beltmap: schematic pictures of Factorio belt layouts

Landon Fox

version 0.6, 2026-11-23

Factorio is a factory-building game in which items ride on conveyor belts. A *splitter* sits across two adjacent belts, takes what arrives on both, and sends half out of each side. A *balancer* is a network of splitters whose outputs each carry an equal share of every input. beltmap draws these networks in LuaLaTeX and computes what they do.

Contents

1 Getting started **2**

2 Reading a picture **3**

3 Putting a layout in the document **3**

4 Changing the look **5**

5 Checking a layout **8**

6 Pictures of the network **12**

7 Blueprints **14**

8 Galleries, animations, and export **14**

9 Game icons **15**

10 Accessibility and speed **15**

11 Reference **16**

12 Installing **18**

13 Known limitations **18**

1 Getting started

One splitter is a balancer of two belts:

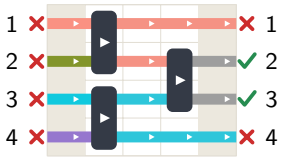
```
\usepackage{beltmap}
...
\begin{beltgrid}[labels, expect=balanced]
  > S> > \\
  > S> >
\end{beltgrid}
```



Each cell of the grid is one tile: `>` is a belt running right and `S>` is half of a splitter. The key `expect=balanced` makes LuaLaTeX stop with an error if the layout is not a balancer.

Four belts take more care. This layout has three splitters, and nothing from inputs 3 and 4 reaches output 1:

```
\begin{beltgrid}[labels, mix=check, marks]
  > S> > > > \\
  > S> > S> > \\
  > S> > S> > \\
  > S> > > >
\end{beltgrid}
\beltmapmatrix
```

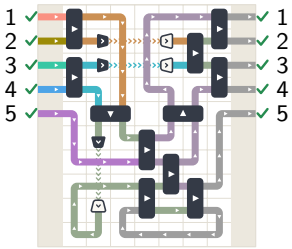


	output			
input	1	2	3	4
1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0
2	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	0
3	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$
4	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$

The table has a row per input and a column per output, and each entry is the exact share of that input on that output. Outputs 2 and 3 carry 1/4 of every input and get a check. Outputs 1 and 4 carry 1/2 of two inputs and none of the others and get a cross.

Larger layouts usually come from a file or from the game. `\beltlayout` reads a Factorio-SAT tile file or a saved blueprint, and `\beltblueprint` takes a blueprint string pasted into the document. This tile file holds a balancer of five belts:

```
\beltlayout[labels, mix=strict, marks,
  expect=balanced]{examples/f00_9x10.json}
```



To	Use	Section
draw a blueprint string copied from the game	`\beltblueprint`	3
draw a blueprint or Factorio-SAT file	`\beltlayout`	3
type a layout	`beltgrid`	3
change the look	`theme`, `tile`, `flow`	4
print each input's exact share of each output	`\beltmapmatrix`	5.1
color belts so that gray means balanced	`mix=strict`, `marks`	5.1
test whether lanes are kept or evened	`mix lanes`, `lane balance`	5.2
test throughput	`throughput`, `full load`	5.3
test balance with outputs blocked	`blocked`	5.4
compare two layouts	`compare`, `same network`	5.5
draw the network without its positions	`view=graph`, `view=wires`	6
get the blueprint back out of the PDF	`blueprint`, `\beltmapbook`	7

2 Reading a picture

Glyph	Meaning
	A belt. Every belt between two splitters has its own color, through turns and tunnels. Arrowheads give the direction.
	A splitter over two tiles; the arrow is its direction.
	Underground entrance: a solid hood over the half of the tile that faces the tunnel, narrowing toward it.
	Underground exit: the same hood, open, the belt leaving its mouth.
	The tunnel: chevrons in the belt's color, pointing the way items travel.
	Input priority: a triangle at the input edge on the preferred side.
	Output priority: a triangle at the output edge on the preferred side.
	Filter: a diamond where the filtered item leaves; the rest leaves on the other side. One diamond color per item.
	Belt tier: a rim on splitters and a bar over a hood's mouth, in the tier's color (yellow, red, blue, turbo green).
	Side-loading: a belt that ends against the side of another.

Left and right are those of somebody traveling with the belt. The shaded lines of tiles are the entry and exit lines, when the layout has them. `\beltglyph` sets one glyph in running text: `\beltglyph{entrance}` gives .

3 Putting a layout in the document

3.1 Files: `\beltlayout`

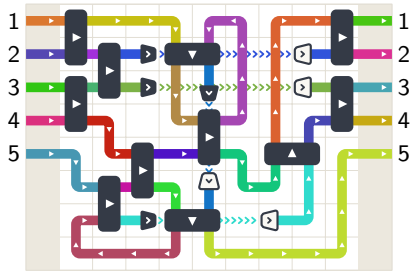
`\beltlayout[keys]{file}` reads, by the file's extension:

- `.json`: tiles in Factorio-SAT's format, the bare list `tiles[y][x]` or an object `{"tiles": ..., "gap": 4, "inputs": [...], "outputs": [...], "tier": ...}`;

- `.jsonl`: search records; the smallest record with status `sat`, or the one picked by `pick` (network file) and `record` (counted from 0);
- anything else: a blueprint string, a blueprint book, or the JSON inside one; `index` (from 0) or `label` (part of the label) picks a blueprint of a book.

