

Flying-Probe Testing: A Trajectory Planner and a Benchmark Suite

Original

Flying-Probe Testing: A Trajectory Planner and a Benchmark Suite / Calabrese, A., Quer, S., Squillero, G.. - In: IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. - ISSN 0278-0070. - STAMPA. - 44:12(2025), pp. 4807-4817. [10.1109/tcad.2025.3567012]

Availability:

This version is available at: 11583/2999987 since: 2025-05-09T05:28:44Z

Publisher:

IEEE

Published

DOI:10.1109/tcad.2025.3567012

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

IEEE postprint/Author's Accepted Manuscript

©2025 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

Flying-Probe Testing: A Trajectory Planner and a Benchmark Suite

Andrea Calabrese, Stefano Quer, *Senior, IEEE*, Giovanni Squillero, *Senior, IEEE*

Index Terms—Circuit Testing, Network theory (graphs), Parallel Algorithms, Testing.

Abstract—The in-circuit test checks whether the board’s electrical and electronic components have been correctly soldered when producing printed circuit boards. When such a test is performed using a flying-probe tester, the cost of testing is mainly related to the time required for moving probes over the board and the time necessary for defining such movements, tuning the optimization on the number of devices that will eventually be tested. Since the 2000s, flying probe testing has been gaining popularity. Still, despite its industrial relevance, the research has been impaired by the lack of publicly available benchmarks for testing the new algorithms and comparing the different ideas. This paper presents an open test set of realistic boards, ranging from a few thousand to half a million test points, together with a tool for generating more samples. It also presents an optimizer for flying probe tests composed of two separate planners: one global detecting test that could be performed together and reordered to obtain a more efficient probing sequence, and one local, implementing the probe movements and taking care of specific board features. The test set will eventually be used to present a quantitative evaluation of the performance of the proposed approach.

I. INTRODUCTION

Printed Circuit Boards (PCBs) are becoming highly complex, and state-of-the-art boards have significantly more components than a few years ago. On the one hand, wearables, smartphones, and IoT devices are pushing the boundaries of miniaturization driven by consumer demand for smaller, lighter, and more portable devices. Their PCBs are smaller and often have high complexity packed into a tiny footprint. In the 2010s, smartphones like the iPhone 6 typically had PCBs with around 1200 components. In contrast, modern smartphones like the iPhone 14 Pro have over 2000 components, reflecting a significant increase in complexity and interconnections. On the other hand, in 5G telecom infrastructure, data centers, and other industrial applications, large boards (up to one square meter) are more typical. These boards are not primarily driven by miniaturization but by the need for high-power processing, cooling, and dense routing for data transmission. This trend is evidenced by the proliferation of relevant industrial tools available [1]–[3], the established approaches dedicated to solving related problems [4], [5], and even research focusing on newer strategies employing vision systems or machine learning techniques [6]–[9].

PCBs require several expensive testing steps to eliminate possible fabrication faults [10]–[12]. Among those, PCB Assembly (PCBA) testing is a crucial part of ensuring the quality and functionality of electronic products. The main phases of PCBA testing include the following:

- Visual inspection to identify visible defects, such as misaligned components, soldering issues, or physical damage.
- In-circuit testing (ICT) to verify that individual components on the PCB are correctly placed, properly soldered, and functioning as expected.
- Functional testing to ensure the assembled PCB performs as intended under real-world operating conditions.
- Boundary Scan testing to test interconnections between integrated circuits without requiring physical probes.
- Burn-in testing tests the board under stress conditions, such as high temperatures and voltages, to ensure reliability and longevity.
- Environmental testing to validate the PCB’s durability in specific ecological conditions.
- Final Quality Assurance testing to ensure the PCB meets all specifications and is ready for deployment.

By progressing through these phases, manufacturers can identify and address issues at various stages, improving reliability and reducing costs associated with defects in the field. A significant trend in this area is that testing strategies are increasingly tailored to the application and board type rather than a one-size-fits-all approach. In bed-of-nails testing, each test point is reached by a specific probe, and all tests can be performed without interruptions. Bed-of-nails in-circuit testing is the preferred method for compact boards and simultaneously testing multiple identical boards. However, fixtures must be designed for each board, making the approach suitable only for significant production levels. In flying-probe testing [13], [14], probes move on the board, contacting sets of test points to apply the required electrical signal and observing the response. This approach is preferred for larger boards with thousands of test points typically spaced further apart than dense, small boards, making the flying probe testing a viable solution. Flying-probe testing eliminates the set-up cost but does not easily scale to test multiple units in parallel. Moreover, testing requires long sequences of tests, and it is slower. Consequently, producers are investing significantly in hardware and software, resulting in testers with increased probes and improved movement speed and accuracy.

Although ICT is widely used in manufacturing, it receives less attention in academic research than specialized techniques

A. Calabrese, S. Quer, G. Squillero are with the Department of Control and Computer Engineering (DAUIN), Politecnico di Torino, Turin, Italy.
Authors contributed equally and are listed in alphabetical order.

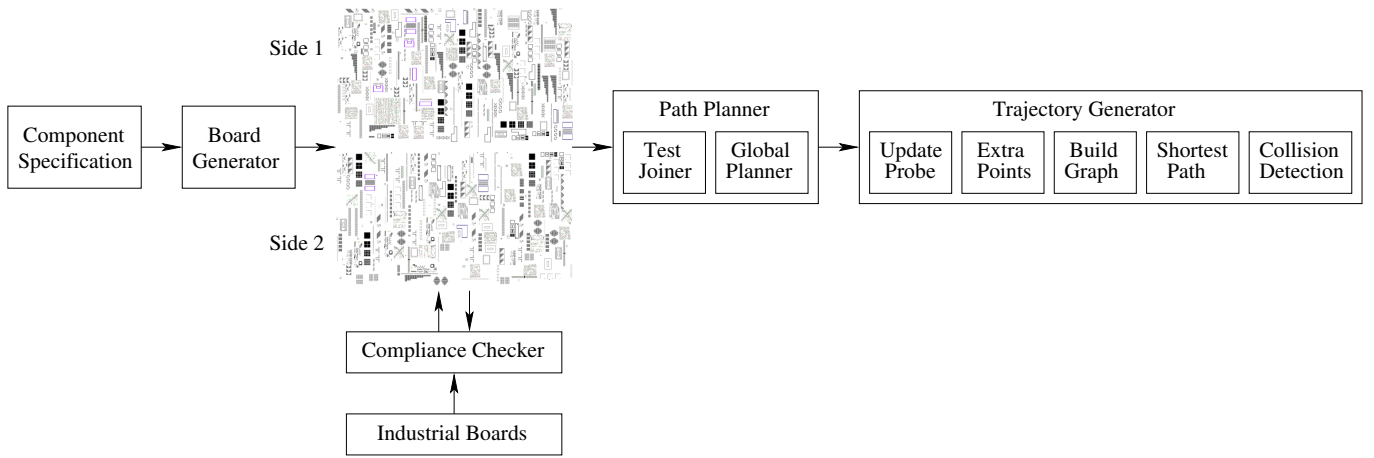


Fig. 1. The picture illustrates the flow of the current work. First, we present a board generator (Section IV) that, starting from pre-defined component specifications, generates two-sided boards and the corresponding test suites. A compliance checker compares the generated boards with a set of original industrial ones. Then, we focus on the path planner and its internal components (a test joiner and a global planner), illustrated in Section V. Finally, we present our trajectory generator (Section VI). Note that the board represented in the figure is the board_10 used in our experimental result section, and it has a width of 125 mm, a height of 225 mm, and 81 818 test points.

like boundary scans or processor-controlled tests. Nevertheless, it is fundamental to detect issues, such as open circuits, soldering defects, and incorrect component placement, before powering up the board and damaging it.

A possible explanation is the lack of publicly available benchmarks. The adopted solutions are deeply bound to the boards' technical details and the machine's peculiarities for performing the test. Therefore, they are hardly generalizable and not suitable for open comparison. This paper aims to provide an unbiased testbench that, while dropping the technical details and perhaps representing a slight oversimplification, is still realistic enough to allow a meaningful comparison of the core algorithms. Besides the set of benchmarks, our contributions include a board generator that can generate boards with additional constraints and a dual-stage planning system featuring a global search and local optimization approach. Figure 1 illustrates the overall methodology and outlines the main phases of our approach.

