Factorio Belt Balancers

A compilation of research and occasionally a useful guide

Contents

Introduction

This "guide" is the sum total of my belt-balancing knowledge after approximately 15 hours of research, testing and investigation. In it, I explain some of the theory behind how belt balancers work and how to design them properly. I have come up with a reliable method to design one-to-many splitters and many-to-one mergers. I have also come up with a method to design many-to-many balancers, but the designs produced are too large to be of practical use.

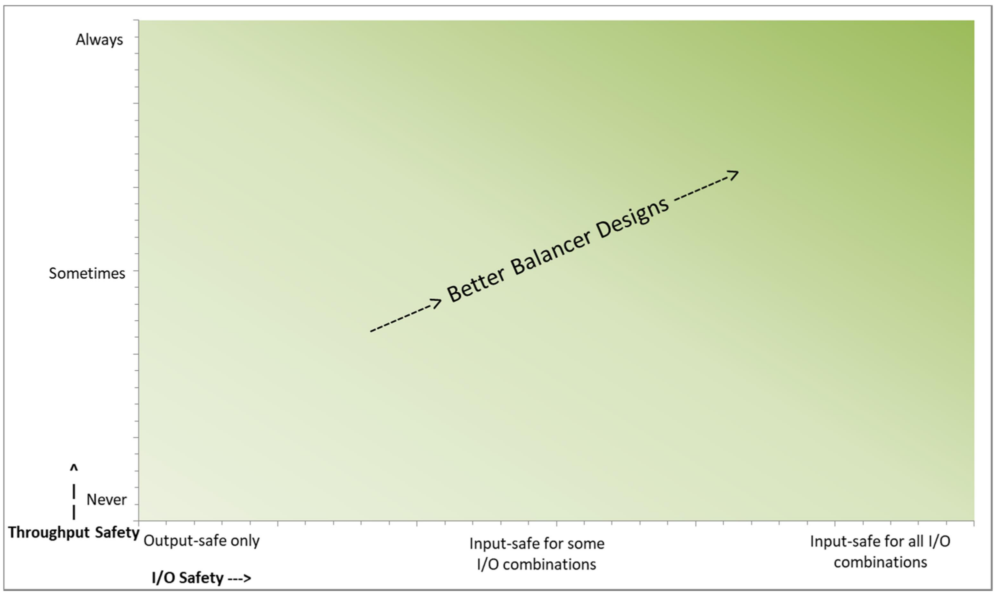
Bear in mind that the following information is merely the conclusions I have reached from my own investigation; I do not have answers to some of the questions raised, and there is always the possibility that I am wrong about something.

However, I hope that at the very least I will shed some light on the mysterious belt spaghetti, and perhaps give you a 15-hour head-start on your own learning.

N.B. Some of the images used show belts with circuit-network scanners on them. These are not part of the designs, and are only there for counting the number of items passing through each line as part of my testing process. The strange boxes visible at each end of the belts in some pictures are Matter Sources and Matter Voids from the Creative Mode mod; they make it easy to test by automatically filling and emptying belts.

Categorising Balancers

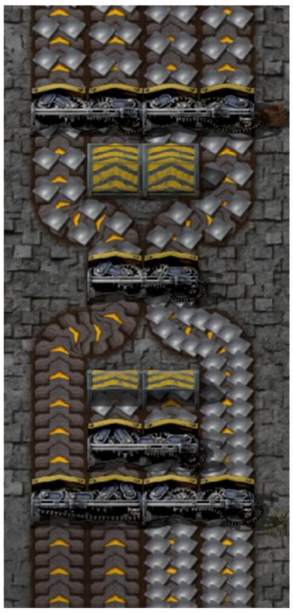
First, I shall explain the different types of balancer, and what the categories mean. All balancers will be located somewhere on the following graph, categorised by Input/Output Safety, and Throughput Safety.



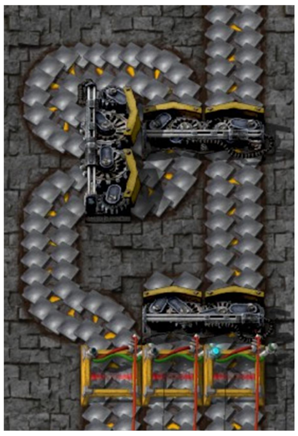
Throughput Safety

Throughput safety is the guarantee that the balancer will not be a choke point; i.e. the items leaving will never be less than the items arriving. There are three possible categories:

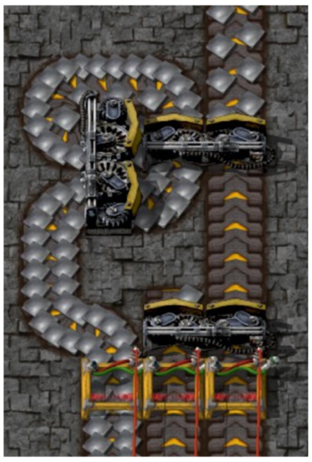
ALWAYS, where the balancer is throughput-safe regardless of how many input or output belts are active. E.g. the classic 4-to-4 balancer:



SOMETIMES, where the balancer is throughput-safe at full capacity (when all input-belts are running), but one or more combinations of input belts being less than full and/or outputs being backed-up will cause a bottleneck. E.g. this 3-to-1 merger:



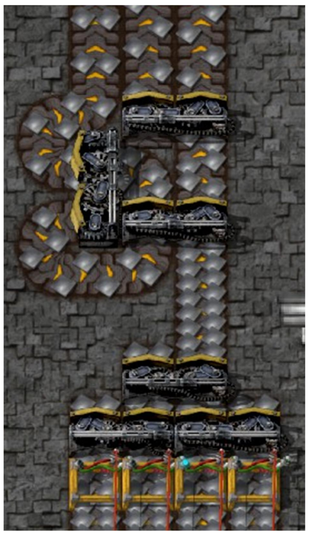
If the two rightmost belts cease to input, then it will only output half a belt, despite a full belt still coming in:



This is because now the only items coming onto the output belt are from a single splitter, which sends half the full belt elsewhere.

