

Fast Filter Synthesis and Placement for Programmable Photonics

F. Vanden Kerchove¹, Zhengqi Gao², Didier Colle¹, *Member, IEEE*, Wouter Tavernier¹,
Wim Bogaerts¹, *Fellow, IEEE*, Duane S. Boning², *Fellow, IEEE*, and Mario Pickavet¹, *Senior Member, IEEE*

Abstract—Programmable photonics is a rapidly evolving field aimed at improving the accessibility and versatility of photonic integrated circuits. Despite recent advances, a major challenge persists: the lack of efficient and scalable algorithms for configuring the circuits to realize complex programmable functionalities. We propose a comprehensive methodology for routing and filtering in programmable photonic integrated circuits. Our approach begins with a high-level description of the filtering and routing requirements and systematically generates a fully functional configuration that satisfies all criteria. Central to our approach is an analytical formulation of the filter response, which not only facilitates rapid evaluation but also enables efficient derivative computation. We employ a divide-and-conquer strategy to calculate these derivatives with little additional overhead. This allows us to utilize derivative-based minimization techniques for filter synthesis, thereby reducing computation time. Furthermore, we demonstrate a simulated-annealing-based placement algorithm that focuses on concurrently placing multiple filters. Prior work has largely focused on small-scale demonstrations, typically involving one or two filters simultaneously. To the best of our knowledge, this is the first method capable of configuring photonic meshes that support the simultaneous realization of over 20 distinct light-processing functions, while maintaining practical run times. In this paper, ChatGPT and Claude were used for editing and grammar enhancement.

Index Terms—Optical filters, optimization, photonic integrated circuits, programmable circuits, signal processing.

I. INTRODUCTION

PROGRAMMABLE photonic integrated circuits (PPICs) have the potential to dramatically speed up and reduce the cost of developing photonic applications [1]. Rather than designing and fabricating a new circuit for every application, a single programmable mesh can be reconfigured in the field to realize a wide range of functionalities, much as a field-programmable gate

array (FPGA) is repurposed across digital designs. This reconfigurability lowers the barrier to entry for photonic design, shortens development cycles, and opens the technology to researchers and engineers who are not specialists in photonic fabrication. This versatility has attracted interest across a broad range of fields, including optical communications [2], photonics for radiofrequency applications [3], microwave photonics [4], quantum information processing [5], [6], optical computing and neuromorphic hardware [7], and sensing [1], making programmable photonics relevant to a correspondingly wide audience. The algorithms that drive the various components are an important part of this ecosystem. When comparing programmable photonics to its electronic counterpart, the algorithmic problems associated with these two systems are substantially differentiated by the ability of light to be folded back onto itself. This creates a wavelength-dependent interference and resonance pattern that enables on-chip optical functionalities, such as wavelength filtering and phase manipulation.

Multiple papers describe algorithmic solutions to various problems related to PPICs. In [8], the authors propose several graph representations that allow graph-based algorithms to realize routing in PPIC, realizing port-to-port connections. Using sequential routing, [9] explores the first combination of routing with optical filtering, while [10] scales the capabilities of the routing algorithms to handle larger meshes. In [8], the authors have already touched upon multicast routing, which is further refined in [11]. Programmable photonic functions are, however, not restricted to only routing.

PPICs can implement linear light-processing functions, e.g., wavelength filtering, phase manipulation, by splitting and recombining light waves. When and how to split, and where to recombine them, is the problem of filter synthesis. A more complete outline of the problem is discussed in Section II. This problem is often regarded as a manual task, but considerable effort has been invested in automating this process. Particle swarm optimization is used in [12] to perform filter synthesis for ring resonators and various other filter structures. In [13], the author proposes both a derivative and derivative-free optimization technique. For the derivative-based technique, a 2- and 3-point approximation is used. In [14], [15], the authors take this a step further and use gradient descent to speed up the minimization algorithm using a differentiable system-level transfer function. The algorithms, as mentioned earlier, have two noticeable shortcomings. The first problem with the algorithms is their runtime. In [12], the authors mention their concern

Received 19 September 2025; revised 25 June 2026; accepted 20 August 2026. Date of publication 26 August 2026; date of current version 16 October 2026. This work was supported in part by Flemish Research Foundation (FWO-Vlaanderen) under Grant G020421 (GRAPHSPAY) and Grant 1100925N and in part by European Research Council (ERC) under Grant 725555 (Photonic-SWARM). (Corresponding author: F. Vanden Kerchove.)

F. Vanden Kerchove, Didier Colle, Wouter Tavernier, and Mario Pickavet are with the IDLab, the Department of Information Technology, Ghent University - imec, 9052 Ghent, Belgium (e-mail: ferre.vandenkerchove@ugent.be).

Zhengqi Gao and Duane S. Boning are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Wim Bogaerts is with the Photonics Research Group, Department of Information Technology, Ghent University - imec, 9052 Ghent, Belgium.

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/JLT.2026.3727869>.

Digital Object Identifier 10.1109/JLT.2026.3727869

about speed, but little other timing-related data is provided. [15] improves the convergence time of their minimization algorithm thanks to their gradient-based optimization; however, they are still hindered by the speed of their filter response calculation. This calculation is based on matrix inversion, a notoriously slow operation that results in running times in the order of hours for medium-sized circuits. In this paper, we address this issue in Section III by revisiting the results of [16], [17] by the iTEAM Research Institute. We modify their approach to derive a symbolic expression for the filter response, enabling the rapid computation of over 100,000 responses in under one second. Our modifications result in a speedup of approximately 10,000, regardless of the number of routing elements involved. Furthermore, we differentiate the filter response symbolically, recognizing the importance of being able to calculate the derivative. However, some care is required to calculate the derivatives. The straightforward but naive way results in unnecessarily high running times. This problem is resolved in Section IV. There, we show that we can reduce the number of multiplications from $\mathcal{O}(m^2)$ to $\mathcal{O}(m \log m)$, where m is an expression of the size of the filter. This sizable reduction in computational complexity allows us to synthesize filters, as discussed in V, between 50 and 250 times faster than the technique discussed in [15].

A second limitation of the aforementioned approaches is their exclusive focus on a single filter at a time. However, a fully functional PPIC must be capable of implementing multiple filters simultaneously. In [12], the authors place a single filter manually; this becomes infeasible to perform effectively for many filters in the same mesh. In [14], the authors show the possibility of applying their technique to two filters, but extending their methods to accommodate many filters while maintaining computational efficiency appears to be a nontrivial challenge. In Section VI, we propose a simulated-annealing-based placement algorithm that works in tandem with the routing algorithm of [10] to place and route every filter. Through our approach, configurations have been found that support more than 20 filters simultaneously.

By addressing these two shortcomings, we present a comprehensive method for configuring programmable photonic integrated circuits with the explicit goal of keeping running times reasonable. In Section VII, we discuss known limitations of our approach and future areas of improvement.

The main contributions of this work are threefold. First, we derive a closed-form symbolic expression for the filter response of compact filter structures and accelerate the response calculation by approximately four orders of magnitude compared to [17]. Second, we introduce a divide-and-conquer scheme that reduces the complexity of computing all response derivatives from $\mathcal{O}(m^2)$ to $\mathcal{O}(m \log m)$, enabling gradient-based synthesis that is 50 to 250 times faster than [15]. Third, we propose a simulated-annealing-based placement algorithm that concurrently places and routes more than 20 filters in a single mesh, whereas prior approaches are limited to one or two.

II. PROBLEM STATEMENT

At the core of the PPIC is the mesh where light is guided in waveguides connected by Tunable Basic Units (TBUs). Each

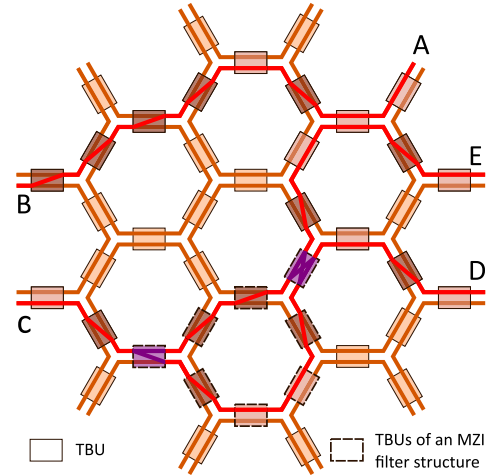


Fig. 1. A small mesh with a hexagonal architecture, realizing one routing demand and one MZI filter structure.