To facilitate realistic testing simulations, we developed a generator capable of producing simplified yet representative PCB models and their corresponding test suites. The generation process begins by strategically placing a set of "initial components" onto the virtual board (visualized as the leftmost block in Figure 1). These components mirror the physical footprints of genuine industrial parts, including their standard pins, power, and ground connections. To mimic the physical challenges encountered by automated test probes, the generator introduces obstructions by placing additional random components. These create specific regions that impact probe navigation. "High-fly" zones contain components that require probes to execute a distinct vertical lift (along the z -axis) to pass over safely. "No-fly" zones are areas blocked by components so tall they are entirely impassable, forcing probes to plot a path completely around them. Subsequently, the generator defines the electrical "nets," the crucial sets of interconnected points (component pins, test pads, etc.) on the board that are intended to be electrically continuous. These

nets are constructed based on the component layout, forming the basis for the test plan. Finally, the generator derives the test suite, i.e., a sequence of probe movements designed to verify each defined net's electrical integrity and continuity. The entire generation process is guided by user-selected parameters, often reflecting the characteristics of target industrial boards, and incorporates controlled randomness to ensure variety in the generated layouts. Users can customize the board's design and defining features by fine-tuning these parameters within the tool. The process is verified by assessing the compatibility of each generated board with real industrial boards. This verification phase is based on several metrics, among which the size, the number of points and density (average and variance), the number and size of nets (average and variance), the number and size of test points (average and variance), etc.

The global search planner (the second to last block on the right of Figure 1) seeks the best schedule to perform all the tests; a heuristic merges and sorts test sets and coordinates the concurrent movements of many entities simultaneously, as modern test devices have up to eight probes. We resort to a parallel implementation to be as fast as possible [15]–[17]. Probes are moved singularly but concurrently, and straight movements may be unfeasible due to the different speeds, accelerations, and decelerations along the two principal axes. To solve this problem, we model movements as rectangles to provide safe areas for the probes. As each probe's movement is not perfectly linear, instead of following the main diagonal of the rectangle, they can, in the worst-case scenario, move along the two sides of the rectangle. The global search planner cannot predict or care about the probe trajectories. Thus, the trajectory is invalidated if the movement between two configurations intersects a no-fly zone or if the probe moves in a position that is not physically allowed.

The local optimization planner (the rightmost block of Figure 1) corrects the trajectories on the fly by adding new steps. Once one step is discovered to be illegal, we add extra steps to the probe path to validate the whole sequence. The

additional steps are unnecessary to perform tests but provide a legal trajectory for the probes. Each step implies a new set of probe movements, which we define as safe and collision-free. Determining the additional steps is complex, and the process may iterate, leading to a solution in which new movements are added repeatedly to make previous steps valid. A prototype of the local optimization planner has been described in [18].

Our results include statistics on the generated benchmarks (e.g., size, number of test points, test sets, etc.), the global search planner (e.g., number of movements, path length for each probe, etc.), and the local optimization planner (number of corrections required, etc.). As the test cost is mainly related to its duration, our algorithms exploit parallel multi-core devices (CPUs) to improve the overall efficiency. In our experiments, we use the sequential implementation as a base comparison tool. Our results show that our parallel CPU-based implementation has a speed-up of 10x compared to the original sequential version. The optimizer and the generator are distributed under GNU General Public License v3.0, the benchmark boards, under Creative Commons Attribution-Share Alike 4.0 International¹.

The paper is organized as follows. Section II reports some considerations on related works, and Section III describes the testing environment. Section V describes our path planning algorithm, divided into a test joiner and a global search router. The result of the planner is the input for the local optimization router, described in Section VI. The local optimization planner emulates the process running on the actual tester. Section IV describes the board generator. Section VII includes our experimental evidence. Finally, Section VIII concludes the paper with summarizing remarks and hints on future works.

II. RELATED WORKS

While extensive research papers on PCB testing delve into boundary scan techniques, processor-controlled testing, and other powered digital methods, there is comparatively less focus on optimizing ICT, including methods like flying probe testing. These latter techniques are critical in the early stages of testing, particularly for detecting manufacturing defects (e.g., open circuits, soldering defects, incorrect component placement) in a non-powered state, which can help prevent damage during subsequent powered tests. Existing literature relevant to these areas can be grouped into several categories.

A. Test Strategy, Coverage, and Reliability Assessment

Fundamental work addresses the effectiveness and scope of board-level testing, is concerned about reliability under operational stress, or presents specific tools to aid test generation. Hird et al. [19] examine the limitations of test coverage metrics in board-level testing. It highlights the significant gap between passing tests and accurate fault detection. They emphasize the importance of using realistic fault models and complementary testing techniques, providing practical recommendations such as refining fault models and utilizing field data to improve coverage. Although the paper could benefit from more quantitative

support and broader applicability, it is also a valuable resource for enhancing the reliability of electronic testing processes. Yang [20] provides significant insights into PCB reliability by focusing on interconnect stress testing, a methodology assessing the durability of interconnections under various thermal and mechanical stresses (like temperature variations and mechanical loading). The research highlights the importance of these stress factors and suggests practical guidelines for improving PCB design and testing procedures toward more robust and durable electronic packaging solutions. Tinca et al. [21] develop predictive models for assessing the reliability of PCBA. The authors predict reliability, presenting a method combining Finite Element Analysis with virtual prototyping to simulate PCB performance under different loads. They aim to offer more accurate PCB behavior and longevity predictions by incorporating fatigue and copper layer calibration factors. The proposed models aim to offer more accurate predictions of PCB behavior, enhancing the design process and improving overall product reliability. The approach provides a detailed framework for understanding and mitigating failure risks. Van Schaaik et al. [12] describe a software tool designed to automatically generate in-circuit tests directly from product design files, notably without requiring probe access on every net. Their application also indicates board sections where fault coverage is suboptimal and additional probe access would improve test quality.

B. Flying Probe Path Planning and Optimization

Several works focus on optimizing probe movement, which is crucial for efficient flying probe testing. Others present more comprehensive frameworks to integrate multiple aspects to improve the efficiency of the process. Some works finally concentrate on the specificity of the problem, determined by the board and the equipment involved in the testing.

Hiratsuka et al. [14] present an algorithmic approach specifically targeting the minimization of test time and cost. They apply an extension of the Traveling Salesman Problem algorithm, formulated as an integer linear programming problem, to minimize the total inspection time. Practical evaluations show potential improvements in test efficiency for complex circuits. The paper provides valuable insights for enhancing the cost-effectiveness of flying probe testing, but it could benefit from broader application scenarios and real-world data validation. Li et al. [22] present a different planning method based on computing the Manhattan distance between test points. They first cover points using a convex polygon path. Then, they insert all remaining points into the polygon to minimize the added subpath length. Gonzalez-Rodrigues et al. [23] introduce a novel framework combining motion planning and scheduling to optimize task execution while avoiding collisions. This framework is particularly relevant for systems with multiple moving entities, like probe arms, i.e., requiring coordination among numerous moving entities. They provide a mathematical model for planning collision-free trajectories under time constraints and present algorithms balancing computational complexity with real-time needs. The authors demonstrate the effectiveness of their approach in

¹<https://github.com/stefanoquer/Flying-Probe-Board-Testing>

industrial applications. Unfortunately, as far as we know, neither the tool nor the benchmarks are publicly available. Bonaria et al. [24] detail a technique to optimize flying probe positioning for SPEA 4080 equipment. The authors rearrange the test sequence to minimize the test time, considering the tester's capabilities, the board layout, and several constraints derived from the environment and the customer. The authors also compare their new algorithm with the previously used one, i.e., the FP2012, whose performances were sub-optimal when the number of test blocks and points was high.

C. Hybrid Testing Approaches and System Enhancements

Several works combine flying probes with other technologies to leverage multiple strengths and avoid pitfalls.

