support intersects $\delta(z)$ (an *augmentation*), removing an arc from F (a *saturation*), or increasing along a stationary circulation of the residual graph whose support does not intersect $\delta(z)$ (a *move*). By increasing along a circulation, we mean augmenting the throughput value of fluid arcs and decreasing those of saturated arcs. The amplitude of the change is always chosen maximum subject to the constraint of obtaining a uniform sub-steady-state.

We recall some basic facts that will be necessary in the following proofs. First, It can be readily checked that all saturations done in the proof of the sub-steady-state algorithm are of one of the three following cases.

Claim 7.1. *During the uniform sub-steady-state algorithm, an arc $e = uv$ becomes saturated when one of these conditions occur:*

(i) *$v \in O$ and $t(e) = c(e)$. Because we modified the residual graph in this section, when we remove e from F, we also remove it from the residual graph instead of reversing it;*
(ii) *e is non-loose: it is in-coupled to an arc $e' \notin F$, and $t(e) = t(e')$;*
(iii) *v is a sink in the residual graph: arcs in $\delta^+(v)$ either are saturated or have throughput 1, and the arc $e' \in \delta^-(v) \setminus \{e\}$ is fluid with $t(e) \geq t(e')$. e' may also be saturated with $t(e) = t(e') = 0$, in which case e is also non-loose.*

We also need to understand when an augmentation or a move happens. Both come from $\mathcal{C}^=$-circulations of the residual graph. The support of $\mathcal{C}^=$-circulations must be strongly connected components (by virtue of being a circulation) of the residual graphs, with no leaving arcs (because the $\mathcal{C}^=$-classes correspond to the out-incidencies of the vertices). Therefore, each such $\mathcal{C}^=$-circulation is (up to a multiplicative factor) a stationary circulation over a sink strongly connected component of the residual graph. This implies:

Claim 7.2. *The supports of $\mathcal{C}^=$-circulations that can be used in a* move *or* augment *operation are edge-disjoint and vertex-disjoint. Moreover, we can increase along a $\mathcal{C}^=$-circulation if and only if its support contains only loose arcs. Therefore, not all $\mathcal{C}^=$-circulation can be used.*

These two claims allow us to prove:

Lemma 7.3 (Local confluence). *Let (t, F), (t_1, F_1) and (t_2, F_2) be three uniform sub-steady-state for a splitter network $(G, \mathbb{1})$, such that (t_1, F_1) and (t_2, F_2) are each derived from (t, F) by a single elementary operation, respectively o_1 and o_2. Then o_2 can be applied to (t_1, F_1) and o_1 can be applied to (t_2, F_2), resulting in the same uniform sub-steady-state (t', F').*

Proof. The proof is by a case analysis, each case corresponding to the choice of which operations where used to derive (t_1, F_1) and (t_2, F_2). Notice that from (t, F), there can be at most one possible augmentation operation, using the stationary circulation from z in the residual graph. Let Z, Z_1, Z_2 be the strongly connected sink components of the residual graphs of (t, F), (t_1, F_1) and (t_2, F_2) containing z, respectively.

Case 1: Saturation of e_1 and saturation of e_2. By Claim 7.1, the conditions that allow the saturation of an arc are local to the head of that arc, hence the saturations of the two arcs are independant as long as the two arcs have no common extremity. Specifically, there are two subcases: $e_1 = uv$ and $e_2 = vw$, or $e_1 = uv$ and $e_2 = wv$.

Consider the former case. Using again the locality of the necessary conditions to saturate e_2, e_2 can be saturated after e_1, leading from $(t, F \setminus \{e_1\})$ to $(t, F \setminus \{e_1, e_2\})$. If e_1 is non-loose (respectively a sink) in (t, F), it is still non-loose (respectively a sink) in (t, F_2), and hence can be removed from F_2 to get $(t, F \setminus \{e_1, e_2\})$.

Suppose now that $e_1 = uv$ and $e_2 = wv$. As the saturation of each arc e_1, e_2, is possible, they are both fluid and loose, v is a sink and $t(e_1) = t(e_2)$. Therefore, e_2 is non-loose in (t, F_1) and therefore can be saturated to get $(t, F \setminus \{e_1, e_2\})$. Symmetrically, e_1 can be saturated in (t, F_2).

Case 2: Move over h_1 and move over h_2. As the supports of h_1 and h_2 are vertex-disjoint by Claim 7.2, the residual graph does not change from (t, F) to (t_1, F) on the support of h_2 and all its incident arcs. Therefore the support of h_2 is still a strongly connected sink component in the residual graph of (t_1, F), with only loose arcs. Hence a move operation with h_2 is possible, and its amplitude is also not changed as the throughput value have also not changed on the support of h_2 and its incident arcs. Similarly, we can apply a move with h_1 from (t_2, F). In both case, we finally obtain $(t_1 \vee t_2, F)$.

Case 3: Augmentation over g and saturation of e. Let (t_1, F) be the uniform sub-steady-state obtained by augmenting over g. We consider the possible cases for e from Claim 7.1

Suppose that e is a non-loose arc from u to an output o. If $u \in V(Z)$, then e is a non-loose arc in the support of g, and augmenting over g would not be possible. Therefore, $u \notin Z$, and the saturation of e is still possible from (t_1, F). Moreover, $Z_2 = Z$, as only e is removed from the residual graph when it is saturated. Thus augmenting over g from $(t, F \setminus e)$ also gives $(t_1, F \setminus \{e\})$.

Suppose that e is a non-loose arc from u to $v \in S$, with in-coupled saturated arc $e' = wv$. Here again u is not in Z (otherwise e is in the support of g), and $v \notin Z$ (otherwise $\overleftarrow{e}$ is in the support of g but is non-loose). Thus we may saturate e from (t_1, F) to obtain $(t_1, F \setminus \{e\})$. Moreover saturating e reverses the orientation of e, and both extremities of e being outside Z, this implies that $Z_2 = Z$, therefore we can augment over g from $(t, F \setminus \{e\})$ to obtain $(t_1, F \setminus \{e\})$.

The last case occurs when $e = uv$ is loose but v is a sink in the residual graph. Hence $u, v \notin Z$, because Z is a sink component. This implies again that $Z = Z_2$, and that the saturation and augmentation can be done in any order.

Case 4: Augmentation over g and move over h. The support of g and h are two distinct strongly connected sink component of the residual graph by Claim 7.2. Hence the two operations are independant and commute.

Case 5: Move over h and saturation of e. The proof is similar to the case augmention and saturation. If e is a non-loose arc, it must be disjoint for the support of h, making the two operations independant. If

e is a loose arc, then v is a sink. As the support of h is a non-trivial sink component, e and the support are vertex-disjoint. Therefore the two operations are independant and commute. $\square$

By Newman's lemma on locally confluent binary relations, we get:

Corollary 7.4. *Let (t_1, F_1) and (t_2, F_2) be two uniform sub-steady-states obtained by two sequences of operations from a common uniform sub-steady-state (t, F). Then there are two sequences of operations, one from (t_1, F_1), the other from (t_2, F_2), ending at a common uniform sub-steady-state (t', F').*

It remains to prove that any uniform-sub-steady-state is obtainable from $(0, E)$.

