Benchmarks and Framework for PCB Layout Automation

ABSTRACT

The wide gap between PCB layout research and commercial practice limits the usefulness of our community's research contributions to industry. PCB layout researchers can emulate the steps taken by the VLSI community to solve this problem: specifically, open benchmarks and open and automated workflows. VLSI workflows are available that can take a design from a netlist to layout with minimal human interaction. These workflows enable the evaluation of a contribution in context, for example, in evaluating whether a change in layout methodology improves routability. They also facilitate research contributions, for example, allowing experimentation with routing heuristics without coding a router from scratch.

We have implemented and are releasing a PCB layout framework, which has the unique combination of being fully-automated and open-source. Ours is the first PCB layout framework that can provide the same benefits to PCB layout research as existing VLSI workflows provide in that domain. We also propose a method for objectively evaluating PCB layouts: a set of benchmarks, a proposal for adopting a standard open benchmark format, and a list of evaluation metrics. The PCB benchmarks are based on real designs, ranging from 19 to 57 components and from 15 to 99 nets. We offer an initial evaluation of our framework using our proposed evaluation method to show that our framework produces layouts comparable to manual layouts. We believe that these contributions will help close the gap between PCB layout research and commercial practice, supporting the production and evaluation of industry-relevant PCB layout research.

CCS CONCEPTS

• **Hardware PCB design and layout**; *Software tools for EDA*.

KEYWORDS

PCB, layout, routing, placement, benchmarks, framework

ACM Reference Format:

. 2020. Benchmarks and Framework for PCB Layout Automation. In *ICCAD '20: International Conference On Computer Aided Design, November 02–05, 2020, San Diego, CA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

ICCAD '20, November 02–05, 2020, San Diego, CA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Automated printed circuit board (PCB) layout has been of interest to the design automation community for decades [7, 8, 18]. Despite such interest, this area of research has fallen far behind current commercial tools. Prima facie, PCB layout is tantalizingly automatable due to its easy-to-grasp problem domain of placing and connecting points in two dimensions, and its apparent similarity to several classic computer science problems, including packing and pathfinding. In practice, complex design rules and special cases that are often never formalized lead to a quagmire of complications that remain obscure to those without extended, practical PCB layout experience. A 2010 survey by Yan and Wong [21] showed that a "huge" amount of research has been devoted to the PCB routing problem. The significant projects addressed in Yan and Wong's survey operate in simplified problem regimes, however, and it is unclear whether they could be integrated into a real PCB layout flow.

For example, many of the problem formulations assume single-layer designs. This is never the case for the high-end designs that might make use of the routing techniques surveyed. The vastly reduced complexity of single-layer routing compared to multi-layer routing makes it unclear if the techniques can ever be applied to real layout problems. The simplification also hides many other nuances, such as the large vias that typify the PCB routing problem. In essence, there is a "reality gap" between current academic work and the practice of designing PCBs.

If a practical, fully-automated PCB layout flow can be achieved, the benefits would be numerous and far-ranging, including the potential for new embedded-system design interfaces, new pedagogical methodologies, and vastly reduced prototype turnaround time.

Our discussions with PCB layout engineers while preparing this work indicate some distrust towards automation techniques in their field. Often cited is a belief that automated tools cannot produce manufacturable design and that a human will inevitably need to rework the automated tools' outputs. Robust, realistic research is needed to overcome this distrust.

For PCB research to be robust, open-source research tools are needed that can produce manufacturable PCB layout. In addition to the availability of the source code, a standard set of metrics and benchmarks must be agreed upon to allow reproducibility and objective measurement of results.

We summarize the key contributions of our work as:

(1) The implementation of an open-source framework for PCB layout automation, including the release of our open-source placer, router, and design database.
(2) A set of metrics useful for evaluating the quality of a low-speed PCB layout.
(3) Eleven real PCB designs for evaluating PCB layout automation, each with manual layout for comparison.
(4) The evaluation of our placer and router with an existing open-source router to demonstrate an end-to-end flow.

The remainder of this paper is organized as follows: Section 2 relates this work to other work in the field; Section 3 compares PCB layout to the similar field of VLSI layout; Section 4 details our open-source PCB layout framework; Sections 5 and 6 briefly discuss the PCB-relevant aspects of our layout algorithms; Section 7 discusses metrics for evaluating PCB layout; Section 8 discusses the preparation and technical details of the benchmark designs we are releasing; Section 9 explains our choice of the KiCad PCB layout format as a standard format for PCB benchmarks; Section 10 presents our evaluation of automated and manual layout for the benchmark designs; Section 11 proposes future directions for our framework and PCB benchmarks.

2 RELATION TO PREVIOUS WORK

The dearth of fully-automated PCB layout tools limits computational design research for embedded systems. Tools like Trigger-Action-Circuits [5], EDG Methodology [17], TinyLink [9] and Geppetto [3] provide novel ways of generating circuit netlists from high-level descriptions, with the goal of enabling the creation of embedded devices by non-experts. Without a fully-automated PCB layout flow, however, these processes cannot fully realize their goal. Geppetto, for example, must offload track routing to human engineers "behind the scenes," increasing both cost and turnaround time.