Directions are numbered 8 ways in Factorio 1.1 and 16 ways in 2.0. The blueprint's `version` field decides, and `factorio=1.1` or `factorio=2.0` overrides it. A 2.0 splitter filter is read as its item name, with a note when its quality is not plain normal, as in *copper-ore (any quality)*, and written back as it was read.

```
\beltlayout[labels, flow=right]{examples/f04_11x8.txt}
```



3.2 Strings: `\beltblueprint`

Paste a blueprint string into the document. Line breaks and spaces inside it are ignored.

```
\beltblueprint[inputs={A,B}, outputs={left, right}]{0eNqtkMEKwjAQRH9F9rxKGxuh+RU
  RaXXVQJqEZCtK6b8bFUWsPYiewmZ2HrPTQW1a8kFbBtWBZmpAvfwhkGXNmiKoZXcfzmvbNjUFUDmCdzH
  ...}
```



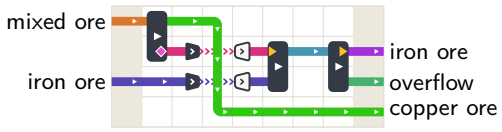
3.3 Typed grids: `beltgrid`

One cell per tile, cells separated by spaces, rows by `\\`. Keep a space between two `^` cells: TeX reads `^^` as a character code.

Cell	Tile	Cell	Tile
`> < ^ v`	belt	`S>`	a splitter tile, on both tiles
`I>`	underground entrance	`S>[in=left]`	with options, on either tile
`O>`	underground exit	`.` or `_`	empty

Splitter options are `in` and `out` (`left` or `right`) and `filter` (an item name); the filtered item leaves on the `out` side, left when `out` is not set. The tiles of a splitter facing right or left are stacked, those facing up or down sit side by side. A belt turns as in the game: when exactly one belt feeds it from a side and none from behind.

```
\begin{beltgrid}[inputs={mixed ore, iron ore},
    outputs={iron ore, overflow, copper ore}]
 >  S>[filter=iron-ore, out=right] > v .  .  ...\\
 .  S>  I> v O> S>[in=left] > S>[out=left] >  \\
 >  >   I> v O> S>          . S>           >  \\
 .  .   .  > >  >           > >            >
\end{beltgrid}
```



3.4 By hand: the TikZ library

`\usetikzlibrary{beltmap}` works without the package, in any engine. One unit is one tile and y points up; `(tile cs:x=3,y=5)` or `\bmtile{3}{5}` is the center of the tile in column 3, row 5, counted from the top left as in Factorio-SAT. The glyphs are pics whose argument names a direction and options: `bm head`, `bm entrance`, `bm exit`, and `bm splitter`, placed at the middle of its two tiles. Belts are paths with the style `bm belt=`*color*; `bmc0` to `bmc11` are the colors of the palette.

```
\begin{tikzpicture}[bm tile=6mm, bm theme=schematic]
  \path[bm belt=bmc0] (0,-0.5) -- (1.5,-0.5);
  \path[bm belt=bmc1] (0,-1.5) -- (1.5,-1.5);
  \pic at (1.5,-1) {bm splitter={right, out=left}};
  \path[bm belt=bmc2] (1.5,-0.5) -- (3,-0.5);
  \path[bm belt=bmc3] (1.5,-1.5) -- (2.5,-1.5);
  \pic at (2.5,-1.5) {bm entrance=right};
  \node[bm label, anchor=west] at (3,-0.5) {first};
  \draw[red, thick, <-] (2.5,-1.8) -- ++(0,-0.6)
    node[below, font=\scriptsize] {goes under};
\end{tikzpicture}
```



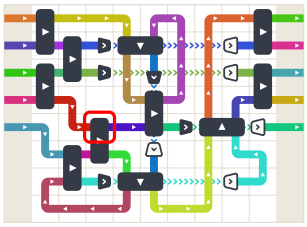
Pictures from the other front ends take TikZ code through the key `annotate`, drawn after the layout in the same coordinates:

```
\beltlayout[annotate={\draw[red, very thick, rounded corners=2pt]
    (tile cs:x=1,y=4) ++(-0.55,0.55) rectangle ++(1.1,-1.1);}]{...}
```

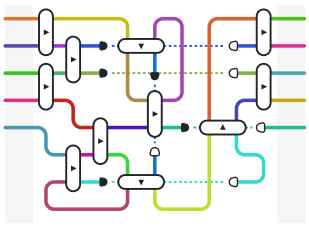
4 Changing the look

4.1 Themes

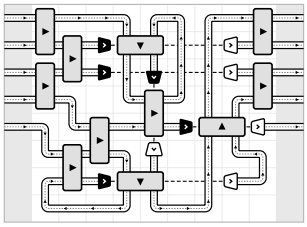
`theme` sets the look and leaves the drawing as it is. The same layout, `f05_11x8.txt`, in each:



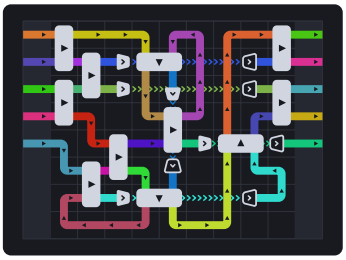
`schematic` (the default; with an `annotate` box)



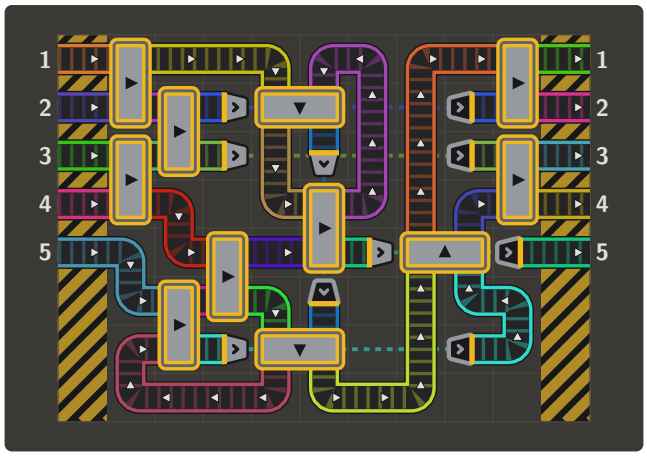
`transit`



`print`: black ink, both lanes of each belt



`dark`



`industrial`: after the feel of the game, drawn from scratch without its graphics

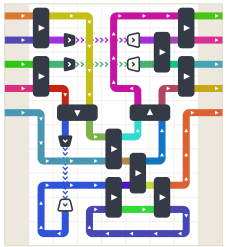
`industrial` draws each belt as a dark track with slats between two rails in the belt's color, splitters and underground belts as steel housings striped in their tier's color (yellow when the layout names none), and hazard stripes on the entry and exit lines.