TBU can be configured to be in either bar mode, cross mode, or have a specific coupling factor. The places where light can enter or exit the mesh are called ports. In fig. 1, the mesh is configured such that light is routed from port A to port B, and a light-processing function is realized from port C to ports D and E.

A *routing demand* specifies a pair of ports (input and output) between which light must be guided. A *light-processing function* consists of a single input port and a set of output ports, where each output is associated with a target filter response and an acceptable tolerance. The desired functionality of the chip is specified as a set of routing demands and light-processing functions with associated input and output ports. The objective is to determine a configuration that realizes this functionality on the photonic integrated circuit. The configuration task, therefore, involves ensuring that these specified filter responses are accurately realized between the designated input and output ports. The techniques used in this paper can be extended to multiple input ports. Similarly, a routing demand could be multipoint to multipoint.

Our key idea is to select a specific filter structure for every light-processing function. A filter structure is a fixed collection of couplers that form a single logical unit, realizing specific light-processing functionality. For example, in fig. 1, the filter structure of a Mach-Zehnder interferometer is indicated. Using filter structures is not only computationally more efficient, but it is also theoretically logical. The light-processing functions are independent of each other; hence, no two functions share waveguides. We can thus realize the light-processing functions by independent filter structures. Furthermore, we can select specific filter structures that are known to be able to realize a wide range of filter responses. This way, we reduce the computational search space by limiting our search to compact filter structures, as mentioned in Section III. There, the key benefits of using filter structure are explained in depth.

Once an adequate filter structure is chosen and the appropriate coupling coefficients have been computed, the structure must be placed in the mesh such that no two structures share the same

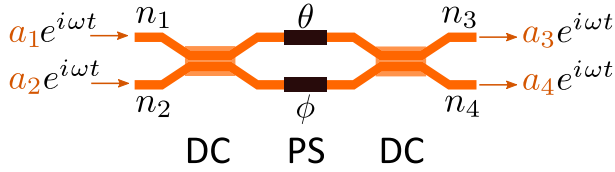


Fig. 2. A schematic of a TBU. This model uses two directional couplers (DC) and a phase shifter (PS) on each arm. The complex coefficients a_1 and a_2 represent the light coming in. The light coming out is represented by a_3 and a_4 .

part of the mesh. Subsequently, we compute the light paths from the designated input ports to the start of the filter structure and from the end of the filter structure to the corresponding output ports. After this final step, our algorithm has obtained a valid configuration of the PPIC. We start with the calculation of the filter response in the next section.

III. FILTER RESPONSE ANALYSIS

We follow a similar strategy to that employed by the iTEAM Research Institute in [16], [17], but improve it such that we can calculate the filter response 4 orders of magnitude faster for similar-sized problems. Our key improvements follow from the idea of using filter structures, as listed below.

- 1) When implementing a filter structure in a programmable photonic mesh, relatively many couplers are in bar or cross mode, and only a few are in coupling mode. This reduces the number of variables and the complexity of the filter response formula, which speeds up construction and calculation.
- 2) Many tried and tested filter structures [12], [18], [19], [20], [21] never have light flowing in two directions in the same waveguide. This allows us to simplify the analytical function, substantially speeding up the calculations. How this is achieved is detailed below.
- 3) Feedback loops are usually only a small part of the entire filter. When filters are cascaded, a well-defined directional flow of light is established. There is recirculation within one filter, but not between two cascaded filters. This enables us to split the filters into separate independent parts, which improves the derivative calculation as detailed in Section IV.

A. Analytical Expression Construction

In this paper, we utilize the common TBU model of a Mach-Zehnder Interferometer with directional couplers and phase shifters, as shown in fig. 2, which is the same model as in [15]. Our results can be easily generalized to any TBU model. We use the term *coupler* and TBU interchangeably.

We start with the base case of one single TBU, as in fig. 2. The phase shifts are parameterized by θ and ϕ , and we assume that their values can be configured independently to any value in $[0, 2\pi]$. Suppose an optical signal with a single frequency $e^{i\omega t}$ and complex coefficient a is coupled into node n_i . Then the resulting signal exiting from node n_j is $\mathbf{F}(n_i, n_j) \cdot a e^{i\omega t}$ where

TABLE I
THE DIFFERENT VALUES FOR A COUPLER IN BAR AND CROSS MODE

input and output	Bar mode	Cross mode
$(n_1 \rightarrow n_3)$	αz^{-1}	0
$(n_1 \rightarrow n_4)$	0	$-i\alpha z^{-1}$
$(n_2 \rightarrow n_3)$	0	$-i\alpha z^{-1}$
$(n_2 \rightarrow n_4)$	$-\alpha z^{-1}$	0

$\mathbf{F}(n_i, n_j)$ is:

$$\begin{aligned}
 \mathbf{F}(n_1, n_3) &= \frac{\alpha}{2} (e^{-i\theta} - e^{-i\phi}) z^{-1} \\
 \mathbf{F}(n_1, n_4) &= \frac{\alpha}{2} (-ie^{-i\theta} - ie^{-i\phi}) z^{-1} \\
 \mathbf{F}(n_2, n_3) &= \frac{\alpha}{2} (-ie^{-i\theta} - ie^{-i\phi}) z^{-1} \\
 \mathbf{F}(n_2, n_4) &= \frac{\alpha}{2} (-e^{-i\theta} + e^{-i\phi}) z^{-1} \quad (1)
 \end{aligned}$$

and $z^{-1} := e^{-i\omega \frac{n_{\text{eff}} L}{c}}$ which represents the phase delay for a single coupler. Here n_{eff} is the effective index of the propagating mode, L is the length of the waveguide in the TBU, c is the speed of light, and α is the transmission loss caused by the waveguides and couplers in the TBU. θ and ϕ are variables that can be tuned by the electronic hardware supporting the photonic chip.

As a consequence of assumption 2) at the beginning of this section, we assume that light is always coupled into n_1 and n_2 and always exits from n_3 and n_4 . We call n_1, n_2 the input nodes, whereas n_3, n_4 are the output nodes. Every node present in our model will have a clear designation as either an input node or an output node, but not both. Given now the complex coefficients of the light being coupled in the input nodes, a_1 and a_2 , the complex coefficients of the light coming out of the output nodes, a_3 and a_4 , can be calculated as follows.

$$\begin{bmatrix} a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} \mathbf{F}(n_1, n_3) & \mathbf{F}(n_2, n_3) \\ \mathbf{F}(n_1, n_4) & \mathbf{F}(n_2, n_4) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

We use the notation $(m \rightarrow n)$, where m is an input node, while n is an output node, to represent the transfer function between m and n . In most filter structures, around two-thirds of the couplers are in bar mode or cross mode. The former occurs when $\theta - \phi = k\pi$ with k an odd integer, whereas the latter occurs when $\theta - \phi = l\pi$ with l an even integer. This greatly simplifies the equations of (1) as summarized in Table I.

An additional phase shift occurs depending on the exact values of θ and ϕ . A coupler in bar or cross mode is always assumed to induce no additional phase shift, except for the couplers following a TBU in coupling mode in the filter structure. These are also modeled to have a possible phase shift of $e^{i\psi}$. By having only one coupler with a phase shift between every two couplers in coupling mode, we reduce the number of variables without diminishing the versatility of the filter structures.

Every filter structure can now be constructed by adding TBUs in one of the following cases.

- 1) Add an unconnected coupler.
- 2) Add a coupler such that one input node is connected with one output node of the already placed structure. This adds one new input and two new outputs.

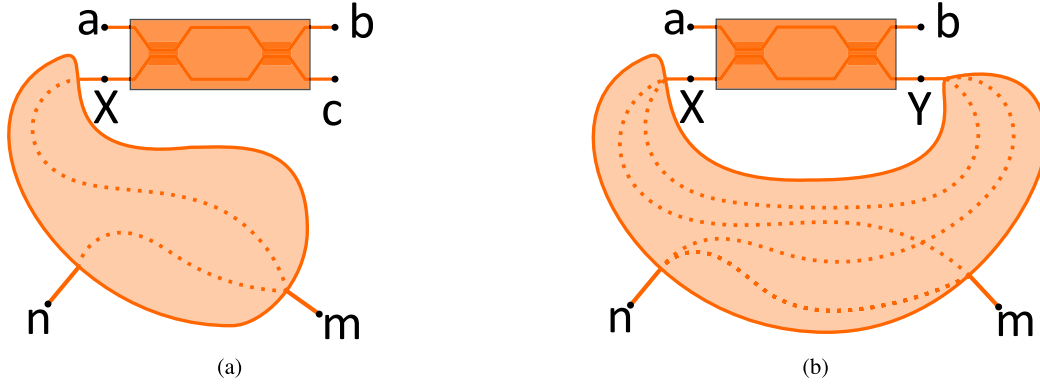


Fig. 3. Different possibilities to add an additional coupler to the mesh. (a) Case 1. (b) Case 3.