For researchers who *have* cobbled together fully-automated PCB flows, the restrictions –availability of appropriate licences, lack of exposed tuning parameters, difficulties with automating and interfacing, no ability to extend capabilities– of closed-source commercial tools limit the complexity of designs that are possible. The tools Echidna [15] and JITPCB [17] use Autodesk's EAGLE autorouter to create a fully-automated layout flow. Those researchers chose EAGLE because EAGLE's plain-text file format creates the possibility of an automated interface. The proprietary format restricts extension through a license agreement, however, and because EAGLE is closed-source, the router cannot be modified or extended.

The OpenROAD [4] project has the ambitious goal of creating a no-human-in-the-loop VLSI flow with a target of $500 for a one-off IC prototype. If the project is successful, the design of the host PCB and package will account for the majority of the remaining hardware costs.

The creation of viable PCBs at "the push of a button," while a major task, would enable a plethora of research and commercial projects. Our work in developing benchmarks and a framework for fully-automated PCB layout is a fundamental step towards achieving that goal.

3 COMPARISON TO VLSI LAYOUT

The previous work on practically applied layout algorithms for PCBs may be sparse, but similar flows exist for VLSI. We have identified key differences in both placement and routing that prevent the direct application of VLSI tools for PCB layout.

Differences between PCB and VLSI placement. As an initial experiment, we applied the state of the art analytical IC placers [6, 14] to the PCB placement problem and identified several issues that prevented convergence to good solutions.

(1) The vanilla RePlace implementation fails to successfully find good rotations for individual components.
(2) The smoothness of the cost and convergence of the solution is significantly dependent on design-dependent parameters, such as filler cells and boundary anchor weights.
(3) RePlace is natively incapable of dealing with non-rectilinear geometries.

The failure of RePlace to find acceptable placement for PCB layouts led us to identify these key difference between PCB and VLSI placement:

(1) Scale: PCB designs contain orders of magnitude fewer components than VLSI. The number of components in a PCB placement problem range from dozens in small designs to thousands in very large designs, whereas VLSI designs may contain billions of components.
(2) Shape of components: PCB components may have complex polygonal shapes, as opposed to the rectilinear cells of IC design. The shape of any single component is a set of arbitrary polygons that may be spatially disconnected or non-convex.
(3) Two-sided and 3D placement: PCB components may be placed on either the top of the bottom of the PCB. When a component is placed on the bottom of the PCB, the component geometry (outline, pin location) is mirrored. In some cases, components may overlap. For example, small components may be placed underneath the overhangs created by daughter-cards. While some PCB tools are able to encode 3D component geometry, we are unaware of any PCB layout tool that assists with 3D component placement.
(4) Rotation of components: PCB components may be freely rotated during placement, and 45-degree rotations are not uncommon in PCB layouts. Human designers generally resist creating designs with odd angles (other than 45-degrees), ostensibly for aesthetic or legibility purposes.

Our work leverages the fact that PCBs are composed of far fewer components compared to VLSI. Our placer uses stochastic combinatorial optimization which, although typically incapable of placing hundreds of thousands of cells, we successfully apply it to PCB designs. Such generic global optimization techniques have been demonstrated to perform well on the PCB placement problem. For example, Jain and Gea[11] demonstrated the efficacy of genetic algorithms on layout problems involving multiple objectives. In this work, we apply simulated annealing[13] to jointly optimize wirelength, overlap/density, and routability.

Differences between PCB and VLSI routing. Similar to the placement problem, there are several key differences that prevent the use of standard VLSI routing tools for PCBs:

(1) Large vias: Vias can be much wider than the routing trace width. Through-hole type vias penetrate the whole PCB, blocking routing on all layers. Laser vias are smaller than through-hole (generally still larger than trace width) and are significantly more expensive.
(2) Free angle routing: Because of the large and expensive vias, PCB traces are commonly routed at odd angles through the conductor layers to limit vias use. Traces are not restricted

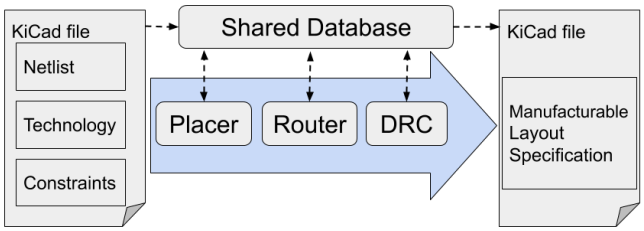

Figure 1: Framework and flow diagram. The dashed arrows represent passing layout information. The large blue arrow shows the flow from incomplete layout to completed layout.

to a layer's "preferred direction." Commonly, PCB traces are routed using "eight-direction routing," or multiples of 45 degrees.

(3) Layer count: Routing layer count should be minimized because the number of conductor layers is the most significant contributor to the cost of PCB manufacture.

Canonical approaches for routing PCBs include Mazeroute and variants [16]. Topological-based routing has emerged as a flexible alternative, able to handle a variety of cases where Maze-routing fails [20]. Our router adopts a grid-based cost-driven rip-up and reroute router.

In conclusion, the differences between VLSI and PCB have led us to create a new placer, router, and design database focused on the issues unique to PCB layout, rather than attempting to modify an existing IC flow.