Lemma 7.5. *For any uniform sub-steady-state (t, F) distinct from $(0, E)$, either there is $e \in E \setminus F$ such that $(t, F \cup \{e\})$ is a uniform sub-steady-state, or there exists a uniform sub-steady-state (t', F) and a non-zero circulation of the residual graph for (t', F), such that (t, F) is obtained by a move or augment operation with that circulation from (t', F).*

Proof. First we may assume that for any arc e with $t(e) = 1$, e is fluid. Otherwise, $(t, F \cup \{e\})$ is a uniform sub-steady-state, as required. Consider the graph H with arcs $\{e \in F : t(e) > 0\} \cup (E \setminus F)$. If H contains a non-zero $\mathcal{C}^=$-circulation, where $\mathcal{C}^=$ are the out-incidencies of the vertices, then we can decrease t along that circulation, decreasing t on fluid arcs and increasing t on backward saturated arcs. This yields a throughput t' with (t', F) a uniform sub-steady-state, whose residual graph admits the same $\mathcal{C}^=$-circulation (because none of the arcs whose throughput differs between t and t' is tight). Hence (t, f) is obtained from (t', F) by a single *augment* or *move* operation.

Therefore we can assume that H as no non-zero $\mathcal{C}^=$-circulation, which implies that it contains a sink vertex u, and because t is non-zero, we may assume that $t(\delta^-(u)) \neq 0$. Because u is a sink, and by the assumption that saturated arcs have throughput less than 1, its incoming arcs must be fluid. Up to reindexing, we assume that $t(e_1) \geq t(e_2)$. Then $t(e_1) \geq \frac{1}{2}t(\delta^+(u)) = \frac{1}{2}t(\delta^-(u)) > 0$, therefore e_1 must be saturated. Then $(t, F \cup \{e_1\})$ is a uniform sub-steady-state: rule R7 holds because $t(e_1) \geq t(e_2)$, and rule R8S holds because $\delta^-(u) \in F$. $\square$

Lemma 7.6. *If (t, F) is a steady-state, no* augment *operation can be applied. Any other operations result into another steady-state with the same throughput on $\delta^+(I) \cup \delta^-(O)$.*

Proof. Because (t, F) is a steady-state, rule R3 holds. Therefore for any input $i \in I$ with outgoing belt e, either $t(e) = c(i) = 1$ (recall that we assumed $c = \mathbb{1}$), or $e \notin F$. Thus z is a sink in the residual graph, proving that no *augment* operation can happen.

Also the throughput on $\delta^+(I) \cup \delta^-(O)$ cannot be changed by a *move* operation, therefore rule R3 will still hold after any valid operation. $\square$

Theorem 7.7. *For any splitter network $G = (I \uplus S \uplus O, E)$ and any capacity function $c : I \cup O \to [0, 1]$, all steady-states have the same restriction on $\delta^+(I) \cup \delta^-(O)$.*

Proof. First assume $c = \mathbb{1}$. Then any steady-state is reachable from $(0, E)$ by Lemma 7.5, and can be modified into a unique steady-state $(t^\uparrow_F, F)$ by Corollary 7.4 (this notation will be justified later). By Lemma 7.6, t and $t^\uparrow_F$ coincides on $\delta^+(I) \cup \delta^-(O)$.

When c is not uniformly 1, we replace each arc in $\delta^+(I) \cup \delta^-(O)$ by a rate-limiting network whose rate is the capacity of the incident input or output. Then there is an obvious morphism between the steady-state of the original splitter network and this extended splitter networks, where the restriction of the throughputs to $\delta^+(I) \cup \delta^-(O)$ coincides. Therefore the result extends to splitter networks with arbitrary capacities. $\square$

7.2 The lattice of uniform sub-steady-states

Another consequence of the local confluence Lemma 7.3 is that the sequences of admissible operations starting from $(0, E)$ are an antimatroid (see Lemma 1.2 in [14]). An *antimatroid* is a family $\mathcal{A}$ of finite sequences of symbols, satisfying the following properties

- (normal) each sequence in $\mathcal{A}$ contains each symbol at most once;
- (hereditary) each prefix of a sequence in $\mathcal{A}$ is in $\mathcal{A}$ ($\mathcal{A}$ is closed by taking prefixes);
- (anti-exchange) for each two sequences distinct R and S of $\mathcal{A}$, such that R contains a symbol not in S, there is a symbol x in R such that the sequence S, x is in $\mathcal{A}$.

By repeatedly applying the anti-exchange property, we can make a sequence R' from all the elements of R there are not in S, such that $S, R' \in \mathcal{A}$. For uniform sub-steady-states, the symbols are the *move*, *augment* and *saturate* operations, and a sequence of operation can be associated to the uniform sub-steady-state obtained by applying these operations from $(0, E)$.

It is known that antimatroids are semimodular lattices, therefore the set of uniform sub-steady-states is a semimodular lattice, and admits meet and join operations. In this section, we define these two operations explicitely, focusing at first on the restriction to a fixed set F of fluid arcs. We consider a splitter network $G = (I \uplus S \uplus O, E)$, with a capacity function on $I \cup O$ and t_1, t_2 be two throughput functions such that (t_1, F) and (t_2, F) are uniform sub-steady-states over the same set F of fluid arcs.

Consider the graph $H = (z \cup S, E^+ \cup E^-)$, where z is the identification of all inputs and outputs, and

$$E^+ = \{e \in F : t_2(e) > t_1(e)\} \cup \{\overleftarrow{e} : e \in E \setminus F, t_2(e) < t_1(e)\},$$
$$E^- = \{\overleftarrow{e} : e \in F, t_2(e) < t_1(e)\} \cup \{e \in E \setminus F : t_2(e) > t_1(e),\}$$

(some arcs are reversed, but for simplicity we will abusively identify e and $\overleftarrow{e}$). Let $E^= = \{e \in E : t_1(e) = t_2(e)\}$, such that $E = E^= \uplus E^+ \uplus E^-$.

