Troubleshooting: when a real circuit doesn't lay out cleanly

v0.72.103.244

Péter Felecan

assisted by Claude Sonnet 5.5 (Anthropic)

October 18, 2026

Contents

1 Purpose of this document **2**

2 Situation 1: placement fails outright **2**
2.1 Read the failure messages themselves first 3
2.2 Try the board first, before touching the circuit 3
2.3 Look at –debug next, before assuming the circuit itself is the problem . . . 3
2.4 Check whether a component's own real size was ever actually measured . . 4
2.5 If a component is being treated as the wrong kind of thing 4

3 Situation 2: it places, but crowded **5**
3.1 Give it more real room, the same way as above 5
3.2 Be genuinely careful with anchor_min_pins – a documented, real cautionary example exists 5

4 Situation 3: it places cleanly, but the layout itself isn't good **5**
4.1 This is an honest, currently real limitation, not a bug to report 5
4.2 What's actually worth trying, and what generally isn't 6
4.3 Reading a –debug reason trail directly is often the fastest way to understand a specific, individual placement decision 6

5 Situation 4: ppresolve_components stops, a component library file is not found **6**

6 If none of this resolves it **7**

List of Listings

1 The connectivity check after a render 2
2 Asking pprender for the computed board 3
3 The message when every tier of the escalation fails 3
4 Rendering on a taller board from a configuration file 3
5 Asking pprender for its debug report 3
6 The debug report: parts relocated for room 4
7 The debug report: sizing fallbacks 4
8 Fetching the component libraries of the test designs 6

1 Purpose of this document

Workflow.pdf shows the pipeline working end to end, on circuits already known to place cleanly. This document is for the other case: a real circuit that doesn't – placement fails outright, the result is technically valid but crammed, or the diagram is correct but clearly not a good one to actually build from. Three real, distinct situations, three different approaches – covered in turn below, each grounded in this project's own real, current diagnostic output, not invented advice.

If a problem doesn't fit cleanly into any of these three, or persists after following the relevant section, see FutureResearch.pdf for this project's own currently open, known limitations before assuming a new bug – and see LibraryReference.pdf's own auto_layout.py section for what each function mentioned below actually does.

2 Situation 1: placement fails outright

`pprender` reports one or more real failures and, honestly, skips rendering rather than risk drawing something invalid (see Listing 1):

Listing 1: The connectivity check after a render

```
connectivity check: 0 failures
board-bounds check: 3 failures
  Net-(D1-K): wire row 5 at column 42 (N) is out of range, ROWS_PER_HALF=5
  PMOS_GATE: wire row 5 at column 31 (N) is out of range, ROWS_PER_HALF=5
  VSW_OUT: wire row 5 at column 32 (S) is out of range, ROWS_PER_HALF=5

Skipping render: at least one check above failed, and rendering an
invalid layout risks an unhandled crash rather than a clean report.
```

2.1 Read the failure messages themselves first

Each one already names the real, specific cause – a component or a wire, a column, and which limit it exceeded. ”Board-bounds” failures (as above) mean something ran out of real vertical room; ”connectivity” failures mean a net’s own members never actually ended up connected in the layout. These are different problems with different real fixes (below) – don’t reach for the same fix for both.

2.2 Try the board first, before touching the circuit

The single most common real cause is simply too little room (see Listing 2):

Listing 2: Asking pprender for the computed board

```
$ pprender --use-computed your-circuit.kicad_sch -o out.svg
```

`--use-computed` escalates the same real board shape’s own sector count automatically (`compute_optimal_config()`) until the circuit fits, or reports, honestly, that it tried every tier up to its own real ceiling and none worked – in which case every attempt’s own real failure is printed, not just the last one (see Listing 3):