4 AN AUTOMATED OPEN-SOURCE FLEXIBLE PCB LAYOUT FRAMEWORK

Our framework is unique in being both open-source and fully-automated. The numerous benefits of open-source are critical to our project and its impact. These benefits have been thoroughly discussed in the context of EDA [10]. A push-button flow enhances the applicability of such a framework, freeing researchers from details that are irrelevant to their particular work, and allowing different parts of the layout process to be isolated and tested across multiple tools in the context of an entire flow [12].

An overview of our framework is illustrated in Figure 1. We are releasing the following modules as part of our framework:

(1) shared design database
(2) placer
(3) autorouter
(4) design rule checker (DRC)

The framework is highly flexible, allowing for each stage – placement, routing, and design rule checking– of the flow to be interchanged between several options as detailed in Sections 4.2, 4.3 and 4.4. Included with this work, we release a fully-automated, open-source tool for each stage. Alternatively, each stage can be completed using other external tools with varying degrees of automation. A scripting API facilitates the creation of additional stages and modification to the flow. Each of the framework components that we have implemented, including the database, exposes both a C++ and Python API. Iteration between stages is scriptable, if desired, and manual placement, manual routing, or design inspection can each be inserted at any stage.

4.1 Database

Our shared design database is a utility for importing and exporting PCB layout designs and exposes an API that enables easy reading and writing of PCB layout information in both C++ and Python. API documentation is available in the project repository. Our database can also interface with other tools (for example, extant placers and routers) using a file-based interface.

The database can read and write to the KiCad PCB layout format (`.kicad_pcb`), a plain-text, fully open-source PCB layout format. See Section 9 for a discussion of our selection of this file format.

Additionally, a partial, intermediate layout can be written out at any stage. These partial layout files can be imported by KiCad and manually inspected or modified using the KiCad GUI. The manually modified files can be reloaded by the database and further processed by the automated tools.

Upon completion of the layout, Gerber files can be generated from KiCad PCB files for multiple manufactures. PCB manufacturers accept Gerber files as a standard manufacturing specification.

4.2 Placer

Our framework may use any combination of the following:

(1) Our placer: We use a simulated annealing (SA) algorithm to find a placement that is optimized with regards to the sum of an overlap penalization term, unrouted air-wire lengths, and a routing congestion prediction metric[19]. Our placer can be used as part of a full-automated, push-button flow. See Sections 5 for algorithm details and Section 10 for evaluation.
(2) KiCad autoplacer: KiCad has a utility for spreading components on the PCB. KiCad's autoplacer attempts to minimize air-wire length, with a preference for compacting components into the lower-right corner of the board. KiCad's autoplacer requires use of the KiCad GUI.
(3) Other external tools: Our framework can be used with any other placement tool compatible with the KiCad file types or capable of being modified to interface with our database API.
(4) Manual placement: Manual placement can be performed by placing and locking components in the design through the KiCad GUI. The design can then be exported to or imported from the KiCad GUI at any stage.

Placement constraints. PCB layout generally starts with a partial placement for off-board interconnects and user interface components. These components are "locked" and may not be moved by the automated placer. Interface components like USB connectors, pin headers, power plugs, reset buttons are commonly locked.

Components must be placed within a board outline, the geometry of which is given as a set of line segments and arcs that form a continuous outline of the PCB's perimeter. All component pads and pins must be placed completely within the PCB's perimeter.

4.3 Router

Our framework may use any combination of the following:

(1) Our autorouter: Our own autorouter uses a grid-based A*-search and cost-based rip-up and reroute. Following the general practice of VLSI routing, our autorouter prioritizes layout-vs-schematic (LvS) completion over design rule violation (DRV) cleanness, differentiating it from existing PCB autorouters, which refuse to route nets unless a DRV-clean solution can be found. Our autorouter can be used as part of a full-automated, push-button flow. Details of the algorithm can be found in Section 6. Our autorouter is evaluated in Section 10.
(2) FreeRouting: Multiple stand-alone autorouters are compatible with KiCad, the most prominent being FreeRouting, which uses a topological routing algorithm. FreeRouting only performs DRV-clean routing, and will not output routes that violate the design rules. This is because FreeRouting assumes active human interaction and verification. FreeRouting can be run in an automated way, but requires file conversion through KiCad. FreeRouting is open-source software, but does not appear to be actively maintained. We evaluate FreeRouting in Section 10.
(3) Other stand-alone autorouters: Any autorouter can be used with our flow provided that it is either compatible with KiCad files or that it can be modified to interface with our database's API.
(4) Manual routing: Routing can be performed manually using the KiCad GUI, or using any graphical layout program that is compatible with KiCad.

4.4 Design Rule Checker

We have implemented an automated design rule checker (DRC) that can interact with our database. KiCad also has an integrated DRC, but it has been designed to be used within the KiCad GUI and is difficult to script and modify.

4.5 Public availability

We offer all parts of our framework with source code and the BSD 3 license[1]. The framework source code is available on GitHub[2]. We have attempted to set up this work to be a maintainable open-source project. Continuous integration tests run through the Jenkins framework and API documentation is available in the GitHub repository. An install script is available for Ubuntu.

5 SIMULATED ANNEALING FOR PCB PLACEMENT

Our placer uses a simulated annealing strategy with a mixed objective function. Simulated annealing is a standard algorithm [13]; this section briefly outlines some of the issues specific to PCB placement.