- 3) Add a coupler such that two input nodes are connected with two output nodes of the already placed structure. This adds two new outputs.
- 4) Add a coupler such that one input node is connected with an output node of the structure, and one output node of the coupler is connected with an input node of the coupler. This adds one new input and one new output.
- 5) Add a coupler such that two input nodes are connected with two output nodes of the structure, and one output node of the coupler is connected with an input node of the structure. This adds one new output.

Every time a coupler is added, the existing scattering matrix is updated. This is similar, albeit simplified, to what is done by the authors from iTEAM Research Institute in [17]; thus, we only treat cases 1 and 3 here. A sketch of the proof why these five cases suffice can be found in the Appendix A.

1) *Case 1. 1 New Input, 2 New Outputs:* We are working inductively; thus, we have the current (old) scattering response between all input and output node pairs, denoted by $(m \rightarrow n)_o$.

The coupler connects to the existing structure as seen in fig. 3(a). We calculate the new responses for every input and output node $(m \rightarrow n)_n$.

- 1) Existing input nodes to existing output. There are no other paths possible than the ones that are already there, thus:

$$(m \rightarrow n)_n = (m \rightarrow n)_o.$$

- 2) Existing inputs to new outputs. For all inputs m , go to b and c through X , hence:

$$(m \rightarrow b)_n = (m \rightarrow X)_o \mathbf{F}(X, b)$$

$$(m \rightarrow c)_n = (m \rightarrow X)_o \mathbf{F}(X, c).$$

- 3) New inputs to new outputs. This is the response of the coupler itself.

$$(a \rightarrow b)_n = \mathbf{F}(a, b)$$

$$(a \rightarrow c)_n = \mathbf{F}(a, c)$$

- 4) New inputs to existing outputs. There is no path to go from a to the output n , hence:

$$(a \rightarrow n)_n = 0.$$

If this coupler is in cross mode, then $\mathbf{F}(X, c) = 0$ and $(m \rightarrow c)_n = 0$. This effect cascades to every time $(m \rightarrow c)$ is used in a calculation, causing many terms to become 0 and our computation to speed up considerably. Similar effects occur if the coupler is in bar mode, as $\mathbf{F}(X, b) = 0$.

2) *Case 3. 1 New Input, 1 New Output:* In this case, a recirculating loop is possible, and a sum with infinitely many terms appears. We need the following identity for the sum of a geometric sequence:

$$1 + r + r^2 + r^3 + \dots = \sum_{n=0}^{\infty} r^n = \frac{1}{1-r} \quad \text{if } |r| < 1.$$

The coupler now connects as seen in fig. 3(b). The node X is an existing output node, and Y is an existing input node. Hence $(m \rightarrow X)_o$, $(Y \rightarrow n)_o$ and $(Y \rightarrow X)_o$ have already been calculated. Just as in the previous case, 4 types of paths need to be calculated and updated.

- 1) Existing inputs to existing outputs. The following paths between m and n are possible.

- a) The already existing path from m to n : $(m \rightarrow n)_o$.
- b) A path from m to X , then from X to Y , and lastly from Y to n : $(m \rightarrow X)_o \mathbf{F}(X, Y) (Y \rightarrow n)_o$
- c) A path from m to X , from X to Y , a loop going through X back to Y and then from Y to n :

$$(m \rightarrow X)_o \mathbf{F}(X, Y) (Y \rightarrow X)_o \mathbf{F}(X, Y) (Y \rightarrow n)_o.$$

- d) A second loop starting in Y , going through X and ending in Y again, which we write as $L := (Y \rightarrow X)_o \mathbf{F}(X, Y)$ giving an additional

$$(m \rightarrow X)_o \mathbf{F}(X, Y) L^2 (Y \rightarrow n)_o.$$

- e) This continues with higher powers of L being added each time.

Summing all these gives

$$\begin{aligned} (m \rightarrow n)_n &= (m \rightarrow n)_o + (m \rightarrow X)_o \mathbf{F}(X, Y) (Y \rightarrow n)_o \\ &\quad \times (1 + L + L^2 + \dots) \\ &= (m \rightarrow n)_o \\ &\quad + \frac{(m \rightarrow X)_o \mathbf{F}(X, Y) (Y \rightarrow n)_o}{1 - L}. (*) \end{aligned}$$

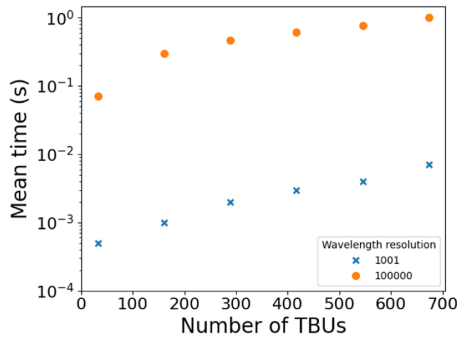


Fig. 4. Time needed to calculate the filter response with our technique for differently sized meshes at a varying number of wavelengths.

- 2) Existing inputs to new outputs. A similar reasoning gives us

$$(m \rightarrow b)_n = \frac{(m \rightarrow X)_o \mathbf{F}(X, b)}{1 - (Y \rightarrow X)_o \mathbf{F}(X, Y)}.$$

- 3) New inputs to new outputs:

$$(a \rightarrow b)_n = \mathbf{F}(a, b) + \frac{\mathbf{F}(a, Y)(Y \rightarrow X)_o \mathbf{F}(X, b)}{1 - (Y \rightarrow X)_o \mathbf{F}(X, Y)}.$$

- 4) New inputs to existing outputs:

$$(a \rightarrow n)_n = \frac{\mathbf{F}(a, Y)(Y \rightarrow n)_o}{1 - (Y \rightarrow X)_o \mathbf{F}(X, Y)}.$$

We again want to stress the effect of the many TBUs that are set beforehand in either bar or cross mode. Here, if this coupler is in cross mode, then $\mathbf{F}(X, Y)$ is equal to 0, and many of these formulas would simplify drastically. For example, (*) simplifies to $(m \rightarrow n)_n = (m \rightarrow n)_o$.

Adding every coupler and updating the scattering matrix according to this procedure produces an analytical expression of the scattering matrix of the filter structure.

B. Experimental Evaluation of the Analytical Expression Implementation

To ensure correctness, we implemented the filter response calculation method from [14] and verified that the numerical results are identical. We compare our results with those of the iTEAM Research Institute, as described in [17]. There, the authors calculate the mesh response at 1001 different wavelengths, requiring approximately 200 seconds for a 400 TBU mesh and around 600 seconds for a 600 TBU mesh. As seen in fig. 4, we can calculate the same number of wavelengths for a similar amount of TBUs in less than 10 milliseconds. 100,000 different wavelengths are calculated in 0.77 seconds for a filter structure that contains 417 TBUs, and 0.84 seconds for a 545 TBU filter. We argue that this speed-up of around 4 orders of magnitude is caused by 1) reduced complexity of the formula and 2) the reduction of the number of variables. The code was written in Python using NumPy and run on an Intel i7-1185G7 at 3 GHz. Compared to the other experiment, we argue that this difference is not caused by a hardware or programming language difference

and is unlikely to be an overstatement of the actual performance difference.

Often devices have fabrication imperfections. We discuss the impact of these imperfections on our filter response simulation in Appendix B.

When performing the filter synthesis, it is not only advantageous to calculate the filter response, but the derivatives as well. In the next section, we show how we calculate this efficiently.

IV. EFFICIENT DERIVATIVE COMPUTATION

We want to calculate the derivatives of the filter response. Given that we derived an analytical expression in the previous section, this is easily achieved. However, there are two derivatives for every coupler in coupling mode, namely one in ϕ and one in θ . Additionally, there is one more derivative for every TBU that realizes a phase shift in either bar or cross mode. Suppose that there are in total m variables. The time to compute the response scales linearly with the number of TBUs and hence linearly with m , as can be seen in fig. 4. We need to compute now m derivatives where every computation takes approximately the same amount of time as the original filter response, $\mathcal{O}(m)$ time. In total, this gives us a quadratic complexity or $\mathcal{O}(m^2)$. In this section, we reduce the complexity to $\mathcal{O}(m \log(m))$, resulting in noticeable time gains. The term $\log$ refers to the base-2 logarithm as customary in complexity theory.