SOMETIMES is less of a discrete category and more of a continuous scale, as a design may be throughput safe with a greater or lesser number of its I/O combinations, but still not all.

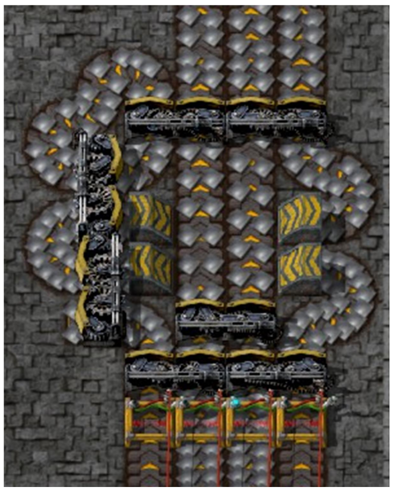
NEVER, where the balancer does not achieve full throughput when all input belts are active. These balancers are pretty much useless, as they will never achieve full throughput. E.g. the following 4-to-3 balancer:



This bottleneck is due to the whole lot having to pass through a single belt in the middle. This demonstrates why you can't create an x-to-y balancer just by chaining an x-to-1 merger and a 1-to-y splitter.

Better throughput safety can often be achieved if you rework the design; however, the safer designs tend to be much larger and more complex than their less safe variants, so unless you need that guarantee, it's probably not worth the extra space and effort.

For example, here is a sometimes-throughput-safe 4-to-3 balancer:



The above design is safe for one, two, or four full input belts, but not three. Outputs backing up however will not cause problems.

Very few balancers are truly always-throughput-safe, although many are safe enough for general use, as their throughput-limiting conditions are few and unlikely to occur.

Input / Output Safety

All proper balancers pull evenly from all of their inputs under ideal conditions: when all input belts are full and no outputs are backed-up. This is guaranteed by definition; any belt setup that does not do this is not balancing the belts, and so is not a balancer.

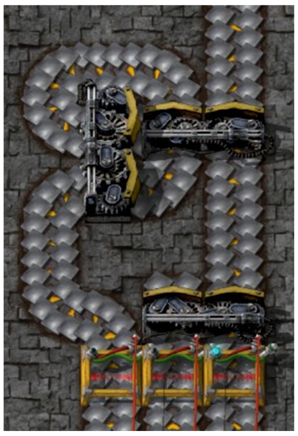
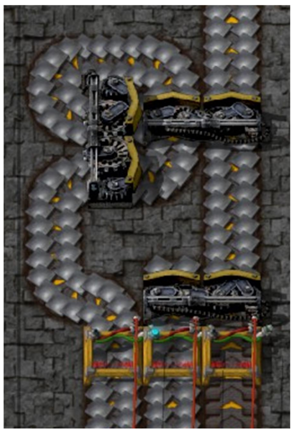
Input safety guarantees that if one or more of the input belts are not filled, the balancer will still pull evenly from the remaining belts.

Output safety guarantees that if the output is backed-up and not moving at full speed, the balancer will still pull evenly from all input belts.

Note that backed-up output can still cause problems with throughput on an output-safe balancer; the issues are unrelated.

There are three possible categories (two, really, as I'm fairly sure the last one is impossible):

OUTPUT-SAFE ONLY, where supply exceeding demand leading to slow output does not cause a problem, but demand exceeding supply can if this leads to uneven input belts. I have been unable to come up with a design that isn't output-safe for full input belts, so it is (probably) safe to assume all balancers are. E.g. the 3-to-1 merger:

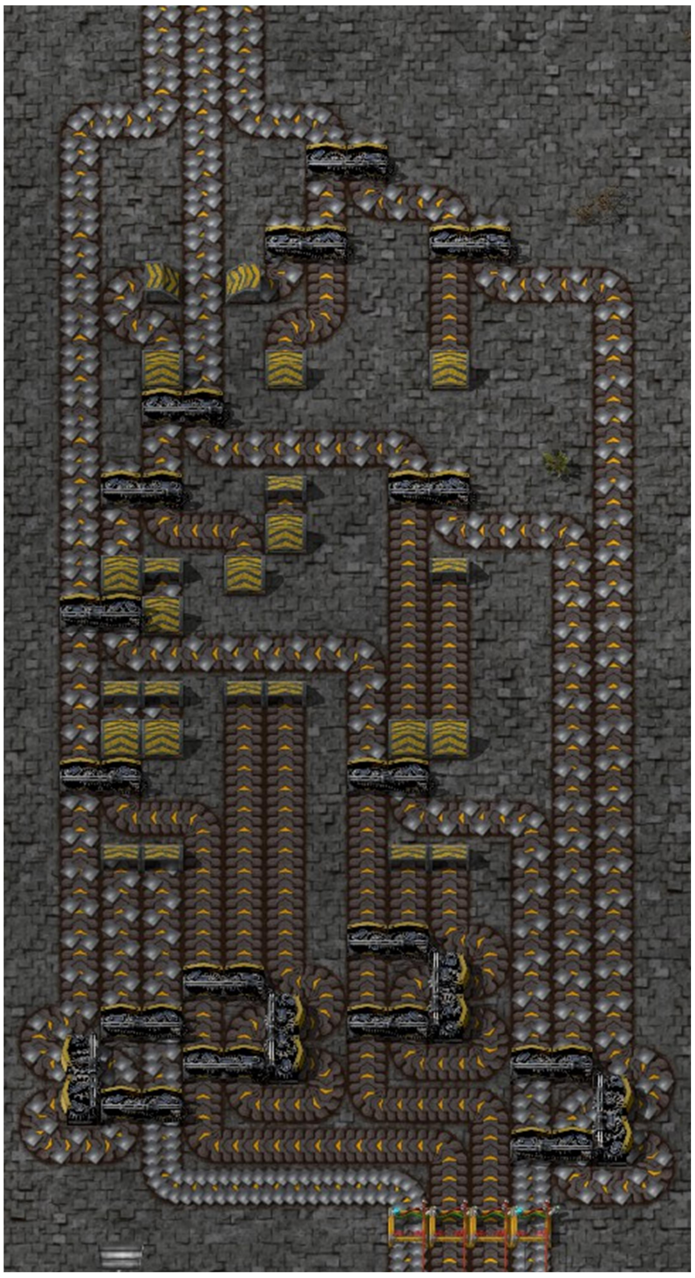



In the left image, all belts are filled and the output will pull one third from each of the inputs.

