Improved Networks Routing Using Link Addition

Cristian E. Onete
Former NXP Semiconductors,
The Netherlands
cristian.oneteATgmail.com

Maria Cristina C. Onete
XLIM/Univ. of Limoges/CNRS 7252
France
cristina.oneteATgmail.com

***Abstract*—In this paper an improved routing algorithm suitable for planar networks, static Zigbee and mesh networks included, is shown. The algorithm is based on the cycle description of the graph, and on a new graph model based on arrow description, which is outlined. We show that the newly developed model allows for a faster algorithm for finding a direct and a return path in the network. The newly developed model allows further interpretations of the relationships in any simple planar graphs. Examples showing the implementation of the newly developed model are presented, too. Keywords—graph cycles, path-finding, routing, arrow model, group theory.**

I. Introduction

Routing is an essential feature of any communication network. Almost any networks-related standard, including those by IEEE (Zigbee), recommends the existence of routing protocols. The cornerstone of routing is the ability to find paths between two given end nodes. Most such algorithms rely on node hopping and are based on the node description of the graph associated to the network [6-9]. We note that many types of networks deployed in everyday use (including home networks, static sensor networks, etc.) can be modelled as a connected planar graph.

In a previous paper [10] it has been shown that a routing algorithm can be developed using the cycle description of the underlying graph. This method has several advantages over algorithms based on node-hopping, such as its lack of back-tracking.

However, one disadvantage of the method presented in [10] was that the data obtained as the result of the algorithm is difficult to process. In this paper we address this shortcoming by means of a new "arrows" model for the graph. This model significantly improves the path finding algorithm based on cycles mergers.

The paper is organized as follows:

Paragraph II introduces the basic notations used in the paper.

Paragraph III describes the arrows group, while paragraph IV describes the use of the arrows group.

Paragraph V describes the routing algorithm using the arrows group and Paragraph VI is dedicated to conclusions and future work.

II. Basic Notations

A. Basic Definitions and Notations

It is known that any graph is represented by a set of links, which are connected to each other via nodes [1]. The number of links departing or arriving in a node is called the *degree* of the node. A link connected to a node is said to be *incident* to the node. A suite of links connecting two end nodes by passing through other nodes is called a *path*. A closed path, i.e., a path starting and ending in the same node is called a *cycle.* A cycle that visits every node of the graph exactly once is called a *Hamiltonian* cycle; a graph that has at least one Hamiltonian cycle is called a *Hamiltonian graph*. A link connected to a cycle, i.e., a component of a given cycle, is said to be *incident* to the cycle. Two cycles are *adjacent* if they share only a single link.

The total number of nodes in the graph is denoted as N and the number of links is denoted as L. In this paper we shall use simple planar connected graphs, which have no self-looping nodes, no leaves (nodes of degree 1) and no parallel links between any two nodes.

Algebraically, the graph's connectivity can be described by using either the nodes-links or the cycles-links incidence matrix. The nodes-links incidence matrix **A** is an NxL matrix (it has N rows and L columns), whose elements describe which links connect which nodes. The cycle-links incidence matrix **B** is a CxL matrix, where C is the number of independent cycles, and L, the number of links of the graph. In a planar graph, the values C, L, and N are such that [5] C=L-N+2.

Labelling is used to uniquely identify different parts of the graph, such as nodes, links, and cycles. In addition, graphs representing nodes are usually undirected. However, a randomly chosen direction is used in order to, e.g., analyze network functions. Neither the chosen labelling, nor the added direction, directly influence the properties of the graph.

In a planar graph, its cycles divide the geometric plane into an internal and an external region, and a cycle defines the border between the two regions. In this paper we assume that the cycle with the highest index $\underline{C}$ is the border cycle. Recall that a planar graph is a graph that can be drawn in a planar embedding such that no link crossings are present.

More on the cycles description will be presented in the subsequent paragraphs. In the simple planar connected graphs this is always the case [12].

In this paper we will use an incidence matrix describing how the links are related to the cycles. This matrix is the cycles-links incidence matrix $\mathbf{B}_{CxL}$ of the associated graph, which will be presented in a later paragraph. It is orthogonal to the nodes-links incidence matrix **A**. [1, 5].

