Graph Representations for Programmable Photonic Circuits

Xiangfeng Chen, *Student Member, IEEE*, Pieter Stroobant, Mario Pickavet, and Wim Bogaerts, *Senior Member, IEEE, Member, OSA*

***Abstract*—We propose graph representations for reconfigurable photonic mesh circuits. Waveguide mesh circuits are abstracted into a graph to highlight the connectivity and topology. We model the optical ports as graph nodes. Performance metrics for each connection are incorporated into the edge attributes in categories such as propagation loss, crosstalk penalty, power consumption, phase accumulation, and so on. We use three types of graph representations for tunable couplers to model the flow of light and create a circuit graph representation to an example hexagonal mesh. The representation should respect the physics of waveguide circuits (e.g. directional flow of light). Of the three types, the directed graph with eight artificial nodes performs best for solving light distributions with feedback paths. This graph representation is demonstrated in four distribution cases: a single pair input-output, multi-pair inputs and outputs without collisions, a single input to multiple outputs (distribution), and multiple distributions without collisions. The programming tolerance against malfunctioning tunable elements is also demonstrated. With this circuit representation, we can reduce all these distribution cases to different graph problems and leverage a wealth of existing algorithms developed in graph theory to program the photonic mesh. Thus we provide a systematical strategy to design and program complex reconfigurable photonic circuits, especially in photonic meshes with feedback paths.**

***Index Terms*—Graph theory, optical routing, programmable photonics, photonic integrated circuits, silicon photonics.**

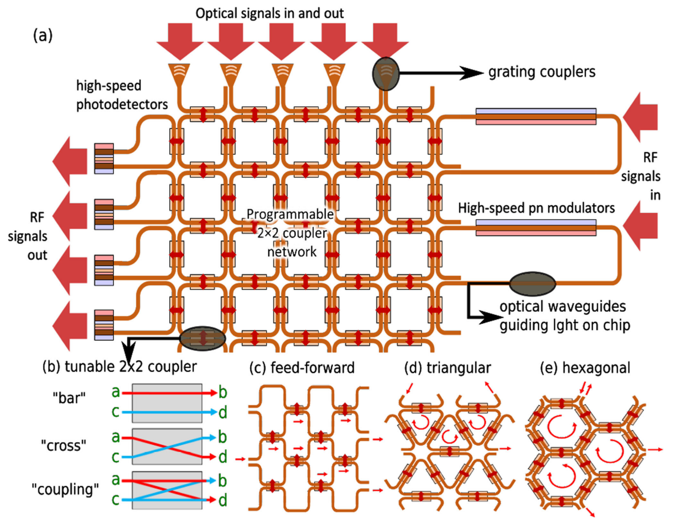

Fig. 1. Programmable photonic circuits: (a) the circuit connects inputs, outputs and functional blocks (modulators, detectors). (b) internally it has couplers that can be put in different states. The topology can be forward-only (c) [12] or use feedback rings such (a) square [9], (d) triangular [8] or (e) hexagonal ring meshes [10], [11].

I. Introduction

In the past few years, the field of programmable photonics has gained a lot of momentum, evolving from a blueprint to a realistic platform for new research and applications, especially in the domains of quantum optics and neuromorphic computing [1], [2]. Similar to an electronic integrated circuit, a photonic integrated circuit (PIC) integrates many optical components on a single chip. It is especially the silicon photonics technology platform that is enabling the rapid scaling of PICs, allowing the placement of hundreds or thousands of functional elements on a single chip [3]. Today, most PICs are application-specific PICs (ASPIC), and the light paths are mostly fixed during chip design. Programmable photonic circuits is introducing a change here: with reconfigurable optical pathways, a programmable PIC can in principle realize the same functions as a large variety of ASPICs. Programmable PICs consist of a mesh of electrically tunable optical couplers where the coupling ratio can be programmed at the time of operation [4]–[6]. This is illustrated in Fig. 1(b): a tunable coupler can be in bar state (no coupling), cross state (full coupling), or in a fractional coupling state. The maturing PIC technology makes it possible to integrate many such couplers in a large-scale connected mesh of waveguides and thus distribute light through a multitude of paths which are reconfigurable. We can generally identify two large classes of such programmable PICs: Forward-only meshes and meshes with optical feedback (recirculating loops). Forward-only meshes, in which light flows in one direction, are today the most commonly used [1], [5], [7]. They can implement any transfer matrix between a set of input waveguides and a set of output waveguides. While they are fairly easy to configure, they are limited in functionality: optical path length differences

Manuscript received September 24, 2019; revised January 13, 2020 and February 21, 2020; accepted March 29, 2020. Date of publication April 3, 2020; date of current version July 23, 2020. This work was supported in part by the European Research Council through under Grant Agreement 725555 (PhotonicSWARM) and in part by the European Union's H2020 program through Grant 780283 (MORPHIC). The work of Pieter Stroobant was supported by a Ph.D. grant of Ghent University, Special Research Fund (BOF). *(Corresponding author: Xiangfeng Chen.)*

Xiangfeng Chen and Wim Bogaerts are with Photonics Research Group, Department of Information Technology, Ghent University-IMEC, 9052 Ghent, Belgium (e-mail: xiangfeng.chen@ugent.be; wim.bogaerts@ugent.be).

Pieter Stroobant and Mario Pickavet are with IDlab, Department of Information Technology, Ghent University-IMEC, 9052 Ghent, Belgium (e-mail: pieter.stroobant@ugent.be; mario.pickavet@ugent.be).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JLT.2020.2984990

are usually limited to a $0-2\pi$ phase shift, and therefore it is not straightforward to implement wavelength filtering functions.

An alternative architecture uses tunable directional coupler organized in loops of triangles [8], squares [9] or hexagons [10], [11]. In such recirculating architectures, light thus folds onto itself, creating optical resonances. In this scheme all the ports are equivalent, and can serve as either input or output, and the mesh makes it easy to define discrete delay lines, enabling the implementation of wavelength filters. Inside this feedback mesh, the transport and distribution of light is governed entirely by the connectivity of the waveguides and the states of the tunable couplers. The phase relations are governed by integrated phase shifters. Fig. 1(a) shows a generic programmable PIC which contains a core waveguide mesh, connecting together optical inputs/outputs, as well as high-speed radio-frequency inputs and outputs enabled by high-speed electro-optic modulators and balanced photodetectors, respectively.

Programmable meshes are proving very useful in many domains such as artificial intelligence [1], quantum optics [13] and microwave photonics [14]. Rapid progress in driving electronics and packaging technologies makes it possible to build programmable PICs of increasing sizes. However, as meshes scale up, the degrees of freedom become overwhelmingly large. Configuration algorithms have been proposed to define certain functions in forward-only meshes with several complex demonstrations of linear operations [1], [13], [15]–[17]. Whereas for meshes with feedback loops, demonstrations have been mostly limited to reconfigurable wavelength filters [2] and time delay lines [4], which are manually programmed. This is neither sustainable nor practical when the complexity of the meshes scales up and multiple functions need to be defined. Recently, automated control algorithms have been proposed for beam-forming application [18]. However, for each reconfigurable function, the chip needs to be trained by these control algorithms. Also, graph-based algorithms are reported for auto-routing. [19] The systematical strategy is still lacking to address placement and routing of different filters [11] simultaneously in the circuit, especially when one wants eliminating using components to use the full capacity of the mesh and to trade off among circuit performance metrics that an S-matrix [20] can not model (e.g. power consumption). Thus todays programmable PICs are mostly forward-only circuits [1], [5], [7], [13], [21], [22], with demonstrations of recirculating meshes limited to a 7-cell hexagonal mesh [2], [14].