In the right image, one of the belts has become empty, and so due to the splitter layout, the output will pull one third from the left input, and two thirds from the centre input. Both behaviours are unchanged if the output is backed-up.

Note that output-safe balancers can be input-safe for some combinations only; the 3-to-1 merger above is still input safe if the left-most input belt becomes empty, as the other two will be pulled from evenly.

INPUT OR OUTPUT SAFE (MUTUALLY EXCLUSIVE), where you can have EITHER the output back up, OR any combination of input belts become empty and the balancer will still pull evenly from the inputs. If both happen at the same time (e.g. one of the inputs is from a mine that has just run out, and the ore is still being produced faster than it is being consumed), then the balancer will pull unevenly between the inputs. E.g. this 4-to-3 balancer:



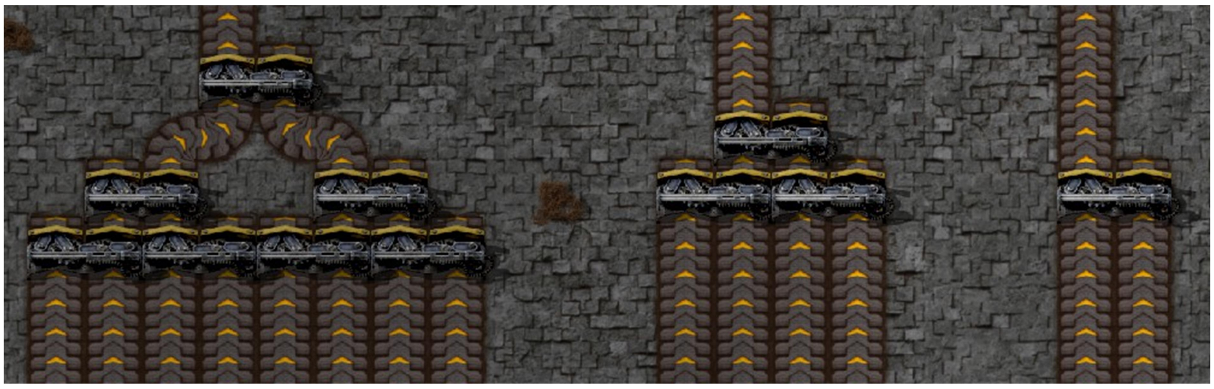
However, if the output becomes backed-up, and the items on the middle sections become fully compressed, the balancer will stop working, and will pull unevenly until full output speed is restored.

INPUT AND OUTPUT SAFE (SIMULTANEOUSLY), where even in the case of a backed-up output and empty belt(s) in the input, the balancer will still pull evenly from all working inputs. I have not been able to create any balancers which fulfil these criteria, and I'm fairly sure it is impossible.

Input safety is obviously a desirable property for a balancer, as (most of the time) you don't want to deplete some inputs faster than others. However, it is difficult to achieve (impossible for some combinations?), and so most designs are a compromise that are only input-safe in some circumstances.

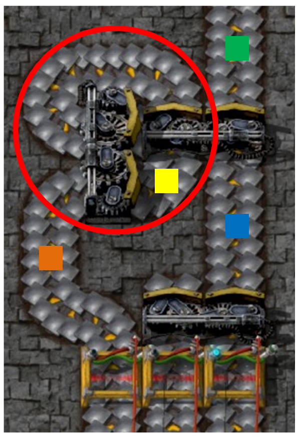
Designing x-to-1 mergers

Powers of two (2,4,8,16,etc.) belts can easily be merged simply by stacking splitters into a big pyramid, as each splitter takes two inputs and produces one output. Other than the 2-to-1, these are not input-safe for all combinations.



Things which are not powers of two must be broken down into powers of two in order to merge them. For example, 5 becomes 4+1 (2^2+2^0). Then, the smaller parts are boosted via *feedback loops* to become equivalent to the larger parts, whereupon they can be merged as equals.

A feedback loop can be seen in the red circle in the following image:



They originate where a smaller branch is merged into a bigger one (typically they originate at the very final merge if there is more than one merge). They then loop back and re-join the branch they left. This causes the effective "strength" of the branch to be doubled each time the loop re-joins. By "strength", I mean the number of inputs contained within this branch.

Taking the above example, the belt marked with the blue rectangle is formed by merging two inputs, and thus has a strength of 2 (both real and effective). The orange belt contains only one input, and thus has a strength of 1 (both real and effective).

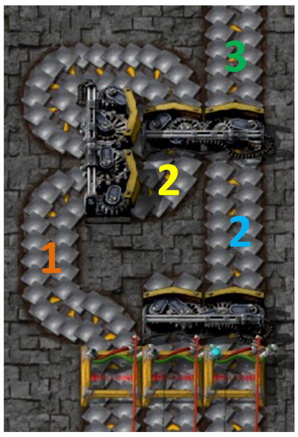
If the orange and blue belts were to be merged directly without the feedback loop, the mismatch in effective strength would cause twice as much to be pulled from the orange belt as the blue one, rendering our balancer, well, unbalanced.

However, at the location of the yellow rectangle, the feedback loop has re-joined the branch, doubling its effective strength. Thus, that singular tile of belt has real strength 1 and effective strength 2.

