Optimizing wire layout in quantum processors with SciPy

Adyoth Sural

Abstract—This paper presents Qartographer, an open-source computational tool for designing and optimizing control component layouts in quantum chips. We address the critical challenge of creating compact, high-performance designs by systematically arranging control and readout lines. The method focuses on minimizing crosstalk and noise, with a primary goal of providing a robust, repeatable process for exploring various wire layouts.

Keywords—*Quantum Computing, Software, Optimization*

1. Introduction

The promise of quantum computing to revolutionize fields from cryptography to materials science has driven a global effort to build powerful, scalable quantum computers. Unlike classical computers that store information in bits, quantum computers use qubits, which can exist in a superposition of states, offering the potential for exponential speedups on certain computational problems. This transformative potential was first highlighted by landmark theoretical breakthroughs, such as **Shor's algorithm** for factoring large numbers [1] and **Grover's algorithm** for unstructured search [2]. These foundational works sparked intense interest and laid the groundwork for the field of quantum information science, as comprehensively detailed in the seminal textbook by Nielsen and Chuang [3].

While theoretical research has progressed, a significant bottleneck remains in designing and optimizing the physical chip layout. Crafting a quantum chip is a complex puzzle, requiring not only the careful placement of qubits but also the precise routing of control and readout lines. Improper routing can introduce crosstalk and signal interference, compromising qubit performance [5]. As quantum processors scale, the need for automated design tools becomes critical, especially as systems transition from a handful of qubits to the thousands or even millions required for practical quantum algorithms [4]. The manual routing of these control components is a tedious and time-consuming task, with the number of connections growing quadratically, leading to increased risk of crosstalk and signal interference. To tackle these problems, we developed **Qartographer**, a computational tool that automates the wire routing process. Qartographer focuses on optimizing the layout of ancillary components, providing a streamlined workflow for exploring scalable, high-performance quantum processor designs.

2. Methodology

Our approach uses an optimization framework to systematically design and refine quantum chip layouts. This method begins with a pre-optimized qubit layout, then proceeds with an efficient control line routing process.

3. Control Line Routing

The routing process takes a fixed qubit layout and the Drive SMPM connection points, and designs the paths for the drive and readout lines. We achieve this by minimizing a multi-objective cost function that balances several key design constraints. A gradient-based optimizer is well-suited for this task because the routing cost function is generally smooth and differentiable under typical layout conditions.

The cost function is a composite of several terms:

- **Proximity Penalty:** This term penalizes control lines that pass too close to qubits or to each other, which helps prevent unwanted crosstalk and signal interference.
- **Length Cost:** The length of the multiplexer and drive lines is minimized to reduce signal loss and latency, thereby promoting high-fidelity operations.
- **Readout Line Alignment:** This term encourages readout lines to connect efficiently from each qubit to its nearest multiplexer line, targeting an ideal line length.

Each of these penalties can be weighted to emphasize different aspects of the chip structure, and choosing the best weight values for the task at hand is critical for the usage of this tool. For the optimization process, we use the *L-BFGS-B* algorithm, a memory-limited version of the Broyden-Fletcher-Goldfarb-Shanno algorithm. This was chosen to handle the large number of variables in the routing problem efficiently, avoiding the computational inefficiency of a full-memory algorithm.

4. Computational Details

The Qartographer optimization framework is implemented in **Python**, leveraging key libraries for complex calculations. At its core, the process relies on **SciPy's optimization module** [6].

For **Control Line Routing**, we specifically use the `scipy.optimize.minimize` function with the `method='L-BFGS-B'` parameter. This is a gradient-based optimizer that handles a large number of variables, such as the coordinates that define the wire paths. Its memory-limited nature makes it an effective choice for the high-dimensionality of this routing problem.

5. JSON File Structure

The Qartographer workflow uses a structured **JSON** format to store and transfer data. This modular design helps in managing the data for different stages of the design process.

5.1. Qubit Placement Data File

This input file contains the pre-optimized qubit arrangement. It includes:

- `optimized_qubit_coordinates`: A dictionary mapping each qubit (e.g., "Qubit_0") to its final x and y coordinates.
- `optimized_couplings`: A list of objects representing couplings between qubits, specifying the `qubit1_index` and `qubit2_index`.