We use the third observation of Section III: feedback loops are often limited in size. This enables us to split the filter into independent, roughly equally sized parts. We can construct the scattering matrix for every independent part separately. If a filter can be split into two scattering matrices A_1 and A_2 , then the full scattering matrix is $A_1 \cdot A_2$. More importantly, the matrices do not share any variables, except for ω and α , which represent the wavelength and transmission loss, respectively. We do not need the derivatives in terms of ω or α . Suppose θ appears in A_1 but not in A_2 . The partial derivative with regards to θ is then $\frac{\partial}{\partial \theta}(A_1 \cdot A_2)$ which simplifies to $(\frac{\partial}{\partial \theta} A_1) \cdot A_2$ through the product rule and the fact that $\frac{\partial}{\partial \theta} A_2 = \mathbf{0}$. Similarly, if θ only appears in A_i :

$$\begin{aligned} \frac{\partial}{\partial \theta}(A_1 \cdot \dots \cdot A_i \cdot \dots \cdot A_N) \\ = A_1 \cdot A_2 \cdot \dots \cdot \frac{\partial}{\partial \theta}(A_i) \cdot \dots \cdot A_N. \end{aligned}$$

This allows us to reuse large parts of the calculation for the filter response in the calculation of the derivatives.

We modify the order of calculation of the main filter response to accelerate the computation of derivatives using a divide-and-conquer strategy. We showcase our calculation method on $A_1, \dots, A_n$ for $n = 8$, and we assume that every scattering matrix contains roughly the same number of variables. We construct a binary tree that splits the calculation into two equal parts. On the lowest level, we calculate and store the pairs $A_1 \cdot A_2$, $A_3 \cdot A_4$, $A_5 \cdot A_6$, and $A_7 \cdot A_8$. This takes 4 multiplications. For the second level, we calculate $(A_1 A_2) \cdot (A_3 A_4)$ and $(A_5 A_6) \cdot (A_7 A_8)$. This only takes 2 multiplications because of the earlier computed results. Lastly, the full expression is

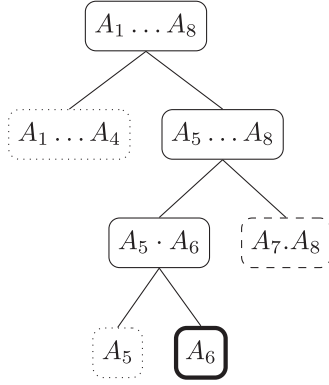


Fig. 5. An example of the tree in which the different parts of the calculation can be easily identified. At every level below the root, one node is needed in the calculation. This results in $\log(n)$ nodes. Here, the tree is shown for the case of A_6 . The dotted nodes form the left part of the calculation, and the dashed nodes are the right part.

$(A_1 A_2 A_3 A_4) \cdot (A_5 A_6 A_7 A_8)$, which requires one additional multiplication. This is $7 = n - 1$ multiplications in total, the same as if we multiplied from left to right.

Suppose we now want to differentiate the main expression with respect to θ , which only appears in A_6 . The desired computation is $A_1 \cdot A_2 \cdot \dots \cdot \frac{\partial A_6}{\partial \theta} \cdot \dots \cdot A_8$. We compute the expression to the left $= (A_1 \cdot \dots \cdot A_5)$ and to the right $= (A_7 \cdot A_8)$ of $\frac{\partial A_6}{\partial \theta}$ separately. In fig. 5 we show the parts needed. On every level below the root, exactly one node will be needed in either the left or right part of the calculation. A node represents a multiplication that has already happened when we calculated the full expression. Here, we already calculated $A_1 \cdot A_2 \cdot A_3 \cdot A_4$, as well as $A_7 \cdot A_8$. The total number of multiplications needed is equal to the number of nodes in the left part minus one, plus the number of nodes in the right part minus one. Since there is precisely one node at each level of the tree beneath the root involved in either the left or right portion of the calculation, the total number of multiplications is given by $\log(n) - 2$. If there is no left or right part, then there are $\log(n) - 1$ multiplications needed. This is the case if we repeat the calculation for A_1 or A_8 . In the example of fig. 5 for A_6 , this translates to multiplying $(A_1 \cdot \dots \cdot A_4) \cdot (A_5)$ which results in $1 = \log(n) - 2$ multiplication. For A_1 and A_8 we need $2 = \log(n) - 1$ multiplications. The total number of matrices in this example is a power of two. If this is not the case, $\log(n)$ should be replaced by $\lceil \log(n) \rceil$, which is the smallest integer that is greater than or equal to $\log(n)$.

For every matrix A_i , we compute the left and right side; this is in total $n \log(n) - 2n + 2$ multiplications. For every derivative, at most two multiplications are needed to properly multiply the stored left part $= (A_1 A_2 A_3 A_4) \cdot A_5$ and right part $(= A_7 A_8)$ with the derivative matrix $\frac{\partial A_6}{\partial \theta}$. Thus the total number of multiplications is at most $n \log(n) - 2n + 2 + 2m = \mathcal{O}(n \log(n) + m)$ where m is the number of derivatives. Without this optimization, the original naive computation needs $\mathcal{O}(n + m \cdot n)$ matrix multiplications. As a last step, we simplify this complexity to an expression of only m . We assume that the filter parts all contain an approximately similar number of variables, thus $n \sim m$. The

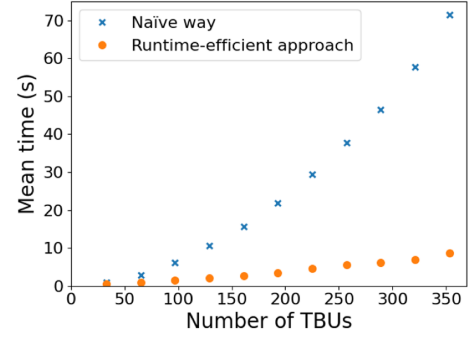


Fig. 6. The time needed to calculate the function and all derivatives of differently sized meshes, comparing the naive way and our optimized method.

total number of multiplications is then $\mathcal{O}(m \log(m))$ instead of $\mathcal{O}(m^2)$ the naive way.

A. Experimental Confirmation of Analytical Speed-Up

This theoretical speed-up is reflected in practical performance gains. In fig. 6, we compare the computation time required to evaluate the filter response and its derivatives using two approaches. The first corresponds to the naive method described in the previous section, while the second employs our divide-and-conquer strategy. The plot clearly illustrates a significant speed-up consistent with the theoretical complexity analysis: the naive approach exhibits quadratic scaling, whereas our method demonstrates a markedly lower increase in running time.

Furthermore, when comparing these results to our implementation of the matrix-based technique proposed in [15], our implementation achieves a speed-up of over two orders of magnitude, underscoring the efficiency of the proposed method.

Armed with the ability to calculate the filter response and derivatives quickly, we turn towards the problem of filter synthesis.

V. FILTER SYNTHESIS

Currently, the filter response can be calculated, given a specified configuration. We want to tackle the opposite problem, namely, calculate the configuration of the filter, given a desired filter response. In this section, we define how the filter response is specified and explain how this is handled.

A fundamental problem is now to establish the desired filter response between two ports, i and j . This desired filter response is specified as a complex function $U_{ij}(\omega)$ which contains both the magnitude and phase response on a domain $[\omega_{ij}^{\min}, \omega_{ij}^{\max}]$. We use $O_{ij}(\omega, \mathbf{p})$ to denote the frequency response between those two ports for a given frequency ω and configuration of the filter structure $\mathbf{p}$. If only the magnitude response is relevant, then $U_{ij}(\omega)$ is a real function, and likewise, only the magnitude of $O_{ij}(\omega, \mathbf{p})$ is kept. Likely, the desired filter response cannot be exactly implemented in the mesh. Hence, an acceptable error on the approximation of the desired filter response is supplied, e.g., a maximal error and accompanying error function, or a maximal passband loss, and/or a maximal stopband gain, if and in which band ripple is allowed, etc.