Because the yellow and blue belts both have an effective strength of 2, they can be merged perfectly, and all three inputs will be pulled from equally.

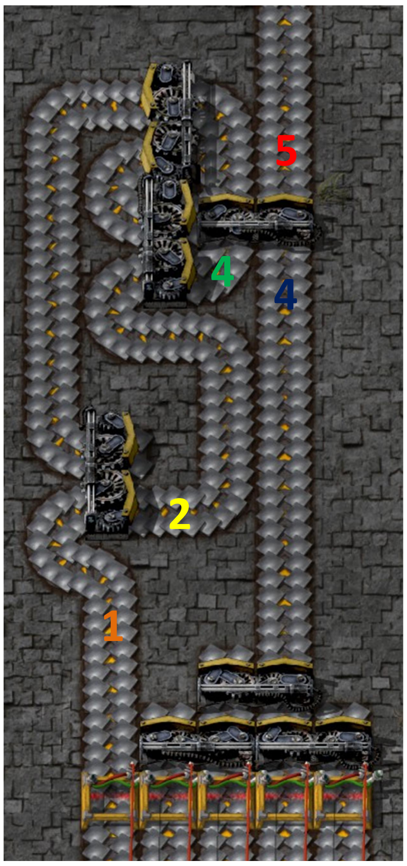
Now for something incredibly important but not obvious. <u>Once the belt has proceeded past the origin of the feedback loop, the effect on the effective strength is lost</u>. Thus, if the belt marked with the green rectangle was to be merged further, it would have an effective strength of 3 (its real strength), <u>not</u> 4 (from merging two 2s).

See below for the effective strength of each belt at each point:



Thus, 3 belts (2+1) can be evenly merged by boosting the 1 into an effective 2.

This quirk is important when one needs to double the effective strength of a branch twice. The correct layout for a 5-to-1 merger is shown below, along with effective strength numbers:

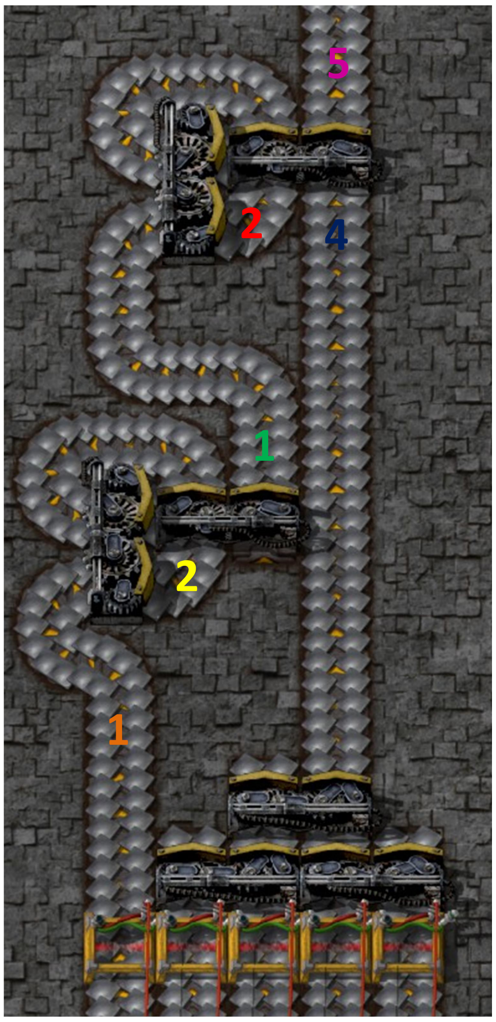


This is the 5-belt merge; 4+1 where the 1 is doubled twice into an effective 4.

Note that the doubling effect persists until it has gone beyond the origin of the loop, whereupon it reverts to its true strength of 5.

Note also that to achieve this layout, the feedback loop itself was split into two equal portions. If three re-joins were desired, an accurate 1-to-3 split would be needed.

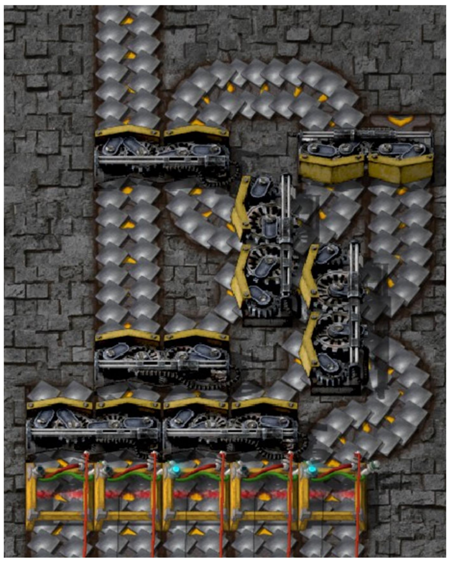
An easy mistake to make is to use two separate feedback loops, pictured below:



Although there are the same number of feedback loop re-joins, the effect of the first loop expires before the second is reached. At the green number the belt has moved beyond the origin of the loop, so its effect is lost and the belt reverts back to an effective strength of 1.

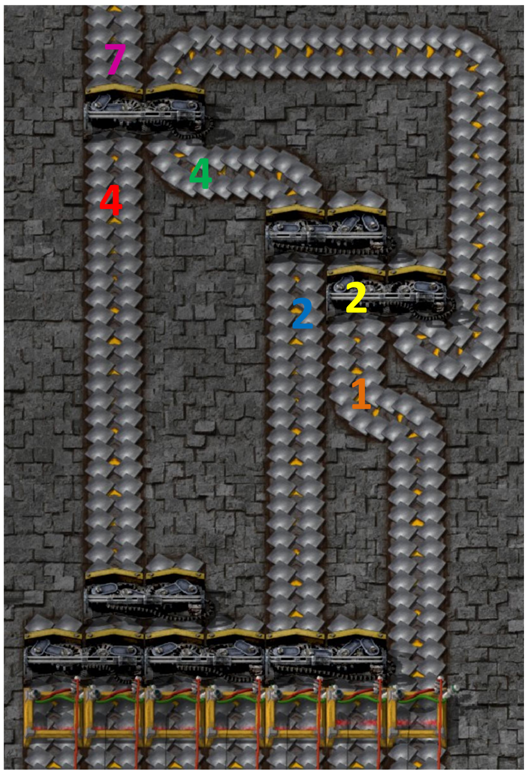
Thus, the final merge is an effective 2 with an effective 4, so the merge is uneven, and the leftmost input gets drawn from twice as much as each of the others.