Lemma 7.8. *$|t_2 - t_1|$ is a circulation in H. Therefore the connected components of H are strongly connected.*

Proof. Indeed, it follows from the conservation rule R5 applied to t_1 and t_2 at u that

$$(t_2 - t_1)(\delta^+_G(u)) = (t_2 - t_1)(\delta^-_G(u)),$$

which, by the choice of reversing arcs e with $t_2(e) < t_1(e)$, is equivalent to

$$|t_2 - t_1|(\delta^+_H(u)) = |t_2 - t_1|(\delta^-_H(u)).$$

□

Lemma 7.9. *For each connected component C of H, either $E(C) \subseteq E^+$ or $E(C) \subseteq E^-$.*

Proof. Let $u \in V(H)$ and $e_1 \in \delta^-_H(u)$ and $e_2 \in \delta^+_H(u)$. We prove that if $e_1 \in E^-$ then $e_2 \in E^-$. Because we consider uniform sub-steady-states, $u \neq z$. For the sake of contradiction, assume that $e_1 \in E^-$ and $e_2 \in E^+$. We consider four cases:

Case 1: $e_1, e_2 \in F$. Then $\delta^+_G(u) = \{\overleftarrow{e_3}, e_4\}$, and applying rule R7 we get a contradiction:

$$t_1(e_4) > t_2(e_4) = t_2(\overleftarrow{e_3}) > t_1(\overleftarrow{e_3}) = t_1(e_4).$$

Case 2: $e_3, e_4 \in E \setminus F$. Then $\delta^-_G(u) = \{\overleftarrow{e_3}, e_4\}$, and applying R6 we get a contradiction:

$$t_1(e_4) < t_2(e_4) = t_2(\overleftarrow{e_3}) < t_1(\overleftarrow{e_3}) = t_1(e_4).$$

Case 3: $e_3 \in F$, $e_4 \in E \setminus F$. Then $e_4 \in \delta_G^-(u)$ and $\overleftarrow{e_3} \in \delta_G^+(u)$. But $t_1(e_4) < t_2(e_4) \leq 1$ and $t_1(\overleftarrow{e_3}) < t_2(\overleftarrow{e_3}) \leq 1$, therefore t_1 contradicts rule R8S.

Case 4: $e_3 \in E \setminus F$, $e_4 \in F$. Then $\overleftarrow{e_3} \in \delta_G^-(u)$ and $e_4 \in \delta_G^+(u)$. Then $t_2(\overleftarrow{e_3}) < t_1(\overleftarrow{e_3}) \leq 1$ and $t_2(e_4) < t_2(e_3) \leq 1$, therefore t_2 contradicts rule R8S.

Next we prove that if $e_1 \in E^+$ then $e_2 \in E^+$. Otherwise, $e_1 \in E^+$ and $e_2 \in E^-$ appears consecutively in a cycle of H, as the connected components of H are strongly connected by Lemma 7.8. This cycle must contain a pair of consecutive arcs $e_3 \in E^-$, $e_4 \in E^+$, which cannot happen as we have already proved.

As each connected component of H is strongly connected, this implies that each component of H is either in E^+ or in E^-. ☐

We define two new throughput functions from t_1 and t_2.

Definition 7.10. Let (t_1, F), (t_2, F) be two uniform sub-steady-states over a capacitated splitter network (G, c). Let $t_1 \wedge t_2$ and $t_1 \vee t_2$ be the throughput functions defined by

$$(t_1 \wedge t_2)(e) = \begin{cases} \min\{t_1(e), t_2(e)\} & \text{if } e \in F; \\ \max\{t_1(e), t_2(e)\} & \text{if } e \notin F. \end{cases}$$

and

$$(t_1 \vee t_2)(e) = \begin{cases} \max\{t_1(e), t_2(e)\} & \text{if } e \in F; \\ \min\{t_1(e), t_2(e)\} & \text{if } e \notin F. \end{cases}.$$

Proposition 7.11. *$(t_1 \wedge t_2, F)$ and $(t_1 \vee t_2, F)$ are uniform sub-steady-states.*

Proof. For each vertex $u \in V(H)$, by Lemma 7.9, all arcs incident to u are in either $E^- \cup E^=$ or $E^+ \cup E^0$. In the former case, the throughput of its incident arcs in $t_1 \wedge t_2$ are equals to those in t_2, in the latter case to those in t_1. Therefore, all the rules defining a uniform sub-steady-state hold for $t_1 \wedge t_2$. The same reasoning applies for $t_1 \vee t_2$, because $t_1 \vee t_2$ agrees with t_1 on vertices incident to E^-, and with t_2 on vertices incident to E^+. ☐

Consequently, for any set $F \subset E$ for which a uniform sub-steady-state exists, there are two special uniform sub-steady-states $(t_F^\downarrow, F)$ and $(t_F^\uparrow, F)$, defined by $t_F^\downarrow = \bigwedge\{t : (t, F) \text{ uniform sub-steady-state}\}$ and $t_F^\uparrow = \bigvee\{t : (t, F) \text{ uniform sub-steady-state}\}$. When all the input capacities are equal, these corresponds to the two extremal solutions (minimum and maximum) corresponding to optimizing the linear form $\sum_{e \in F} t(e) - \sum_{e \in E \setminus F} t(e)$ over the polyhedron of uniform sub-steady-states. Observe that with non-uniform input capacities, the set of uniform sub-steady-states, for a given F, is not necessarily a polyhedron. When F is not fixed, let $c_1 < c_2 < \ldots < c_l$ be the values of input capacities, the uniform sub-steady-states can be described as the union of several polyhedra, each defined by adding a variable γ whose value is the common throughput to all non-limited inputs, and adding constraints $c_k \leq \gamma \leq c_{k+1}$, and for each fluid arc e leaving an input $i \in I$, either $t(e) = c(i)$ (if $c(i) < \gamma$), or $t(e) = \gamma$ (if $c(i) \geq \gamma$). Then $t_F^\uparrow$ and $t_F^\downarrow$ are the extremal points for the same linear form, in this union of polyhedra. Therefore, $t_F^\uparrow$ and $t_F^\downarrow$ can also be computed using linear programming.

We extend the definition of $\vee$ and $\wedge$ to arbitrary uniform sub-steady-states. Let (t_1, F_1) and (t_2, F_2) be two uniform sub-steady states. We define $(t_1, F_1) \wedge (t_2, F_2)$ as $(t_0, F_1 \cup F_2)$ where for each $e \in E$,

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

and we define $(t_1, F_1) \vee (t_2, F_2)$ as $(t', F_1 \cap F_2)$ where for each $e \in E$,

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

These two operations coincides with the previous definitions when $F_1 = F_2$.

By the anti-exchange property of antimatroids, if R and S are two sequences of operations from $(0, E)$ (or any uniform sub-steady-state (t_0, F_0), by removing a common prefix to both sequences), and r is an operation in R that does not appear in S, then r appears in a sequence S, R', therefore appears after any element in S. We use this property several times in the proof of the next theorem, and say that an operation τ is *in the future* of a sequence R if there exists S containing τ such that R, S is a valid sequence.

Theorem 7.12. *The operations $\wedge$ and $\vee$ are the meet and join operations of the semimodular lattice of uniform sub-steady states.*

Proof. Let (t_0, F_0) be the meet of (t_1, F_1) and (t_2, F_2), and let's prove that it satisfies the formula given above. There are two sequences of operations R and S, such that (t_1, F_1) arises from (t_0, F_0) by the sequence R and (t_2, F_2) arises from (t_0, F_0) by the sequence S.