5.2. Wiring Data File

This is the primary output file from the control line routing process. It details the paths for all control lines and includes:

- `fixed_drive_points`: Coordinates for the fixed endpoints of drive lines on the chip boundary.
- `optimized_multiplexer_lines`: Optimized routes of the multiplexer lines.
- `optimized_readout_to_multiplexer_lines`: Connections from each qubit to a multiplexer line.
- `optimized_drive_paths`: Optimized routes of drive lines from each qubit to its drive point.

5.3. Device Map File

The merger tool combines the data from the qubit placement and wiring files into a single, comprehensive device map. This file serves as a complete blueprint of the entire quantum chip, consolidating all information into one file for easy use with the visualization tools.

6. The Qartographer Tool Suite

Beyond the core optimization algorithms, Qartographer includes a suite of tools for managing the design workflow. This suite, comprising of a merger, a splitter, and a plotter, ensures a modular and robust process.

6.1. The Merger

The **merger** combines data from the qubit placement and wiring files into a single, comprehensive device map JSON. This creates a unified blueprint for the entire chip layout.

6.2. The Splitter

The **splitter** performs the inverse operation, taking a unified device map and separating it into its constituent JSON files. This is useful for iterative design; for instance, a designer can extract only the wiring data to perform a new round of routing without affecting the existing qubit couplings. This modularity allows for focused exploration and modification.

6.3. The Plotter

The **plotter** visualizes the chip layout from a device map JSON file. It displays qubit locations, couplings, and the intricate routes of all control lines. This visual feedback is essential for design verification, helping designers quickly spot potential issues like overcrowding or closely routed lines that could cause crosstalk.

7. Results

The Qartographer tool successfully optimizes control line layouts for quantum processors, as demonstrated by applying the methodology to various qubit array configurations. We specifically tested the framework on two distinct square lattice sizes: a **3x3 array** (9 qubits) and a **5x5 array** (25 qubits). In both cases, the optimization process effectively minimized the composite cost function across the large number of variables that define the wire paths.

For the 3x3 array, the optimization produced a compact and efficient arrangement. The drive paths and the readout and multiplexer lines had minimal crossings and it reached the global minimum faster.

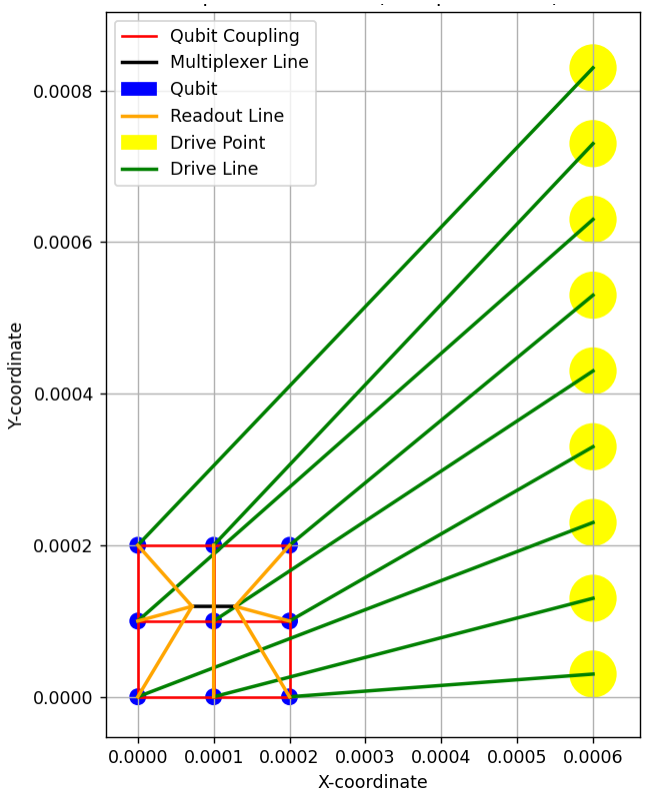

Figure 1. Optimized wire layout for a 3x3 qubit array, showcasing compact and efficient routing with minimal crossings.

While the smaller array appeared to reach a global minimum efficiently, the 5x5 array highlights the increased complexity of larger systems. The multi-objective cost function enabled balanced optimization, but tuning the weights of proximity penalty, length cost, and readout alignment remains a critical hyperparameter that requires further study.