Seah et al. [13] explore a hybrid approach integrating Automated Test Equipment (ATE) and flying probe in-circuit testing for load board verification. They demonstrate how this combination uses ATE's speed and the flying probe's flexibility to enhance test coverage and fault detection. Real-world implementation examples showcase improved efficiency and reduced costs in testing complex boards. While offering practical insights into optimizing load board verification strategies, the paper could provide more detailed comparisons with traditional methods. Radev and Shirvaikar [11] investigate enhancing a flying probe tester by adding an automated optical inspection module. Adapting face recognition strategies, the authors first perform training using several board images. Then, to achieve a wide range of defect detection and decrease inspection times, they verify their method with different noise levels, occlusion, position shift, rotation, and lighting variation. Jurj et al. [25] design a hybrid sensor-less tester by combining the features of flying probe testing and the capabilities of a coordinate measuring machine. The experimental results show that the proposed tester is suitable for small boards with good precision, speed, power consumption, and cost.

D. Advanced Defect Detection Techniques

Recent works also use machine learning for improved defect detection or focus on scenarios requiring high precision but limited computational resources.

To detect minor defects on PCB surfaces, Hu et al. [6] enhance a Faster-RCNN architecture by incorporating a Feature Pyramid Network. This addition ensures higher precision in identifying defects of varying sizes. The authors replace the traditional intersection over Union loss function with a Weighted intersection to improve detection robustness and accuracy. The study introduces structural improvements to reduce computational complexity and resource requirements, making the model suitable for embedded systems and mobile devices. Niu et al. [8] employ the k-means++ algorithm to improve the location matching of prior anchor boxes, leading to better bounding box predictions for PCB defects. The authors replace the traditional Generalized Intersection over Union loss function with Focal-EIoU. This modification addresses degeneration issues and enhances the model's robustness in defect localization. The proposed method was tested on a public PCB defect dataset, achieving a high mean average

precision and improving the YOLOv5-based approach for real-time PCB defect detection tasks in industrial settings. Yang [26] introduces a novel approach to detecting defects in PCBA. The key contribution is the application of semantic segmentation models, i.e., a deep learning technique typically used in image processing, to automate the detection of defects in PCBA during manufacturing. This method significantly enhances the precision and efficiency of automated optical inspection systems. The approach not only improves defect recognition but also helps reduce reliance on manual inspection, which is prone to errors. The paper demonstrates the robustness of this method in real-world industrial settings, proving its applicability for improving the quality control process in PCBA manufacturing.

E. Summary

To summarize, while state-of-the-art research in PCBA testing has improved, the absence of well-recognized standard benchmarks specifically for comparing flying probe optimization algorithms limits progress in efficiency and cost-effectiveness, leaving the challenges posed by high-density and complex boards somewhat unmet.

III. PROBLEM SPECIFICATION

The proposed approach focuses on a typical high-end flying probe tester with eight arms, four on each side of the board, which can simultaneously test a board's top and bottom sides. Our planner can handle test sets, including simultaneous test points on both sides. However, as probes mounted on one side do not interfere with probes mounted on the other side, for the sake of simplicity in the description of the algorithms, we consider only one side (even if the procedures have been applied on both sides of the board in the experimental section). This restriction is introduced solely to facilitate the discussion and to streamline our methodology and algorithms for clarity and focus and not a limitation of the planner. Similarly, algorithms might be easily extended to a different number of probes.

PCBA test involves performing all in-circuit tests $t_i \in \mathbf{T}_{\text{tot}}$. When using flying probes, the test plan $\mathbf{S}$ can be described as a sequence of steps $s_i = (\mathbf{p}_i, \mathbf{t}_i)$, where $\mathbf{p}_i$ is the position of the four probes $\mathbf{p}_i = (p^0, p^1, p^2, p^3)$ with $p^j = (x^j, y^j)$; and $\mathbf{t}_i = \{t_{n_0}, t_{n_1}, \dots, t_{n_k}\}$, the set of tests that can be performed with the probes in position $\mathbf{p}_i$. Indeed, $\bigcup_i \mathbf{t}_i = \mathbf{T}_{\text{tot}}$ and $\forall i, j : i \neq j \implies \mathbf{t}_i \cap \mathbf{t}_j = \emptyset$. Minimizing the test time requires first defining a set of steps $\{s_0, s_1, \dots, s_n\}$ and then ordering them in the test plan $\mathbf{B} = (s_{b_0}, s_{b_1}, \dots, s_{b_n})$ to minimize the total time for probes movements $(\mathbf{p}_{b_0}, \mathbf{p}_{b_1}, \dots, \mathbf{p}_{b_n})$. The time required to perform the actual tests is irrelevant, as it is constant, regardless of the order of the tests.

To move from positioning $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$, all probes k must be physically relocated from position p_i^k to position p_{i+1}^k . The tester and the board layout may dictate physical constraints: Probes cannot change their positional order along the x axis; they move independently along the y axis and synchronously along the z axis. Boards may include components taller than average, thus requiring extra displacement on the z axis to be

flown over (*high-fly zone*), or even so tall that they cannot be flown over at all (*no-fly zone*).

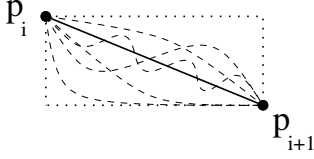


Fig. 2. A probe moves from the source p_i to the destination p_{i+1} . The theoretical straight segment can degenerate into paths within an occupation rectangle with p_i and p_{i+1} as opposite corners.

Probes may have different speeds on the three axes. Movements on the x axis are generally faster than on the y axis, and movements on the z axis are generally the slowest (thus, avoiding high-fly zones may require significant time). Moreover, as relative speeds are not known with absolute precision, to be safe, all trajectories inside the rectangle p_i – p_{i+1} should be considered potentially possible (Figure 2). That is, it would not be safe to move probes directly from p_i to p_{i+1} if the rectangle is p_i – p_{i+1} intersects a no-fly zone, and the direct movement would require extra displacement on z if the rectangle intersects a high-fly zone.

IV. THE BOARD GENERATOR

In this section, we first present a generator to create realistic synthetic boards for objective benchmarks. Then, we describe a small tool to verify the correspondence of these boards with realistic ones coming from the industry.

Based on industrial specifications, boards have a customizable size. Following these specifications, the generator places a predefined set of components on the board. The generator also generates test points, nets, and tests based on the components. It saves them in various formats, from simple text files to JSON² formats to bitmap, including the corresponding board images.

The board generator requires the correct set of components. These components are designed manually following the ones used in the boards we analyzed. Components include high-fly and no-fly zones and may consist of devices with random points and nets, introducing a small amount of non-determinism. These values may be controlled by including the components into classes (e.g., chip, resistors, high- and no-fly areas, etc.) and using components from the different classes with a distinct probability. Many components are reused in developing electronic systems, and our tool allows us to replicate modules and sub-modules. Nets are randomly created using the existing points; the number and size of the nets are selected randomly, but they include at least the ground and the power supply.

The ground net usually touches many components; sometimes, the ground consists of many test points within the same component. A practical example is the GPIO on a Raspberry Pi board, which presents multiple pins that provide voltage

or enable access to the ground. Usually, the number of nets is between 10% and 25% of the total number of points, with smaller boards closer to the former value and larger boards closer to the latter. The test set size is also designed after realistic boards, assuming the availability of four probes for each side of the board.

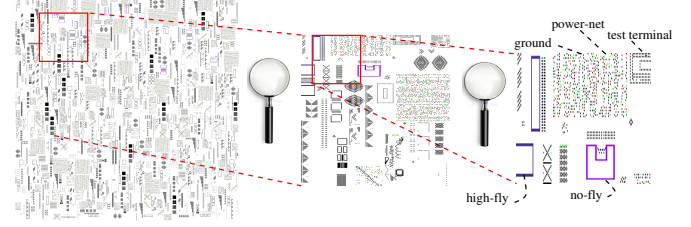


Fig. 3. The image on the right displays board number 48. Alongside it, two progressively zoomed-in views (center and left) provide greater detail. The board has a width of 375 mm, a height of 375 mm, 406 784 points, and 18 693 nets. Key features are color-coded: Black points represent normal test terminals, red points represent the power net, green points represent the ground net, electric violet areas indicate no-fly zones, and dark slate blue areas indicate high-fly zones. All these elements are highlighted in the leftmost, most magnified view (the second zoom level).