Our objective function penalizes the sum of half-perimeter wire length (HPWL) over all nets, a routing congestion metric[19], and a component overlap score. We use the Boost.Polygon library for calculating component overlap. At the beginning of annealing, the objective function places 100% of weight on HPWL and congestion and none on overlap. As annealing progresses, the objective function is reweighted, placing more weight on overlap, proportional to the remaining annealing iterations. Annealing ends with overlap weighted at 100%. To maintain the layout solution in a low HPWL-congestion space, the move size is reduced as HPWL-congestion weight is decreased. At the end of annealing, the allowed component move radius (in Manhattan space) is one millimeter.

We have experimented with several formulas for penalizing overlap. Our best results have been achieved using the following three-term measure: Let $o_{a,b}$ be the area of overlap between the courtyard polygons of components (in mm^2) a and b, $d_{a,b}$ be the distance between the geometric centers of the courtyard polygons of components a and b, and $k_{a,b}$ be the indicator function $k_{a,b} = 1$ if $o_{a,b} > 0$; else 0.

$$overlap\ penalty = \sum_{i,j} o_{i,j}^2 + o_{i,j} + \frac{k_{i,j}}{1 + d_{i,j}}$$

Penalizing the overlap quadratically encourages small overlap between all the components. The linear term produces better behavior when overlap between two components is less than 1 mm^2 (when the edges are barely overlapping). Also, when the allowed move size is small, the last term has the effect of "pushing" small components towards the edge of larger components in the case that their courtyard polygon is wholly within the larger components.

Components are also allowed to rotate during placement. We fix the ratio of rotation moves at 25%. We reject all moves that would place a component outside the board outline.

6 A* SEARCH FOR PCB ROUTING

Our router uses an iterative cost-based rip-up and reroute strategy based on A* search. This section The vector-based PCB design is rasterized onto a grid. The pads of each component are rasterized and represent obstacles during routing. Each net is then routed using minimum-cost graph search. When a point-to-point connection is made, a cost is added to each grid-cell that the new trace occupies. For multi-pin nets, the search frontier is initialized as the set of grid-cells occupied by previously routed connections for that net. After every net has been routed to minimum cost, nets are ripped up (their costs removed from the routing grid) and rerouted, one by one. Ripping up and rerouting all nets once represents one iteration of our router algorithm. The results for our router, presented in Section 10, were routed with 20 iterations.

Because the trace-width is wider than the grid resolution (typically by a factor of 6 or more) we use a partial cost update method for the search-frontier, based on a rasterized circle whose radius is the trace width. That is, when we expand into a new cell, we only add and subtract the cost of the grid-cells at the edge of the circle. Large vias cause a considerable slow-down for this algorithm. They require a much larger cost update (their radius is larger than the trace width) and they require cost lookups on all layers. Allowing only laser vias (which are smaller than through-hole vias and only penetrate one layer) results in an approximate 4x speedup

[1] The BSD 3 license is a permissive open-source license.
[2] https://github.com/ICCAD2020-Submission271/PCB-Layout-Framework

over through-hole vias. To handle 45-degree routing we allow traces to expand into their 45-degree corner neighbors with an increased cost of factor $\sqrt{2}$. This requires the use of floating-point numbers to store cost.

Adding support for high-speed nets and other advanced constraints is ongoing.

Our discussions with layout engineers have led us to penalize trace bending. The layout engineers that examined our results commented that the result looked "cleaner" and they were more accepting of the automated layouts after this change. However, they acknowledged that the bends would not affect electrical correctness.

7 METRICS

Metrics are standards used to objectively evaluate layout quality and quantitative comparison between systems without relying on aesthetics. We identified design-rule violations (DRVs), through-hole via count, laser via count, and laser via layer count as the key metrics because they are direct outputs of the layout flow and each affects either design correctness or design cost.

7.1 DRVs

In the context of routing, DRVs include pad-to-trace, pad-to-via, trace-to-trace, trace-to-via, and via-to-via clearance violations.

In the context of placement, DRVs include pad-to-pad clearance violations, overlap of component outlines, and off-board placement (placement of components completely or partially outside of the board outline).

Numerous other advanced design rules affect PCB layout quality, including but not limited to power delivery, thermal dissipation, differential pair routing quality, bus length matching, signal integrity, pad entry, high-speed net topology, and decoupling capacitor placement. Commercial layout tools have mixed support for these advanced rules. To the best of our knowledge, no commercial layout tool is able to encode and check for all of the advanced design rules required of a modern, high-speed PCB design. These tools typically rely on "app notes" written and checked by human engineers. KiCad, specifically, does not currently support advanced constraints, but support is in the road map for Version 6, the next major version.

The majority of our benchmark designs do not require advanced design rules. The few that do did not have these parameters formally specified in their design files.

Our framework, which supports simple design rules, is a necessary first step towards implementing a future framework that supports more advanced rules. We are currently adding support for some advanced rules to our placer and router (see Section 11).

7.2 LvS errors

An LvS error occurs when a connection that appears in a design's netlist does not have a corresponding electrical connection after routing. Our experience is that power and ground nets are most likely to be subject to a LvS error due to their relatively large number of connections. Our router prioritizes LvS connectivity over DRV cleanness and never produces LvS errors, but all other extant autorouters will produce LvS errors if they cannot find a DRV clean solution.