We can separate mesh programming problems into two classes: the distribution of light in the mesh (routing and distribution) and the definition of interferometric functions such as wavelength filters, which are phase sensitive. We introduced the basic idea to use graph representations for such large programmable circuits. [23] In this paper, we describe in more detail several approaches for graph-based design and programming of PIC meshes, and validate that the graph representation is capable of solving light distribution problems in programmable meshes with feedback loops. Thus we highlight the connections and topology and translate the feedback-mesh into a graph. This way, we make programming and design of the large-scale photonic mesh agnostic of the technology platform of the chip. For example, a tunable coupler can be based on the thermo-optic effect or microelectromechanical systems(MEMS) technology. The abstracted connectivity has intrinsic flexibility for circuit malfunctions. It is reasonable to assume that a portion of optical elements can be malfunctioning in large photonic circuits due to various reasons such as fabrication defects or fatigue which causes broken paths or even couplers stuck in a cross or bar state (e.g. in MEMS devices). The connections are then updated in our graph by removing nodes and edges blocked by malfunctions after the hardware calibration. The routing and distribution is similar to programming strategies for field-programmable gate arrays (FPGA) in electronics [24], [25], but with very different constraints, mainly due to the reciprocal nature of light propagation. The objectives also differ, ranging from optimizing the circuit performance to fabrication tolerance, flexibility given certain malfunctions and to robustness against parasitic interferences. The performance of the overall function is a trade-off of metrics such as propagation losses, crosstalk, and power consumption. We will demonstrate that many functions in programmable photonic circuits can be related to a corresponding routing or flow problem and we can leverage existing powerful graph tools to solve it. We deliberately separate the graph layer from the hardware layer. The solution from the graph layer dictates which couplers and with what state to employ for such a distribution problem. By mapping the identity (e.g. unique name string) of each connection in this solution to the individual components in the circuit, we can then assign the actual coupling values of tunable 2×2 couplers to the hardware layer. For this, tunable elements on the chip need to be pre-calibrated one time on the hardware level and then leave the programming tasks to the graph layer. The application of graph theory to this new field will help us create programming paradigms for systematically controlled programmable meshes.

II. Graph Representation

For the discussion in this paper, we will use a hexagonal mesh for our graph abstraction, as shown in the schematic drawing in Fig. 1(e). The presented technique however is applicable to all types of meshes of waveguides and tunable couplers, including the forward-only meshes. The mapping onto a graph is guided by preserving the connectivity and performance properties of the photonic building blocks. There can be a wide variety of building blocks in a photonic circuits, including phase shifters (PSs), modulators, semiconductor optical amplifiers, photodetectors, and waveguides (WGs), but from the point of view of a graph these are similar, in the sense that they have only two optical ports. Other components, such as the tunable couplers (CPs), have four ports. For instance, a tunable coupler based on the thermo-optic effect consists of a Mach-Zehnder interferometer with a thermo-optic phase shifter in one or both arms. The tunable coupler is capable of distributing light by electrically tuning the phase shifter(s). In a programmable mesh circuit, the tunable couplers are the elements that define the connectivity, so their correct graph representation is crucial to the definition of a full graph of the programmable circuit.

A graph consists of nodes, and edges which connect the nodes. Light flows along the edges, following a path through the graph. Since we are not just interested in which coupler to address but also in its state (cross, bar, partial coupling), a logical choice is to represent the optical ports of the building blocks as nodes. Light flow is then physically represented from port to port. For the two-ports building blocks, the graph connections are straightforward: the component is represented as two nodes which are internally connected. And each node can then be externally connected to exactly one node of an adjacent circuit block. For the four-port 2×2 couplers, things become more complicated, as the coupler can be configured to couple the ports on one side (a, c) to different ports on the other side (b, d). Internally, there can be more than one connection for each port. More precisely, the coupler can be in 'bar' state, 'cross' state, or in a 'fractional coupling' state as shown in Fig. 1(b). A simple graph representation which captures this connectivity is shown in Fig. 2(e), where the coupler consists of 4 nodes a, b, c, d connected in a cross/bar configuration. When the coupler is in 'bar' state, light may travel from a to b or b to a with weight W_{ab} and from c to d or from d to c with weight W_{cd}. When the coupler is in 'cross' state, light may travel from a to d or from d to a with weight W_{ad} and the other signal traverses from c to b or from b to c with weight W_{bc}. When the coupler is in a 'fractional coupling' state, any port can distribute light forward to the two opposing ports by a programmable ratio like from a to both b and d or both from b and d to a.

For the connections between building blocks, light can traverse from or to a port of a building block, so we represent these two light flows in the graph. For example, in Fig. 2(e), arrow $a1$ stands for light travelling towards to port a while arrow $a2$ stands for the direction away from the port a. Analogous definitions apply to port b, c, and d. As the edges between nodes represent connections, either between building blocks, or a port-to-port coupling within a building block, each connection can have a different influence on the circuit performance. Measurements or simulated component performance metrics can be incorporated into the graph by assigning specific attributes to the edges, such as insertion loss (I_l), power consumption (P_c), crosstalk and phase accumulation which can be represented by optical path length (L_o). The desired circuit performance metrics is expressed in edge weights for graph tool to optimize. One possible weight assignment to the edges (W_e) is linear weighted (w) contribution from these attributes:

$$W_e = w_1 \cdot I_l + w_2 \cdot P_c + w_3 \cdot L_o + \cdots \quad (1)$$

This edge weight is used as a cost in the algorithms for routing and flow problems. The result of those algorithms will then be a feasible solution which minimizes the total cost. From such distribution solution, we can recalculate the impact on circuit performance for each category of these attributes. For demonstration purposes, in our mapping, we choose w_1 as 100% and all other w are 0%, and assign different propagation loss weights to different connections (e.g. 10 for all coupler 'cross' connections, 5 for all coupler 'bar' connections, 4 for all phase shifters and 1 for all waveguide connections). For phase sensitive programming such as path-balancing distributions and optical filter synthesis, the optical path length is a key attribute. However, in the scope of this work we mainly focus on graph representations and the verification of phase-independent problems (routing and distribution), so we do not take L_o into account ($w_3 = 0\%$) in our demonstration. Nonlinear weight assignment is also possible. For example, addressing crosstalk or signal congestion problems requires the consideration of edge interactions. This can be realized through weight updating which we will discuss in Section III-B for multi pair signals routing.