Several error functions are possible and dictate how the implemented filter response can differ from the desired filter response. As an example, take the error function that measures the square of the absolute value of the differences. The error is then calculated as follows. Take N equidistant frequency points $\{\omega_{ij1} = \omega_{ij}^{\min}, \omega_{ij2}, \dots, \omega_{ijN} = \omega_{ij}^{\max}\}$ and define the error

$$\text{Error}(\mathbf{p}) := \sum_{k=1}^N |O_{ij}(\omega_k, \mathbf{p}) - U_{ij}(\omega_k)|^2.$$

As mentioned in [15], a logarithmic error function $|\log(O_{ij}(\omega_k, \mathbf{p})) - \log(U_{ij}(\omega_k))|$ is better suited to focus on stopband losses or a weighted sum of the two as proposed in [18], balances the needs of the stop- and passband. Furthermore, for a stopband, once the response dips below the required loss, these points do not contribute to the error. In [14], the authors remark that this could theoretically cause problems as the error function is no longer differentiable everywhere. However, the error function is still differentiable almost everywhere, and in practice, this does not pose a problem.

We propose a second improvement. If the desired filter response is a pass- or stopband with maximum ripples allowed, we convert the desired filter response to an elliptical filter. Theoretically, no other filter of the same order can have a faster transition in gain between the passband and stopband. Furthermore, in [18], the authors propose a filter structure that can realize this type of filter response. If the ripple requirements differ, then a Butterworth, Chebyshev Type I, or Type II filter can also be realized in the mesh. From the ripple and gain restrictions, a clear starting point and minimal order for the filter structure can be calculated by the function “ellipord” in Scipy [22].

The following approach chooses the appropriate filter structure. A possible filter structure is selected from a list of filter structures, which increases in complexity and size. If the current filter structure cannot realize the desired filter response within the specified constraints, then the next structure is tried. When needed, structures are cascaded such that the output of one structure is fed into the input of the next.

A. Evaluation Relative to State-of-The-Art Technique

Firstly, we compare our filter synthesis to our implementation of the approach described in [14]. We run both filter synthesis methods, varying the filter structure, the desired filter response and the error function. For each combination, the synthesis is repeated five times with randomized initial parameter values, and all runs are included in the reported statistics. The filter structures used in these benchmarks are composed of a small set of well-established building blocks: cascaded double ring-loaded Mach–Zehnder Interferometers (MZI), cascaded with balanced and imbalanced MZIs and/or ring resonators [18], [21]. The individual structures differ in the number and combination of these blocks, which determines the number of TBUs reported on the horizontal axis of fig. 7 and fig. 8. Cascading additional stages yields sharper, higher-order responses at the cost of a larger structure, whereas a single ring resonator or double ring-loaded MZI provides a compact, lower-order filter. We plot the number of TBUs in the filter structure and the relative speed-up. Both

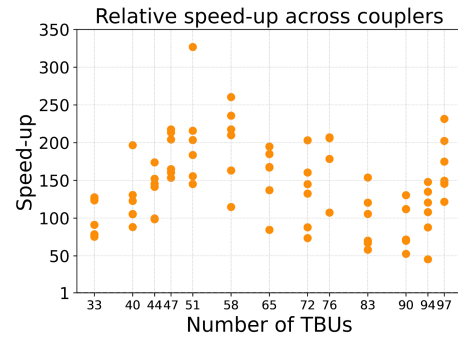


Fig. 7. A comparison of our synthesis approach with the approach implemented in [15].

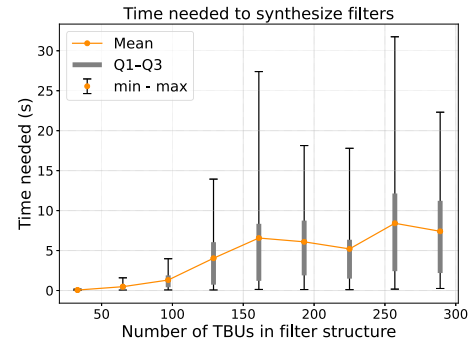


Fig. 8. The time needed to calculate an acceptable solution on varying filter structures and for various desired filter responses.

use the same multivariate optimization technique; thus, the error of the produced filter response to the desired filter response is the same. T_g represents the time taken by their approach, while T_o represents the time taken by our approach. The speed-up is expressed as $\frac{T_g}{T_o}$. As shown in fig. 7, our method achieves a significant reduction in execution time in empirical tests. Our computation time is generally between 50 and 250 times faster for a wide variety of different desired filter responses and filter structures.

In fig. 8, the average time is shown to synthesize a desired filter response on varying filter structures of different sizes. Even for very large filter structures, good coupling ratios are calculated within 30 seconds. As the filters increase in size, the total time required remains stable. This can be explained by the fact that it is easier for larger filters to achieve a desired filter response within the specified error. The number of iterations that the minimization algorithm needs decreases, and thus the total run time decreases or stays largely the same.

The speed-up reported in fig. 7 varies considerably at a fixed number of TBUs and decreases on average as the filter structures grow. Both effects follow from the structure of the computation rather than from measurement noise. Our method incurs a one-time cost C_s to construct the analytical expression of a filter structure s , after which each evaluation of the response and its derivatives is cheap, whereas the matrix-based baseline of [14] pays a large cost at every evaluation. Writing $T_o \approx C_s + Ne_o$ and $T_g \approx Ne_g$, with N the number of optimizer iterations and $e_o \ll e_g$ the per-evaluation costs, the

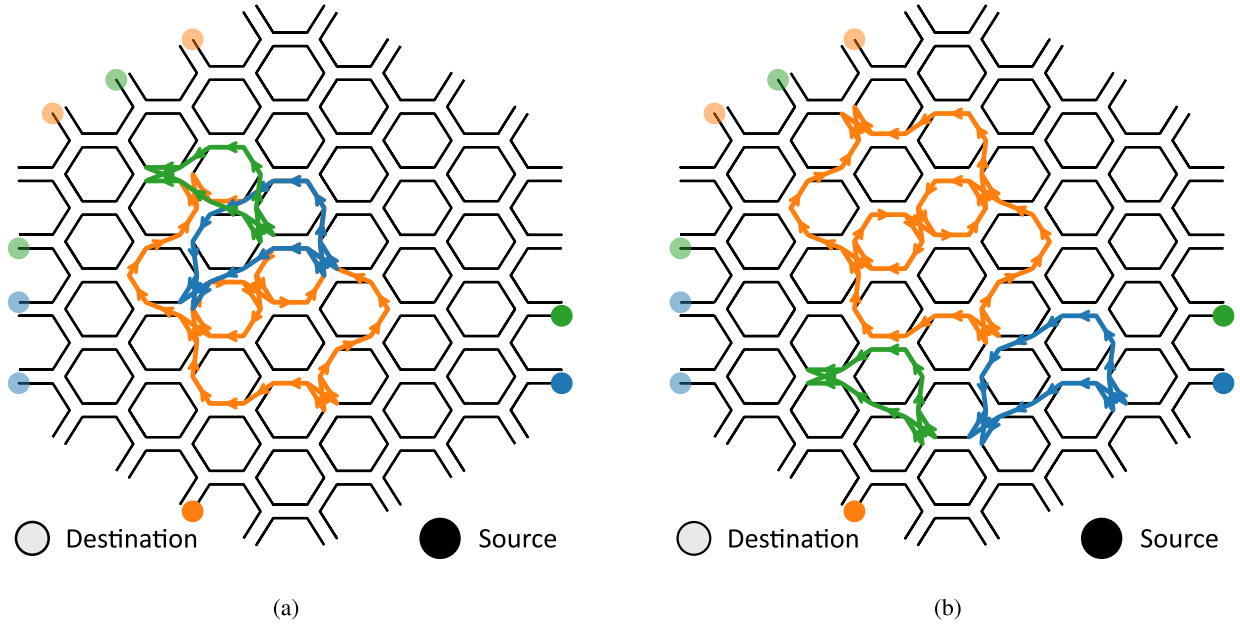


Fig. 9. (a) An initial starting configuration. (b) A conflict-free placement. However, when trying to route the wiring paths, this results in a failure.

speed-up equals $Ne_g/(C_s + Ne_o)$. The spread at a fixed size arises because structures of equal TBU count differ in C_s , e_o , and e_g , depending on the fraction of couplers held in bar or cross mode and the size of their feedback loops, and because N varies with the desired response and error function.