7.3 Via count

Vias are conductive connections between conductive layers and can be either through-hole or laser. Per-piece board cost increases with each through-hole via in a design. Through-hole vias penetrate and connect every layer of a PCB; laser vias connect only two adjacent layers and can be made smaller than through-hole vias. Our benchmarks include design rules that specify minimum via sizes.

Through-hole via count equals the number of through-hole vias used in a layout.

Minimum through-hole via diameter increases with the number of conductor layers due to the difficulty of plating vias with tall aspect ratios in thick PCBs. Exact minimum through-hole via diameter varies between manufacturers, but our benchmarks contain per-net minimum through-hole via sizes as design rules.

Laser via count equals the number of laser vias used in a layout and should be minimized because they are a common failure point in PCBs due to thermal and mechanical stress.

Laser via layer count equals the number of adjacent layer pairs connected by laser vias. For example, if three adjacent conductor layers (with two layers of dielectric) are connected by laser vias, the laser via layer count is two.

Each pair of layers connected by laser vias increases the manufacturing cost by approximately 40% due to process cost and yield losses.

7.4 Other metrics

We report two other metrics to differentiate the tools evaluated in this work:

CPU time is the total run time of a layout program. The scale of run times reported (less than one hour) have little effect on cost, but run time may have more significant consequences for designs more complex than those discussed in this work.

Total trace length is the combined linear length of all traces in a routed design. This metric does not affect design correctness or cost, but may indicate general "competence" of a router in avoiding detours.

8 PCB BENCHMARKS

In this work, benchmarks serve as standard problems against which to measure the performance of competing tools. (N.B. This differs from the use of the word benchmark to indicate searching the industry for best practices[1].) We present a collection of 11 PCB designs to serve as benchmarks for automated layout[3].

These designs were selected because they contain a variety of complexity levels, are small enough to be completed in a reasonable time using our automated framework, and are simple enough that they do not contain many advanced constraints, such as trace-length matching or differential pair coupling. Each benchmark design is presented with a full manual placement and routing with the intent of providing a "known good" point of comparison for automated layouts. The benchmarks presented

[3] https://github.com/ICCAD2020-Submission271/PCBBenchmarks

Table 1: Design characteristics of PCB benchmarks.

design	#components	#nets	#pins	#locked	design area
bm1	60	99	319	18	4890.44
bm2	19	34	77	4	874.01
bm3	58	80	229	10	1358.78
bm4	48	54	163	7	1432.23
bm5	34	38	138	4	1220.87
bm6	28	52	140	6	1143.52
bm7	8	15	40	1	193.01
bm8	36	70	188	4	4342.17
bm9	61	63	314	4	5520.65
bm10	58	35	233	6	2920.12
bm11	46	69	207	7	1176.24

#pins includes unconnected pins. #locked is the number of fixed location components. Design area is the area of the smallest rectangle that covers the board outline, in square millimeters.

were prepared from layouts that have been manufactured and functionally tested. Further details on each design, including license information, are available in the GitHub repository.

8.1 Summary of benchmark designs

For each benchmark design we present a short summary of the features of the design. The designs and some of their characteristics are listed in Table 1. Additional information on each design is available in the design repository.

bm1. Based on a well known microcontroller (MCU) carrier board (Arduino Mega), bm1 contains a high pincount MCU.

bm2. Based on the Adalogger FeatherWing, a commercial design, bm2 contains two large, oddly shaped components: an SD card slot, and a coin cell battery. The remaining area available for placement is fully enclosed by these two components and interface pins along the sides.

bm3. A quadcopter controller board, bm3 contains an antenna and four motor driver circuits to power the propellers.

bm4. An audio processor breakout board, bm4 contains a headphone jack and a MCU. This MCU has fewer pins than the MCU in bm1, however, this MCU has a large ground pad that prevents routing through the MCU's footprint.

bm5. A USB to lithium battery charger, bm5 contains several different connector components and the components of the charging circuit have unusual footprints.

bm6. An MCU carrier board, bm6 has a very narrow aspect ratio. We have found that narrow PCBs are difficult to place with simulated annealing. Random moves often go outside of the board boundary and components can become stuck in local minima as the movement step size decreases during cooling.

bm7. An accelerometer breakout board, bm7 is useful as a sanity check and can be automatically completed in less than one minute. It can be routed using only one layer.

bm8. An MCU carrier board, bm8 only contains through-hole components and does not contain any surface mount components (SMDs).

bm9. A remote control board, bm9 contains many locked user-interface components including buttons, switches, and LEDs. It is also a four-layer design.

bm10. Another quadcopter design, bm10 contains larger motor driver components and four routing layers.

bm11. Based on the Adafruit Feather nRF52 Pro Bluetooth LE development board, bm11 contains a high pincount Bluetooth modem and a very tight placement area.

8.2 Benchmark preparation

Human-generated PCB designs lack the regularity that would make them suitable for automatic layout. Human designers are inconsistent in their use of the layout format features. We took care, therefore, to process each design file to conform to a set of stylistic standards. We also added missing information that was not explicitly encoded in the original design files.