While a graph can be multi-dimensional, for visualization purposes we project our photonic graph network on a plane and

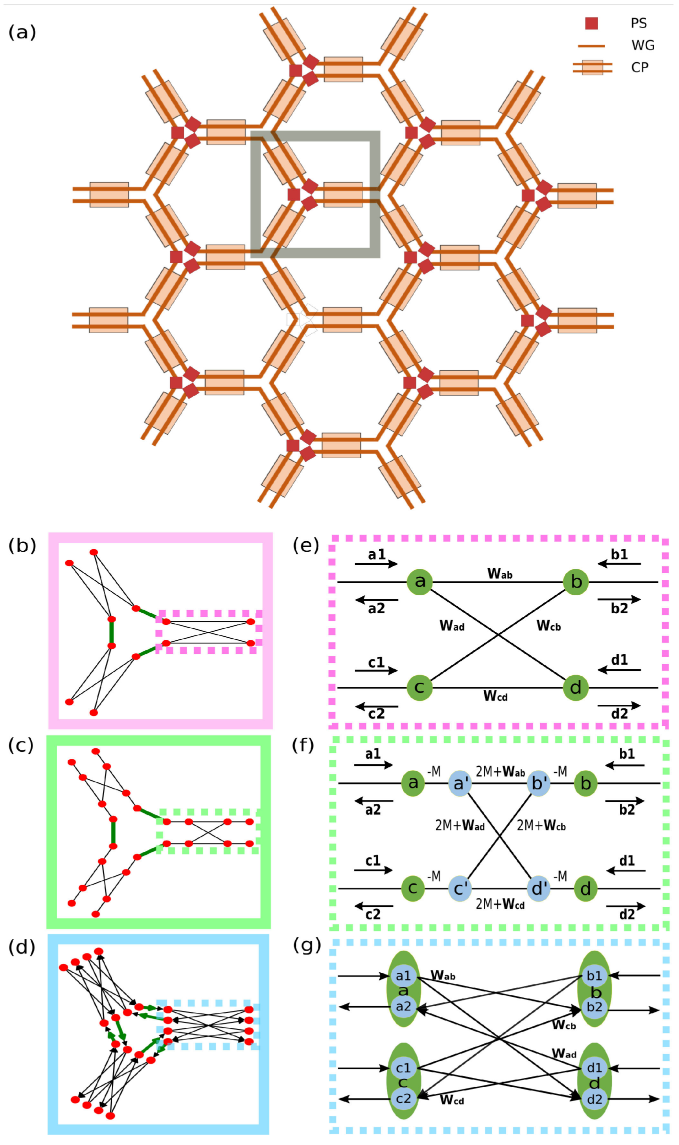

Fig. 2. (a) Schematic drawing of our example hexagonal mesh with highlighted repeatable cell in gray. In this repeatable cells, three couplers are interconnected with phaseshifters which are marked in green. These cells are mapped to three subgraphs using three different coupler representations: (b) original undirected subgraph using the coupler representation (e); (c) undirected subgraph with four auxiliary internal nodes introducing negative weights using the coupler representation (f); (d) directed subgraph using coupler representation (g) with eight artificial nodes.

assign coordinates to each node which correspond to the real hexagonal mesh. We used the python package *NetworkX* for graph visualization and algorithm customization [26].

Graphs can be characterized as either undirected graphs or directed graphs. Due to the reciprocal nature of waveguides, we originally chose an undirected graph as presented in Fig. 2(e), where light can flow through the edges in both directions. However, this choice allows unphysical paths inside the tunable coupler. For instance, in an undirected graph, light is allowed to travel from a to c by means of node b and/or d, with a weight of $W_{ab} + W_{bc}$ or $W_{ad} + W_{dc}$. In a real tunable coupler, this path is not allowed: it would require that the light abruptly reverses direction in a nonreciprocal way, and it would use a 'bar' and 'cross' connection at the same time, which is not possible. We will explain this in details in Section III-A and in Fig. 4.

One way to address this problem is to use weights to make unphysical paths prohibitively expensive. Scheme (f) in Fig. 2 illustrates an approach which applies arbitrary weights to favor certain possible connections. Four more auxiliary nodes (a', b', c' and d') are added next to the original port nodes. All possible coupling connections used to connect to original nodes (a, b, c, and d) are connected to these auxiliary nodes instead. The weight between the port nodes and auxiliary internal nodes has a negative value of $-M$, where $M \in \mathbb{R}^+$ is an arbitrary number chosen to be larger than the sum of all weights in the original graph representation. The weights of the edges between the four auxiliary internal nodes are changed to the original weights of the port-to-port connections with addition of $2 \times M$. The introduction of the negative weights now favors the connections that do not mix the two coupling states in one coupler. For example, the nonphysical path from a' to c' taking edges a'-b' and b'-c' becomes prohibitively expensive because of the high additional costs 2 M. In the section below, we will explain in detail how the arbitrary weights together with a minimum weight path algorithm achieves selectivity of nonphysical paths. Also, we will discuss further in the next section that undirected graphs generate problems, especially for meshes with feedback loops.

Because of these issues, the scheme in Fig. 2(g) is proposed to fully represent the possible light traversal directions between the four ports of a coupler. Instead of four nodes, eight nodes are utilized in this coupler representation, two for each port representing the inbound and outbound light. The arrows representing incoming or outgoing light in the original scheme are replaced by eight corresponding nodes which represent the direction of light towards to or away from the port of the coupler respectively. For instance, the arrow $a1$ is replaced by node $a1$ for external connections. Node $a1$ only has directional edges pointing towards node $b2$ and node $d2$ for connections within the coupler, and likewise, node $a2$ only receives connections from nodes $b1$ and $d1$. Note that this last graph representation also requires that we change the representation of two-port devices such as waveguides and phase shifters. These two-port devices need to be represented by 4 graph nodes, two for each propagation direction.

To assess the effectiveness of these graph implementations in real circuit problems, we performed several tests on a hexagonal programmable circuit with feedback loops. A possible

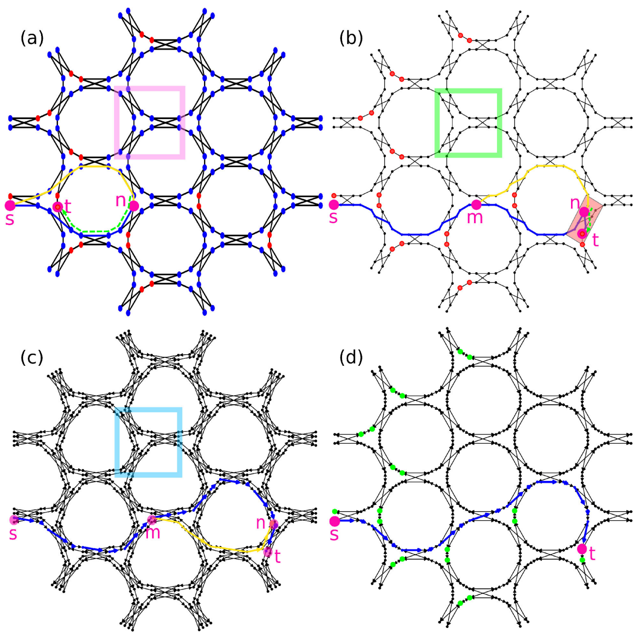

Fig. 3. Three graph representations are compared for single input-output pair case, which is reduced to a constrained single pair shortest path routing problem. The blue dots in (a) are nodes that can be reached by the algorithm. The red dots in (a) show nodes that cannot be reached by the customized Dijkstra's algorithm, and red dots in (b) are marked as unreachable as well because the blue path by the standard Dijkstra's algorithm is unphysical and is then discarded. The blue route is the solution that these algorithms provide. The yellow paths in (b) and (c) are valid alternatives of the blue paths as explained in the text. The alternative path starts at node m. Node n is the key node where the directed and undirected graph representations make a difference, which is further illustrated in 7(a–c). (d) is the same graph as (c) except that nodes representing the same port are projected to the same location. And green dots are valid destination as opposed to red counterparts in (a) and (b). [27].