Figure 3 shows a board generated using the previously described functions. It includes some significant components, some small ones, and some randomly generated test points. A section of the board is zoomed (and then zoomed again) to give a better flavor of the disposition of the soldered points. Black points represent standard test terminals, red points correspond to the power net, and green points belong to the ground. Electric violet areas indicate no-fly zones, and dark slate blue areas indicate high-fly zones. The layout does not show the nets, i.e., the connections between the ground or power points, as the image would become too complex.

To perform a compliance check for the generated PCBs against the industry-standard boards, we focus on several parameters encompassing the boards' physical, electrical, and functional aspects. We first perform a visual inspection of the board; then, our checker considers the following parameters:

- Physical Design Compliance: Dimensions (overall PCB size, i.e., length, width, and thickness), tolerances for placement and alignment, component placement (positioning and orientation of components, clearances between components).
- Electrical Design Compliance: Netlist Match: (the logical connections must match the industry-standard reference, ground and power design, and component specifications (package types and footprint compatibility)).
- Assembly Compliance, i.e., component placement accuracy and compliance with soldering profiles.

Notice that our board design, once completed, does not produce boards equivalent to actual synthesized boards in every possible design aspect; it statistically generates equivalent boards from the perspective of PCBA testing.

V. THE PATH PLANNING

A valid probe position $\mathbf{p}_i = (p_i^0, p_i^1, p_i^2, p_i^3) = ((x_i^0, y_i^0), (x_i^1, y_i^1), (x_i^2, y_i^2), (x_i^3, y_i^3))$ is a configuration where

²<https://ecma-international.org/publications-and-standards/standards/ecma-404/>

$\forall \alpha, \beta : \alpha < \beta \implies x_i^\alpha \leq x_i^\beta$, and no probe is inside a no-fly zone.

The global search path planning is divided into two phases: A test joiner and a global search router. The test joiner is in charge of merging as many tests as possible in a single probe configuration, such that for each position of the probes $\mathbf{p}_i$, the set of tests performed $\mathbf{t}_i$ is maximized. Using an algorithm similar to the well-known Traveling Salesman Problem (TSP) [27], the global search router finds a reasonably short path touching all positions $\mathbf{p}_i$.

A. The Test-Set Joiner

As most tests require a subset of the probes, and different tests may have one or more test points in common, tests that can be performed with probes in the same position are grouped. For instance, if two tests require only two probes each, they can always be executed simultaneously in the same step, and if they share one test point, they can be executed together using only three probes.

All test points p_i are connected by some electrical net n_j , and points on the same net may be considered equivalent for some tests. The test-set joiner works at the level of *nets*, that is, the joining procedure continues until all probes have been set to a position n_i and all test t_{i_j} that can be executed with the probes in that position have been determined $T_i = \{n_{i_0}, n_{i_1}, \dots, n_{i_k}\}$. The positions are a list of nets, each corresponding to one or more different physical test points of the probes. If a test point in a specific test cannot be considered equivalent to the other test points on the same net, it is assigned to a new virtual net containing only that test point.

It must be noted that joining tests t_i and t_j may prevent joining t_i and t_k , and finding the optimal grouping T_i is an NP problem. We implemented a $\mathcal{O}(N^2)$ greedy heuristic: The tests that require more probes are considered first and joined with the ones sharing more test probes with them.

B. The Global Search Router

Once the set of n_i has been defined, the global search router converts it to a list of steps $((p_0, T_0), (p_1, T_1), \dots, (p_n, T_n))$ that minimize the overall moving time. The global search router must select the best point within each net involved in the test. After that, it sorts them and chooses the effective path, minimizing the traveling time of the probes. These targets could not be separated into different tasks, as the choice of the point in the nets may significantly change the routing outcome.

This problem is far more complex than the Traveling Salesman Problem (TSP), already known as NP-hard. First, the best points are chosen among equivalent ones, while the TSP locations are unique and must all be reached once; moreover, the four probes need to move synchronously, adding new dynamic constraints.

Among the many alternative heuristics developed for the TSP, the router is based on Nearest Neighbor. This constructive methodology is known to produce a result that is 25% longer than the shortest path on average [28], a result inferior to 2-opt [29], 3-opt (or Lin-Kernighan) [30] that focus on optimizing an already constructed route.

As the computational complexity of the global search router is higher than that of the joiner, the eventual performance strongly depends on the number of tests that appear after the joining phase. Thus, improving the joiner also positively affects the time spent by the router performing all tests.

VI. THE TRAJECTORY GENERATION

The router computes the final trajectories of all probes in the selected path, taking into consideration all constraints, such as the test order given by the global search planner, the order of the probes along the x axis, the presence of no-fly (and high-fly zones) forcing probes to longer routes to avoid them.

The trajectory t is the list of movements necessary to perform step s_{i+1} after step s_i . It is defined as a list of positions, with p_i as the first and s_{i+1} as the last; intermediate positions are added as needed and do not correspond to steps (i.e., no test is performed).

```

PERFORMSTEP(s, d)
1:  $Z \leftarrow \{ \text{all no-fly zones on the board} \}$ 
2:  $t \leftarrow (s, d)$ 
3: while not VALID( $t, Z$ ) do
4:    $j \leftarrow \text{SELECTPROBE}(t, Z)$ 
5:    $t \leftarrow \text{UPDATEPROBE}(t, j, s, d, Z)$ 
6: end while
7: STOREMOVEMENTS( $t$ )

```

Algorithm 1: First, we use a straight line to move the probes from source position s to destination d . Then, all illegal movements are iteratively patched. The final legal trajectory t is eventually stored.

The path planning algorithm for finding t works as follows (Algorithm 1). First, it stores the set of all no-fly zones in Z (line 1). Then, an initial trajectory T from the current position p_i (in s_i) to the new positions p_{i+1} (in s_{i+1}) is found by moving all probes on a straight line (line 2). If T is invalid due to the collision between a probe and a no-fly zone or an illicit probe configuration (line 3), the trajectory is updated iteratively. The core idea of the process (lines 3-5) is to select invalid movements and patch them. Thus, in each iteration of the while loop, the function SELECTPROBE (line 4) selects a probe (that is, probe j) that performs an invalid movement. Then, the function UPDATEPROBE evaluates an effective trajectory bringing the flying probe j from the test point p_i to the test point p_{i+1} , avoiding all previous conflicts. Function UPDATEPROBE is detailed in Algorithm 2. Finally, when all illegal probe movements are rectified, the probes are moved along the trajectory T (line 7).

A. Procedure UPDATEPROBE

The procedure UPDATEPROBE generates paths with increasing intermediate points until a valid path is found. The set P (line 1) initially includes only the source p_i and the destination p_{i+1} of the trajectory. The function EXTRAPPOINTS finds the new intermediate points that should be added to the previous path (initially a straight line) to find a permissible path. This function is described in Section VI-B and adds n points to the original trajectory to make it feasible. As the value of

n increases at each iteration, it is always guaranteed that an admissible path is found. Given this new set of points, the function **BUILDGRAPH** (detailed in Section VI-C) returns a graph G representing the shortest path problem. More specifically, G is a weighted, undirected, and incomplete graph $G = (P, E, w)$: Its vertices P are the test points to consider for defining the trajectory, the edges E represent all possible segments connecting two test points in P (excluding the ones crossing a no-flight zone), and the weights w represent the time required to the probes to move from one vertex to the other.

```

UPDATEPROBE( $\mathcal{T}, j, p_i, p_{i+1}, Z$ )
1:  $\mathbf{p} \leftarrow \{p_i, p_{i+1}\}$ 
2:  $\mathcal{S} \leftarrow \emptyset$ 
3:  $n \leftarrow 0$ 
4: while  $\mathcal{S} = \emptyset$  do
5:    $n \leftarrow n + 1$ 
6:    $\mathbf{p} \leftarrow \mathbf{p} \cup \text{EXTRAPOINTS}(p_i, p_{i+1}, n)$ 
7:    $\mathbf{G} \leftarrow \text{BUILDGRAPH}(P, Z)$ 
8:    $\mathcal{S} \leftarrow \text{SHORTESTPATH}(p_i, p_{i+1}, \mathbf{G})$ 
9: end while
10:  $\mathcal{T} \leftarrow \text{ADDSTATES}(j, \mathcal{T}, \mathcal{S})$ 
11: return  $\mathcal{T}$ 

```

Algorithm 2: Update the trajectory $\mathcal{T}$ to get a valid set of probe movements to reach the test point p_{i+1} from the test point p_i , considering no-fly zones Z .

