A Graph-based Design and Programming Strategy for Reconfigurable Photonic Circuits

Xiangfeng Chen and Wim Bogaerts
Ghent University - IMEC, Photonic Research Group, Department of Information Technology, Gent, Belgium.
Center of Nano and Biophotonics, Ghent University, Belgium.
Email: xiangfeng.chen@ugent.be

***Abstract*—We have developed a graph representation of programmable photonic mesh circuits that can be used by pathfinding algorithms. We modified the Dijkstra algorithm to observe only physically possible connections in the graph, demonstrating flexible rerouting in case of node malfunction.**

***Index Terms*—Programmable Photonics, Graph Theory, Customized Dijkstra,**

I. Introduction

Graph theory forms the basis of cost managements in a wide range of fields such as transportation scheduling, logistics planning, network flow assignments. Among these, it stands out by playing an substantial role in architecture designs of integrated electronic circuits [1]. Designers of field-programmable gate arrays (FPGA) employ graph theory to solve placement and routing issues. In photonic integrated circuits, we encounter similar, but still significantly different, connection and arrangement problems of photonic building blocks. And in the new field of programmable photonic circuits, the problem shifts from laying out circuits to programming the circuit connectivity in an optical mesh. A graph representation of a (programmable) photonic circuit makes it possible to leverage decades of research in graph theory [2]. By abstracting interactions of photonic building blocks to a graph network, well-developed algorithms for graph analysis can enable us to solve placement, routing and programming dilemmas. In this paper, we report our work in progress towards a graphing implementation for programmable photonic meshes which takes into account the particularities of optical waveguides, such as the directional flow of light.

II. Photonic Graph Implementation: Mapping

There are many topology designs for reconfigurable optical circuits, which can be separated into feed-forward ('left-to-right') topologies [3] and feed-back ('loops') topologies with square or hexagonal meshes [4]. For demonstration purpose we will use a hexagonal mesh in our graph abstraction, which consists of vertices and edges. A possible implementation of the hexagonal mesh consists of repeatable unit cells which have 3 programmable 2×2 couplers with 3 phase shifters to interconnect them. The mapping onto a graph is guided by preserving the connectivity and performance properties of photonic building blocks. Thus waveguide ports are chosen to be vertices in our photonic graph. As shown in Fig. 1, every coupler of this unit cell has 4 ports, namely ”in1”, ”out1” , ”in2” and ”out2”. The edges between the nodes represent connections either between building blocks, or port-to-port coupling within a building block, and each connection has a different influence on the circuit performance, expressed in figures of merit like propagation loss, crosstalk, and power consumption. Thus edges are assigned with attributes which are translated into weights. For example, in our mapping, we assign different propagation loss weights for different connections (10 for coupler cross connections, 5 for coupler bar connections, 4.5 for phase shifters and 1 for waveguides). This edge weight is used as a penalty in the routing algorithm. The result is the path with the lowest propagation loss, as shown in section III. For state of a tunable 2×2 coupler is initially left undecided, and the cross or bar state is chosen when the coupler is first used in a light path. Partial coupling can be realized by solving subgraph routing problems, but in this paper, we limit the scope to pure cross and bar states for the couplers.

Graphs can be characterized as undirected graphs, directed graphs and multi-graphs. Due to the reciprocal nature of waveguides, an undirected graph is chosen, but as we discuss further, this generates problems in the graph representation of the couplers. While a graph can be multi-dimensional, for visualization purposes we project our photonic graph network on a plane and assign vertices with coordinates that correspond to a real hexagonal mesh. We used the python package *NetworkX* for graph visualization and algorithm customization [5].

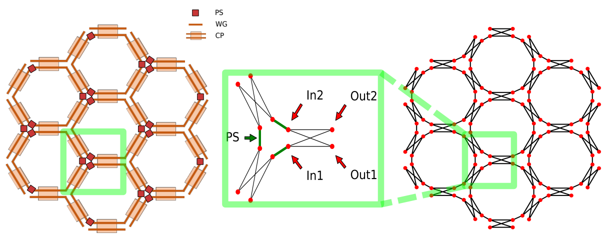

Fig. 1. Green boxes highlight the mapping steps from the hexagonal mesh to photonic graph by abstracting subgraph representation of the repeatable unit cells

This work has received funding from the European Research Council through grant agreement No 725555 (PhotonicSWARM).

III. Photonic Graph Traversal: Dijkstra

A key function of programmable photonic circuits is routing, i.e. guiding light from one port to another. For this, we can use established pathfinding algorithms, such as the Dijkstra algorithm [6]. We can use this sequentially for multiple paths by updating the graph when a path is added, by removing all used edges. In the case of the 2×2 coupler, we also need to remove the edges of the states that are no longer possible. If a 'cross' edge used, the only other available edge remaining is the other 'cross' edge, and 'bar' edges are removed.