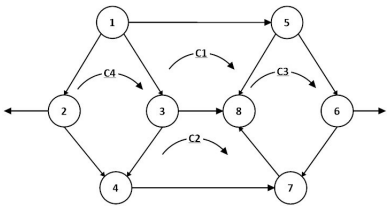


Fig. 1 Adjacency definition

B. Adjacency and cycles merger

Onete and Onete defined [2,3] the notion of cycle merger, which is applicable to adjacent cycles only. We only briefly describe this notion below, as it was already described in [10].

Fig.1 depicts a subnetwork in which the nodes and the cycles are labelled and a direction was defined for the links and cycles as previously discussed. Note that for example cycles c_1 and c_2 are adjacent through the common link connecting nodes 3 and 8. The two cycles can be *merged* as shown in Fig.2, following the method of [2,3]. Notably, we obtain a larger cycle by removing the adjacent link of the two cycles. Algebraically this can be obtained by adding either the rows or the columns describing c_1 and c_2 in the matrix $\mathbf{L_C}$, as described by [2]. Notably, $\mathbf{L_C}$ is the cycles Laplacian of the network and it is obtained using the cycles-links incidence matrix: $\mathbf{L_C=B*B^T}$.

We depict the result of the merger, denoted $c_1|c_2$ in Fig.2.

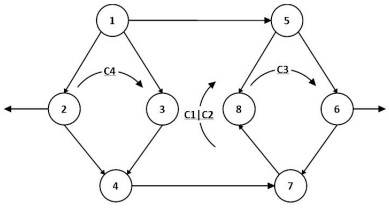


Fig.2 Merging two cycles

The papers [2,3] describe an algorithm of obtaining the Hamiltonian cycles in a graph by using cycle merger. This method needs no backtracking and thus the algorithm is parallelizable.

III. ARROWS GROUP

A. Graph Arrows

In this section we define graph arrows as an algebraic object, then proceed to describe operations on such objects. We depart from a (finite) set of nodes N. We define an arrow as an ordered tuple $\overrightarrow{(I,J)}$ with $I,J \in N$. Intuitively this object will correspond to a directed arrow from node I to node J, as depicted in Fig.3:

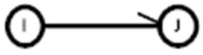


Fig.3 Graph arrow from node I to node J

In addition to arrows of the form $\overrightarrow{(\iota,J)}$, we define two special types of arrows. The first is called the arrow at infinity[1] $\overrightarrow{\infty}$, defined as an arrow from any node to itself. Thus, $\overrightarrow{(\iota,\iota)} = \overrightarrow{(J,J)} = \overrightarrow{\infty}$. The second is the zero arrow, denoted $\vec{0}$.

We define an "addition" operation on graph arrows, which we denote as $\vec{+}$.

Definition 1. Let N be a node set and let $i,j,k,l \in N$ be nodes in that node set. Let $\overrightarrow{\infty}$ and $\vec{0}$ be the two special nodes defined above. We define the addition operation $\vec{+}$ as follows:

- $\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(J,k)} = \overrightarrow{(\iota,k)}$;
- $\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(\iota,J)} = \overrightarrow{(\iota,J)}$;
- $\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(k,\iota)} = \overrightarrow{(k,J)}$;
- $\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(k,l)} = \vec{0}$ if $i \neq l$ and $j \neq k$;
- $\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{\infty} = \overrightarrow{\infty} \vec{+} \overrightarrow{(\iota,J)} = \overrightarrow{(\iota,J)}$;
- $\vec{0} \vec{+} \overrightarrow{(\iota,J)} = \overrightarrow{(\iota,J)} \vec{+} \vec{0} = \vec{0}$;
- $\vec{0} \vec{+} \overrightarrow{\infty} = \overrightarrow{\infty} \vec{+} \vec{0} = \vec{0}$.
- $\overrightarrow{\infty} \vec{+} \overrightarrow{\infty} = \overrightarrow{\infty}$

Consider the set of arrows defined by $(N\ x\ N) \cup \vec{0} \cup \overrightarrow{\infty}$ under the operation of arrow addition $\vec{+}$. This structure is closed under addition, it has an identity element $\overrightarrow{\infty}$, and it is commutative. Moreover, using the first rule it holds that $\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(J,\iota)} = \overrightarrow{(\iota,\iota)} = \overrightarrow{\infty}$. As a result, we usually call $\overrightarrow{(J,\iota)}$ the inverse of $\overrightarrow{(\iota,J)}$. Note that for each arrow of the form $\overrightarrow{(\iota,J)}$ there is a unique inverse. The inverse of $\overrightarrow{\infty}$ is itself. However, the zero element has no inverse. In addition, the addition operation we have defined is not generally associative. Indeed, it holds that:

$$\left[\overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(k,l)}\right] \vec{+} \overrightarrow{(J,k)} = \vec{0} \vec{+} \overrightarrow{(J,k)} = \vec{0}$$