It is perfectly possible to compress these designs by moving the belts around and reusing bits of belt for multiple loop paths. For example, the wiki gives this design for a 5-to-1 merger:



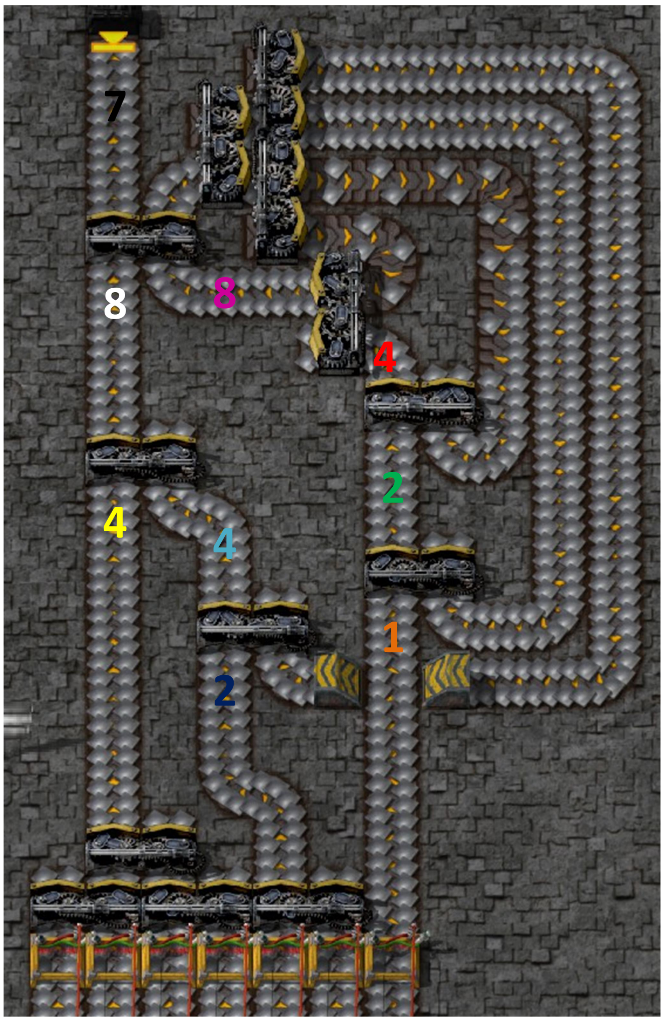
It is functionally equivalent to my design, and if you follow the paths, you can see that there is indeed a single feedback loop that re-joins the 1-real-strength branch twice.

There is one additional complication to be aware of: beware of splitting a single feedback loop into too many different re-joins. Take the following 1-to-7 splitter, which works correctly. It operates on a 4+2+1 basis, where the 1 is doubled once, combines with the 2 into a 4, and then combines with the real 4:



Note that because the loop origin is at the final merge, after the yellow+blue merge happens (real 2 + effective 2) it stays at an effective strength of 4 rather than reverting straight to 3.

The next design looks like it should work. However, it does not.



The idea is that the 2 is doubled into a 4, combined with the real 4 to make an 8, and then the 1 is doubled three times into an 8, and the two 8s are finally merged together.

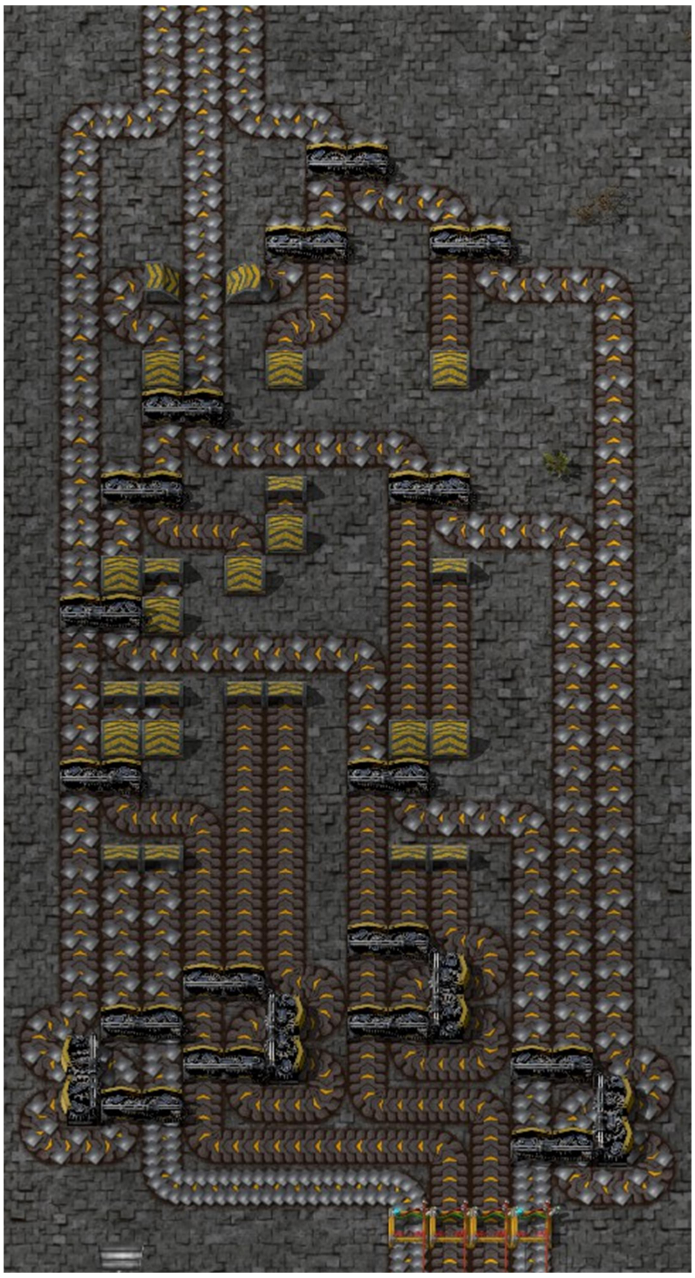
When run, this balancer does not work properly and pulls unevenly from the rightmost belt compared to all the others. I will admit, I cannot figure out why this is the case. Even the ratio with which it pulls from that belt compared to the others is difficult to determine, as it seems to vary considerably over time.