We regularized each design with the following steps:

(1) Designs not originally in the KiCad layout format were converted to that format;
(2) Images, text, and other non-functional copper elements were removed;
(3) Copper pour areas (also known as copper fill or zones) were removed, and nets that relied on copper pour for connectivity were manually routed;
(4) Silk-screen-only components and other non-copper components were removed;
(5) Silk-screen outlines and designator text were replaced with polygons in the front and back courtyard layers (CrtYd.F and CrtYd.B). These courtyard polygon serve as component outlines for placement purposes;
(6) Board outlines were simplified in some cases and made continuous;
(7) Connectors and interface components were marked as locked as appropriate;
(8) Net classes were added to specify trace width;
(9) Components, traces, and design rules were moved or modified so that the KiCad DRC reports zero errors.

9 OPEN-SOURCE PCB LAYOUT FORMAT

The IC community has standardized on LEF/DEF as an open layout format, enabling research and commercial tools to be interoperable. Interoperability allows research tools to be inserted into, and even replace parts of, commercial tools chains. Also, LEF/DEF can be freely extended by anyone to add advanced features as the needs of IC design evolve. PCB layout research would also realize significant benefit from a standard layout format.

We propose the KiCad PCB layout format (`.kicad_pcb`) as a preferred standard format for PCB layout benchmarks due to:

(1) an active development community,
(2) the openness of the format, and
(3) its availability as a widely compatible, open-source tool.

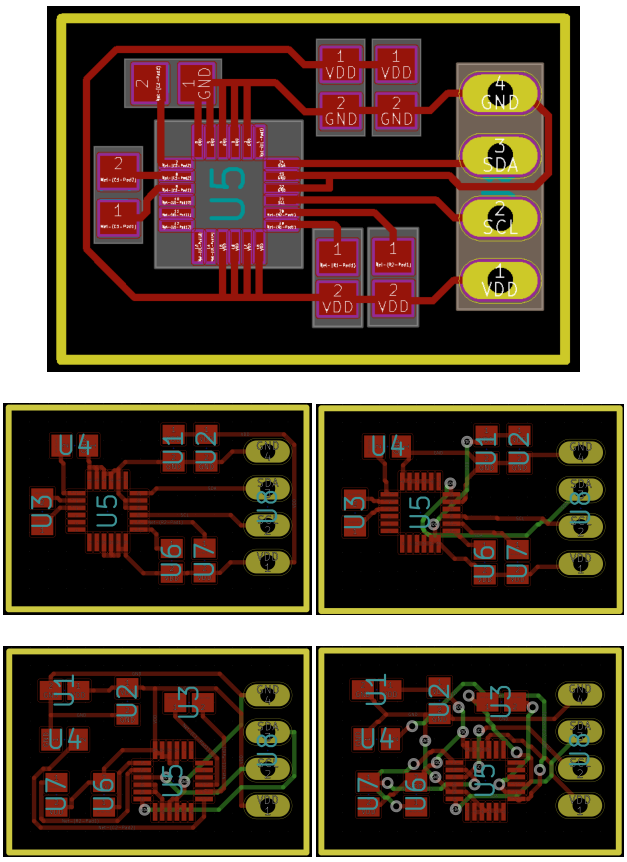

Figure 2: bm7 layouts. Manual placement and route (top); manual placement and FreeRouting (middle left); manual placement and our router (middle right); SA placer and FreeRouting (bottom left); SA placement and our router (bottom right).

KiCad is a full EDA suite for PCB, including schematic capture, manual layout tools, 3D board modeling, and CAM generation. Created in 1992, KiCad is an open-source tool for PCB layout with a very active development community that provides regular updates. The Git repository shows dozens of updates and bug fixes per week[4]. The European Organization for Nuclear Research (CERN) is an active contributor and recommends its use for open hardware projects[2].

Unlike commercial PCB layout tools, which each have their own proprietary, often encoded file format, the KiCad layout format is plain-text and can be freely extended. KiCad format documentation is licensed under Creative Commons Attribution License version 3.0.

10 LAYOUT RESULTS

We present comparative PCB layout results for each benchmark design in Tables 2 (via counts), 3 (total trace length), and 4 (CPU time) using the metrics for five different combinations of placers and routers:

(1) manual placement and routing,
(2) manual placement with FreeRouting (FR) autorouter,
(3) our placer with FreeRouting autorouter,
(4) manual placement with our autorouter, and
(5) a fully automatic layout using our framework.

[4]https://gitlab.com/kicad/code/kicad

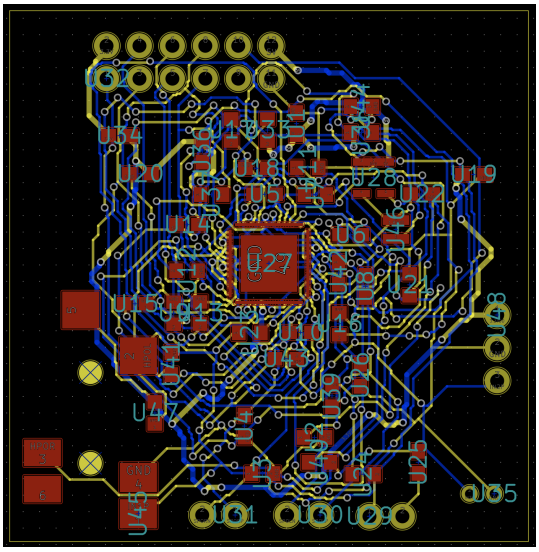

Figure 3: Layout of benchmark design bm4 using our placer and router. This figure shows a design of medium complexity and a layout result generated without any human input.