4.2 Splitter options in four directions

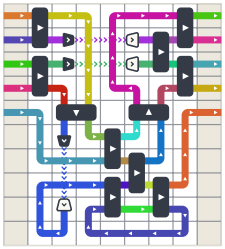
	in left	in right	out left	out right	filter left	filter right	in left, out right
right							
up							
left							
down							

4.3 Counting tiles

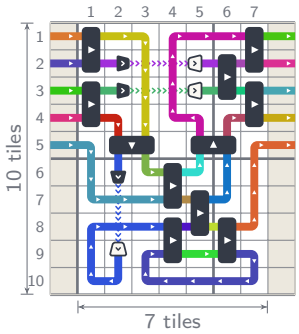
The size of a layout is counted inside its entry and exit lines, as the search records count it. `grid` sets how much the tile grid shows, from `faint` to `strong`, which also runs over the entry and exit lines. `grid major=5` draws a heavier line every five tiles from the inside corner, `rulers` numbers the inside tiles along the top and left (every one, or every fifth with `rulers=5`), and `dimensions` marks the inside size in tiles. `\beltmapcaption` gives the same size in words.



`grid=faint`

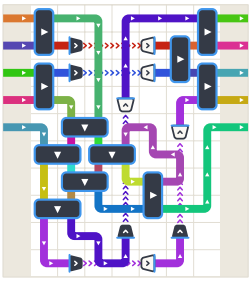


`grid=strong`

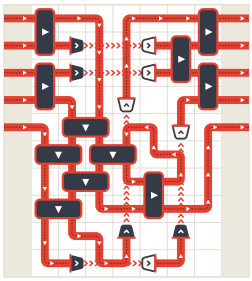


`grid=strong, grid major=5, rulers, dimensions`

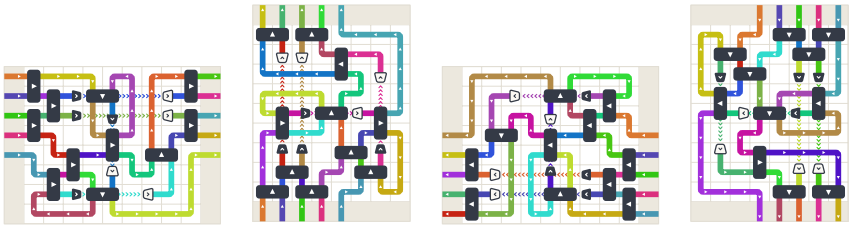
4.4 Tiers, lanes, and direction



`tier=blue`



`tier=red, color=tier, lanes`



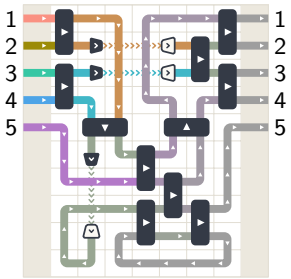
`flow=right` `up` `left` `down`

5 Checking a layout

Each test below sets a macro with its verdict after the picture, adds the verdict to the picture's alternative text, and can stop the document through `expect`.

Question	Key	Verdict in	`expect` values
Does every output carry $1/m$ of every input?	`mix`	`\beltmapverdict`	`balanced`, `unbalanced`
Does every input supply $1/n$ of the demand?	`marks`	`\beltmapinputs`	`input-balanced`, `input-unbalanced`
Are the two lanes kept apart?	`mix lanes`	`\beltmaplanes`	`lanes`
Are the two lanes evened?	`lane balance`	`\beltmaplanebalance`	`lane-balanced`, `lane-unbalanced`
Does any cut hold back flow?	`throughput`	`\beltmapthroughput`	`throughput`, `limited`
Do full belts get through?	`full load`	`\beltmapfullload`	`full-load`, `full-load-limited`
Does it balance with outputs blocked?	`blocked`	`\beltmapblocked`	`blocked`, `blocked-unbalanced`
Is it this other network?	`same network`	`\beltmapsame`	`same-network`, `different-network`

`expect` takes a list and computes what it checks, whether or not the key for that test is on. `\beltmapcaption` is a one-line summary of the last picture: inputs and outputs, the size between the entry and exit lines, splitters, tunnels, and every verdict the picture computed.



	output				
input	1	2	3	4	5
1	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
2	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
3	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
4	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
5	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$

5–5 · 7×10 · 11 splitters · 3 tunnels · balanced · passes max-flow

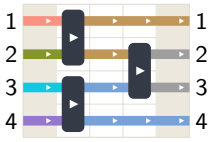
5.1 Balance

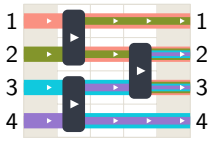
Each input carries one unit of an item of its own. A splitter sends half of what enters it out of each side, all of it to its output-priority side, or all of it to its one side that leads anywhere. A side-load adds to the belt it joins, and a belt that leads nowhere carries nothing. A loop fed only through output-priority sides fills up in the game, and its splitter then sends everything to the other side; beltmap finds such loops and counts them full. A layout is *balanced* when each of its m outputs carries $1/m$ of every input. The shares are exact fractions, and floating point only when a fraction outgrows 62 bits.