It is possible that there is not enough throughput in the feedback loop to power four re-joins; note that two of the re-join loops are gradually emptying.

It is also possible that the issue lies with the location of the re-joins, as they take different paths back to the origin of the loop. However, I think the first hypothesis is more likely, as the gradual emptying of the feedback loop could feasibly cause the gradual shifts in input pull ratio observed.

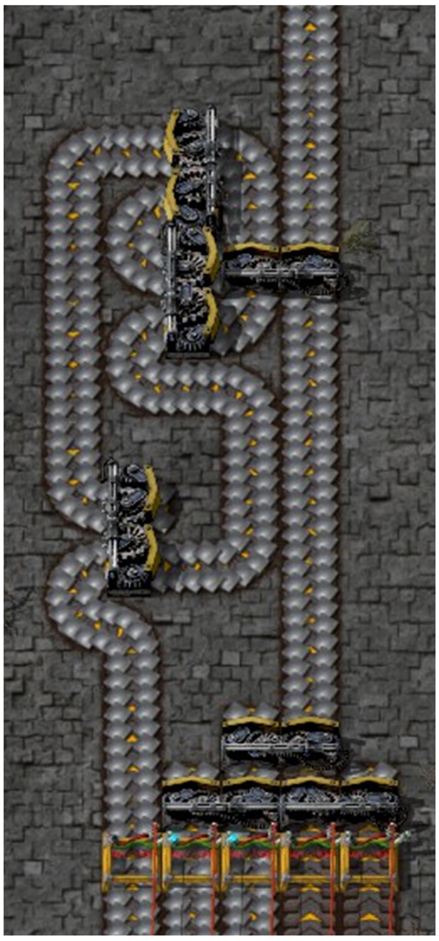
Creating fully-input-safe balancers is much more difficult, and I have yet to find a reliable method. My current theory is that this is simply not possible for some configurations, such as 3-to-1. Input-safety is definitely linked to whether the balancer is backed-up or not. As conjectured in the first table, it appears to me that a balancer that is input-safe when the output is backed-up is impossible.

The only fully-input-safe balancer I have made (by accident) is the following 4-to-3, where each input is split into three, and then the four sets of three are merged into the three outputs:



Note that the whole balancer is freely-moving; there are no sections at a standstill or moving in jumps. As far as I can tell, this is a requirement for input-safe balancing. I have no idea how to reliably achieve it.

The following 5-to-1 balancer is not input-safe:



It is not freely-moving, and belts move in jerky steps.

Freely-moving balancers are only possible when the output capacity >= the current input. Therefore, cramming three lanes into one as above is always going to back-up internally even with a free-flowing output, and so looks like it will never be input-safe for three inputs, and likely not for four either (unless the four in question are the rightmost four belts, as then it would effectively become a four-belt merger with the whole of the left section irrelevant).

This supports the hypothesis that some balancers can never be fully-input-safe.

So, in summary, to reliably create an output-safe, sometimes-throughput-safe, x-to-1 merger:

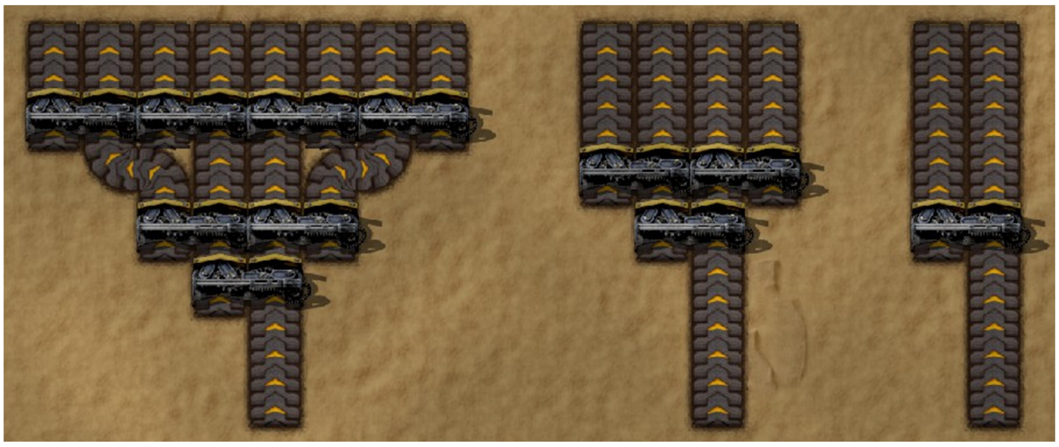
1. Split x down into the largest possible powers of 2 (e.g. 11 = 8+2+1).
2. Create feedback loops to boost the effective strength of the smaller branches, then merge them into ones of equal effective strength.
3. Remember that the boosting effect of loops expires once the belt goes beyond the origin of the loop.
4. Don’t create more feedback loops than your throughput can support.
5. Keep your input belts filled if possible.
6. Keep your output belts flowing if possible.