The weights w are computed by adopting a non-linear function that considers the length of the movements, the speed, the acceleration, and the deceleration of the probes along the x and y axes. The function **SHORTESTPATH** (Section VI-D) returns the shortest path from $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$ in the graph G . If $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ are not connected, it returns an empty path. Once a valid path $\mathcal{S}$ has been found, function **ADDSTATES** embeds the path $\mathcal{S}$ into the path $\mathcal{T}$, and the function returns the resulting path. This way, the function ensures the trajectory comprises a sequence of valid configurations.

B. Procedure EXTRAPOINTS

Given a straight path connecting the point set p_i to the point set p_{i+1} , function **EXTRAPOINTS** adds intermediate points to the path to make it feasible. Given the new set of points, finding a path reaching them is expressed and solved as a shortest-path problem on the weighted graph G . Although considering a large number of points would lead to a very effective and possibly optimized trajectory, the size of the graph directly influences the computational effort. Therefore, to trade off the computation time with the movement time, we add the minimum number of points, trying to find a solution with the minimum effort. Thus, the function may be called more than once with the same source p_i and the exact destination p_{i+1} . The value of n controls the number of points returned and records the number of attempts already made. The probability of finding an admissible solution and the computation time to apply the Dijkstra shortest path algorithm [27] on the resulting graph increases at each iteration.

Most invalid trajectories could be grouped in just a few cases (Figure 4). This figure illustrates the four most common

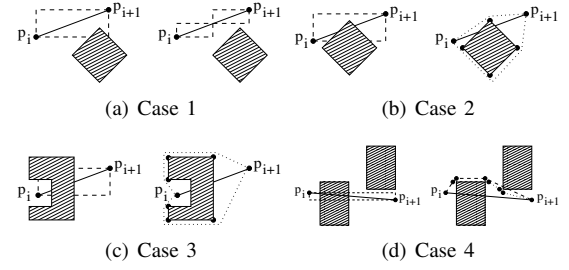


Fig. 4. This figure shows four common collision scenarios between a probe trajectory and a no-fly zone. Each scenario contrasts the original, invalid straight path (left side) with a viable, albeit potentially longer, alternative shown as a dotted line (right side). In case 1, the collision can be resolved simply by splitting the trajectory into two segments. For the other cases (2, 3, and 4), resolution typically involves introducing a few intermediate waypoints, which are not test targets, and then calculating the shortest valid path incorporating these points. Notably, resolving cases 2 and 3 generally requires waypoints related to only one nearby component, whereas case 4 often necessitates using waypoints associated with two components.

configurations where a probe trajectory collides with a no-fly zone. Each configuration's left side depicts the original, invalid straight-line trajectory. The right side shows a potential valid alternative path (dotted line), which may be longer:

- Case 1: Splitting the original movement into two segments creates a valid path.
- Cases 2 and 3: A valid trajectory is typically found by adding a few intermediate waypoints related to a nearby component. These points are not associated with test targets. We compute the shortest valid path that includes them.
- Case 4: We proceed as in cases 2 and 3, but waypoints associated with two different components are often required.

The function **EXTRAPOINTS** (Algorithm 3) first checks if the segment does not intersect a no-fly zone (line 2). In this case, the trajectory can be valid by simply breaking down the movement into different sub-segments; thus, the function returns $1 + 2(n - 1)$ points equally spaced over the original segment (Figure 4, Case 1). When the segment $\overline{p_i p_{i+1}}$ intersects a no-fly zone and $n = 1$, the algorithm returns points around the edges of the relevant no-fly zone, hoping that the following algorithm call will be able to find a trajectory that circumnavigates it (Figure 4, Case 2 and 3). When the segment $\overline{p_i p_{i+1}}$ intersects a no-fly zone and $n = 2$, the algorithm returns the points around the edges of all no-fly zones inside a certain radius of the segment, allowing to find solutions for intricate cases (Figure 4, Case 4). Finally, when invoked with $n = 3$, the function returns a pre-defined set of *safe* points from which a valid albeit non-optimal trajectory can always be found.

C. Procedure BUILDGRAPH

The function **BUILDGRAPH** builds the graph $G = (P, E, w)$ to find the shortest path between the initial and final probe configuration. Given the points in P , the function creates all admissible edges E connecting two test points in P and labels them with the weights w representing the time required by the

EXTRAPOINTS (p_i, p_{i+1}, n)

- 1: $T \leftarrow \emptyset$
- 2: **if** (no intersection with no-fly zones) **then**
- 3: **return** { 1 + 2($n - 1$) points of $\overline{p_i p_{i+1}}$ }
- 4: **else if** ($n = 1$) **then**
- 5: $Z \leftarrow$ intersecting no-fly zones
- 6: **return** EDGES(Z)
- 7: **else if** ($n = 2$) **then**
- 8: $Z \leftarrow$ all nearby no-fly zones
- 9: **return** EDGES(Z)
- 10: **else if** ($n \geq 3$) **then**
- 11: **return** {set of safe points}
- 12: **end if**

Algorithm 3: Update the trajectory T to get a valid movement from position p_i to p_{i+1} .

probes to move from the first to the second extreme of the vertex. The resulting graph is undirected and incomplete, as edges referring to invalid movements are not considered. The procedure is straightforward but can significantly benefit from parallelism (see Section VI-E).

D. Procedure SHORTESTPATH

The function SHORTESTPATH finds the shortest path from the source p_i to the destination p_{i+1} in the weighted graph G . It adopts a modified version of Dijkstra's single-source shortest-path algorithm [31] where the search is interrupted as soon as the destination p_{i+1} is reached. Although the algorithm is exact, the result may not be optimal because the paths are related to the points selected in the previous steps.

E. Collision Detection

The previous algorithms have to verify whether trajectories are valid, i.e., do not cross a no-fly zone and do not change the positional order of the probes. As no-fly zones are modeled as rectangles, procedure SEGMENTCOLLISIONDETECTION looks for any intersection between two segments $\overline{s_1 e_1}$ and $\overline{s_2 e_2}$. We can suppose the first one is indeed our trajectory, whereas the second one is one of the diagonals of the no-fly zone (Algorithm 4).

Let L_1^x and L_1^y (L_2^x and L_2^y) be the length of the first (second) segment on the x and y axes, respectively. The value of d , computed in line 3, is used to detect whether the segments are parallel (line 4). If they are parallel, they do not touch (line 5). If the segments are not parallel, we find the collision's x and y coordinates. Since we are interested in segments instead of lines, even if the segments are not parallel, they might not collide. The part with the checks ensures this is respected by calculating the distance ratio between the collision point and the one using the extremes. If this ratio is not between 1 and 0, the segments do not collide; otherwise, they do. In the final result, we are not interested in the collision point; thus, a simple Boolean value is returned as the result of the check.