The decrease with size follows from the interplay between C_s and N . As noted above, larger structures reach the desired response in fewer iterations, so N decreases with size. The baseline cost $T_g = Ne_g$ consists entirely of expensive per-iteration work and therefore benefits strongly from this reduction. Our method benefits far less, since its per-iteration cost e_o is already small and its time is increasingly set by the construction cost C_s , which grows with structure size and is independent of N . This growth of C_s offsets the reduction in Ne_o , which is why our total synthesis time increases only mildly as seen in fig. 8 even as the structures grow. As filters become larger, the baseline time consequently falls toward our construction-dominated time and the observed speed-up diminishes. We emphasize that this construction cost is paid only once: in settings where a structure is evaluated many times, the full per-evaluation advantage e_g/e_o is recovered.

VI. PLACEMENT ALGORITHM

This section adapts, simplifies, and expands on the methods presented earlier in [23]. Once a filter structure has been synthesized for each desired filter response, it must be strategically placed within the photonic mesh. Subsequently, the corresponding *wiring paths*, i.e., the connections from the input port to the beginning of the filter, and from the end of the filter to the output ports, must be routed. Both placement and routing are computationally hard problems in general. To address this complexity, we adopt a simulated-annealing-based methodology comprising three successive stages. Each stage tackles an

increasingly complex subproblem, building upon the output of the preceding one. Although we only discuss our algorithm here in the context of a hexagonal architecture, the results discussed here can be transferred to any regular architecture, be it triangles, squares, or a more esoteric architecture such as ring-connected straight lines introduced in [24].

Firstly, the filter structures require an initial configuration that serves as a good starting point. Every filter structure is placed in the mesh such that the wiring paths are minimized, without considering the other structures. This is easily calculated by greedily minimizing the distance for every rotation and reflection of each filter structure and then choosing the best candidate. In fig. 9(a), the initial placement is shown.

The second stage searches for a placement such that no two filter structures have a conflict. Two filters have a conflict when they both assign the same TBU a different mode or coupling coefficient. This criterion considers shared TBUs only; the impact of optical crosstalk between structures is discussed in Appendix B. For the general placement problem, simulated annealing [25] is used. The pseudo-code is shown in Algorithm 1.

Simulated annealing is mainly characterized by its starting point s , the neighborhood function N , and the score function f . Besides this, in Section VI-C, we discuss the effects of the choice of the starting temperature, the decay parameter α , and the number of iterations on the quality of the output. At every iteration, a random neighbor from the neighborhood function is selected. This consists of choosing a filter structure that has a conflict and moving that structure in one of the following ways:

- 1) Shift the filter in a random direction over a small distance.
- 2) Mirror the filter over a random axis in the mesh.
- 3) Rotate the filter around a nearby hexagon.

This produces a first placement, but without any wiring paths, see fig. 9(b). If no placement is found at this stage, we can assume there is no feasible solution and terminate the program.

Algorithm 1: Simulated Annealing.

```

1: Input: Initial starting point  $s$ , initial temperature  $T$ ,
   cooling schedule  $\alpha$ , score function  $f$ 
2: Output: Approximate global optimum solution  $s^*$ 
3:  $s^* \leftarrow s$ 
4: while  $s$  has conflicts do
5:   choose neighbor  $s'$  in  $N(s)$ 
6:    $\Delta \leftarrow f(s') - f(s)$ 
7:   if  $\Delta < 0$  then
8:      $s \leftarrow s'$ 
9:   else
10:    Accept  $s'$  with probability  $e^{-\Delta/T}$ 
11:   end if
12:   if  $f(s) < f(s^*)$  then
13:      $s^* \leftarrow s$ 
14:   end if
15:   Update temperature:  $T \leftarrow \alpha \cdot T$ 
16: end while
17: return  $s^*$ 

```

If a placement is found, the program continues to the third stage.

The third stage has as a goal finding a feasible configuration, that is, both the placement of the filter structures and the routing of the corresponding wiring paths. Routing these, as studied in [10], is computationally expensive. Therefore, we adapt the scoring function to minimize the number of times this is attempted. The scoring function checks the following:

- 1) Do the filter structures have a conflict-free placement?
- 2) Can every wiring path be routed independently?
- 3) Can all wiring paths be routed at the same time?

The other routing demands, where no filtering is happening, are also included in the last check. It is only this last check that uses a computationally expensive routing algorithm. The other two checks are significantly faster; hence, if an earlier check has a negative answer, the later conditions are not checked in this iteration. Furthermore, the state of the mesh is recorded for the third check, and the result is cached for future reference. Simulated annealing can reexamine the same state multiple times, and recomputing the wiring paths for the same state should be avoided. Around 10% to 20% of the time, the routing algorithm would run, and results are retrieved from the cache, instead of being recomputed.

Each iteration, a filter structure is chosen to be moved. The list of possible candidates to move consists of all the filters that caused one of the above conditions to fail. If all three conditions are met, a feasible solution is found and returned, for example, in fig. 10. One can appreciate the non-triviality of this solution. This configuration would likely have never been discovered if manual placement had been used for a relatively simple three-filter problem. Granted, there are plenty of other solutions. If no feasible solution is found after a set number of iterations, the algorithm returns failure. However, this does not guarantee that no feasible solution exists.

In the next section, we discuss the results of this algorithm.

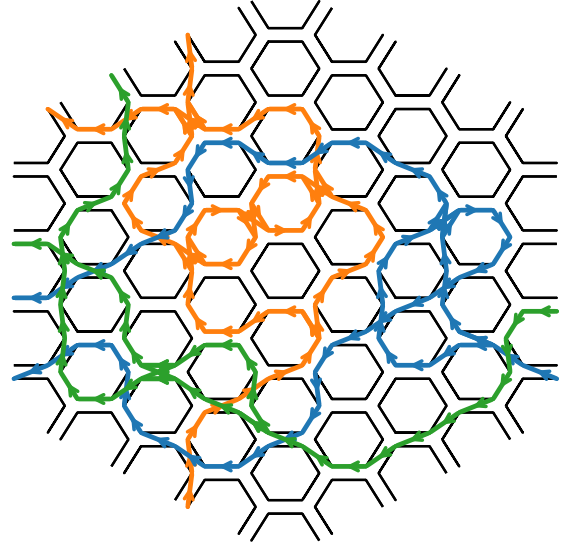


Fig. 10. The final configuration with wiring paths, realizing the desired filter responses.

A. Benchmark Design

Currently, as mentioned in [23], there is a lack of benchmark test sets that accurately reflect realistic usage scenarios of programmable photonic integrated circuits. Furthermore, the absence of comparable algorithms in the literature leaves us without a proper baseline. To address this limitation, we construct two distinct types of test sets.

The first test set consists of simple yet feasible problem instances. We construct a list of possible filter structures and corresponding ports. From this list, we greedily construct a subset by placing and routing these sequentially until no additional filters can be accommodated within the mesh. Each resulting configuration is guaranteed to be solvable and is subsequently included in the test set. This dataset is primarily used to assess the ability of our method to identify valid solutions systematically.

The second test set is generated randomly. Input and output ports are selected uniformly at random from the complete set of available ports. The mesh size, as well as the number of problem instances, varies across this dataset. For this test set, we report only the run-time performance of the algorithm on instances where a solution is found. We do not report on the success rate, as the feasibility of each instance is unknown. Although an integer programming formulation capable of solving these instances optimally was developed, its computational complexity is prohibitively high for usage on problem sets of practical size.

B. Evaluation of the Placement Algorithm

On the first sequentially generated test set, which contains 50 large problems with approximately 13 filters each, our method achieves a 100% success rate, finding a solution for every instance. This outcome is consistent with expectations, as the construction of this specific test set results in meshes that are not too densely populated.

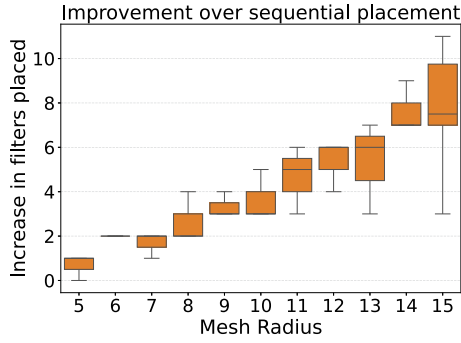


Fig. 11. The improvement of our method over sequential placement in absolute numbers. On average, our method can place 53% more filters in the same mesh.