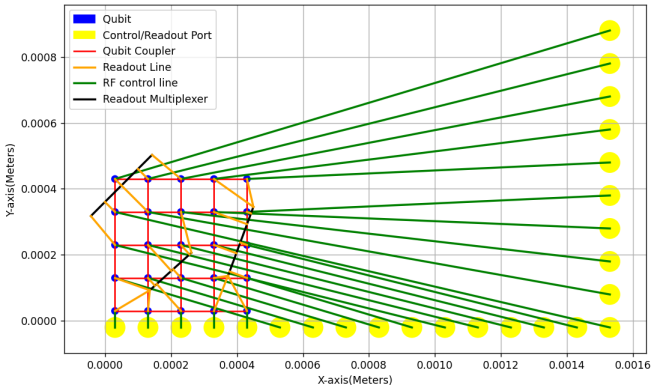

Figure 2. Optimized wire layout for a 5x5 qubit array. The complexity of the routing increases with the number of qubits, but the tool still produces an organized layout.

8. Discussion and Future Work

While the Qartographer tool effectively optimizes control line layouts, several areas for improvement can be explored in future work. The current framework optimizes idealized wire paths, so a critical next step is to integrate real-world fabrication constraints directly into the cost function. This includes adding penalties for sharp turns, minimum trace widths, and other lithography-related limitations to make designs more robust and manufacturable. Additionally, incorporating more sophisticated physical models, such as calculating capacitance or inductance between wires, would improve the designs' physical accuracy by allowing the optimizer to minimize crosstalk and other noise sources more directly. Future research could also extend the framework to include readout multiplexer SMPM port integration, a crucial step toward generating designs more directly translatable to physical fabrication.

The Qartographer project provides a foundational framework for tackling the automated physical design of quantum processors. Future research can build upon this foundation by extending the core capabilities to address more complex, real-world design problems. These research directions can be explored as independent modules or as part of a more comprehensive, integrated design system.

The following are key areas for this future research framework:

1. **Integration of Physical Models:** The present cost function uses simplified geometric metrics. Future research could focus on integrating more sophisticated physical models directly into the optimization. For example, a module could be developed to calculate mutual inductance and capacitance between wire segments, providing a more accurate measure of crosstalk. The output of this module could be incorporated as a penalty term in the cost function, allowing the optimizer to minimize physical noise sources more directly.
2. **Constraint-Aware Optimization:** The current tool does not account for many fabrication-specific constraints. A valuable research project would be to develop a **constraint-aware optimization** layer. This would involve building a set of rules and penalties for lithography-related limitations, such as minimum wire spacing, bend radii, and layer restrictions. The optimization would then find the most efficient layout that adheres to a given fabrication process's design rules, producing designs that are not just theoretically optimal but also physically manufacturable.
3. **Algorithmic Scalability:** As quantum processors scale to thousands of qubits, the dimensionality of the optimization problem

will grow quadratically. A critical area of research is to investigate and implement more scalable optimization algorithms. This could include developing specialized decomposition techniques to break the problem into smaller, more manageable sub-problems, or exploring the application of machine learning methods, such as deep reinforcement learning, to discover routing strategies that generalize to large-scale systems.

9. Conclusion

We have introduced Qartographer, a robust computational tool for optimizing the wire layout in quantum processors. By leveraging **SciPy's optimization module** and a carefully crafted multi-objective cost function, Qartographer automates a complex and critical aspect of quantum chip design. The tool's modular architecture, supported by its merger, splitter, and plotter components, provides a flexible and repeatable workflow for exploring scalable, high-performance designs. As quantum hardware continues to evolve, the Qartographer framework offers a powerful, open-source solution to overcome the bottleneck of physical design, paving the way for more compact, efficient, and higher-fidelity quantum computers.

10. Code Availability

The Qartographer tool, including the source code and documentation, is publicly available on the GitHub repository at https://github.com/AYDOSUL/Qartographer.

References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring", *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search", *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

[3] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th Anniversary. Cambridge University Press, 2010, ISBN: 978-1-107-00217-3.

[4] C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits", *Quantum*, vol. 5, p. 433, Apr. 2021, ISSN: 2521-327X. DOI: 10.22331/q-2021-04-15-433. [Online]. Available: https://doi.org/10.22331/q-2021-04-15-433.

[5] V. Tripathi, H. Chen, M. Khezri, K.-W. Yip, E. Levenson-Falk, and D. A. Lidar, "Suppression of crosstalk in superconducting qubits using dynamical decoupling", *Phys. Rev. Appl.*, vol. 18, p. 024 068, 2 Aug. 2022. DOI: 10.1103/PhysRevApplied.18.024068. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.18.024068.

[6] SciPy. "Optimization and root finding". Accessed on 11 July 2025. (2025), [Online]. Available: https://docs.scipy.org/doc/scipy/tutorial/optimize.html.