When no collision is detected, we compute the actual distance. This value is not a topological distance but a measure of the time required to perform the movement. As the probes have different speeds and distinct accelerations on the other axes, we use a function to compute this value. However, its

SEGMENTCOLLISIONDETECTION($\overline{s_1 e_1}, \overline{s_2 e_2}$)

- 1: $c_1 \leftarrow L_1^y \cdot s_1^x + L_1^x s_1^y$
- 2: $c_2 \leftarrow L_2^y \cdot s_2^x + L_2^x s_2^y$
- 3: $d \leftarrow L_1^y \cdot L_2^x - L_2^y \cdot L_1^x$
- 4: **if** ($d = 0$) **then**
- 5: **return** FALSE
- 6: **end if**
- 7: $x \leftarrow \frac{L_2^x \cdot c_1 - L_1^x \cdot c_2}{d}$
- 8: $y \leftarrow \frac{L_1^y \cdot c_2 - L_2^y \cdot c_1}{d}$
- 9: **if** ($e_1^x \neq s_1^x$) **then**
- 10: $r \leftarrow \frac{x - s_1^x}{L_1^x}$
- 11: $v_0 \leftarrow (r \geq 0 \text{ and } r \leq 1) ? \text{TRUE} : \text{FALSE}$
- 12: **end if**
- 13: **if** ($e_1^y \neq s_1^y$) **then**
- 14: $r \leftarrow \frac{y - s_1^y}{L_1^y}$
- 15: $v_1 \leftarrow (r \geq 0 \text{ and } r \leq 1) ? \text{TRUE} : \text{FALSE}$
- 16: **end if**
- 17: **if** ($e_2^x \neq s_2^x$) **then**
- 18: $r \leftarrow \frac{x - s_2^x}{L_2^x}$
- 19: $v_2 \leftarrow (r \geq 0 \text{ and } r \leq 1) ? \text{TRUE} : \text{FALSE}$
- 20: **end if**
- 21: **if** ($e_2^y \neq s_2^y$) **then**
- 22: $r \leftarrow \frac{y - s_2^y}{L_2^y}$
- 23: $v_3 \leftarrow (r \geq 0 \text{ and } r \leq 1) ? \text{TRUE} : \text{FALSE}$
- 24: **end if**
- 25: **return** ($v_0 \text{ or } v_1$) **and** ($v_2 \text{ or } v_3$)

Algorithm 4: Segment-to-segment collision detection.

computation does not present any particular complexity, and we do not describe it here for the sake of space.

VII. EXPERIMENTAL RESULTS

All tests have been performed using a desktop with a CPU Intel Core i9-10900KF (10 cores, 20 threads) with 64 GiB DDR4 of RAM.

The generator of benchmark boards, the global search planner, and the local optimization trajectory planner have all been implemented in C++. Both planners are time-critical; considering the complexity of their tasks, we resort to an efficient multi-core CPU-based implementation. On the contrary, the generator is not critical, and we performed no specific optimization.

Section VII-A presents some statistics on the board generator and the PCBs under test. Section VII-B reports the results of the global search planner. Finally, Section VII-C collects statistics on the overall trajectory evaluation.

A. Benchmark Suite

Table I shows the generated benchmark suite: 49 boards with different sizes and test-point density, with statistics quite close to actual boards. The pseudo-random generator with embedded components automatically generates test points, nets, and test sets with different complexities. Benchmarks are numbered from 0 to 48 and have increasing dimensions on the x and y axes. The size (column S) varies from 75 mm x 75 mm to 375 mm x 375 mm. The smaller has about 10 000 test points (column $\#TP$), and the larger one has up to 400 000.

TABLE I
FEATURES OF THE BENCHMARK SUITE: SIZE (S), THE NUMBER OF TEST POINTS (#TP), AND THE NUMBER OF NETS (#N). EXPERIMENTAL RESULTS ON THE “SMALL”, “MEDIUM”, AND “LARGE” TEST SETS IN TERMS OF THE NUMBER OF TESTS (#T), PERCENTAGE OF GROUPED TESTS IDENTIFIED BY THE JOINER (%G), TOTAL TIME REQUIRED FOR POSITIONING THE PROBES DURING THE TEST (T^{ALL}), TIME REQUIRED BY THE TEST-POINT JOINER (T^J), AND TIME REQUIRED BY THE PATH PLANNER (T^{PP}).

BOARD	S [mm ²]	#TP	#N	#T	%G	Small T^{ALL} [ms]	T^J [s]	T^{PP} [s]	#T	%G	Medium T^{ALL} [ms]	T^J [s]	T^{PP} [s]	#T	%G	Large T^{ALL} [ms]	T^J [s]	T^{PP} [s]
board_00	75x75	10 230	446	800	68	9738	< 1	< 1	1757	65	18 364	< 1	< 1	2343	66	24 014	1	1
board_01	75x125	22 324	1319	1366	69	25 577	< 1	< 1	2817	69	46 258	1	1	4573	68	70 646	2	1
board_02	75x175	34 387	1661	3339	69	69 683	1	2	5212	68	101 265	3	3	8209	67	147 559	8	7
board_03	75x225	44 547	2468	2936	67	79 790	2	1	6109	68	145 936	7	3	9201	67	204 634	15	5
board_04	75x275	59 953	3355	3169	68	98 050	2	1	9532	68	243 971	19	12	13 143	67	314 232	36	21
board_05	75x325	55 543	2952	5227	68	170 615	6	5	7577	68	233 581	13	10	11 522	67	327 027	28	25
board_06	75x375	78 709	4406	6335	68	224 983	23	11	11 181	68	355 065	74	32	16 511	67	487 809	163	64
board_07	125x75	25 468	1461	1550	69	26 128	< 1	< 1	4187	68	61 624	2	1	5665	68	79 294	3	1
board_08	125x125	45 149	2544	2839	68	60 244	1	1	7031	68	132 944	7	7	10 736	67	188 542	16	7
board_09	125x175	57 668	2872	3740	68	88 538	7	5	7254	68	155 286	24	16	13 236	66	254 397	74	49
board_10	125x225	81 818	4781	5495	69	156 371	10	5	11 667	68	296 208	44	22	16 869	68	404 075	95	39
board_11	125x275	97 854	5129	6931	68	215 169	56	19	16 412	68	443 102	289	106	20 093	67	527 946	397	150
board_12	125x325	98 262	5362	6127	69	214 000	24	16	16 759	67	497 719	159	103	21 938	67	625 035	277	170
board_13	125x375	127 796	7162	9376	69	355 596	57	31	18 827	68	637 680	247	116	31 542	67	976 735	731	293
board_14	175x75	34 644	1900	2913	70	53 976	2	1	4717	68	81 440	4	3	8451	66	132 660	13	7
board_15	175x125	57 446	3001	3685	68	83 864	5	5	8126	68	169 811	22	23	13 766	66	262 391	59	60
board_16	175x175	85 137	4647	7694	68	199 139	22	16	11 611	69	283 692	59	32	19 384	67	437 345	162	80
board_17	175x225	111 939	5295	5981	69	187 134	27	14	15 890	68	426 881	204	91	24 255	67	614 156	567	200
board_18	175x275	125 638	6756	12 430	68	389 347	169	107	17 478	68	519 421	396	217	31 407	67	852 790	1184	594
board_19	175x325	163 376	6871	11 616	68	420 588	162	95	21 397	68	714 723	622	277	35 834	67	1 108 926	1831	737
board_20	175x375	184 398	8241	15 414	68	573 541	385	183	23 807	67	819 155	844	390	43 107	67	1 367 621	2990	1223
board_21	225x75	42 362	2220	2696	70	58 175	2	1	5313	67	101 608	7	5	8542	68	154 675	18	11
board_22	225x125	80 801	3762	7974	69	193 550	20	11	13 564	68	304 988	62	30	18 507	67	396 466	122	58
board_23	225x175	113 981	5355	8495	67	241 748	56	37	14 541	67	386 719	172	103	23 629	67	589 217	462	256
board_24	225x225	146 365	6950	8594	69	284 214	100	53	24 836	67	704 838	798	371	30 142	67	83 148 491	1377	548
board_25	225x275	169 408	10 227	10 822	68	389 802	186	113	24 043	67	774 217	965	512	36 966	66	1 118 468	2220	1127
board_26	225x325	197 260	9464	18 874	68	695 150	722	277	33 406	67	1 130 892	2241	830	49 066	67	1 581 462	5050	1641
board_27	225x375	238 835	10 625	18 121	68	745 340	972	296	34 814	67	1 300 701	2770	1035	56 017	67	1 963 347	9031	2514
board_28	275x75	57 069	3079	3147	68	70 821	6	3	8975	67	174 091	44	18	11 521	67	216 683	76	28
board_29	275x125	91 163	5107	4848	70	136 971	17	10	14 213	68	345 327	147	70	20 642	67	477 069	305	134
board_30	275x175	138 137	7417	10 994	68	336 934	115	63	20 077	67	566 969	463	192	30 007	67	805 731	941	438
board_31	275x225	179 365	9659	10 675	69	381 071	140	73	27 859	68	874 131	790	468	37 260	67	1 127 380	1456	779
board_32	275x275	216 017	9657	11 248	68	432 043	265	128	34 192	67	1 140 574	2967	1047	47 784	67	1 512 446	6062	2025
board_33	275x325	240 686	14 373	20 088	68	809 764	1148	477	39 001	67	1 431 979	4617	1636	50 395	66	1 780 406	7565	2712
board_34	275x375	292 963	13 292	26 535	68	1 100 192	2029	694	38 229	68	1 515 310	4043	1391	68 858	67	2 514 024	13 530	4301
board_35	325x75	63 857	2971	3956	69	91 886	9	6	9893	67	201 767	48	32	15 275	66	293 514	119	69
board_36	325x125	113 953	6303	7700	68	216 814	61	36	16 169	67	412 049	341	146	28 359	66	670 833	853	399
board_37	325x175	150 145	7431	10 661	68	351 482	173	75	24 478	67	725 533	1240	373	32 701	67	932 409	1955	636
board_38	325x225	204 825	11 027	16 688	68	603 408	523	212	33 991	67	1 121 759	2237	827	42 773	67	1 367 143	3413	1265
board_39	325x275	255 132	14 453	17 850	69	618 942	470	156	36 348	68	1 355 769	2696	872	52 559	67	1 865 216	6496	1713
board_40	325x325	294 332	13 876	27 212	68	1 115 845	2674	817	42 578	68	1 653 640	7089	1932	60 437	67	2 228 489	13 822	3711
board_41	325x375	354 618	19 878	30 800	68	1 377 866	3805	1471	45 822	68	1 938 994	8210	3075	71 341	67	2 840 873	23 157	7187
board_42	375x75	83 452	4938	5292	69	132 813	13	9	11 746	68	264 839	65	48	19 485	66	406 418	161	130
board_43	375x125	133 558	7445	9151	68	276 635	81	41	17 387	67	480 868	300	136	31 043	67	795 841	899	436
board_44	375x175	195 884	10 185	17 173	68	563 254	615	252	26 036	68	814 963	1222	541	39 317	67	1 159 011	3531	1153
board_45	375x225	244 110	13 076	17 850	69	685 458	770	336	33 349	67	1 32 836 456	4033	1416	52 380	67	1 732 017	8034	2542
board_46	375x275	302 178	16 013	25 859	68	1 038 840	2564	975	51 435	67	1 900 516	9913	3596	68 021	66	2 411 096	18 169	6101
board_47	375x325	344 642	19 881	25 692	68	1 161 034	3398	1109	50 623	67	2 075 027	14 628	4113	82 296	67	3 167 940	34 413	10 012
board_48	375x375	406 784	18 693	25 693	69	1 248 617	3749	1137	53 424	67	2 343 187	14 508	4544	85 207	67	3 507 331	39 145	11 010