We claim that R and S contain distinct operations. Suppose, for the sake of contradiction, that there is a common operation ρ. We may assume that ρ appears as soon as possible in both $R = R', \rho, R''$ and $S = S', \rho, S''$, and no common operation appears earlier in both sequences. Hence ρ commutes with neither $\tau_1 := \text{last}(R')$ nor $\tau_2 := \text{last}(S')$ (and R' and S' cannot be empty). If ρ is a *move* or *augment* operation, then as τ_1 does not commute with ρ, there must be an arc uv, with u in the support of ρ, such that

- either τ_1 is the saturation of uv. As τ_1 is not in S', uv is not saturated after applying S', therefore uv is not in the support of ρ. Thus to apply ρ after S', we must have $t(uv) = c(uv)$. As an *augment* or *move* operation whose support contain uv cannot be in the future of R', and by our choice of the earliest common operation ρ, no such operation occurs in S'. This implies that $t(uv)$ is not increased by S', and $t_0(uv) = c(uv)$. Hence τ_1 and ρ commutes in R, contradiction;

- or τ_1 is an *augment* or *move* that increases the throughput value of uv to $c(uv)$. As τ_1 is in the future of S', the throughput value of uv after S' is less than $c(uv)$, and ρ is not applicable, contradiction.

Else ρ is the saturation of some arc uv. Then τ_1 is

- either an *augment* or a *move* whose support contains uv. But then τ_1 cannot be in the future of S', ρ (where uv is saturated), therefore τ_1 must be in S', contradicting the choice of the earliest common operation ρ;r

- or the saturation of an arc vw. Then after R', $t(vw) < 1$ as otherwise τ_1 and ρ commutes. But then after S', vw is not saturated as τ_1 is in the future of S', thus by Rule R8 $t(vw) = 1$, implying that $S' \setminus R'$ contains an *augment* of *move* operation whose support contains vw. But such an operation cannot be in the future of R', as vw is saturated after R', contradiction;

- or an *augment* or *move* operation whose support contains an arc vw, such that after R', $t(vw) = 1$. As τ_1 is in the future of S', after S' vw is neither saturated nor at throughput 1, therefore ρ is not applicable, contradiction.

Therefore, R and S contain distinct operations.

Next we claim that for any arc e, the first operations ρ in R, τ in S, that increase the throughput of e, must be equal. To prove this, we decompose $R = R', \rho, R''$ and $S = S', \tau, S''$. By the anti-exchange property, there is a permutation $\tilde{S}'$ of S' such that $R', \tilde{S}'$ is applicable, and moreover both ρ and τ may be applied immediately after $R', \tilde{S}'$. Therefore as the support of ρ and τ intersects at e, they must be equal. Because R and S have distinct operations, this implies that only one of R and S can modify the throughput of any given arc.

It follows from the fact that R and S have distinct operations, hence distinct saturations, that $F_0 = F_1 \cup F_2$. Now consider some operation ρ in R that increase the throughput of an arc e. Then ρ is in the future of S, therefore $e \in F_2$. By contraposition, if $e \notin F_2$, the throughput on e does not increase when applying R, hence $t_1(e) = t_0(e)$, proving that $t_0(e) = t_1(e)$ when $e \in F_1 \setminus F_2$ (and its symmetric case). If $e \in F_1 \cap F_2$, e does not increase in either R or S, hence $t_0(e) = \min\{t_1(e), t_2(e)\}$. The case $e \notin F_1 \cup F_2$ is symmetric, either R or S does not decrease e, therefore $t_0(e) = \max\{t_1(e), t_2(e)\}$, proving the formula for $\wedge$.

Finally, $t_1 \vee t_2$ is obtained by applying the sequences $R, \tilde{S}$ or $S, \tilde{R}$, for some permutations $\tilde{S}$ and $\tilde{R}$ of S and R respectively. The effects of $\tilde{R}$ (respectively $\tilde{S}$) on a uniform sub-steady state are the same as applying R (respectively S). The formula for $\vee$ immediately follows. ☐

8 Priority splitters

8.1 Definition and algorithms

Factorio allows players to assign priorities to splitters, altering their behavior. A splitter that prioritizes some output belt will send as much throughput to that belt, until saturating or reaching its capacity. Any excess flow will then be sent on the other output belt. Similarly, a splitter will pull its flow from the prioritized input belt, and use the other output belt only to complete its throughput.

We modify the definition of steady-states to accomodate priority splitters.

Definition 8.1. Let $G = (I \uplus S \uplus O, E)$ be a splitter network, $c : I \cup O \to [0, 1]$ a capacity function, $p^+, p^- : S \to E \cup \{\bot\}$ the so-called the output-priority and input-priority functions, such that for each $s \in S$, $p^+(s) \in \delta^+(s) \cup \{\bot\}$ and $p^-(s) \in \delta^-(s) \cup \{\bot\}$. A *steady-state* for (G, c, p^+, p^-) is a pair (t, F) where

- P1 $t : E \to [0, 1]$ is the throughput function;
- P2 $F \subseteq E$ is the set of *fluid arcs*, $E \setminus F$ is the set of *saturated* arcs;
- P3 for each $i \in I$ with $\delta^+(i) = \{e\}$, $t(e) \leq c(i)$ and moreover if $e \in F$ then $t(e) = c(i)$;
- P4 for each $o \in O$ with $\delta^-(o) = \{e\}$, $t(e) \leq c(o)$ and moreover if $e \notin F$ then $t(e) = c(o)$;
- P5 for each $s \in S$, with $\delta^-(s) = \{e_1, e_2\}$ and $\delta^+(s) = \{e_3, e_4\}$, $t(e_1) + t(e_2) = t(e_3) + t(e_4)$;
- P6 for any $e_1, e_2 \in E$ with $\{e_1, e_2\} = \delta^-(s)$ and $e_1 \notin F$, if $p^-(s) = e_1$ then $t(e_1) = 1$ or $t(e_2) = 0$, and if $p^-(s) \neq e_2$ then $t(e_1) \geq t(e_2)$;
- P7 for any $e_1, e_2 \in E$ with $\{e_1, e_2\} = \delta^+(s)$ and $e_1 \in F$, if $p^+(s) = e_1$ then $t(e_2) = 0$ or $t(e_1) = 1$, and if $p^+(s) \neq e_2$ then $t(e_1) \geq t(e_2)$;
- P8 for any $uv \in E \setminus F$ and $vw \in F$, $t(uv) = 1$ or $t(vw) = 1$.

Compared to the definition of steady-states for (G, c), all rules are identical except rules P6 and P7 which generalizes rules R6 and R7. Therefore when p^- and p^+ are uniformly $\bot$, the two notions of steady-states coincide.

The extended definition of steady-state preserves the symmetry already observed, therefore if (t, F) is a steady-state for (G, c, p^+, p^-), then $(t, \overleftarrow{E \setminus F})$ is a steady-state for $(\overleftarrow{G}, c, \overleftarrow{p}^-, \overleftarrow{p}^+)$, where $\overleftarrow{p}(e) = \overleftarrow{(}p(e))$.