implementation of the hexagonal mesh consists of repeatable unit cells which have 3 programmable 2×2 couplers with 3 phase shifters to interconnect them. [4] This repeatable unit cell is highlighted in gray in the mesh in Fig. 2(a). For this unit cell, we apply the different graph implementations of the tunable coupler. Fig. 2(b) uses the coupler representation in Fig. 2(e); Fig. 2(c) is abstracted using the scheme in Fig. 2(f); and Fig. 2(d) uses the representation in Fig. 2(g). Thus, the three couplers in dotted rectangles in Fig. 2(e–g) are interconnected with phase-shifters, forming an unit cell as in Fig. 2(b–d), respectively. The interconnect components are two-port components such as phase shifters and waveguides. In our case, phaseshifter connections are chosen and marked in green. With repeatable unit cells, we construct three graph networks based on these coupler implementations. The mapping of the unit cells in Fig. 2(b–d) to the graph representations in Fig. 3(a–c) is visualised by applying the same color to the unit cell. Fig. 3(d) is the same graph as (c) except it projects nodes that represent the incoming and outgoing direction of the same port to the same location, which simplifies the visualization but hides some of the complexity.

III. Light Distribution Problems

A key function of programmable photonic circuits is light distribution, i.e. guiding light from one port to another, or route it between functional blocks within the mesh. Being able to solve a variety of light distribution cases using graph algorithms demonstrates the usefulness of graph-based strategies for programmable photonic meshes. Four elementary cases are investigated in the subsections below.

A. Single Input-Output Pair

This problem relates to the single pair source-target shortest path problem, for which many well-established pathfinding algorithms exist, such as Dijkstra's algorithm [28]. This algorithm is able to find a shortest distance path (if such a path exists) for a given single source-target pair. Here, 'shortest distance' indicates that the sum of the weights of all edges in the path is minimal among all possible paths between the specified source-destination pair. Edge weights are linear user defined costs which may be chosen to represent trade-offs among various parameters of circuit performance. For instance, if we choose propagation loss as the edge weight, the solution of this algorithm will provide us a path with minimal total propagation loss.

There is a complication, however: in the context of single input-output problems, the 2×2 coupler can only be programmed in either 'cross' state or 'bar' state at any given time. Therefore, we will need to eliminate (during the execution of the algorithm) the edges of the states that are no longer possible. Once a 'cross' edge of a coupler is used, the only other available edge remaining is the other 'cross' edge, and the 'bar' edges are no longer accessible. Without this condition, the standard Dijkstra routing algorithms may yield nonphysical paths as highlighted in red in Fig. 4(b) and (c). The same undirected graph representation is used as in Fig. 3(a) but 12 couplers at the outside of the mesh were removed: In the resulting mesh configuration unphysical routing cases appear more frequently. With this in mind, one solution is to customize the Dijkstra algorithm while still using the original graph representation. This can be achieved by eliminating cases where three sequential nodes from one coupler are put into the priority queue of the algorithm. Note that it is impossible for a shortest path to pass through a certain coupler, subsequently pass by some other edges, and then pass through the same coupler again but in incompatible state, due to the fact that such a path would need to visit the same node twice (meaning that it cannot be a shortest path). Thus, our customized Dijkstra-based heuristic guarantees valid physical paths and coupler states, as highlighted in green in Fig. 4 (d) and (e). The ability to find optimal solutions in the presence of broken couplers is also illustrated in Fig. 4 and demonstrates the intrinsic flexibility of the graph-based approach.

Instead of customizing the algorithm, we investigate whether we can introduce a weighting scheme that would penalize those nonphysical internal paths that mix both coupling states. We introduce a graph model where each coupler is represented as illustrated in Fig. 2(f). Clearly, a path which contains a 'cross' edge immediately followed by a 'bar' edge through the same coupler (or vice versa) will have a cost of at least 2 M. For example, port a has possible connection a-a'-b'-b (the total weight for this connection is equal to W_{ab}), a-a'-d'-d (the weight is W_{ad}), a-a'-b'-c'-c (the weight would be arbitrarily large,

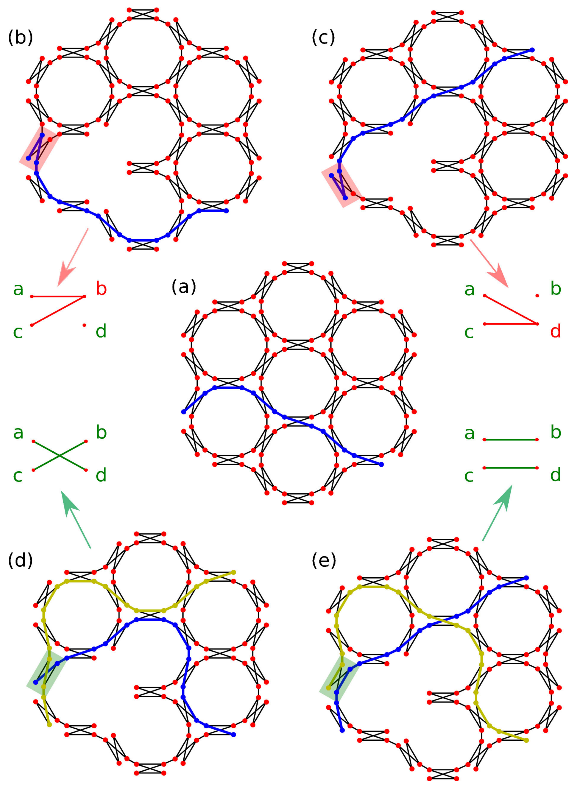

Fig. 4. Outer couplers were removed from Fig. 3(a) to make unphysical routing solutions more likely. (a) and (b) are routes with same start and end points, allowing to compare routes found by Dijkstra's algorithm with and without malfunctioning nodes. Nonphysical connections in the light paths of the couplers are highlighted in (b) and (c). The valid minimum cost solutions for feed-forward path are visualised in (d) and (e). These sequential routing results were found by a customized Dijkstra algorithm for the same two source-target pairs but were found by looking for the paths in a different order: the blue path is routed first, and the yellow path is routed after removing any prohibited edges resulting from the blue route.