Furthermore, we repeat this greedy procedure to construct a maximally placeable set as described for the construction of the first test set. But here, we use our method instead of the sequential placement. This approach can place, on average, 53% more filters in the same mesh. On larger meshes with radius $r \geq 12$, this increases to 61%. In fig. 11, we show the number of filters our method can place more than the sequential method on meshes of varying radii. Our method consistently places more filters, and this increase is more pronounced as the meshes become larger.

To effectively report runtime performance as a function of both mesh size and the number of filters, it is crucial to consider the interdependence between these parameters. Keeping the mesh size constant while increasing the number of filters often leads to infeasible instances due to constrained routing capacity. Conversely, enlarging the mesh while keeping the number of filters fixed increases the solution space, simplifying the problem and resulting in reduced runtimes. To normalize this trade-off and allow for fair performance comparisons, we introduce the concept of *density*.

Density quantifies how intensively the routing resources are utilized. Specifically, we construct a solution in which each functionality is routed independently of the others, and count the number of routing resources used, denoted R_u . This is compared against the total number of available routing resources in the mesh, R_t . We define the density as the ratio $\frac{R_u}{R_t}$, representing the minimum fraction of resources that any feasible solution must occupy.

To ensure the benchmark problems are non-trivial and present a meaningful computational challenge, we restrict our analysis to instances with a density of at least 0.4. We empirically observed that instances below this threshold are solved almost immediately and would dilute the timing statistics, whereas this threshold retains computationally challenging instances while providing a sufficient number of samples per mesh radius for the reported statistics to be representative.

Fig. 12 illustrates the runtime required for solving the placement problem across mesh radii. The time axis is logarithmic and individual problem instances are overlaid on the boxplots. The runtime of the simulated annealing procedure on dense instances follows a heavy-tailed distribution: most instances are solved quickly, while a few require substantially longer. The median runtime grows with the mesh size. On small- and medium-sized

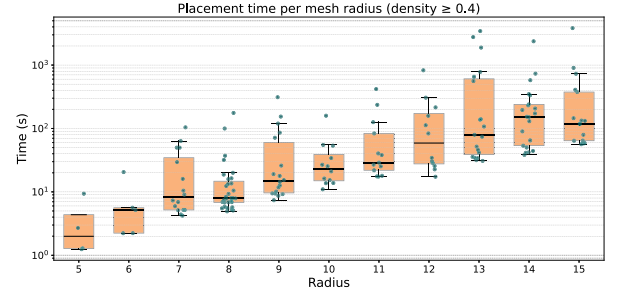


Fig. 12. Time needed to place various problems that are deemed difficult (density ≥ 0.4) for varying mesh radii. Individual problem instances are overlaid on the boxplots.

TABLE II
TOP 5 CONFIGURATIONS BY AVERAGE NUMBER OF FILTERS PLACED

T_0	T decay	iterations	restarts	avg filters	avg time
20	0.95	400	1	16.3	324s
20	0.90	400	1	15.7	1624s
20	0.90	700	1	15.7	608s
20	0.90	700	0	15.3	680s
45	0.90	100	1	15.3	579s

meshes, solutions are typically obtained within seconds to a few minutes. For larger meshes, the required runtime increases substantially. Although this increase is not ideal, a key observation is that feasible solutions are still found within a reasonable timeframe. Notably, for the largest problem instances, more than 20 filters are placed simultaneously. This is substantially more than what the current state of the art proposes to solve.

C. Simulated Annealing Schedule Design

To evaluate the impact of simulated annealing parameters on solution quality, we systematically vary the starting temperature T_0 , decay parameter, number of iterations, and number of restarts. Table II presents the top configurations based on the average number of filters placed. The following parameters are considered, and all combinations are examined. The starting temperature varies between $[20, 45, 70]$ and the decay parameter is either 0.9, 0.95, or 0.99. The number of iterations is 100, 400, or 700. There is either no restarting or restarting once, which resets the temperature and number of iterations, but continues with the current best solution. In practice, this means that the maximum number of iterations doubles as well. The test set is a reduced version of the one in the previous section.

The results indicate that starting temperature has a modest effect, with lower temperatures (e.g., 20) slightly outperforming higher ones (e.g., 70) in terms of filter placements. This suggests that a more conservative exploration at the start can lead to improved outcomes in this setting.

The decay parameter, which controls the rate of temperature decrease, had a minor effect in isolation. Configurations with the decay parameter values of 0.90 and 0.95 both appeared frequently among the top performers, indicating that moderate decay rates may be sufficient to balance exploration and exploitation.

In contrast, the number of iterations had a more pronounced influence: configurations with 400 or more iterations generally outperformed those with fewer, underscoring the benefit of prolonged optimization. However, performance gains between 400 and 700 iterations appeared marginal, suggesting diminishing returns beyond a certain point.

The number of restarts also contributed positively, with most top-performing configurations utilizing at least one restart. Restarts likely helped the search escape local optima, especially in more constrained configurations with fewer iterations.

Overall, while no single parameter dominates performance, the combination of moderate starting temperatures (around 20–45), controlled decay that is not too slow (0.90–0.95), sufficient iterations (400+), and one restart yields consistently strong results. This also highlights the robustness of our method, as it is not dependent on one single parameter set.

VII. DISCUSSION AND FUTURE WORK

We present a general approach to configuring programmable photonic integrated circuits that supports the routing and placement of multiple light-processing functions simultaneously. The introduction of filter structures gives an increase of four orders of magnitude in the calculation speed of the filter response in comparison to the state of the art. Calculating the derivatives incurs limited additional overhead thanks to the introduction of the divide-and-conquer approach. The combination of these two parts results in a filter synthesis routine that is 50 to 250 times faster than prior work.

Filter structures are not only a key part in filter synthesis, but they are also crucial for the placement and routing algorithm. For small and medium-sized meshes, our simulated annealing algorithm finds solutions in minutes and clearly outperforms a more basic implementation. For larger meshes, our approach can easily realize more than 20 light-processing functions in the same mesh.

One of the largest shortcomings of our approach is its dependency on the regularity of the mesh for the placement algorithm. The algorithm relies heavily on the numerous symmetries present in regular meshes. Currently, most papers study regular meshes; however, adding irregularity can enhance the generality of the chip. Adapting our method to an irregular mesh appears to be a non-trivial task, likely requiring a redesign of the placement algorithm.

A growing body of work employs machine-learning-assisted methods for the synthesis of photonic structures, frequently reporting strong performance in terms of insertion loss, coupling, and design accuracy, and notably without being restricted to a predefined catalogue of structure designs [26], [27]. These approaches typically operate at the level of the physical device geometry, freely shaping waveguides, couplers, or metasurface features, whereas our method operates at the level of configuring a fixed, reconfigurable mesh. The two are therefore complementary rather than competing, and our framework can support learning-based design in two respects. First, because our method is analytical, deterministic, and produces both filter responses and configurations rapidly and reproducibly, it can serve as a

strong baseline against which learning-based synthesis methods are benchmarked, both in solution quality and in runtime. Second, the speed of our response calculation makes it well suited as a data-generation engine: large, labeled datasets pairing mesh configurations with their corresponding filter responses can be produced at low cost, supplying the training and validation data that learning-based models typically require. We consider the synergy between such analytical frameworks and learning-based synthesis a promising direction for future work.

Finally, while all results in this work are obtained through simulation, the underlying TBU model is the standard model used in prior experimental demonstrations of programmable meshes [12], [13], and the configurations produced by our method can serve as starting points for on-chip self-configuration and fine-tuning procedures that compensate residual deviations of the fabricated device.

APPENDIX A

PROOF: THE FIVE CASES SUFFICE.

All the cases always have at least one output that is not connected to the thus-far constructed structure. Thus, we need to demonstrate that we can build every filter structure by adding couplers, such that every time a coupler is added, it has at least one output that is free. The key trick is realizing that the overall filter structure always has inputs where the light is meant to be coupled in, and outputs where the light exits the filter structure. Now, where the light exits the structure, this coupler has at least one output not connected to others. This coupler can thus be placed last. Now, looking at the structure without this last coupler, we again have light exiting the structure somewhere (at least the coupler feeding the light into the now removed coupler), and thus we can recursively remove coupler after coupler that always has at least one output free, until there are no more couplers. Constructing the filter structure is now the reverse of the removal process and always uses one of the 5 cases.