`mix` picks how the shares are colored:

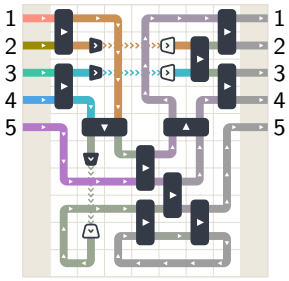
Value	Each belt is drawn	Gray means balanced?
`blend`	as the average of its input colors in OKLab	no
`bands`	as stripes, a pair per input, as wide as that input's share	no
`strict`	as the blend, with unbalanced belts kept a visible step from gray and outputs lighter or darker by amount	yes
`check`	gray, or the color of the input most in excess, at a strength that grows with the error	yes

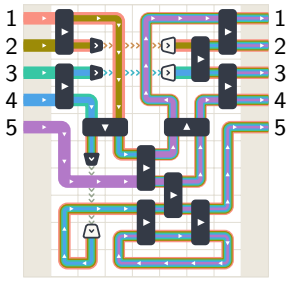
Gray `mix=blend` outputs do not prove balance. A color has three coordinates, so from 5 inputs up many uneven mixes blend to exactly the gray of the even one: a belt with 12.18%, 15.46%, 35.17%, 0%, and 37.19% of inputs 1 to 5 is that gray. Equal `mix=bands` stripes do not prove it either, because bands show shares and leave out amounts. In the last picture below, every output has the same stripes and output 1 carries twice as much as each of the others.



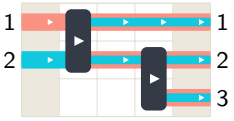


`mix` and `mix=bands`: not balanced: output 1 carries 50% of input 1, not 25%



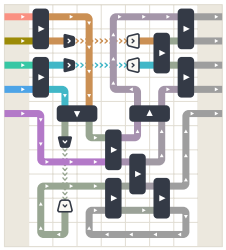


balanced: each of the 5 outputs carries 20% of each of the 5 inputs

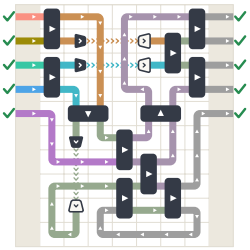


The same stripes on every output, but output 1 carries twice as much as each of the others.
not balanced: output 1 carries 50% of input 1, not 33.3%

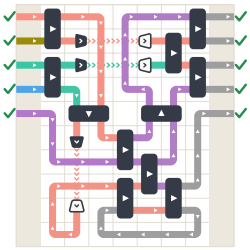
`marks` puts a check or a cross on every output and input. An output passes when it carries $1/m$ of every input. An input passes when it supplies $1/n$ of every output's demand, read backwards at full backpressure: every output asks for one unit, and a backed-up splitter or merge draws evenly from the belts that feed it, or all of it from its input-priority side. For splitters that each take two belts in and send two out, with no priorities, one balance implies the other. Priorities break that, as in the last picture below.



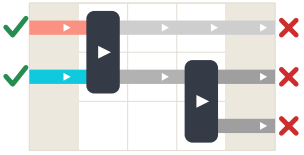
`mix=blend`



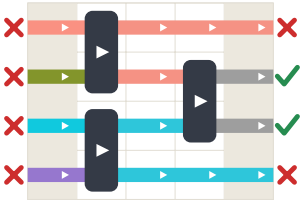
`mix=strict, marks`



`mix=check, marks`



input balanced, outputs not: output 1 is lighter, 2 and 3 darker

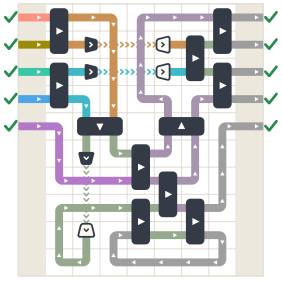


neither: outputs 2 and 3 pass, the rest fail

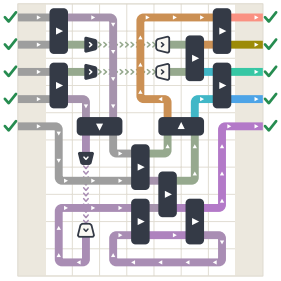


input priority: outputs pass, inputs fail

`demand` colors the belts by that demand: each output gets a color, and a belt is drawn in the colors of the outputs whose demand passes through it. It works with `mix=blend`, `strict`, and `check`.



supply: gray where the inputs are mixed



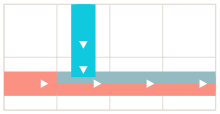
`demand`: gray where the outputs' demand is spread

`focus=`i shades each belt by how much of input i it carries, and the output labels give the share of input i on each output. `palette` picks the input colors: `wheel` (the default, hues evenly round a circle), `okabe-ito` (Okabe and Ito's colors for readers with color blindness), `lightness` (dark blue to light yellow), or `gray` (black to light gray, the default in `print`). Away from `wheel`, a balancer's outputs look alike, and their color is not gray.

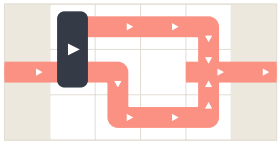
5.2 Lanes

A belt has two lanes. Belts, turns, underground belts, and splitters keep each lane. A side-load puts both lanes of the incoming belt onto the near lane of the belt it joins. Onto an underground belt only one incoming lane gets on, because the hood blocks the other: an entrance takes the upstream lane and an exit the downstream lane.