which is $2M + W_{ab} + W_{bc}$), a-a'-d'-c'-c (the weight is equal to $2M + W_{ad} + W_{cd}$). Similarly as before, a shortest path which passes twice through the same coupler in different states, with some other edges in between is impossible, since such a path would visit the same node twice (and hence is not a *simple* path, i.e. a path in which no node is visited twice). Since the cost of valid paths remains unaffected by M while the cost of an invalid path is at least 2 M, choosing M large enough will ensure that the shortest simple path between a given source-destination pair will also be physically valid. Unfortunately, Dijkstra's algorithm does not guarantee an optimal path when negative weights are present. And the graph contains negative weight cycles (e.g. travelling from node a to a' and immediately returning), which excludes other well-known shortest path algorithms. However, finding the shortest *simple* path in a graph with negative weight cycles is NP-hard [29], [30]. Since Dijkstra's algorithm guarantees that no node will be visited twice, we can technically still apply the standard Dijkstra algorithm to a network where each coupler is represented as in Fig. 2(f). Such compatibility with negative weights is easily achieved by ensuring that only distances corresponding to nodes to which the algorithm has not found a path may be updated.

The issue for graph representations which implement coupler scheme (e) and (f) in Fig. 2 is illustrated in Fig. 3(a) and (b), respectively. The simplified Dijkstra's breadth-first searching is provided in Fig. 7(a) and (b). The red dots in the figures mark nodes to which no valid path can be found with the corresponding algorithms, despite the fact that such a path does exist. The algorithm eliminates these nodes from its consideration list when the breadth-first searching tree traverses the node in one direction, but this also eliminates the node from being traversed in the other direction. Therefore, these unreachable nodes are mainly an artifact arising from the use of an undirected graph representation for an application in which directionality is critical. Light has two possible directions for one port of a coupler: towards to or away from the coupler. In an undirected graph, Dijkstra's algorithm will however mark a node only once as 'visited,' when a minimum cost path is found to that node. The problem is that if this node has been marked as 'visited' by a path passing through the coupler in any one of these two directions, the algorithm will not search for a path to this node in which the light flow is coming from the other direction which connection is marked under red crosses in the Fig. 7. The terminal nodes are "unvisited" but the feedback architecture represented by undirected graphs intrinsically hinders valid paths towards terminal nodes(t) through "visited" nodes(n) which are prohibited in Dijkstra's algorithm. The through nodes(n) is just an arbitrary node in feedback loops, it can be any node in the set of visited nodes from breadth-first searching tree from arbitrary source nodes(s). For some cases(e.g. terminals like "t" in Fig. 3(a)), it requires a route through multiple "visited" nodes to yield a valid path. This results in unreachable nodes in graph representation of Fig. 3(a) and Fig. 3(b). The blue path is unphysical path which is discarded due to the same reason, thus t is marked red as unreachable in the representation. An detailed example is visualised in Fig. 3(b) and abstracted in Fig. 7(b). When looking from a path from node s to t, the subpath from node m to n in blue costs less than the path in yellow, since we chose the weight of 'cross' edges to be larger than that of 'bar' edges when we assigned the propagation loss attribute to edge weights. Thus, in an undirected graph model, the algorithm will store the blue path and mark n as 'visited'. Since travelling from node n directly to node t after arriving in n along the blue path results in an unphysical path, the option of travelling from s to t over node n is discarded. The red highlighted box in Fig. 3(b) shows the violation of the physical light propagation conditions in such a case. Despite this violation, the correct minimum weight $s - t$ path does includes node n, but reaches the node along a different direction, using the yellow path as green arrow indicates in 7.

The third graph model builds on this observation and incorporates the directionality of light by splitting a port into two nodes, resulting in the tunable coupler model shown in Fig. 2(g). Thus, all possible directions of light are represented by a node and connected according to that direction. This graph representation ensures that the standard Dijkstra algorithm stores a route passing through each port corresponding to both light directions, thus solving the issue of the undirected graph models. Constraints are needed to avoid nonphysical edges in this case as well. Since we arbitrarily use two nodes to present two directions for only one port, we also need to eliminate those cases where two nodes representing the same port are path of a shortest path. We use an iterative heuristic, in which each iteration corresponds to an execution of Dijkstra's algorithm to find a specific path. If an unphysical path is found, the conflicting edges are detected and the edge weights are adapted to make it less likely that they will be included in the shortest path of a subsequent iteration. Edges are penalized by updating the edge costs (W_p), replacing the cost by the sum of the original weight (W_e) and the product of the number of times the specific edge caused a violation (h_e) and a user-defined violation cost (p_e) [24], [31]:

$$W_p = W_e + h_e \times p_e + \cdots \tag{2}$$

The history of violations is essential in this fast heuristic algorithm, because it keeps track of how many times a specific edge has been included in nonphysical paths. The violation cost is user defined. The heuristic program keeps iterating the standard Dijkstra algorithm with updated penalties until a solution path is found without any violations or it reaches the iteration number that a user sets.

Fig. 3(c) shows that the directed graph model finds the correct result for the scenario which was problematic for the undirected counterpart, which was introduced in Fig. 3(b). The breadth-first searching tree schematic is provided in Fig. 7(c) and (b) for comparison. The algorithm finds both the yellow and the blue arrow path towards node n and is able to find an optimal path to reach node t through n. For a better visualisation of the mesh, we map node pairs representing the same port to the same location, resulting in Fig. 3(d). The green nodes mark all valid destinations which used to be unreachable with the undirected graph models when starting from node s. [27] A minimum length path is now found towards all nodes for which such a path exists, thus indicating that the directed graph model is well suited to solve single-pair shortest path problems for meshes with feedback loops.

B. Multiple Input-Output Pairs

In many cases, programmable circuits need to route multiple paths at the same time. Since tunable couplers can function as waveguide crossings and paths may cross, there are many ways to route these paths. One possible approach is to solve the single-pair shortest path algorithm sequentially for each of the paths, while removing an edge from the graph as soon as it is included in a specific path. However, the order in which these single-pair shortest path problems are solved influences the final routing results, as demonstrated in Fig. 4(d) and (e). Finding a minimum cost solution for a given set of pairs of input-output ports is essentially an integer multi-commodity flow problem in a graph with extra restrictions due to the constraints to avoid unphysical paths. In this problem, each input-output pair corresponds to a separate commodity. Since the integer multi-commodity problem is well known to be NP-hard, even in the case of only two commodities, we propose a heuristic that builds on top of our solution for the single-pair shortest path problem.

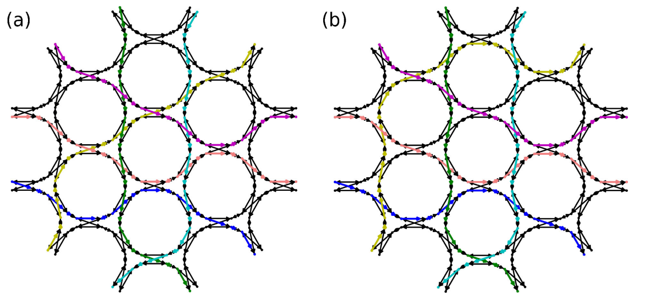

Fig. 5. Different valid solutions are found for the same 6 input-output pairs. Respectively, for (a) and (b), the arbitrary congestion cost is 1 and 2; Total cost is 550 and 565; the iteration time is 92 and 18.