Designing 1-to-x splitters

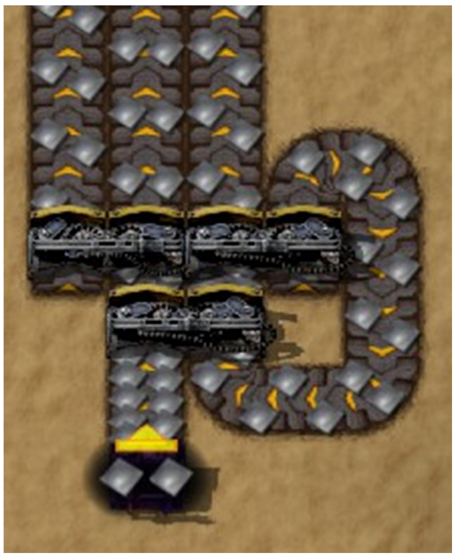
I/O safety is meaningless for 1-to-x splitters as they have only one input belt; they cannot pull unevenly from their inputs. Throughput safety is also easier to achieve, since you are typically expanding the throughput at each stage, so bottlenecks do not form.

This makes splitters much easier to build than mergers or balancers, although getting them compact takes a lot of skill (more than I have), and can introduce throughput-safety issues.

As before, powers of two are easy; just create an upside-down pyramid of splitters:

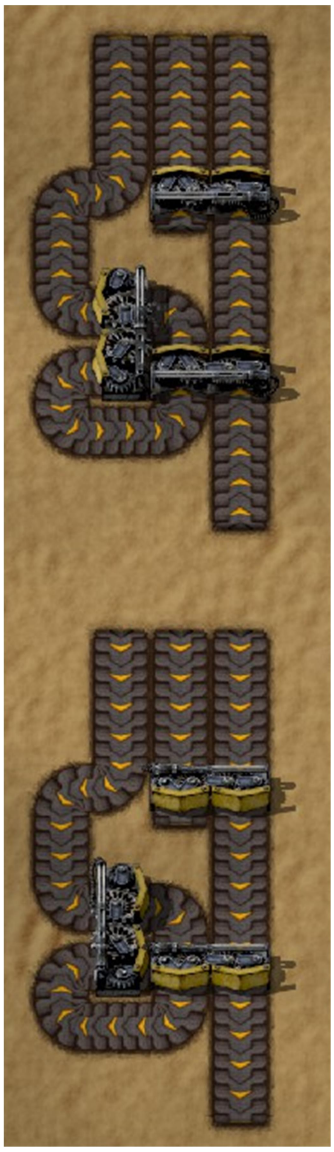


For other types, simply split to the closest power of two above the x you are trying to achieve, and then fold the extra outputs back into your original input. E.g. a 1-to-3 splitter:

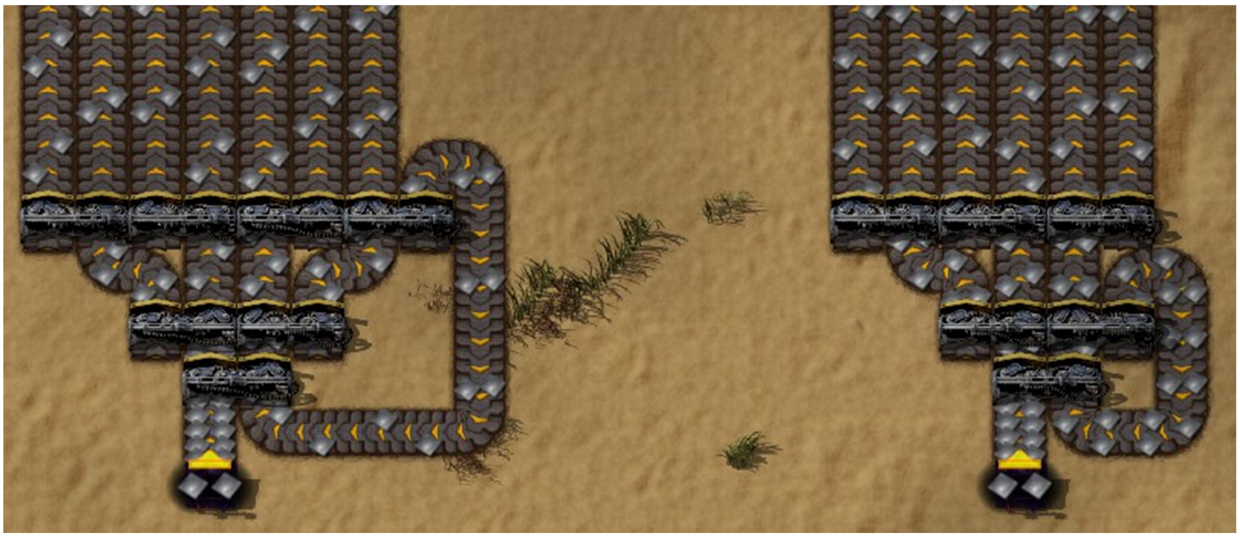


The input is simply split into quarters, and then the extra quarter is folded back to the start, giving three equal outputs. Note the similarity between the fold-back here and the feedback loops present in mergers.