Listing 3: The message when every tier of the escalation fails

```
pprender: compute_optimal_config: exhausted every tier (8 tried, up
to sectors=8) without a clean placement.

--- sectors=1 ---
Net-(D1-K): wire row 5 at column 42 (N) is out of range, ROWS_PER_HALF=5
...
--- sectors=8 ---
Net-(D1-K): wire row 5 at column 42 (N) is out of range, ROWS_PER_HALF=5
```

If the SAME real failure repeats at every tier, adding sectors genuinely isn’t the fix – more sectors add HORIZONTAL room; a board-bounds failure is a VERTICAL (rows_per_half) limit, which no number of sectors can relieve on its own (see breadboard_config(5) for the real difference). Try a real config with a larger `rows_per_half` instead (see Listing 4):

Listing 4: Rendering on a taller board from a configuration file

```
$ pprender --config a-taller-board.conf your-circuit.kicad_sch -o out.svg
```

2.3 Look at –debug next, before assuming the circuit itself is the problem

Listing 5 asks for the debug report:

Listing 5: Asking pprender for its debug report

```
$ pprender --debug your-circuit.kicad_sch -o out.svg
```

This prints, for every real component, which column and row it landed on, which net put it there, and – important – whether it had to be RELOCATED for room (see Listing 6):

Listing 6: The debug report: parts relocated for room

```
C2: col 13 row 1 (net Net-(U1-VREG_2V7) shares with U1.23 (col 10)
    -- relocated to col 13 for room (col 10 overflowed rows_per_half=5))
D2: col 52 row 1 (net VOUT_4V2 shares with C4.2 (col 53)
    -- relocated to col 52 for room (col 53 had no space left for 3 rows))
```

A component with no "relocated" note landed exactly where its own first, natural choice put it; one that DOES have this note is already a sign that column got crowded – several of these clustered together is a real, direct signal of where the circuit itself is dense, useful even before anything actually fails.

2.4 Check whether a component's own real size was ever actually measured

Still in the same `--debug` output, a real, separate section (see Listing 7):

Listing 7: The debug report: sizing fallbacks

```
sizing fallbacks (real footprint data not used -- see real_size_rows()'s
own diagnostics for the full account):
  U1 (STUSB4500QTR): no category matched this value
```

This means U1's own real physical size was never actually measured – its value string didn't match any of this project's own known component categories (resistor, capacitor, diode, inductor, trimmer, mosfet), so a rough, generic estimate was used instead, which can be meaningfully wrong for a genuinely different real part (a connector, an IC, a breakout module). If a component near a real failure appears here, that's worth checking BEFORE assuming the algorithm itself is at fault – an inaccurate size estimate can be the actual, root cause of a downstream overflow. There is currently no way to correct this per-component from the command line (see LibraryReference.pdf's own real_size_rows() section for `category_footprint`, its own programmatic override, and Concept.pdf for how to reach packpay's own Python API directly if this matters for a specific, real circuit).

2.5 If a component is being treated as the wrong kind of thing

`classify()` decides ANCHOR vs. PASSIVE by real pin count alone (`anchor_min_pins`, default 3) – a genuinely 2-pin part that should conceptually anchor its own corner of the board (a large connector, a relay) can end up folded in as an ordinary passive instead, changing where and how it's placed. This is a real, available knob, but see the very next section before reaching for it – it can make a technically-passing result meaningfully worse to actually look at.

3 Situation 2: it places, but crowded

Not a failure – `pprender` reports 0/0 and writes a real SVG – but the result is visually dense: many components sharing few columns, numerous relocation notes in `--debug` output, wires doubling back.

3.1 Give it more real room, the same way as above

The most direct fix: a taller board (larger `rows_per_half`), more sectors, or splitting the circuit across a genuine, real, multi-board assembly (breadboard_config(5)'s own `[multiboard]` section) if one physical board is honestly too small for what's being built. Crowding is very often simply "not enough real physical space for this many real components," not a placement defect.