To optimize which edges are assigned to which path, we can leverage the Lagrangian relaxation method in congestion negotiation algorithms [25], [32]. Congestion occurs when paths are competing for the same edges or nodes in the photonic graph. We implement a heuristic algorithm aiming to reduce congestion which is similar to the heuristic which avoids unphysical paths in the single input-output pair case. The algorithm will keep running until a solution is found in which no nodes are congested. The algorithm keeps track of a history of congested edges, similar to the history of path violations in the heuristic for the single-pair shortest path problem. The algorithm keeps iterating the single-pair shortest path algorithm while penalizing edges according to their congestion. The added cost is calculated by multiplying the number of times an edge was congested with a user-defined congestion penalty. Since the single-pair shortest path algorithm prefers paths with lower cost, increasing the cost of congested edges gradually directs paths towards other edges if these are available. Fig. 5 illustrates, for the same user-chosen input-output pairs, that different valid solutions are found. For this case, a larger congestion penalty decreases the number of iterations before convergence, but also results in less optimal solutions. Further improvements on congestion penalties are possible. [32] A movie is uploaded on DataPort for visualizing realtime algorithms on congestion negotiation with 6 input-output pairs.

C. Single Input With Multiple Outputs (Distribution)

Both when solving the single pair shortest path problem and when looking for a solution for the multi-pair problem, the couplers function as switches and thus are considered to be in either ‘bar’ or ‘cross’ state. In a real mesh however, there are cases where they will need to function in a ‘fractional coupling’ state. For instance, a crucial application for a mesh is to transport a single input light signal to multiple output ports. Therefore we will demonstrate that the graph representation can also generate valid solutions for such distribution problems.

Finding a distribution tree which minimizes a linear penalty, such as linear propagation loss in photonic mesh network, translates to a minimum-cost flow problem (with unlimited edge capacities). The cost of a specific edge is proportional to the number of source-destination paths passing through that edge. A heuristic approach for this problem is to apply Dijkstra’s algorithm to create a shortest path tree starting from the source node. A distribution tree is found by considering the union of the paths to each of the destinations. Without physicality constraints, the optimality of such a tree would be guaranteed. However, this tree may be unphysical due to conflicting edges, which may occur when nodes are used which correspond to the same port, but with light travelling in opposite directions. The corresponding edges are subsequently penalized, similar to the heuristic for the single source-destination shortest path problem. These steps are repeated until a valid solution is found.

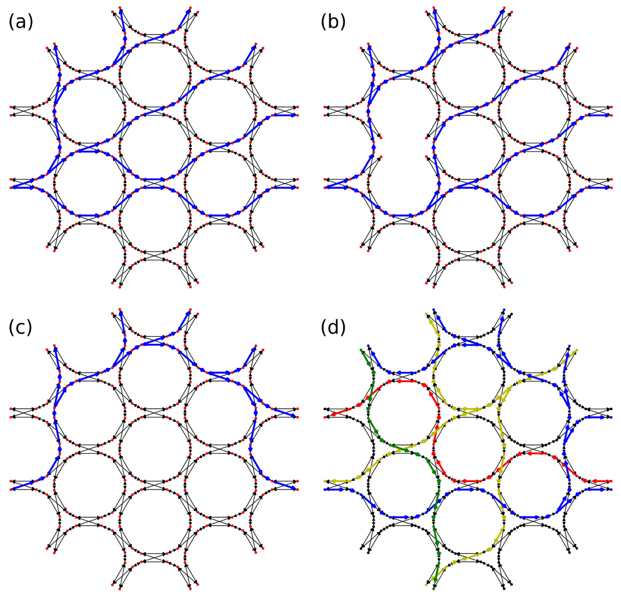

Fig. 6. Example solution for the one input to multiple outputs light distribution problem, found using various integer program objectives: (a) edge costs are proportional to the flow passing through the edge; (b) the same algorithm applied to a graph representation with malfunctioning couplers; (c) edge cost count only once for any flows using that edge; (d) multiple distributions without signal collisions.

An example distribution is shown in Fig. 6(a). The broken coupler in Fig. 6(b) demonstrates the flexibility of graph based methods when dealing with malfunctioning building blocks, similar to the example routing problems in Fig. 4. This reduction to minimum-cost flow problem in graph might be useful for a circuit operating at a high power level which introduces nonlinear losses. In reality, at certain power level, nonlinear losses can emerge, such as two-photon absorption (TPA). Note that although the signal is distributing like a tree in this example, this algorithm does not apply any heuristic to prefer splits as early as possible to balance power level at each couplers.

However, in cases where light signals are being distributed at low power level, we don’t need to worry about nonlinear absorption or saturation effects. In such cases, an optimal solution would employ as few couplers as possible. This way, more couplers are available for programming other functional blocks while being agnostic about any saturation or capacity effects (e.g. when TPA would start to dominate the losses). This case is similar to a directed Steiner Tree problem, with extra constraints to guarantee physical solutions. We formulate an integer program to solve this problem [33], where a binary decision variable is associated with each edge. These binary variables are used to calculate the cost objective, so the cost of each edge may be counted at most once, no matter how many flows use that edge. The program was solved using the *CP-SAT* solver of *Google OR-Tools* [34]. This results in the solution shown in Fig. 6(c). This algorithm tends to save available resources as much as possible and hence distributes the light along a line-shape, gradually tapping off light to the different outputs.

D. Multiple Distributions

The multiple distribution problem requires us to transport multiple input signals each to their own set of output ports. The result is a set of directed trees, each one corresponding to a specific signal, with no two trees sharing any nodes which correspond to the same port. This problem extends both the single distribution problem (trivially), and multiple input-output pairs problem, since a rooted Steiner tree with two nodes is always a path. The NP-hardness of this most general routing problem follows from the reduction to the latter case.

For most of distribution cases, we try to minimize the number of components, and assign a fixed cost to any used edge. The problem is similar to a directed Steiner Tree problem with multiple roots [35] and extra physicality constraints. Based on these similarities, we extend the integer program for single distribution problem to the multiple distributions cases, whilst avoiding shared nodes between different trees. An example solution is shown in Fig. 6(d). The solution for multiple distributions uses the same distribution assignment for the blue distribution in Fig. 6(c). It tackles the distribution dilemma with extra signal distribution in yellow and two more single pair signal assignments in red and green, generating a valid solution without signal collisions.

IV. Discussion

Translating a mesh of tunable couplers with feedback loops into a graph representation is nontrivial. The flow of light in a tunable coupler imposes additional constraints onto the graph model, meaning that we cannot apply standard graph algorithms without modification. In the sections above, we proposed 3 different implementations for a tunable coupler, resulting in 3 different corresponding photonic graph representations. Simple one-pair source-target routing solutions were compared for these different representations. There are two types of physical violations when we program single pair shortest path in the initial graph representations:

- Type 1, light abruptly changes direction inside a coupler (reflection into the other waveguide as marked with the red boxes)
- Type 2, some nodes are not reachable by the algorithm while they should be accessible in the actual mesh which is represented by red terminal nodes.