APPENDIX B

ROBUSTNESS TO FABRICATION VARIATIONS

Our results are obtained under an idealized device model, whereas fabricated devices exhibit waveguide loss, random phase errors, and non-ideal coupling. A key observation is that a programmable mesh is tuned after fabrication: synthesis can therefore be carried out against a characterized device rather than the idealized model, so that variations which are measurable or tunable are absorbed into the model instead of degrading the realized response. Waveguide loss is captured directly by the per-TBU transmission parameter α , for which measured values can be substituted. Random phase errors likewise remain within our model, since θ and ϕ are free parameters, and are compensated by the tunable phase shifters through calibration [28], [29].

Non-ideal coupling is the limiting case and defines the validity boundary of our method. A TBU deviating from its nominal bar or cross state introduces a parasitic light path, and outside the filter structures such paths constitute exactly the bidirectional propagation that our analytical formulation deliberately excludes for speed. The effect of this parasitic coupling has

been characterized experimentally in [30], where the authors also propose a mitigation strategy: leaving all unused couplers in cross mode, which partially suppresses the parasitic interference. Our method is therefore accurate in the regime of negligible parasitic coupling and a characterized device, within which loss and phase variations are absorbed by the model and by post-fabrication tuning.

REFERENCES

- [1] W. Bogaerts and A. Rahim, "Programmable photonics: An opportunity for an accessible large-volume PIC ecosystem," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 5, pp. 1–17, Sep./Oct. 2020.
- [2] D. Pérez et al., "Multipurpose silicon photonics signal processor core," *Nature Commun.*, vol. 8, no. 1, Sep. 2017, Art. no. 636. [Online]. Available: <https://www.nature.com/articles/s41467-017-00714-1>
- [3] D. Pérez-López et al., "General-purpose programmable photonic processor for advanced radiofrequency applications," *Nature Commun.*, vol. 15, no. 1, Feb. 2024, Art. no. 1563. [Online]. Available: <https://www.nature.com/articles/s41467-024-45888-7>
- [4] D. Marpaung, J. Yao, and J. Capmany, "Integrated microwave photonics," *Nature Photon.*, vol. 13, no. 2, pp. 80–90, Feb. 2019. [Online]. Available: <https://www.nature.com/articles/s41566-018-0310-5>
- [5] N. Harris et al., "Linear programmable nanophotonic processors," *Optica*, vol. 5, pp. 1623–1631, Dec. 2018.
- [6] I. Aharonovich, K. B. Crozier, and D. Neshev, "Programmable integrated quantum photonics," *Nature Photon.*, vol. 20, no. 3, pp. 254–265, Mar. 2026. [Online]. Available: <https://www.nature.com/articles/s41566-025-01830-x>
- [7] B. J. Shastri et al., "Photonics for artificial intelligence and neuromorphic computing," *Nature Photon.*, vol. 15, no. 2, pp. 102–114, Feb. 2021. [Online]. Available: <https://www.nature.com/articles/s41566-020-00754-y>
- [8] X. Chen, P. Stroobant, M. Pickavet, and W. Bogaerts, "Graph representations for programmable photonic circuits," *J. Lightw. Technol.*, vol. 38, no. 15, pp. 4009–4018, Aug. 2020.
- [9] A. López, D. Pérez, P. DasMahapatra, and J. Capmany, "Auto-routing algorithm for field-programmable photonic gate arrays," *Opt. Exp.*, vol. 28, no. 1, Jan. 2020, Art. no. 737.
- [10] F. Vanden Kerchove, X. Chen, D. Colle, W. Tavernier, W. Bogaerts, and M. Pickavet, "An automated router with optical resource adaptation," *J. Lightw. Technol.*, vol. 41, no. 18, pp. 5807–5819, Sep. 2023.
- [11] X. Wang, F. Vanden Kerchove, R. Raikar, M. Pickavet, W. Bogaerts, and D. Stroobandt, "A novel connection-based multicasting router for programmable photonic circuits," *J. Lightw. Technol.*, vol. 43, no. 6, pp. 2518–2527, Mar. 2025.
- [12] D. Pérez-López, A. López, P. DasMahapatra, and J. Capmany, "Multi-purpose self-configuration of programmable photonic circuits," *Nature Commun.*, vol. 11, no. 1, Dec. 2020, Art. no. 6359.
- [13] D. Pérez López, "Programmable Integrated Silicon Photonics Waveguide meshes: Optimized designs and control algorithms," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 2, pp. 1–12, Mar. 2020.
- [14] Z. Gao, X. Chen, Z. Zhang, U. Chakraborty, W. Bogaerts, and D. S. Boning, "Automatic synthesis of light-processing functions for programmable photonics: Theory and realization," *Photon. Res.*, vol. 11, no. 4, Apr. 2023, Art. no. 643.
- [15] Z. Gao, X. Chen, Z. Zhang, U. Chakraborty, W. Bogaerts, and D. S. Boning, "Gradient-based power efficient functional synthesis for programmable photonic circuits," *J. Lightw. Technol.*, vol. 42, no. 17, pp. 5956–5965, Sep. 2024.
- [16] D. Pérez and J. Capmany, "Scalable analysis for arbitrary photonic integrated waveguide meshes," *Optica*, vol. 6, no. 1, Jan. 2019, Art. no. 19.
- [17] E. Sanchez, A. Lopez, and D. Perez-Lopez, "Simulation of highly coupled programmable photonic circuits," *J. Lightw. Technol.*, vol. 40, no. 19, pp. 6423–6434, Oct. 2022.
- [18] M. Wang, C. Xiangfeng, U. Khan, and W. Bogaerts, "Programmable wavelength filter with double ring loaded MZI," *Sci. Rep.*, vol. 12, 2022, Art. no. 1482.
- [19] K. Jingui and M. Oguma, "Optical half-band filters," *J. Lightw. Technol.*, vol. 18, no. 2, pp. 252–259, Feb. 2000.
- [20] S.-Y. Cho and R. Soref, "Apodized SCISSORS for filtering and switching," *Opt. Exp.*, vol. 16, no. 23, 2008, Art. no. 19078.
- [21] J. C. C. Mak, A. Bois, and J. K. S. Poon, "Programmable multiring butterfly filters with automated resonance and coupling tuning," *IEEE J. Sel. Topics Quantum Electron.*, vol. 22, no. 6, pp. 232–240, Nov./Dec. 2016.
- [22] P. Virtanen et al., SciPy 1.0 Contributors, "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020.
- [23] F. Vanden Kerchove, D. Colle, W. Tavernier, W. Bogaerts, and M. Pickavet, "Routing and placing filters for programmable photonics," in *Proc. 25th Anniversary Int. Conf. Transparent Opt. Netw.*, 2025, pp. 1–4.
- [24] F. Vanden Kerchove, D. Colle, W. Tavernier, W. Bogaerts, and M. Pickavet, "Routing impact of architecture and damage in programmable photonic meshes," *Photon. Res.*, vol. 12, no. 9, Sep. 2024, Art. no. 1999.
- [25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
- [26] W. Ma, Z. Liu, Z. A. Kudyshev, A. Boltasseva, W. Cai, and Y. Liu, "Deep learning for the design of photonic structures," *Nature Photon.*, vol. 15, no. 2, pp. 77–90, Feb. 2021. [Online]. Available: <https://www.nature.com/articles/s41566-020-0685-y>
- [27] J. Jiang, M. Chen, and J. A. Fan, "Deep neural networks for the evaluation and design of photonic devices," *Nature Rev. Mater.*, vol. 6, no. 8, pp. 679–700, Aug. 2021. [Online]. Available: <https://www.nature.com/articles/s41578-020-00260-1>
- [28] R. Tao, J. Qiu, Y. Chen, Y. Li, H. Guo, and J. Wu, "A parameter-space-reduction-technique-assisted optimization method for characterizing recirculating waveguide meshes," *Chin. Opt. Lett.*, vol. 23, no. 2, Feb. 2025, Art. no. 021301. [Online]. Available: <https://opg.optica.org/col/abstract.cfm?uri=col-23-2-021301>
- [29] S. Bandyopadhyay, R. Hamerly, and D. Englund, "Hardware error correction for programmable photonics," *Optica*, vol. 8, no. 10, Oct. 2021, Art. no. 1247.
- [30] I. Zand and W. Bogaerts, "Effects of coupling and phase imperfections in programmable photonic hexagonal waveguide meshes," *Photon. Res.*, vol. 8, no. 2, pp. 211–218, Feb. 2020. [Online]. Available: <https://opg.optica.org/prj/abstract.cfm?uri=prj-8-2-211>