In the rule P7, for $\delta^+(s) = \{e_1, e_2\}$, when $e_1 \in F$ and $p^+(s) = e_1$, the constraint $t(e_1) = 1$ or $t(e_2) = 0$ is not immediately representable as a linear constraint. However, in the context of iteratively solving the linear system, and removing arcs from F at each step, at the start of an iteration, if $e_1 \in F$ and $t(e_1) < 1$, we may force the constraint $t(e_2) = 0$. Hence $t(e_2)$ will stay null until e_1 becomes tight or saturated. Then we can remove the constraint $t(e_2) = 0$ as it is no longer induced by the definition of steady-state. The same principle applies to rule P6. Therefore, the LP-based algorithms can be adapted to compute the steady-state of a splitter network with priorities. Likewise, the definition of residual graph can be adapted, to accomodate priorities.

Priorities allow to change the feasible steady-states. Therefore, the question of finding a steady-state optimizing the total throughput becomes relevant in the context of splitter network with priorities. Formally, the input of a throughput maximization problem consists in a splitter network (G, c). Its output is a quadruple (p^+, p^-, t, F) such that (t, F) is a steady-state for (G, c, p^+, p^-). The goal is to maximize the global throughput $t(\delta^+(I))$.

We study several possible scenarii for this optimization problem: in Section 8.2 when we may choose all the priorities, in and out of each splitter; in Section 8.3 we impose the in priorities and ask for maximizing out priorities; and finally in Section 8.4 we impose some priorities arbitrarily, and must choose the other priorities.

8.2 Choose input and output priorities for each splitter

Proposition 8.2. *The throughput maximization problem is polynomial-time solvable when $c = \mathbb{1}$. Let $G = (I \uplus S \uplus O, E)$ be a splitter network. The maximum throughput equals the value of a maximum flow from I to O in G with unit capacities.*

Proof. Consider an integral maximum flow t from I to O. Because there is a minimum cut having the same value, the maximum throughput cannot exceed $t(\delta^+(I))$. We show that we can choose F, p^+ and p^- such that (t, F) is a steady-state in (G, c, p^+, p^-).

For each splitter $s \in S$, with $\delta^+(s) = \{e_1, e_2\}$, we define

$$p^+(s) = \begin{cases} \perp & \text{if } t(e_1) = t(e_2); \\ e_1 & \text{if } t(e_1) = 0 \text{ and } t(e_2) = 0; \\ e_2 & \text{if } t(e_2) = 1 \text{ and } t(e_1) = 1. \end{cases}$$

Define $p^-(s)$ similarly. Finally, we define F by setting saturated arcs to be precisely the arcs e with $t(e) = 0$ that are reachable from I in the residual graph of the maximal flow t. This choice guarantees that rule P8 is satisfied. All the other rules are readily checked. □

8.3 Choose output priority for each splitter

Proposition 8.3. *Given $(G, c = \mathbb{1})$ and input priorities $p^- : S \to E$, the problem of finding a steady-state (t, F, p^+, p^-) with maximum throughput is polynomial-time solvable. The maximal throughput equals the value of a maximum flow from I to O in G with unit capacities.*

Proof. For an integral maximum flow t from I to O, and setting $p^+(s)$ as in the proof of Proposition 8.2, (t, E) is a sub-steady-state, as can be readily checked. We initiate the sub-steady-state algorithm with (t, E), yielding a steady-state (t', F). Because the sub-steady-state cannot reduce the global throughput, and by the existence of a minimum cut of size $t(\delta^+(I))$, the throughput of (t', F) is precisely the value of the maximum flow. □

Corollary 8.4. *Given $(G, c = \mathbb{1})$ and output priorities $p^+ : S \to E$, the problem of finding a steady-state (t, F, p^+, p^-) with maximum throughput is polynomial-time solvable. The maximal throughput equals the value of a maximum flow from I to O in G with unit capacities.*

Proof. Apply Proposition 8.3 to the reverse splitter network. □

In contrast, when we force each splitter to have an output priority distinct from $\perp$ and the input capacities are not uniform, finding a maximal throughput is hard.

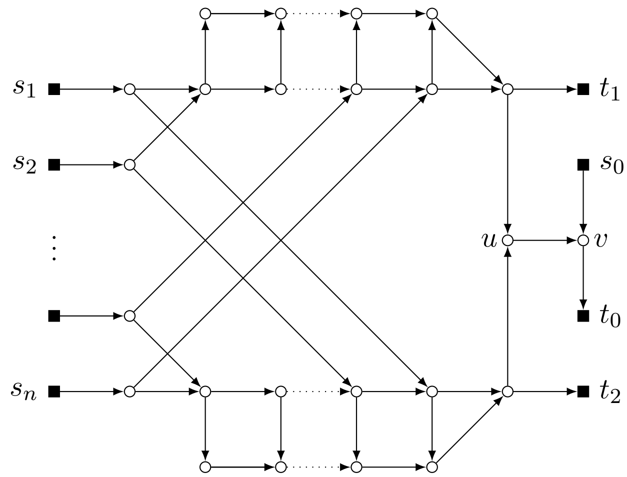

Figure 23: An illustration of the reduction from the partition problem to finding output priorities with maximal throughput.

Proposition 8.5. *Let $G = (I \uplus S \uplus O, E)$ be a splitter network, capacities $c : I \cup O \to [0, 1]$ and input priorities p^-, with $c(O) = 1$ for each $o \in o$. The problem of finding output priorities $p^+ : S \to E$ and a steady-state (t, F, p^-, p^+) of maximum throughput is NP-hard, even when $p^-(s) = \perp$ for each splitter $s \in S$.*

Proof. We reduce the partition problem: given $s_1, \ldots, s_n$ with $\sum_{i=1}^{n} s_i = 2$, is there a subset $I \subseteq [\![1, n]\!]$ with $\sum_{i \in I} s_i = 1$. The reduction is straightforward, and illustrated in Figure 23. The n leftmost terminal have output capacities s_1, $s_2, \ldots$, s_n respectively. s_0 and the outputs have capacity 1. The maximal throughput is 3 if and only if the partition instance has a solution. Indeed, when a partition exists, routing all the flows from terminals s_i with $i \in I$ up, and all the other flows down, provides a routing where no flow goes through u. Inversely, if any flow goes through u, it must go through v too. That flow will concur with the one unit flow between s_0 and t_0, therefore forbidding the total throughput out of s_0 to be one. Howerever, in order to avoid u to receive any flow, the output flow of each input s_i must be fairly split between the bottom and top parts of the splitter network, hereby defining a partition. □

8.4 Choose output priorities for some splitters

When an arbitrary subset of priorities are imposed, the maximum throughput problem becomes intractable, stays so even when we only choose some of the output priorities and all other splitters act fairly.

Theorem 8.6. *Let $G = (I \uplus S \uplus O, E)$ be a splitter network, $c : I \cup O \to [0, 1]$ a capacity function, $p^- : S \to E \cup \{\perp\}$ input priorities for each splitter, and $p_0^+ : S' \to E \cup \{\perp\}$ be a partial output-priority function, where $S' \subseteq S$. The problem of finding a steady-state (t, F, p^-, p^+) of maximum throughput with p_0^+ being the restriction of p^+ to S' is NP-hard, even when p^- and p_0^+ are uniformly $\perp$ and $c = \mathbb{1}$.*

The reduction is based on two gadgets, representing respectively the variables and the clauses of a 3-SAT formula. Before establishing the reduction, we study the two properties of the two gadgets. The variable gadget is based on the splitter network described in Figure 24. We will exploit the fact that its global throughput is not a convex function of the input capacities. Indeed, as depicted in Figure 24, for the input capacities $(1, 1, 0)$, $(\frac{1}{2}, 1, \frac{1}{2})$ and $(0, 0, 1)$, the global throughput are respectively $\frac{11}{4}$, $\frac{10}{4}$ and $\frac{11}{4}$. Therefore, by adding a splitter with undetermined priority, to provide the flow to the two opposite inputs, we incentivize a choice of priority different from $\perp$ for that additional splitter. Implementing this modification yields the variable gadget, defined in Figure 25.