The number of nets (columnn $\#N$) ranges from about 400 to almost 19 000. All boards' average number of test points is about 140 000. The number of no-fly areas varies from 0 to 5, and the number of high-fly zones varies from 0 to 3. For each board, we generated three test suites with increasing tests: a *small* test set, a *medium*, and a *large* one. While the smallest test set comprises about 800 test points (column $\#T$), the largest includes more than 85 000 test points.

We simulate that each board has to be tested for soldering defects on both sides. The designer also defines equivalent test points for each side, such as the one for the ground node, the power supply, and other typical nodes. This information is encoded into nets. As with actual boards, few nets, like ground and power lines, can be huge, while all the others are significantly smaller and may count as one or two test points.

B. The Global Search Planner

Table I shows the results gathered with the global search procedure. Results show the test joiner and the pathfinder; all data are reported on the set of boards described in the previous section. The $\%G$ columns report the percentage of tests after the joining procedure. In almost all benchmarks, the heuristic can reduce the number of tests to slightly more than 2/3 of the original test suite. The columns T^{ALL} show the time (in milliseconds) for moving all eight probes over all necessary test points as computed by the global search router. These values depend on the distance of the points, their number, the number and size of the nets, and the number of tests. Overall, probes can have a high variance in moving from one point to the next. For example, for the largest board, the minimum displacement time is about 4 ms, whereas the latest one is about 393 ms. Similar differences are observed in small and medium-sized boards. The columns T^J and T^{PP} indicate the elapsed time (in seconds) required to, respectively, join the test sets and find the final path. In all tests, the joiner's execution time never exceeds the pathfinder's by more than 300%. However, joining the test sets is fundamental to obtaining good performance in the final practical testing phase. Results suggest that the time reduction obtained using the joiner is exponential in the number of merged tests. The total running time to optimize the test set and find the best possible path ranges from a fraction of a second, for the smaller board, to a few hours, for the larger ones.

C. Local Optimization Planner

Table II reports the wall-clock time³ (in seconds) required to generate a valid trajectory for eight probes (four on each side of the board) using the algorithms of Sections VI-B and VI-E.

The first column ($\#Steps$) reports the number of steps, that is, the number of different positions ($p_0, p_1, \dots, p_n$) the probes have to reach following the previously computed path. Notice that the reported times consider only the displacement times of the probes, not the testing time (i.e., not the times to apply

the stimuli and verify the results, whose value depends on the electrical features of the PBC and not on our planner). We consider up to 30 000 steps, as this number is sufficient for the largest board with the most extended test sequence. The reported times are averaged over fifty different runs, using boards of varying sizes extracted randomly from the benchmark set. Column *Full* indicates the wall-clock times required to compute the trajectory using the complete set of additional points, that is, calling EXTRAPPOINTS directly with a value of n equal to 3 to find a collision-free trajectory. Column *Smart* shows the incremental version of the procedure EXTRAPPOINTS, that is, the value of n is increased from one to three only when needed. The *smart* algorithm behaves like the *full* one after two iterations, but it first looks for a trivial solution. For the *parallel* version, one thread was used to calculate the path of each probe, i.e., eight threads overall. This allows us to run the extended test sequence on the larger boards in less than ten seconds.

Table II details the performance enhancements of the various approaches, reporting speedups calculated against the basic sequential implementation. Introducing the “smart technique” alone provides an approximate 8x speedup over the sequential baseline. This advantage is amplified to around 10x when the “smart technique” is combined with the parallel execution strategy.

TABLE II
TOTAL TIME (FIRST COLUMN) TO PERFORM AN INCREASING NUMBER OF TESTS AND THE SPEEDUP (SECOND COLUMN, WHEN MEANINGFUL) RELATIVE TO THE “SEQUENTIAL FULL” APPROACH.

#Steps	Sequential		Parallel			
	Full	Smart	Full	Smart	Full	Smart
500	1.68 s	0.21 s	8.00	0.85 s	1.98	0.16 s
1000	3.52 s	0.41 s	8.59	1.53 s	2.30	0.31 s
5000	17.42 s	2.14 s	8.14	6.88 s	2.53	1.71 s
10 000	33.12 s	4.20 s	7.89	13.00 s	2.55	3.43 s
15 000	46.40 s	6.69 s	6.94	18.75 s	2.47	4.98 s
30 000	65.50 s	8.91 s	7.35	23.51 s	2.79	6.33 s

VIII. CONCLUSIONS

This paper describes a motion-planning application in which eight probes must move over a printed circuit board to reach a sequence of test configurations. We designed a benchmark generator and a global search and local optimization algorithm that optimized the steps required to move between those configurations without the probes crossing each other. We propose a solution that offers a trade-off between the computational time and the probe repositioning time, finding a reliable local solution for each probe and optimizing their simultaneous change of position for that movement. Experimental results show that the multi-core approach reduces the average path-finding time significantly.

REFERENCES

- [1] C. PCB, “The Ultimate Guide to PCB Testing Techniques,” 2025, accessed April 2025.
- [2] JHDPBCB, “The Ultimate Guide to Flying Probe Testing (FPT) of PCBs,” 2023, accessed April 2025.

³The wall-clock time is the time necessary for a (mono-thread or multi-thread) process to complete the task, i.e., the difference between the time the task finishes and the time the task starts. For this reason, the wall-clock time is also known as elapsed time.