Figure 2 shows bm7 routed with each combination, for visual comparison. Figure 3 shows a larger design (bm4) with fully automated layout.

We modified the basic implementation of FreeRouting to allow FreeRouting to be run in "batch mode" without a GUI. Even so, evaluating the output of FreeRouting requires manually integrating the output with the original design using the KiCad GUI. We used an automated script to collect via and trace-length information after manual conversion.

Via count. Via counts for each placer and router combination are presented in Table 2. Manual routing uses through-hole vias; our router and FreeRouting use laser vias. FreeRouting out-performs the manual result for most designs, perhaps because the human designers were trying to optimize for legibility over via count. Our router uses 11.2 times more vias when routing bm10. In this layout, the configuration of the two high pin count components causes our router to take many layer-changing detours.

Our router's solution for bm8's automatic placement uses 41 times more vias than FreeRouting. In this placement the large through-hole MCU and headers are placed to cut across the middle of the PCB, causing a routing blockage. Our placer also compacts the components in this design, leaving too-little room for routing between them; this is a serious issue for through-hole components because their pins block routing on all layers.

Our router uses 6.5 times more vias to route, the automatic placement of bm11. This layout also contains approximately 120

Table 2: Layout results by via count for each benchmark.

Router: / **Placer:**	manual manual	FR manual	FR ours	ours manual	ours ours
bm1	129	*	*	280	331
bm2	4	4	6	11	18
bm3	44	*	*	73	97
bm4	37	*	*	49	61
bm5	33	*	*	27	37
bm6	30	6	13	47	68
bm7	0	0	2	10	11
bm8	0	0	1	3	41
bm9	166	*	*	128	128
bm10	37	25	27	280	36
bm11	75	*	*	48	310

DRVs. The high pin count components and small space make bm11 very challenging to place.

As our router uses A* search, via count is heavily dependent on a "layer change cost" parameter that penalizes expanding search to other layers while routing. Tuning the layer change cost may improve via count at the expense of run time. We did not experiment with tuning routing parameter in this work.

Total trace length. Routed wirelength for each placer and router combination are presented in Table 3. Manual layouts generally result in less total trace length compared to automatic placement. The automatic placements bm8 and bm11 have especially high trace length compared to the manual placement due to the layout issues that also affected via count. For bm8, the trade off between trace length and via count is evident. FreeRouting prefers to take more detours on the same layer, while our router prefers vias. This behavior is dependant on tuning parameters for both routers.

CPU time. CPU routing time for each router on each placement is presented in Table 4. The manual placement of bm9 has comparatively long pin-to-pin distances. This challenges our routers grid-based search algorithm. The topological routing algorithm of FreeRouting is able to find a solution comparatively quickly.

Our router ran with 20 iterations of rip-up and reroute. More iterations may improve metrics at the cost of CPU time. We terminated FreeRouting if it did not complete in less than 5 hours, but more routing time may have allowed FreeRouting to find a viable solution. FreeRouting often becomes "stuck" if it cannot find a DRV-free solution. Both FreeRouting and our router run single-threaded. Routing was run on Google Cloud Platform virtual instances (type n1-standard-16, Intel Haswell).

DRVs. Layouts generated with FreeRouting do not contain DRVs, but FreeRouting may not terminate if a DRV-free solution cannot be found. Layouts generated with our autorouter have complete LvS connectivity but contain clearance-related DRC errors (except for bm7, which has none).

We suspect that DRVs can be reduced by tuning algorithm parameters for each design but a full evaluation of these effects is beyond the scope of this work.

Table 3: Layout results by total trace length (mm) for each benchmark.

Router: / **Placer:**	manual manual	FR manual	FR ours	ours manual	ours ours
bm1	5034	*	*	5710	5326
bm2	395	323	503	432	591
bm3	1300	*	*	1380	1421
bm4	1043	*	*	1117	1054
bm5	681	*	*	578	600
bm6	778	834	1009	1021	1239
bm7	128	110	110	97	87
bm8	1483	1110	3960	1529	2633
bm9	3721	*	*	4043	3548
bm10	1743	1527	1396	2135	1980
bm11	1120	*	*	1216	2158

Table 4: Layout results by CPU routing time (s) for each benchmark.

Router: / **Placer:**	FR manual	FR ours	ours manual	ours ours
bm1	*	*	1275	1196
bm2	47	51	71	129
bm3	*	*	654	688
bm4	*	*	291	412
bm5	*	*	336	305
bm6	174	224	200	167
bm7	7	14	17	28
bm8	50	130	117	373
bm9	*	*	3574	66
bm10	537	620	2720	2601
bm11	*	*	251	315

* entries did not complete in less than 5 hours.

11 FUTURE AND CONTINUING WORK

Overall, our framework demonstrates significant progress in developing a completely automated end-to-end PCB layout flow. However, producing a placer and router that can rival manual layout is still a significant challenge for the future.