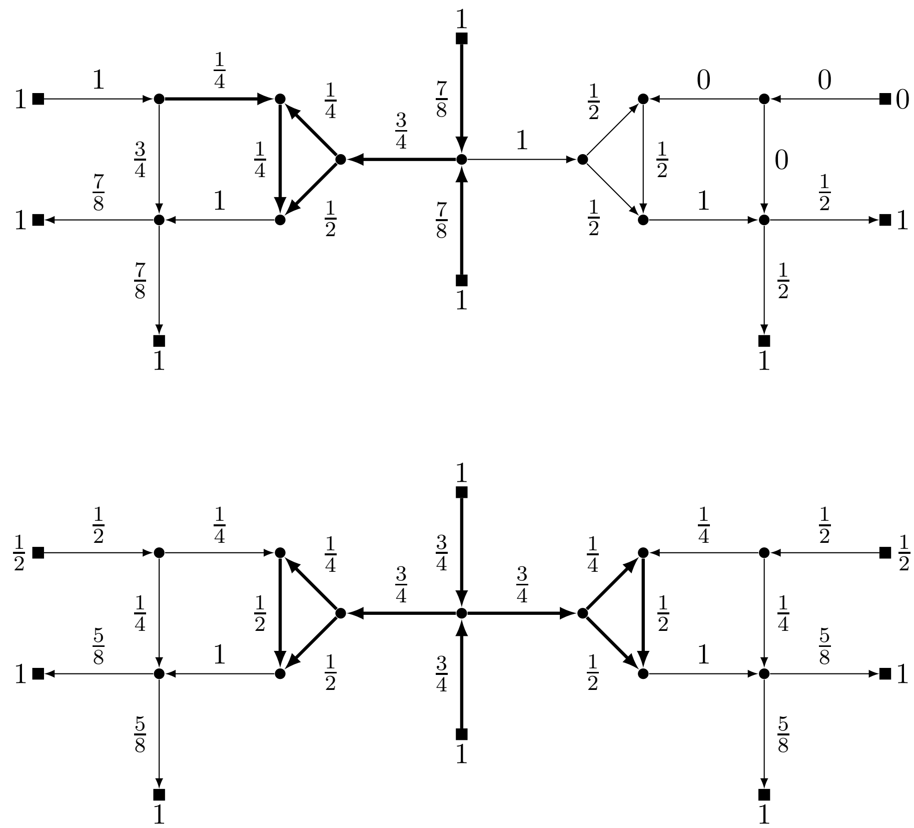

Figure 24: A network splitter with two capacity functions and their steady-states.

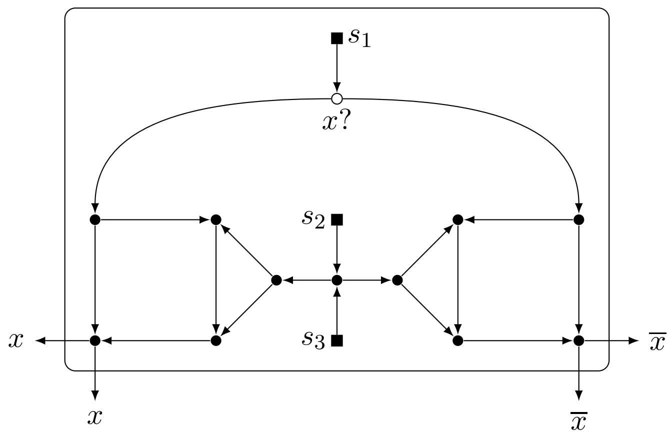

Figure 25: The variable gadget. The node $x?$ represents a splitter whose output priority is not predetermined. All other splitters s are fair: $p_0^+(s) = p^-(s) = \perp$. The leaving arcs indexed by x are connected to gadgets of clauses containing x positively, those indexed by $\overline{x}$ are connected to gadgets of clauses containing x negatively. If there are not enough clauses, each superfluous arc has a new terminal as destination.

Proposition 8.7. *Depending on the choice of $p^+(x?)$, and assuming that each arc leaving the gadget is fluid, the throughput on the leaving arcs of the splitter network depicted in Figure 25 are either $(\frac{7}{8}, \frac{7}{8}, \frac{1}{2}, \frac{1}{2})$, or $(\frac{5}{8}, \frac{5}{8}, \frac{5}{8}, \frac{5}{8})$, or $(\frac{1}{2}, \frac{1}{2}, \frac{7}{8}, \frac{7}{8})$.*

Proof. This follows immediately from the steady-states depicted in Figure 24, considering the throughput on the leaving arc of $x?$ will be either $1, 0$, or $\frac{1}{2}, \frac{1}{2}$, or $0, 1$, depending on the choice of out-priority. $\square$

Hence, each variable gadget induces a choice, that translates into some arcs carrying a throughput of $\frac{7}{8}$ and others $\frac{1}{2}$. We propose a clause gadget that can accept a throughput of up to $\frac{9}{4} = \frac{1}{2} + \frac{7}{8} + \frac{7}{8}$, forbidding the three inputs from having all a throughput of $\frac{7}{8}$. The splitter network presented in Figure 26 features two parts: the upper part receives the flow from the variable gadget, and reduces it to an eighth of its original value. Hopefully the remaining throughput should be at most $\frac{9}{32} < \frac{1}{2}$. The bottom part consists in a capacity simulating network, with capacity $1 - \frac{9}{32} = \frac{23}{32} > \frac{1}{2}$. Then we join this two flows toward the output t_1 in the splitter p. Increasing the input flow from the variable gadgets will increase the flow entering p from above. As $\frac{9}{32} < \frac{23}{32}$, by respecting rule P6, the throughput on the arc entering p from the right would then decrease, and thus s_2 would provide strictly less than $\frac{23}{32}$.

Proposition 8.8. *For any throughput values f_1, f_2, f_3 on the three entering arcs of the clause gadget, depicted in Figure 26, there is a steady-state (t, F) for which the three entering arcs e_1, e_2, e_3 are fluid, with $t(e_i) = f_i$ for each $i \in \{1, 2, 3\}$. Moreover, $f_1 + f_2 + f_3 \leq \frac{9}{4}$ if and only if $t(\delta^+(s)) = \frac{23}{32}$.*

Proof. In order to show that the arcs entering the gadget are always fluid, we consider the case when the throughput on the three entering arcs e_1, e_2 and e_3 are uniformly 1. Then the throughput on each arc of the upper part of the gadget can be easily computed, and the throughput on the arc e entering p from above is $\frac{3}{8}$. This arc is fluid, and the other arc e' entering p from the right is saturated with throughput $\frac{5}{8}$. Therefore e_1, e_2 and e_3 are fluid in this steady-state.