Problems with undirected graph models have been explained in detail, and an undirected graph representation on feedback architectures has also been published [19]. In this work, extra efforts are needed to address the nodes that cannot be reached

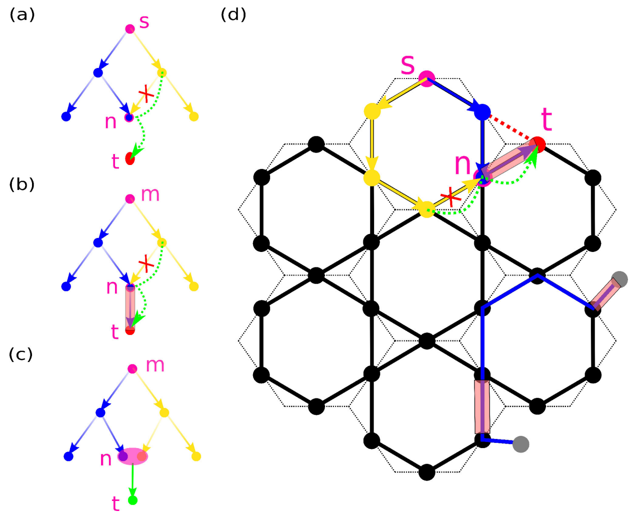

Fig. 7. Simplified breadth-first searching schematic shows the key issue for programming undirected graphs in feedback architectures. (a–c) in this figure corresponds to the Dijkstra's tree searching in graph representations of Fig. 3(a–c). (d) is an undirected graph representation if we consider couplers as nodes [17]. Gray nodes are arbitrary nodes which can be user-defined input and output for outer couplers of the mesh(e.g. grating couplers). violation type 1 is demonstrated. We remove one of the possible coupler connections(red dash edge) to show that violation type 2 causes type 1 violation as summarized in discussion session: type 1 violation is marked in red box and type 2 violation is given by red terminal coupler t. The valid connection is indicated by green arrows in dash line.

by the algorithms, such as the t node in red in Fig. 3(a). These are artifacts caused by applying a breadth-first searching based algorithm to an undirected graph representation of feedback architectures.

As an alternative, one may consider modelling the coupler components themselves as nodes (instead of the ports are we present it here). In Fig. 7 we represented the circuit using nodes to represent the tunable couplers. We can see the routing still suffers from the same issue if the undirected graph is representing the mesh with optical feedback loops. For example, the blue arrow path in Fig. 7(d) from s to n is the shortest path in the breadth-first searching tree. Thus n is marked "visited" and the yellow path can not get to node t through it. This violation of type 2 causes another violation of type 1: the blue arrow path through coupler "n" towards terminal coupler "t" is invalid since the light abruptly changes direction within the coupler "n". The acyclic directed graph modelling coupler as nodes is presented in [17] for forward-only meshes, which propagate light in only one direction between a set of input ports and a set of output ports. However, this representation is not directly applicable to meshes with feedback loops as we explore here. In our case, a 4-port 2×2 tunable coupler has 2 valid connections coexisting for signal routing (2 bar edges, 2 cross edges available) and we prefer not to lose the capability to use coexisting connections at the graph layer.

The proposed directed graph representation using ports solves the both types of violation and the representation is transformed: The node representing the same port of the directed graph overlap with each other for better user visualization. The resulting directed graph representation was further validated by solving shortest path problems with multiple input-output pairs, where the paths are routed simultaneously as opposed to sequentially [19], and by looking at the single distribution problem, where one input has to be transported to multiple outputs, and multiple distributions problem, where multiple input signals each have to be transported to their own set of output ports. Further improvement on the calculation time of different graph algorithms is possible for the individual graph problem in which each distribution case is reduced to. We provided here as a reference run time for the algorithms that we used on 2018 model laptop with 4 core CPU at 2.80 GHz: 100 calculations for single pair shortest path routing as example in Fig. 3(d) cost 0.031 s. For multiple input-output pairs in the mesh with 6-pair congestion negotiation built on top of the single pair, 100 runs in Fig. 5(a) cost 78.846 s while Fig. 5(b) cost 13.931s [27]. 100 calculations for single-rooted Steiner tree in Fig. 6(c) cost 46.921 s. And 100 calculations for multi-rooted Steiner tree problem in Fig. 6(d) cost 515.006 s.

The single input-output pair and single distribution problems are fundamental for the programming of switching and filtering functions. As constraints increase, exact solutions might not be feasible, but heuristic algorithms may provide near-optimal solutions for light distributions inside the mesh circuit. This was demonstrated through congestion negotiation heuristics for multiple input-output pairs and distributions without collisions, which were able to solve cases whenever couplers are in switching or fractional coupling states. The term, input and output, means any node selected as source and terminal for routing and flow problems in graph. We are able to program any light distribution assignment, if applicable, of given port/ports to the other/others. Thus the validation of the four distribution cases using this graph representation forms a backbone for complex programming of more sophisticated functions such as filter placement and routing. Furthermore, optical phase accumulation is already one of the edge attributes in our graph. Light distributions from any port to another also shows the potential to program filters on the graph layer. Our programming scheme could employ extra phase information in the edge weight: This could form the basis for filter synthesis using a combination of the single routing steps [19] and distribution algorithms we discussed above, and with congestion negotiation for coexisting multiple filters. Graph-based strategies will help us to take the next step towards systematically controlled programmable optical cores with complex functionalities.

REFERENCES

[1] N. C. Harris *et al.*, "Linear programmable nanophotonic processors," *Optica*, vol. 5, no. 12, 2018, Art. no. 1623.

[2] D. Pérez, I. Gasulla, and J. Capmany, "Programmable multifunctional integrated nanophotonics," *Nanophotonics*, vol. 7, no. 8, pp. 1351–1371, 2018.

[3] X. Chen *et al.*, "The emergence of silicon photonics as a flexible technology platform," *Proc. IEEE*, vol. 106, no. 12, pp. 2101–2116, 2018.

[4] D. Perez-Lopez, E. Sanchez, and J. Capmany, "Programmable true time delay lines using integrated waveguide meshes," *J. Lightw. Technol.*, vol. 36, no. 19, pp. 4591–4601, Oct. 2018.

[5] D. A. B. Miller, "Self-configuring universal linear optical component [Invited]," *Photon. Res.*, vol. 1, no. 1, pp. 1–15, 2013.

[6] W. Bogaerts and A. Rahim, "Programmable photonics: An opportunity for an accessible large-volume PIC ecosystem," *IEEE J. Sel. Topics Quantum Electron.*, to be published.

[7] A. Ribeiro, A. Ruocco, L. Vanacker, and W. Bogaerts, "Demonstration of a 4×4-port universal linear circuit," *Optica*, vol. 3, no. 12, pp. 1348–1357, 2016.