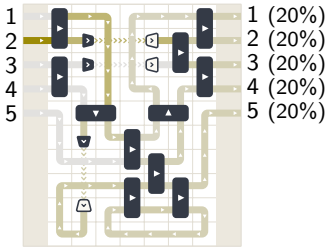
`mix lanes` colors the two lanes of each belt on their own. `\beltmaplanes` says whether lanes are *kept*: each output lane carries $1/m$ of each input lane on its own side and nothing of the other. `lane balance` tests the opposite property, that a layout *evens* its lanes: on every output, the left and right lanes carry the same amount of every input lane. Both are light-load verdicts.



`mix lanes`: a side-load



`lane balance`: a lane balancer



`focus=2`

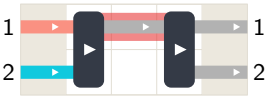
5.3 Throughput

A balancer can balance and still not pass full belts. beltmap has two tests, for every choice of open inputs I and outputs O, each asking whether $\min(|I|, |O|)$ belts get through.

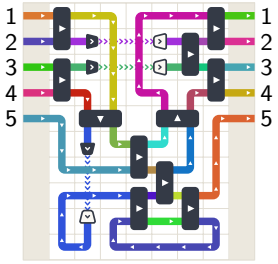
`throughput` is the max-flow test. Every lane has a capacity of one. A choice is *limited* when a *cut*, a set of lanes whose removal separates I from O, has less capacity than the choice needs; the picture shows the cut of the worst choice glowing under its belts. A limit it finds is a proof. Passing is necessary for full throughput and not sufficient, because a splitter does not route along a maximum flow. Choices with $|I| = |O|$ decide all the others, since flow only grows with I and O. Up to 8 inputs and 8 outputs, every such choice is tried. Beyond 20000 choices the test stops: a limit found is still a proof, and with none found the verdict is *undecided*, which `expect=throughput` refuses.

`full load` runs a fluid model of the belts. Each lane passes at most one unit per step and holds two. A straight belt goes before anything side-loading onto it. A splitter passes what its inputs offer and its outputs have room for, evenly on each side unless a priority says otherwise, and what one side cannot take goes to the other. Every choice is tried up to 4096 choices (6 inputs and 6 outputs). Beyond that the test tries as many as that, and with no shortfall the verdict is *undecided*.

The model was compared with Factorio 2.1.24 (base game) on 16 layouts and 1392 choices of inputs and outputs, each measured for two minutes of game time. It was never more than 0.001 belts below the game and at most 0.101 belts above it. All 240 choices it called short were short in the game, and each of the 1152 it passed reached at least 90% of the belts asked for. The measurements are in the repository, and CI compares the model with them.



balanced, but max-flow finds the cut



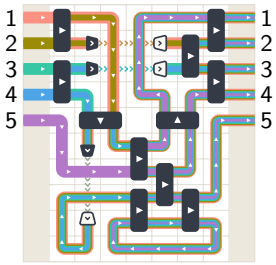
f00 passes max-flow. limited at full load: inputs 5 to outputs 5 pass 0.67 of 1 belt in the fluid model

5.4 Blocked outputs

A blocked output's belt is full, so the splitter that feeds it sends everything to its other side, and a belt that leads only into full belts fills too. `blocked` tries every nonempty set of outputs short of all of them, computes the shares again, and checks that every open output carries one over the number of

open outputs of every input. `\beltmapblocked` names the worst choice, with as few outputs blocked as possible. Blocked inputs need no test, because without belt capacity each input's flow does not depend on the others. Up to 10 outputs (1022 choices) are tested.

This is a light-load test. At full load a belt that cannot take more backs up into the splitters before it and shifts the flow.



f00, a balancer: not balanced with output 1 blocked: output 2 carries 40% of input 1, not 25% (18 of 30 choices fail, at light load)

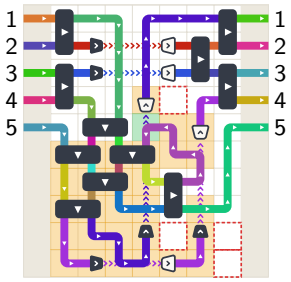


one splitter: stays balanced with any outputs blocked: every open output carries an even share of every input (2 choices, at light load)

5.5 Comparing two layouts

`same network=`*file* tests whether two layouts are one network: the same inputs, outputs, splitters, and belts between them, up to renaming. Priorities, filters, and side-loads count; positions, tiers, and mirror images do not. `compare=`*file* marks how the tiles differ: tiles the other layout lacks are shaded green, tiles that hold something else are shaded amber, and tiles only the other layout uses are outlined in red. The two are lined up by their top left entities, and `flow` turns both.

```
\beltlayout[compare=examples/f00_9x10.json,
  same network=examples/f00_9x10.json, expect=same-network]{examples/f01_9x10.json}
```