A. Flexibility

This same approach can be used for flexible rerouting. It is reasonable to assume that in large photonic circuits individual elements can malfunction for various reasons such as fabrication defects. In our photonic graph, we then update the graph to remove vertices and edges blocked by malfunctions. This way, we can flexibly bypass malfunctioning nodes in circuits. Figure 2(b) shows a rerouting of a connection after malfunctioning nodes. We can identify multiple types of malfunctions, caused by broken paths or by couplers stuck in a cross or bar state.

B. Eliminating Nonphysical Solutions

While reciprocal circuit elements can propagate light in two directions, and thus the graph cannot be directional, light is still restricted to directional paths. In the middle of Fig. 2, green edges represent valid states of the subgraph of a 2×2 directional coupler, whereas red edges denote connections that are not physical. This is illustrated in Fig. 2(b), where the blue route shows the result of the standard Dijkstra algorithm. The routing in the red-highlighted coupler is logically allowed, but nonphysical: light cannot just reverse direction, and it is also not allowed to use the 'bar' edges and the 'cross' edges at the same time. Removing nonphysical connection choices by choosing the state of the coupler upfront solves this, but then requires an additional selection algorithm. This becomes unpractical for scaling up to multiple pathfinding. A similar reasoning holds for changing the edge weights upfront for extra penalty. Therefore, we chose to customize the original algorithm.

C. Customized Dijkstra Pathfinding

In the standard Dijkstra algorithm, the current node can be routed to all adjacent nodes which have not been visited. Recursively, weights for neighboring edges are dynamically recalled and compared. In order to filter out nonphysical path choices within one source-target pair, a third node is added to be dynamically recalled in each recursion of our algorithm. This third node is the predecessor of the current node in the shortest path. By applying a Boolean condition between the predecessor and every adjacent node of the current node, our customized Dijkstra thus eliminates the chance of having a path with three nodes of the same 2×2 coupler. The blue route in Fig. 2(c) shows that, for the same source-target assignment in Fig. 2(a) and (b), the customized algorithm chooses the correct shortest path without losing the flexibility in mesh with malfunction blocks. Couplers highlighted in green emphasize that the algorithm does yield valid connections. The sequence routing in Fig. 2(c) and (d) further validates our customized algorithm for routing in a programmable photonic circuit, irrespective of the circuit topology.

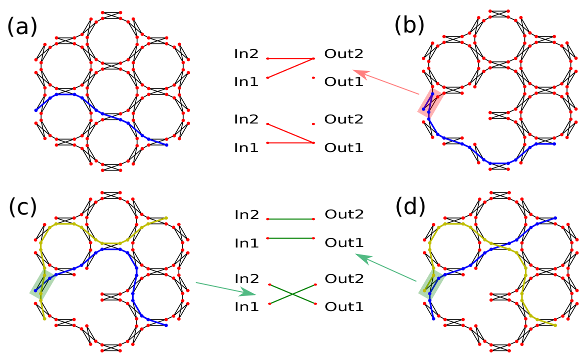

Fig. 2. (a) and (b) are route examples that compare routing condition with and without malfunctioning nodes by Dijkstra algorithm; (c) and (d) are sequential routing results by customized algorithm for the same two source-target pairs with different order: blue path is routed first, and yellow path is routed with removal of prohibited edges given routing result of the blue path

IV. Summary

We abstracted a hexagonal programmable photonic mesh to a graph implementation. We demonstrated that the standard Dijkstra pathfinding algorithm results in nonphysical paths, so we customized algorithm to capture the directionality of the light flow, inheriting the flexibility regarding to malfunctioning elements from the original algorithm. As meshes scale up, Dijkstra has to take the whole mesh as a search space to find solutions. We need to tailor this search space. Heuristic algorithms such as A* are better suited for this task. Also, algorithms for simultaneously resolving multiple paths are needed for congestion negotiation. Statistical analysis of such paths would help us draw a blueprint for mesh sizes, channel numbers, input-output assignments, power and latency management and would provide design guidance on architectures for programmable photonics.

References

[1] V. Betz and J. Rose, "Vpr: a new packing, placement and routing tool for fpga research," in *Field-Programmable Logic and Applications*. Berlin, Heidelberg: Springer, 1997, pp. 213–222.

[2] M. C. Golumbic and I. B. Hartman, Eds., *Graph Theory, Combinatorics and Algorithms*. Springer US, 2005.

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

[4] 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.

[5] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA, 2008, pp. 11 – 15.

[6] D. E. Knuth, "A generalization of dijkstra's algorithm," *Information Processing Letters*, vol. 6, no. 1, pp. 1 – 5, 1977.