3.2 Be genuinely careful with anchor_min_pins – a documented, real cautionary example exists

Lowering `anchor_min_pins` (reclassifying more parts as anchors) can spread a tightly-packed circuit out – but this project's own real field experience (dudhya's own report; full account in ResearchJournal.pdf's own 2026-09-06 entry) found that an aggressive change here (every 2-pin passive reclassified as its own anchor) DID reach a clean, zero-violation bounds check, while the actual, rendered result was visibly WORSE: many small, sprawling anchors and long, crossing wires – a real diagram nobody would actually want to build from. The bounds/connectivity check passing is necessary, but not sufficient, evidence that a layout is genuinely good – always look at the real, rendered SVG itself before trusting a parameter change, not just its own pass/fail result.

4 Situation 3: it places cleanly, but the layout itself isn't good

No failures, no obvious crowding, but the diagram is still not something a person would want to actually build from – long wires where a short one would do, a component landing far from the rest of what it connects to, more sectors used than seem necessary.

4.1 This is an honest, currently real limitation, not a bug to report

packpay's own placement is a real, greedy, heuristic algorithm – room-search and relief actively try to relocate a crowded component and add bridging wires where column-sharing alone doesn't unify a net, but there is no real backtracking or broader search behind any of it (FutureResearch.pdf's own "Backtracking… for anchor sector assignment" item covers

this precisely, along with the real criteria under which building one would become worth attempting). A locally reasonable decision, made early, can turn out non-optimal once the rest of the circuit is placed around it, with nothing currently able to revisit it afterward.

4.2 What's actually worth trying, and what generally isn't

Reordering how anchors connect to each other rarely helps directly – `order_anchors_by_connectivity()` already places electrically- related anchors near each other by design. What's more often worth trying: a different `anchor_spacing` (more or less deliberate room between anchors up front, rather than left to be discovered during room-search later), or, per the previous section, simply confirming the real board itself is a reasonable real size for this real circuit – a layout that looks sprawling on a board far larger than the circuit needs is a real board-choice problem, not an algorithm one.

4.3 Reading a –debug reason trail directly is often the fastest way to understand a specific, individual placement decision

Every component's own line in `--debug` output names exactly which real net put it where it is, and against which other, specific component – tracing this by hand for the one or two components that look obviously out of place is often faster, and more conclusive, than experimenting with global tuning parameters that affect the entire circuit at once.

5 Situation 4: ppresolve_components stops, a component library file is not found

`ppresolve_components(1)` reads the symbol of each part from the file that its sources file names. A relative path in that file is looked for in the component library directories, and a file that is not there stops the run with one line that says which file, every place it was looked for, whether `PACKPAY_COMPONENT_LIBS` was set, and what to do (a KiCad variable such as `${KICAD_SYM_DIR}` that is not set is named too). Two causes cover nearly every case: the third-party libraries that the project's own test designs use are not on this machine, or `PACKPAY_COMPONENT_LIBS` points somewhere else. Listing 8 puts the libraries in a directory; point `PACKPAY_COMPONENT_LIBS` at it, or give it to `./configure` (`--with-component-libs=DIR`), whose directory is the default of the script and of `make`.

Listing 8: Fetching the component libraries of the test designs

```
$ fetch-component-libs --dir $HOME/component_libs
```

`fetch-component-libs(1)` needs git and a network. With no option it fetches what is missing; `--update` checks what is there against its upstream and updates only what is behind; `--refresh` fetches everything again. After `./bootstrap` and `./configure`, `./fetch-component-libs --update` uses the directory that configure was given.

6 If none of this resolves it

Check FutureResearch.pdf directly – several of this project's own currently open items are exactly this kind of case (dense, adversarial, or otherwise unusual real circuits), already investigated and precisely scoped, not simply unnoticed. If a real circuit exposes something that genuinely isn't already covered there, it's worth reporting – see PackpayFieldReport-dudhya.org (this project's own root directory) for a real, worked example of what a useful field report from an external, real user looks like.