More precisely, the throughput on e is $t(e) = \frac{f_1+f_2+f_3}{8}$, and the throughput on e' is $t(e') = \max\{\frac{23}{32}, 1 - t(e)\}$, with e' saturated when $t(e) > \frac{9}{32}$. The steady-state at the critical point, when $f_1 + f_2 + f_3 = \frac{9}{4}$, is illustrated in Figure 26. $\square$

Using the two gadgets, we prove that the throughput maximization problem is NP-hard.

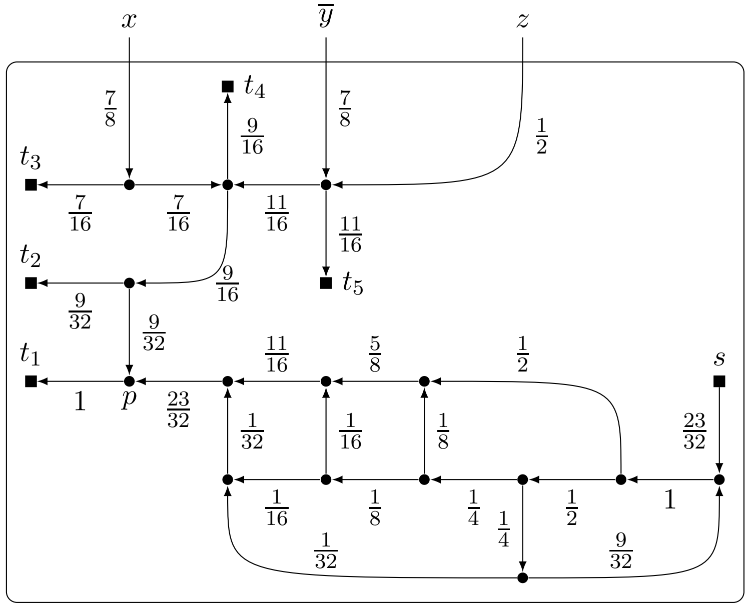

Figure 26: The clause gadget, for a clause $x \vee \overline{y} \vee z$.

Proof of Theorem 8.6. We reduce 3SAT, where each variable appears at most twice positively and twice negatively. Let ϕ be a set of 3-clauses on variables $x_1, \ldots, x_n$. We define a splitter network, with one copy of the variable gadget for each variable in the formula, and one copy of the clause gadget for each of its clauses. A leaving arc labeled x (respectively $\overline{x}$) is identified with an entering arc of a clause gadget containing the literal x (respectively $\overline{x}$). Any superfluous leaving arc is redirected to a new distinct output node. The priorities of every splitter are set to $\perp$, except for the output priority of the splitter $x?$ from the variable gadget, for each variable x. The output priority of that splitter is free. Formally $S' = S \setminus \{x? \ : \ x \in \{x_1, \ldots, x_n\}\}$, p^- is uniformly $\perp$ on S, and p_0^+ is uniformly $\perp$ on S'. The size of the splitter network thus constructed is a polynomial in $|\phi|$ and n.

Suppose that ϕ is satisfiable, and consider a satisfying assignment of the variables. For each variable x, set the output priority $p^+(x?)$ to the arc toward the right if x is assigned to *true*, and to the arc toward the left if x is assigned to *false*. As a consequence, in the steady-state, by Proposition 8.7, the throughput on the arcs labeled x is $\frac{1}{2}$ if x is *true*, and $\frac{3}{8}$ otherwise, and symmetrically for the arcs labeled $\overline{x}$. Then, because each clause is satisfied, at least one arc entering each clause has throughput $\frac{1}{2}$, and the two others have throughput at most $\frac{7}{8}$. Hence each clause receives at most $\frac{9}{4}$, and by Proposition 8.8, the throughput of its input s is $\frac{23}{32}$. Thus the total throughput is $\frac{11}{4}n + \frac{23}{32}|\phi|$.

Conversely, we claim that if their is a choice of priority with a steady-state whose global throughput is at least $\frac{11}{4}n+\frac{23}{32}|\phi|$, then ϕ is satisfiable. By Proposition 8.7, a variable gadget can produce at most $\frac{11}{4}$ unit of flow from its input, and a clause gadget at most $\frac{23}{32}$ by Proposition 8.8. Therefore such a throughput can only happen if every variable gadget produces $\frac{11}{4}$, which means that the priority $p^+(x?)$ is not set to $\perp$. It is also necessary that each clause receives at most $\frac{9}{4}$ unit of flows from the variable gadgets. The priorities of the splitters $x?$ induce an assignment of truth values to the variable (*true* if the priority is to the arc to the right, and *false* to the left). The $\frac{9}{4}$-bound imposes that for each clause, there is a arc from a variable gadget entering that clause's gadget with throughput $\frac{1}{2}$. By construction, that arc corresponds to a literal that is satisfied by the assignment. Therefore ϕ is satisfiable. The splitter network admits a choice of priorities with a steady-state whose global throughput is at least $\frac{11}{4}n + \frac{23}{32}|\phi|$ if and only if ϕ is satisfiable, therefore the throughput maximization problem is NP-hard. $\square$

As the gadgets are planarly embedded, reducing from planar-3SAT is also possible, and thus the maximization throughput problem stays NP-hard with the constraint that the splitter network is planar.

9 A balancer with saturated arcs

In Theorem 1.12, we proved that a $(2^k, 2^k)$-balancer must have at least $k2^{k-2}$ fair splitters. We also define the simple balancer of order k, a $(2^k, 2^k)$-balancer with $S(k) = (k-1)2^{k-1}$ splitters. In this section we exhibit a balancer with priorities whose number of fair splitters is only $(k+1)2^{k-2}$. Hence this balancer is much closer to the lower bound in regards to the number of fair splitters. Unfortunately, it also contains many splitters with non-fair output priorities, making it asymptotically larger than the simple balancer. The construction is nonetheless interesting, not only for its smaller number of fair splitters, but also in demonstrating that saturation may also be used in an effective way to balance the outputs.

The saturating $(2^k, 2^k)$-balancer is composed of three parts, illustrated in Figure 27. The first part contains the input and a half-grid of priority splitters, whose output priority are set to the same direction (vertical in the picture). This design ensures that the flow is pushed as much as possible to the top of the grid. Therefore the throughputs on the arcs leaving the half-grid, from top to bottom, are $1, 1, \ldots, 1, t, 0, \ldots, 0$, where $t = c(I) - \lfloor c(I) \rfloor$. In particular, if $c(I) \leq 2^{k-1}$, all the flow leaves the half grid from the top 2^{k-1} arcs, else the 2^{k-1} receive 2^{k-1} units of flow.