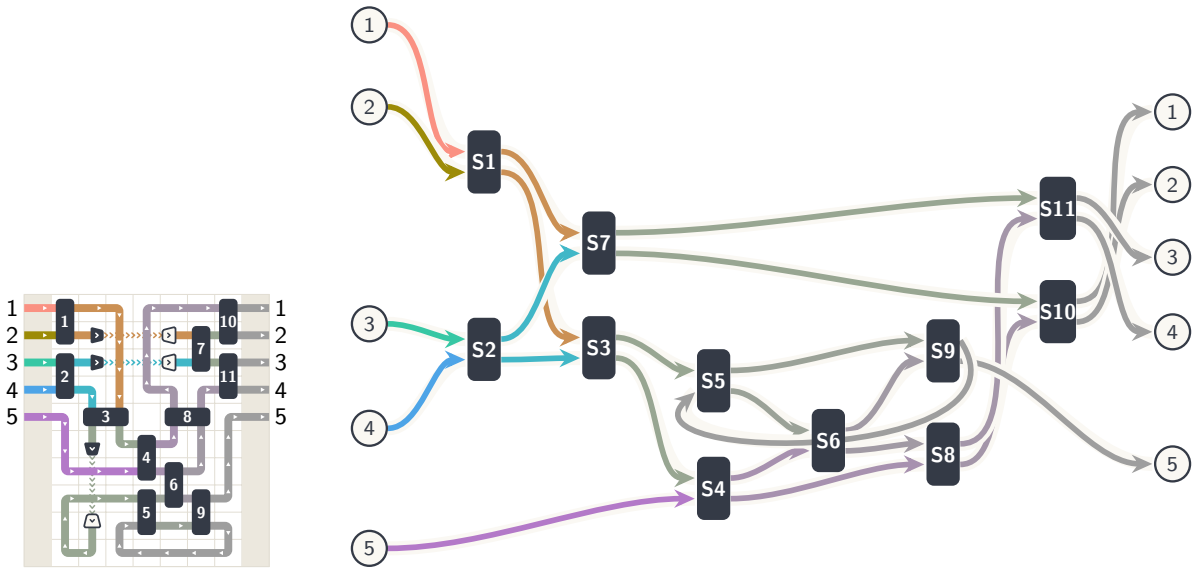


f01 compared with f00: 5–5 · 7×10 · 11 splitters · 5 tunnels · same network · +1 −4 ∼29 tiles

6 Pictures of the network

6.1 The splitter graph

`view=graph` draws the network with none of the layout's positions. Each input, splitter, and output is a node, and so is every point where a belt side-loads. Each piece of belt is an arrow, in the color the layout gives it. Inputs stand in a column on the left and outputs on the right. Each splitter takes the column of its longest path from the inputs, and a force layout sets the heights. `numbers` writes each splitter's number on the layout in place of its arrow, the number its node carries in the graph.

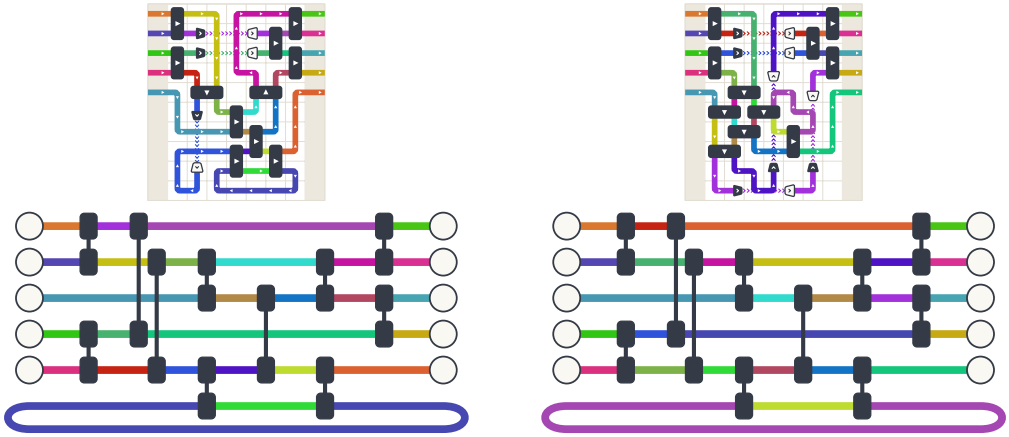


6.2 The wire diagram

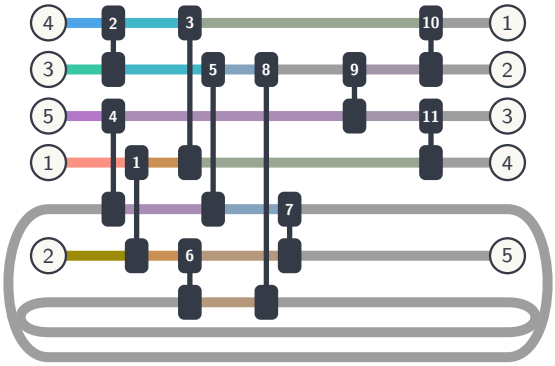
`view=wires` draws the network in the notation of balancing networks: each belt is a horizontal wire, and a splitter is a bar between the two wires it joins. A belt that closes a loop leaves on the right, runs back underneath the diagram, and comes in again on the left.

The picture depends only on the network: two layouts with the same splitter network get the same picture, whatever their positions, numbering, or input order. Loops are cut at every smallest set of belts that leaves no cycle, each splitter takes the column of its longest path from the inputs, and every placement of inputs and loops on the wires is tried. The drawing kept has the least total length of bars, then its loop wires lowest, then the least code, a description of the bars and wires that names no splitter or input.

`labels` and `numbers` say which belt of this layout is which, so they differ between layouts of one network. `mix` colors the wires by what they carry.



two layouts, one network, one wire diagram



`view=wires, mix, labels, numbers`: two loops

The search covers networks whose splitters each take two belts in and send two out, with no side-loading, up to 7 wires (inputs and loops together) and 24 splitters; a 5-to-5 balancer takes a second or two. For any other network the log gets a warning and the picture is drawn as `view=graph`. Priorities and filters are not drawn, so `same network` is the test to use when they matter.

7 Blueprints

After each picture, `\beltmapstring` holds the blueprint string of the layout as drawn (turned by `flow`, tiers given by `tier`), and `\beltmapprintstring[width]` sets it in small type that may break after any character. The string uses Factorio 2.0's numbering unless `factorio=1.1` is given or the layout was read from a 1.1 string. Tiles without a tier become plain (yellow) entities.

`blueprint=file` also writes the string to `\jobname-blueprint-`n`.txt`, beside the PDF, and `blueprint` also attaches it to the PDF. Copy from the attachment or the file: text copied from the page may carry line breaks. `blueprint label` sets the label.