- [3] C. P. Resources, "Circuit Board Testing for Manufacturing Verification," 2021, accessed April 2025.
- [4] G. Lakshmi, V. U. Sankar, and Y. S. Sankar, "A Survey of PCB Defect Detection Algorithms," *Journal of Electronic Testing*, vol. 39, pp. 541–554, 2023.
- [5] M. Prytula and S. Shtefanesa, "Analysis of Testing Methods and Reliability Improvement of Printed Circuit Boards in Measurement Equipment," *Scientific Collection InterConf*, no. 200, p. 216–225, May 2024. [Online]. Available: <https://archive.interconf.center/index.php/conference-proceeding/article/view/6213>
- [6] B. Hu and J. Wang, "Detection of PCB Surface Defects with Improved Faster-RCNN and Feature Pyramid Network," *IEEE Access*, vol. PP, pp. 1–1, 06 2020.
- [7] J. Li, W. Li, Y. Chen, and J. Gu, "A PCB Electronic Components Detection Network Design Based on Effective Receptive Field Size and Anchor Size Matching," *Wiley Online Library*, vol. 2021, p. 6682710, 2021.
- [8] J. Niu, H. Li, X. Chen, and K. Qian, "An Improved YOLOv5 Network for Detection of Printed Circuit Board Defects," *Journal of Sensors*, vol. 2023, pp. 1–10, 03 2023.
- [9] B. T. Sayed, H. A. Alkhalaleh, P. Rodrigues, S. Askar, M. K. Sharma, and S. M. A. and, "Optimising Solder Joint Inspection in Printed Circuit Boards Through X-ray Imaging and Machine Learning Integration," *Nondestructive Testing and Evaluation*, vol. 0, no. 0, pp. 1–21, 2025. [Online]. Available: <https://doi.org/10.1080/10589759.2025.2455451>
- [10] C. F. Coombs, *Printed Circuits Handbook (7th Edition)*. USA: McGraw-Hill, Inc., 2016.
- [11] P. Radev and M. Shirvaikar, "Enhancement of Flying Probe Tester Systems with Automated Optical Inspection," in *Proceeding of the Thirty-Eighth Southeastern Symposium on System Theory*, 2006, pp. 367–371.
- [12] H. van Schaaijk, M. Spierings, and E. J. Marinissen, "Automatic Generation of In-Circuit Tests for Board Assembly Defects," in *IEEE 23rd European Test Symposium (ETS)*, 2018, pp. 1–2.
- [13] Soh Ying Seah, Melanie Po-Leen Ooi, Ye Chow Kuang, Chee Sun See, S. Panchadcharam, and S. Demidenko, "Combining ATE and Flying Probe In-Circuit Test Strategies for Load Board Verification and Test," in *2009 IEEE Instrumentation and Measurement Technology Conference*, May 2009, pp. 1380–1385.
- [14] Y. Hiratsuka, F. Katoh, K. Konishi, and S. Shin, "A Design Method for Minimum Cost Path of Flying Probe In-Circuit Testers," in *Proceedings of SICE Annual Conference 2010*, Aug. 2010, pp. 2933–2936.
- [15] D. Appello, P. Bernardi, A. Calabrese, G. Pollaccia, S. Quer, V. Tancorre, and R. Ugioli, "Parallel multithread analysis of extremely large simulation traces," *IEEE Access*, vol. 10, pp. 56 440–56 457, 2022.
- [16] A. Borione, L. Cardone, A. Calabrese, and S. Quer, "An experimental evaluation of graph coloring heuristics on multi- and many-core architectures," *IEEE Access*, vol. 11, pp. 125 226–125 243, 2023.
- [17] L. Cardone and S. Quer, "The Multi-Maximum and Quasi-Maximum Common Subgraph Problem," *Computation*, vol. 11, no. 4, 2023. [Online]. Available: <https://www.mdpi.com/2079-3197/11/4/69>
- [18] A. Calabrese, S. Quer, and G. Squillero, "Smart Techniques for Flying-probe Testing," in *16th International Conference on Software Technologies (ICSOFT 2021)*, 2021, pp. 285–293.
- [19] K. Hird, K. P. Parker, and B. Follis, "Test coverage: What Does it Mean When a Board Test Passes?" in *Proceedings. International Test Conference*, 2002, pp. 1066–1074.
- [20] Y. Yang, "Reliabilities and Failure Analysis of Printed Circuit Boards Interconnect Stress Test," in *2018 19th International Conference on Electronic Packaging Technology (ICEPT)*, 2018, pp. 428–431.
- [21] Tinca Iulia Eliza and Ailenei Iulian Ionut and Davidescu Arjana, "Printed Circuit Board Assembly Modeling for Predictive Reliability Assessment," in *2022 IEEE 28th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2022, pp. 52–57.
- [22] W. Li, J. Yang, X. Lv, and J. Wang, "A New Path Planning Method for Flying Probe Test Arms," in *Chinese Control And Decision Conference (CCDC)*, 2020, pp. 1552–1556.
- [23] A. G. Gonzalez-Rodriguez and A. Gonzalez-Rodriguez, "Collision-free motion planning and scheduling," *Elsevier*, 2010, further bibliographic details such as the journal name, volume, issue, or DOI are required for a complete citation.
- [24] L. Bonaria, M. Raganato, M. Sonza Reorda, and G. Squillero, "A Dynamic Greedy Test Scheduler for Optimizing Probe Motion in In-Circuit Testers," in *IEEE European Test Symposium (ETS 2019)*, May 2019, pp. 1–2.
- [25] S. L. Jurj, R. Rotar, F. Opritoiu, and M. Vladutiu, "Affordable Flying Probe-Inspired In-Circuit-Tester for Printed Circuit Boards Evaluation with Application in Test Engineering Education," in *2020 IEEE International Conference on Environment and Electrical Engineering and 2020 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*, Jun. 2020, pp. 1–6.
- [26] Yang Li and Xiao Wang and Zhifan He and Ze Wang and Ke Cheng and Sanchuan Ding and Yijing Fan and Xiaotao Li and Yawen Niu and Shanpeng Xiao and Zhenqi Hao and Bin Gao and Huaqiang Wu, "Industry-Oriented Detection Method of PCBA Defects Using Semantic Segmentation Models," *IEEE/CAA Journal of Automatica Sinica*, vol. 11, no. JAS-2023-1360, p. 1438, 2024. [Online]. Available: <https://www.ieee-jas.net/en/article/doi/10.1109/JAS.2024.124422>
- [27] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Fourth Edition*. The MIT Press, 2022.
- [28] D. S. Johnson and L. A. McGeoch, "The Traveling Salesman Problem: A case study in local optimization," *Local search in combinatorial optimization*, vol. 1, no. 1, pp. 215–310, 1997.
- [29] G. A. Croes, "A Method for Solving Traveling-Salesman Problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958. [Online]. Available: <http://www.jstor.org/stable/167074>
- [30] K. Helsgaun, "General k-opt submoves for the Lin-Kernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, pp. 119–163, 10 2009.
- [31] E. W. Dijkstra *et al.*, "A Note on two Problems in Connection with Graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.



Andrea Calabrese graduated in Computer Science from Politecnico di Torino in 2020, where he took his Ph.D. in 2024. His main field of research is software optimization and parallel algorithms. Since the beginning of 2024, he has been working at Amarula Solutions, a company focused on open-source development, as a test engineer, working on BSH services.



Stefano Quer received a M.S. and a Ph.D. degree in Computer Engineering, in 1991 and 1996, respectively. He has been a visiting faculty member at the University of California in Berkeley, an intern with the "Advanced Technology Group" at Synopsys Inc., and with the "Alpha Development Group" at Compaq Computer Corporation. He has been a Compaq Computer Corporation consultant. He is currently a professor with the Department of Control and Computer Engineering at Politecnico di Torino, Italy. His main research interests include systems and tools for CAD for VLSI, formal methods for hardware and software systems, and sequential and concurrent algorithms.



Giovanni Squillero (M'01–SM'14) received both the M.S. (1996) and Ph.D. (2002) degrees in Computer Engineering from the Politecnico di Torino, Italy, where he currently works as a full professor of Computer Science. His research combines artificial intelligence and soft computing, particularly bio-inspired meta-heuristics and multi-agent systems. Squillero also designs approximate optimization techniques to obtain acceptable solutions with reasonable resources. The industrial applications of his work range from electronic CAD to bioinformatics and the cultural sector.