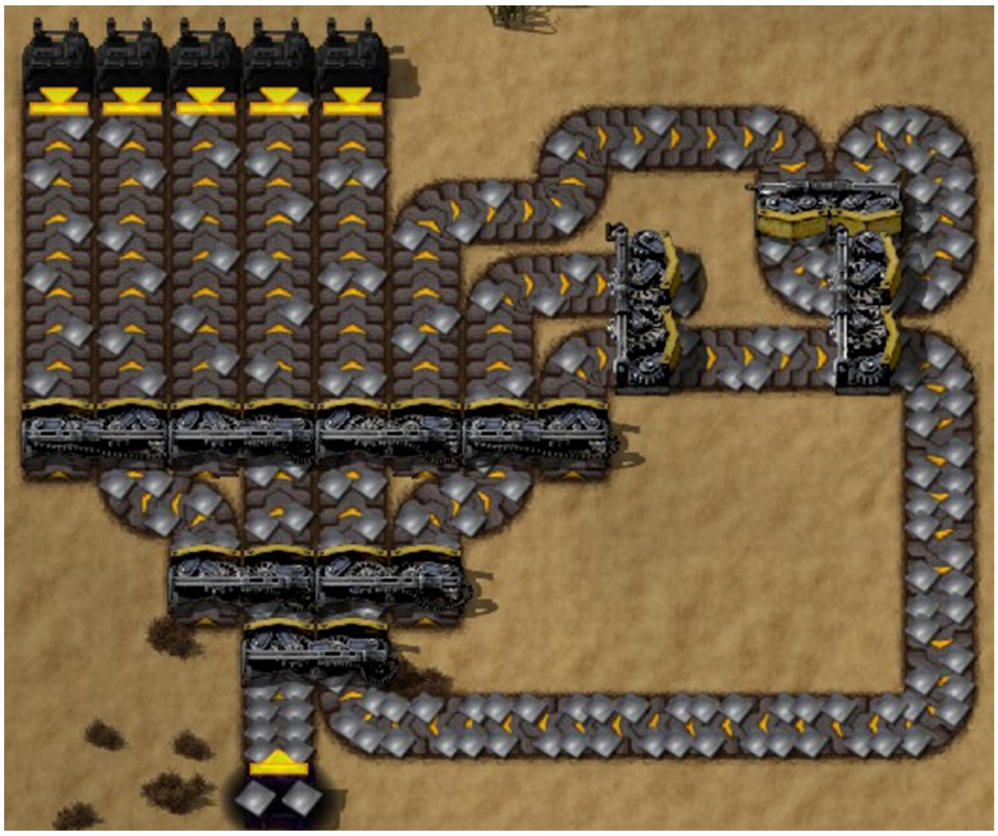
In fact, valid designs for 1-to-x splitters and x-to-1 mergers are often reverse images of each other:



Another example is the 1-to-6 (right) and 1-to-7 (left) below. The input is split into eights, and the extra(s) folded back to the start.



The 1-to-5 is a slightly more complex example, as the 3 extra outputs require a 3-to-1 merge:



The principle is the same.

All of these can be made more compact, though the solutions are not obvious. Look to the wiki for optimal designs.

<u>**Designing x-to-y balancers**</u>

These are tricky. The theory of what they must achieve to work is simple enough; an x-to-y balancer must:

- Have a route from every possible input to every possible output
- Each input must feed all outputs with equal priority

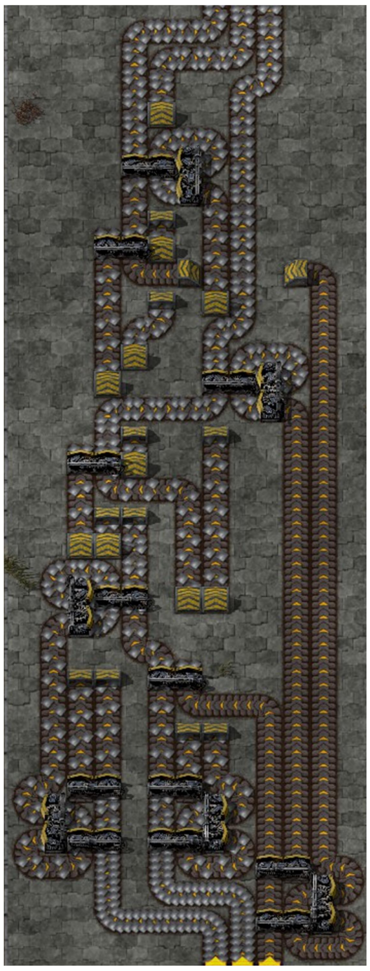
Chaining an x-to-1 merger with a 1-to-y splitter is no use, as the throughput will be limited to one belt maximum.

The only reliable method I have found is to split each input into as many equal parts as there are outputs, and then combine the all the same parts from each input together.

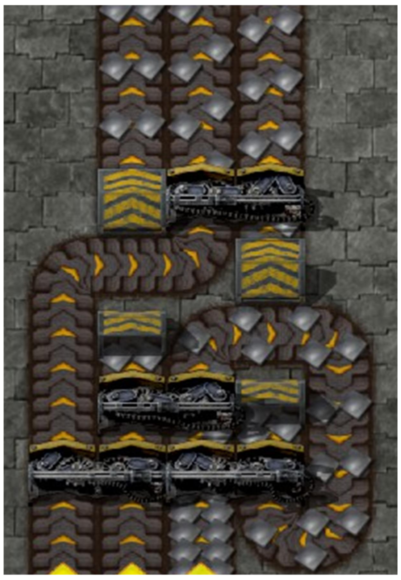
E.g. a 5-to-3:

1. Split each of the five inputs into three parts, numbered 1 to 3.
2. Use a 3-to-1 merger to combine all the number 1s together.
3. Repeat for all the number 2s, and all the number 3s.
4. You now have three balanced outputs.

The drawback to this method is that the balancers end up huge; here is a 3-to-3 balancer:



These designs are not very practical. The internet tells me that the entire setup can be reduced down to this design:



Unfortunately, I have no idea how this design was reached. By following the paths, it can be seen that it is equivalent to the large version; but I don't know how to come up with something like this in the first place.

If you want a compact balancer, you'll have to look to the wiki, where people far better at this than I have figured it out already. It is a shame that they have not shared their methods.

Conclusion

This is the end of the guide. It is unfortunate that I don't have more to offer on the subject of x-to-y balancers, but I simply don't know how to do them any better. If you have anything to add, tips and tricks I missed, have spotted an error etc. please give feedback on the Factorio forums. My forum handle is EX_plode.

Thank you for reading.