A blueprint stores no belt turns, since the game works them out from the neighbors. Each string is read back as soon as it is written, and a layout whose turns would come out differently in the game draws a warning.

`\beltmapbook[label]` collects every blueprint written with `blueprint=file` or `blueprint` into one blueprint book. At the end of the document the book goes to `\jobname-book.txt` and into the PDF's attachments.



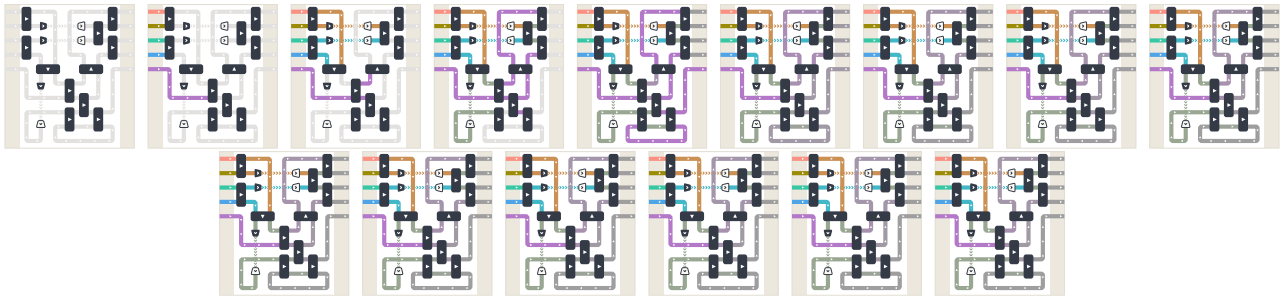
```
0eNqVUtFqwzAM/JWhZ280bl2IH/cbY5Sk1TqBIxtbGQ0h/z412RiMQpsng+50d5I8Qht6
TJlYwI+ALCSEBfzbCCfKeBSKDH5nFmg4cN+1mMFXBrjpEDxIbrikmOW5xSBgIMVCS9sIF
/CbF2dgmN9pMndk7cOydo3sdnXa6hFZrRCnXg66wJgVUYOAH1fZH7uSAoko10Ds5R810/
nzRoTqN8L9AG71uua53jX3MfJyZuITXuaDFjpzE64Nt0WXRsFOob9vYyAOCmvtVUlPoRl
0Uq1+YS6zv9vbelfXzlm339rNNH0Di1LRXzEu
```

8 Galleries, animations, and export

`\beltgallery[keys]{file.jsonl}` draws every record with status `sat` of a search file: `columns` cells to a row (3 by default), each with its network file, size and area, and `\beltmapcaption`. `pick` keeps one network file and `limit` the first so many records; the other keys apply to every cell.

`\beltanimation[keys]{file}` draws one picture per step of the flow. The first has empty belts, each step moves the inputs' colors one piece of belt on, and the last is the steady state. `step=`*k* draws a single step. The script `beltmap_export.py` makes files for use outside LaTeX: a PNG (`--png`, at `--dpi`), an SVG with text as paths (`--svg`), or an animated GIF of the steps (`--gif`).

```
python beltmap_export.py examples/f00_9x10.json --keys "mix, labels" --gif f00.gif
```



9 Game icons

beltmap contains no game graphics. It reads icons from your own copy of Factorio when the document is compiled. `\beltmapicon[width]{item}` sets an item's icon in running text, `1.2em` wide unless a width is given. It works inside `inputs` and `outputs` labels, and a splitter with a filter shows the filtered item's icon over its diamond when the icon is found.

```
\beltmapset{factorio data=C:/Program Files (x86)/Steam/steamapps/common/Factorio/data}
\begin{beltgrid}[inputs={\beltmapicon{iron-ore} iron, \beltmapicon{copper-ore} copper}]
```

`factorio data` names the game's `data` folder, the one that holds `base`. Without it beltmap tries `$FACTORIO_DATA`, then the usual Steam folders on Windows, Linux, and macOS, and the log names the folder it used. Setting the environment variable `BELTMAP_NO_ICON_SEARCH` to `1` skips the usual folders. Item names are the game's internal names: `iron-ore`, `electronic-circuit`, `turbo-splitter`. With no data folder, an unknown name, or `icons=false`, a placeholder with the item's initials takes the icon's place, as everywhere in this manual: IO EC.

A PDF compiled with icons contains Wube Software's artwork. Check the game's terms before you publish one, or set `icons=false`.

10 Accessibility and speed

In a tagged document (`\DocumentMetadata{tagging=on, ...}`) every picture is a figure with alternative text. Without `alt`, the text is generated from the trace, for instance: *Belt layout of 9 by 10 tiles with 11 splitters and 3 underground tunnels; 5 inputs enter on the left and 5 outputs leave on the right.* It also gives the verdict of every test the picture ran, and `alt` replaces it.

The key `cache` keeps the results of `throughput`, `blocked`, and `full load` in `\jobname.beltmap-cache` between runs. An entry is tied to the layout and the file to the version of the code that computed it. The file may be deleted at any time.

11 Reference

11.1 Commands

Command	Effect
`\beltlayout [keys]{file}`	draw a layout from a file
`\beltblueprint [keys]{string}`	draw a blueprint string
`beltgrid` environment	draw a layout typed as a grid
`\beltglyph [keys]{name}`	one glyph in running text
`\beltgallery [keys]{file.jsonl}`	every record of a search file
`\beltanimation [keys]{file}`	one picture per step of the flow
`\beltmapset {keys}`	set keys for the rest of the group
`\beltmapicon [width]{item}`	a game icon in running text
`\beltmapbook [label]`	collect the written blueprints into a book
`\beltmapmatrix`	the last picture's shares, as a table
`\beltmapprintstring [width]`	the last picture's blueprint string, set in type
`\beltmapstring`	the last picture's blueprint string
`\beltmapcaption`	a one-line summary of the last picture
`\beltmapverdict`, `\beltmapinputs`, `\beltmaplanes`, `\beltmaplanebalance`, `\beltmapthroughput`, `\beltmapfullload`, `\beltmapblocked`, `\beltmapsame`	the last picture's verdicts (section 5)

11.2 Keys for drawing

Keys go in the optional argument of each command, in `\beltmapset{...}` for the rest of the group, or as package options.