[8] D. Pérez, I. Gasulla, J. Capmany, and R. A. Soref, "Reconfigurable lattice mesh designs for programmable photonic processors," *Opt. Express*, vol. 24, no. 11, pp. 12 093–12 106, May 2016.

[9] L. Zhuang, C. G. H. Roeloffzen, M. Hoekman, K.-J. Boller, and A. J. Lowery, "Programmable photonic signal processor chip for radiofrequency applications," *Optica*, vol. 2, no. 10, pp. 854–859, Oct. 2015.

[10] D. Pérez *et al.*, "Multipurpose silicon photonics signal processor core," *Nature Commun.*, vol. 8, no. 1, 2017. Art. no. 636.

[11] D. Pérez, I. Gasulla, and J. Capmany, "Field-programmable photonic arrays," *Opt. Express*, vol. 26, no. 21, pp. 27 265–27 278, Oct. 2018.

[12] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, "Optimal design for universal multiport interferometers," *Optica*, vol. 3, no. 12, pp. 1460–1465, Dec. 2016.

[13] J. Carolan *et al.*, "Universal linear optics," *Science*, vol. 349, pp. 711–716, 2015.

[14] J. Capmany, I. Gasulla, and D. Pérez, "Microwave photonics: The programmable processor," *Nature Photon.*, vol. 10, no. 1, pp. 6–8, 2016.

[15] S. Pai, B. Bartlett, O. Solgaard, and D. A. Miller, "Matrix optimization on universal unitary photonic devices," *Physical Rev. Appl.*, vol. 11, no. 6, 2019, Art. no. 064044.

[16] D. A. Miller, "Setting up meshes of interferometers-reversed local light interference method," *Opt. Express*, vol. 25, no. 23, 2017, Art. no. 29233.

[17] S. Pai *et al.*, "Parallel fault-tolerant programming of an arbitrary feedforward photonic network," 2019. [Online]. Available: https://arXiv.org/abs/1909.06179.

[18] D. Pérez, "Programmable integrated silicon photonics waveguide meshes: Optimized designs and control algorithms," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 2, Mar./Apr. 2020, Art. no. 8301312.

[19] A. López, D. Pérez, P. DasMahapatra, and J. Capmany, "Auto-routing algorithm for field-programmable photonic gate arrays," *Opt. Express*, vol. 28, no. 1, pp. 737–752, Jan. 2020.

[20] D. Pérez and J. Capmany, "Scalable analysis for arbitrary photonic integrated waveguide meshes," *Optica*, vol. 6, no. 1, pp. 19–27, Jan. 2019.

[21] D. A. Miller, "Self-aligning universal beam coupler," *Opt. Express*, vol. 21, no. 5, pp. 6360–6370, 2013.

[22] K. Choutagunta, I. Roberts, D. A. B. Miller, and J. M. Kahn, "Adapting Mach–Zehnder Mesh equalizers in direct-detection mode-division multiplexing links," *J. Lightw. Techol.*, vol. 38, no. 4, pp. 723–735, 2020.

[23] X. Chen and W. Bogaerts, "A graph-based design and programming strategy for reconfigurable photonic circuits," in *Proc. IEEE Photon. Soc. Summer Topical Meeting Series (SUM)*, Jul. 2019, pp. 1–2.

[24] V. Betz and J. Rose, "Vpr: A new packing, placement and routing tool for fpga research," in *Proc. Int. Workshop Field Programmable Logic Appl.*, 1997, pp. 213–222.

[25] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Reconfigurable Computing*. Amsterdam, The Netherlands: Elsevier, 2008, pp. 365–381.

[26] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python Sci. Conf. (SciPy)*, 2008, pp. 11–15.

[27] X. Chen, "Supporting data of graph representations for programmable photonic circuits," Jan. 2020. [Online]. Available: http://dx.doi.org/10.21227/t001-kr39

[28] D. E. Knuth, "A generalization of dijkstra's algorithm," *Inf. Process. Lett.*, vol. 6, no. 1, pp. 1–5, 1977.

[29] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1990.

[30] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT, 2009.

[31] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2007, pp. 496–502.

[32] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "Nthu-route 2.0: A fast and stable global router," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2008, pp. 338–343.

[33] S. Bradley, A. Hax, and T. Magnanti, *Network Models*. Reading, MA, USA: Addison-Wesley, 1977.

[34] L. Perron and V. Furnon, "Google's OR-tools," Google, Jul. 19, 2019. [Online]. Avialable: https://developers.google.com/optimization/

[35] O. Suchỳ, "On directed steiner trees with multiple roots," in *Proc. Int. Workshop Graph-Theoretic Concepts Comput. Sci.*, 2016, pp. 257–268.

Xiangfeng Chen (Student Member, IEEE) received the M.Sc. degree from the Center for Optical Materials Science and Engineering Technologies, COMSET, Clemson University, Clemson, SC, USA, in 2018 by carrying out research on array waveguide gratings for III-V on silicon nitride hybrid integration. He is currently working toward the Ph.D. degree with the Photonic Research Group, Ghent University - IMEC, Ghent, Belgium. His research focus is on large scale programmable photonic circuits at both circuit and component levels. He enjoys the interdisciplinary nature of photonic engineering. He was originally trained as a Material Scientist and completed a transition to an Optical Engineer during his years with the Center for Optical Materials Science and Engineering Technologies, COMSET, Clemson University, SC, USA.

Pieter Stroobant received the M.Sc. degree in computer science from Ghent University, Ghent, Belgium, in 2016 where he is currently working toward the Ph.D. degree with the Internet Technology and Data Science Lab (IDLab). His current research interests focus on routing, optimisation, and nano-scale networking and communications.

Mario Pickavet received the M.Sc. and Ph.D. degrees in electrical engineering from Ghent University, Ghent, Belgium, in 1996 and 1999, respectively. Since 2000, he has been a Professor with Ghent University, where he is teaching courses on discrete mathematics and network modeling. He is coleading the research cluster on network modeling, design, and evaluation (NetMoDeL). His main research interests are fixed internet architectures and optical networks, green ICT, and design of network algorithms. In this context, he is currently involved in several European and national projects. He has authored or coauthored about 500 international publications, both in journals and proceedings of conferences.

Wim Bogaerts (Senior Member, IEEE) received the Ph.D. degree in modeling, design and fabrication of silicon nanophotonic components from Ghent University, Ghent, Belgium, in 2004. He is currently a Professor with the Photonics Research Group, Ghent University - IMEC. During this work, he started the first silicon photonics process on IMEC's 200 mm pilot line, which formed the basis of the multi-project-wafer service ePIXfab. His current research interests focus on the challenges for large-scale silicon photonics: Design methodologies and controllability of complex photonic circuits. In 2014, he cofounded Luceda Photonics, a spin-off company of Ghent University, IMEC and the University of Brussels (VUB). Luceda Photonics develops unique software solutions for silicon photonics design, using the IPKISS design framework. Since 2016, he has been a full-time Professor with Ghent University, looking into novel topologies for large-scale programmable photonic circuits, supported by a consolidator grant of the European Research Council (ERC). He has a strong interest in telecommunications, information technology and applied sciences. He is a member of Optical Society of America (OSA) and SPIE.