The next part is made of a simple balancer of order 2^{k-1} from the 2^{k-1} highest rows to the 2^{k-1} lowest rows. The last part is a column of 2^{k-1} fair splitters, whose out-neighbors consist in the network's outputs. From the outputs of the simple balancer to those fair splitters, 2^{k-1} arcs serves as the main bottleneck of this network.

The saturating balancer works under two possible regimes. When $c(I) \leq 2^{k-1}$, all the flow is pushed in the simple balancer, arrives balanced into the bottleneck arcs, and the is split evenly by the last column of splitters toward the output. Then every arc is fluid. The other regime happens when $c(I) > 2^{k-1}$. At $c(I) = 2^{k-1}$, all the arcs in the simple balancer and in the bottleneck reach throughput 1. Increasing the input capacity above 2^{k-1}, the arcs of the simple balancer saturates. The excess flow is pushed into the 2^{k-1} lowest rows, directly toward the bottleneck arcs. Because the bottleneck is already full, the incoming flow is pull backed by the simple balancer. At this point, the simple balancer, which is fully saturated, acts a balancer on the pulled back flow: the pulled back flow is sent back evenly to its input, from which it proceeds on the horizontal arcs of the 2^{k-1} highest rows, to the last column of splitters. Thus, the potential of each fair splitter in the simple balancer is fully used: as long as $c(I) < 2^{k-1}$ it splits the throughput fairly into its two outgoing fluid arcs. When $c(I) > 2^{k-1}$, it splits the pulled-back throughput evenly into its two incoming saturated arcs.

The saturating $(2^k, 2^k)$-splitter contains $2^{k-1} + S(2^{k-1}) = (k+1)2^{k-2}$ fair splitters, which is closer to our lower bound of $k2^{k-2}$ splitters from Theorem 1.12. The difference is explained by the last column of fair splitters, which are only used with fluid outgoing arcs. The total number of splitters is $\Theta(2^{2k})$, dominated by the half-grid. However, this part can be replaced by a network with the following property: the total throughput on the 2^{k-1} highest leaving arcs must be $\min\{c(I), 2^{k-1}\}$. Such a network can be defined with $o(2^{2k})$ priority splitters, but we do not know whether one exists with only $O(k2^k)$ priority splitters.

10 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.

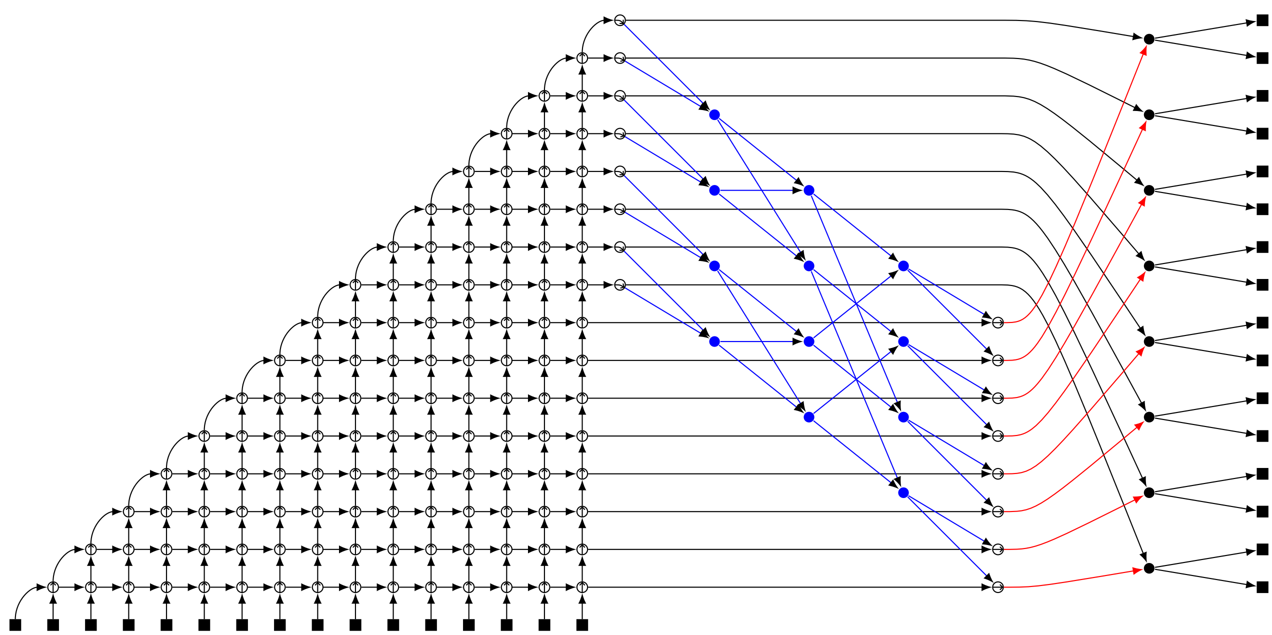

Figure 27: A $(2^k, 2^k)$-balancer utilizing priorities, which contains significantly less fair splitters than the simple balancer. In this example, we set $k = 4$. Fair splitters are represented by full nodes. In each priority splitter, a small arrow indicates which arcs are prioritized: the outgoing arc pointed to by the arrow's head, and the incoming arc aligned with the origin of the arrow. On the right, the half-grid pushes up to 2^{k-1} units of flow to the top rows. In blue, a simple balancer of order 2^{k-1} proceeds to most of the balancing. The red arcs form a bottleneck, when the total throughput exceeds 2^{k-1}. On the right, a last column of fair splitters distributes the flow to the outputs.

Although our continuous model is convenient for modeling the expected throughput of splitter networks, Factorio's belt systems operates discretely. Therefore, the observed throughputs in Factorio's 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.

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. This is partially done in Section 9, at the cost of introducing many priority splitters. 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 [21] 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? We proved several complexity results for the problem of global throughput maximization, when we can choose which arcs to prioritize in each splitter or a subset of those splitters. However, the complexity of global throughput maximization when the input and output capacities are arbitrary is not fully settled yet.

As a last series of questions, 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 deciding 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

[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: *Conference on Systems Engineering Research*. Springer. 2023, pp. 297–311.

[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 A. 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: *Purdue Undergraduate Research Conference* 57 (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

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

[13] Donald E. Knuth and Andrew C. Yao. "The complexity of nonuniform random number generation". In: *Algorithms and Complexity: New Directions and Recent Results*. Ed. by JF Traub. Addison-Wesley, 1976, pp. 357–428.

[14] Bernhard Korte, Rainer Schrader, and László Lovász. "Abstract Convexity — Antimatroids". In: *Greedoids*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 19–43. ISBN: 978-3-642-58191-5. DOI: 10.1007/978-3-642-58191-5_3. URL: https://doi.org/10.1007/978-3-642-58191-5_3.

[15] Legnagi, Alessandro and Montini, Axel. *VeriFactory*. https://github.com/alegnagi/verifactory/.

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

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

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

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

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

[21] pocarski. *Universal 8-8: Perfectly Balanced, as All Things Should Be*. https://web.archive.org/web/20230922

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

[23] 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.

[24] 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.

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