We have also identified several areas that no automated PCB layout tools (commercial or otherwise) address: copper fills, laser via layer count optimization, automated placement-routing iteration, design area optimization, 3D aware placement, and routing-aware placement. Each of these aspects of PCB layout is present in commercial PCBs but must be specified manually by human engineers, increasing cost and turn-around time. Our framework provides an entry point for research into automating solutions to commercially-relevant problems.

Our framework can also help demonstrate the value of the ongoing academic research into escape routing, differential pair routing, and bus length matching, by providing the context of a full layout flow and manufacturable result.

We will continue to augment our PCB benchmark suite with real, open-source designs, with a focus on designs that require advanced constraints. We hope that our work motivates others to release open benchmarks and tools for PCB layout.

REFERENCES

[1] [n.d.]. *Benchmarking Practices and Process for PCB Designers.* https://resources.altium.com/p/benchmarking-practices-and-process-pcb-designers Accessed: 2020-05-28.

[2] [n.d.]. *KiCad software gets the CERN treatment.* https://home.cern/news/news/computing/kicad-software-gets-cern-treatment Accessed: 2020-05-28.

[3] [n.d.]. *Meet Geppetto, IoT Embedded Hardware Design and Production.* https://www.gumstix.com/about-gumstix/ Accessed: 2020-05-28.

[4] Tutu Ajayi, Marina Neseem, Geraldo Pradipta, Sherief Reda, Mehdi Saligane, Sachin S. Sapatnekar, Carl Sechen, Mohamed Shalan, William Swartz, Lutong Wang, and et al. 2019. Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. *Proceedings of the 56th Annual Design Automation Conference 2019 on - DAC 19* (2019). https://doi.org/10.1145/3316781.3326334

[5] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits. *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST 17* (2017). https://doi.org/10.1145/3126594.3126637

[6] C. Cheng, A. B. Kahng, I. Kang, and L. Wang. 2018. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018), 1–1. https://doi.org/10.1109/TCAD.2018.2859220

[7] C. J. Fisk and D. D. Isett. 1965. "ACCEL" automated circuit card etching layout. *Proceedings of the SHARE design automation project on - DAC 65* (1965). https://doi.org/10.1145/800266.810762

[8] J Geyer. 1971. Connection Routing Algorithm for Printed Circuit Boards. *IEEE Transactions on Circuit Theory* (1971). https://doi.org/10.1109/TCT.1971.1083222

[9] Gaoyang Guan, Wei Dong, Yi Gao, Kaibo Fu, and Zhihao Cheng. 2017. TinyLink: A Holistic System for Rapid Development of IoT Applications. *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking - MobiCom 17* (2017). https://doi.org/10.1145/3117811.3117825

[10] Tsung-Wei Huang, Chun Xun Lin, Guannan Guo, and Martin D F Wong. 2019. Essential building blocks for creating an open-source EDA project. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019 (Proceedings - Design Automation Conference)*. Institute of Electrical and Electronics Engineers Inc., United States. https://doi.org/10.1145/3316781.3323477

[11] Sakait Jain and Hae Chang Gea. 1996. PCB Layout Design Using a Genetic Algorithm. *Journal of Electronic Packaging* 118, 1 (03 1996), 11–15. https://doi.org/10.1115/1.2792119 arXiv:https://asmedigitalcollection.asme.org/electronicpackaging/article-pdf/118/1/11/5645131/11_1.pdf

[12] Andrew B. Kahng, Hyein Lee, and Jiajia Li. 2014. Horizontal benchmark extension for improved assessment of physical CAD research. *Proceedings of the 24th edition of the great lakes symposium on VLSI - GLSVLSI 14* (2014). https://doi.org/10.1145/2591513.2591540

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671 arXiv:https://science.sciencemag.org/content/220/4598/671.full.pdf

[14] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *ACM Trans. Des. Autom. Electron. Syst.* 20, 2, Article 17 (March 2015), 34 pages. https://doi.org/10.1145/2699873

[15] Devon J. Merrill, Jorge Garza, and Steven Swanson. 2019. Echidna: Mixed-domain Computational Implementation via Decision Trees. *Proceedings of the ACM Symposium on Computational Fabrication - SCF 19* (2019). https://doi.org/10.1145/3328939.3329004

[16] M. M. Ozdal and M. D. F. Wong. 2006. Algorithmic study of single-layer bus routing for high-speed boards. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 3 (March 2006), 490–503. https://doi.org/10.1109/TCAD.2005.853685

[17] Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. 2017. Turning Coders into Makers: The Promise of Embedded Design Generation. *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication - SCF 17* (2017). https://doi.org/10.1145/3083157.3083159

[18] J. Soukup. 1978. Fast Maze Router. *15th Design Automation Conference* (1978). https://doi.org/10.1109/dac.1978.1585154

[19] P. Spindler and F. M. Johannes. 2007. Fast and Accurate Routing Demand Estimation for Efficient Routability-driven Placement. In *2007 Design, Automation Test in Europe Conference Exhibition*. 1–6.

[20] D. Staepelaere, J. Jue, T. Dayan, and W. W. . Dai. 1993. SURF: rubber-band routing system for multichip modules. *IEEE Design Test of Computers* 10, 4 (Dec 1993), 18–26. https://doi.org/10.1109/54.245960

[21] Tan Yan and Martin D. F. Wong. 2010. Recent research development in PCB layout. *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2010). https://doi.org/10.1109/iccad.2010.5654190