Key	Default	Effect
`theme`	`schematic`	`schematic`, `dark`, `transit`, `print`, or `industrial`
`tile`	`4.4mm`	size of one tile
`labels`	`false`	number the inputs and outputs
`inputs`, `outputs`		labels, in order from top to bottom or left to right; braces protect a comma
`flow`	as drawn	`right`, `up`, `left`, or `down`: turn the layout to run that way
`tier`		`yellow`, `red`, `blue`, or `turbo` for tiles that name none; turns the trim on
`trim`	`auto`	show tier colors on splitters and hoods; `auto` shows them when a tier is given or tiers are mixed, and always in `industrial`
`color`	`belt`	`belt`: one color per belt; `tier`: the color of its tier
`palette`	`wheel`	`wheel`, `okabe-ito`, `lightness`, or `gray` (the default in `print`)
`view`	`layout`	`graph`: the splitter graph; `wires`: the wire diagram
`numbers`	`false`	splitter numbers on the layout, as in the graph
`lanes`	`false`	a line between the two lanes of each belt (always on in `print`)
`grid`	`true`	the tile grid: `faint`, `true`, `strong`, or `false`
`grid major`		a heavier line every n tiles, counted from the inside corner
`rulers`		number the inside tiles along the top and left; `rulers=5`: every fifth
`dimensions`	`false`	the size of the inside in tiles, below and to the left
`ends`, `heads`	`true`	the shaded entry and exit lines, the arrowheads
`openends`	`true`	the stub of belt from a splitter's edge on a belt that runs straight to an output
`icons`	`true`	game icons from your copy of Factorio, or placeholders with `false`
`factorio data`	found	the game's `data` folder
`alt`	generated	alternative text in a tagged PDF
`annotate`		TikZ code drawn after the layout
`tikz`		options for the `tikzpicture`

11.3 Keys for reading and analysis

Key	Default	Effect
`mix`	`none`	`blend` (alone, `mix` means this), `bands`, `strict`, or `check`
`marks`	`false`	a check or a cross on every input and output
`demand`	`false`	color the belts by the demand of the outputs
`mix lanes`	`false`	color the two lanes of each belt on their own
`focus`		an input number: shade each belt by how much of that input it carries
`step`		one step of the flow, as in `\beltanimation`
`expect`		a list of verdicts: stop when the layout is not so (section 5)
`throughput`	`false`	the max-flow test, with the cut marked
`full load`	`false`	the fluid model of full belts
`blocked`	`false`	the balance with every set of outputs blocked
`lane balance`	`false`	whether every output's two lanes carry the same
`same network`		a layout file: whether this is the same network
`compare`		a layout file: mark the tiles that differ from it
`cache`	`false`	keep the exhaustive tests' results between runs
`gap`	file, else 10	longest tunnel for tiles without a tier
`pick`, `record`		which search record of a `.jsonl` file
`index`, `label`		which blueprint of a book
`factorio`		`1.1` or `2.0`: direction numbering, read and written
`columns`, `limit`	3, all	cells per row and number of records of `\beltgallery`
`blueprint`	`none`	`file`: write the string beside the PDF; `attach` (alone, `blueprint` means this): also attach it
`blueprint label`	read, else *Belt layout*	the label of the written blueprint

12 Installing

beltmap is in TeX Live and MiKTEX. By hand, put `beltmap.sty`, `beltmap.lua`, `beltmap-net.lua`, and `tikzlibrarybeltmap.code.tex` beside the document, or in `tex/latex/beltmap` of a local `texmf` tree. It needs LuaLaTeX with a kernel of 2024-06-01 or later, TikZ, and `embedfile` in documents without `\DocumentMetadata`.

Report bugs at `https://github.com/landonfox00/beltmap/issues`.

13 Known limitations

- LuaLaTeX only: the package stops with a message under pdfLaTeX or XeLaTeX. The TikZ library alone runs anywhere.
- Blueprints were tested in Factorio 2.1.24 only. Strings in 1.1 numbering went through that version's import, not into Factorio 1.1. Written in 1.1 numbering, a filter loses its quality.
- Entities other than belts, underground belts, and splitters are left out, and counted in the log.
- The shares, `blocked`, `lane balance`, and `mix` have no belt capacity, so they describe light load. Under full load in the game, each output of f00 carries mostly one input.

- Every test treats all items as one item: filters are ignored, except by `same network`.
- `full load` is a fluid model. It runs at most 0.101 belts above the game on the layouts measured, because real items leave gaps that a fluid does not.
- `throughput` is necessary for full throughput and not sufficient. It stops at 20000 choices and `full load` at 4096; past those, a run with no shortfall is *undecided*.
- Input balance is judged at full backpressure. A belt that side-loads onto a full belt waits for gaps in it, so the verdict for a layout with side-loads is an approximation.
- `view=wires` needs every splitter to take two belts in and send two out, with no side-loads, at most 7 wires and 24 splitters; otherwise it draws `view=graph`. It does not draw priorities or filters.
- `same network` compares belts and ignores which lane a side-load lands on.
- Labels of a layout that runs up or down overlap when longer than a tile is wide.
- In `transit`, belts carry no arrowheads; direction shows on splitters and portals only.