However, associating the second and third terms yields a different result:

$$\overrightarrow{(\iota,J)} \vec{+} \left[\overrightarrow{(k,l)} \vec{+} \overrightarrow{(J,k)}\right] = \overrightarrow{(\iota,J)} \vec{+} \overrightarrow{(J,l)} = \overrightarrow{(\iota,l)}$$

The operation of inverting a graph arrow is shown in Fig. 4, where the inverse of the full arrow is the dotted one.

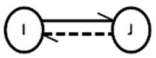


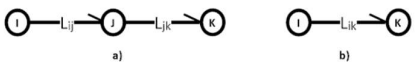
Fig.4 The inverse of a graph arrow

B. Basic graph operations with arrows

Our goal is to ultimately express cycle merger in terms of arrow addition. We first draw a parallel between planar graphs, their nodes and links, and our graph arrow structure. We view the node at infinity as a single node, defined more particularly by the addition that created it. We will additionally view links in the graph as a tuple of arrows, one direct, and one inverse, between the nodes -- this will be shown later in Section IV.

1) Serial addition

Consider a graph containing the nodes i,j,k shown in Fig. 5a, linked by graph arrows (representing directed links in the graph). The addition operation yields the arrow in Fig. 5b.



[1] We use the name "arrow at infinity" similarly to how points at infinity are defined for elliptic curves. This special arrow is in fact an equivalence class of self - loops.

Fig.5 Serial composition of graph arrows

2) Parallel composition of graph arrows

For the most part in this paper we have considered graphs which contain no parallel links. However, the arrow addition operation may yield such parallel links. The second rule of addition shows us that the parallel composition of graph arrows (as depicted in Fig.6a) yields a single arrow (as shown in Fig.6b).

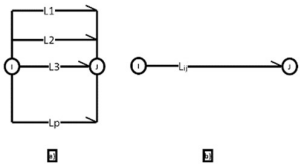


Fig.6 Parallel composition of graph arrows

3) Series-parallel composition of graph arrows

We can combine the two basic operations we described in the previous sections.

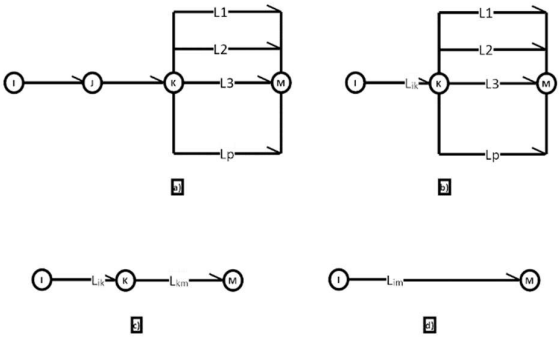


Fig.7 Series-parallel composition of graph arrows

We begin from the complex graph in Fig.7a. By a serial composition on the first three nodes we would obtain the graph in Fig.7b. The parallel composition of the arrows in the right-most node yields Fig.7c. Finally, a renewed serial composition yields the arrow in Fig.7d.

We note that using a sequence of serial and parallel compositions based on arrow addition will ultimately turn a complex planar graph in a simple one, as it will eliminate self-loops and parallel links.

IV. USING GRAPH ARROWS

In this section we show how to use graph arrow addition in finding Hamiltonian circuits, by using as an example the figure of the dodecahedron (shown in Fig. 8). Typically an dodecahedron has single links between the nodes; however, our first step is to de-duplicate each link into two graph arrows, one direct and one inverse.

We label the cycles of the dodecahedron in underlined numbers and choose a random orientation for each cycle (clockwise for the inside cycles, counter-clockwise for the outside one). Adjacent cycles, such as 11 and 6, have one link in common; however, this link is travelled in one direction in one cycle, and in the opposite direction in the adjacent one. As we will indeed show, when de-duplicating the links into graph arrows, this allows us to associate the direct graph arrow to one cycle, and its inverse, to the other. In Table 1 we present the graph-arrow description of the cycles, in which we have deduplicated the links into two arrows which are inverse to one another. This table shows the correspondence between nodes and cycles, and can furthermore be used to generate the cycles – links incidence matrix B. For

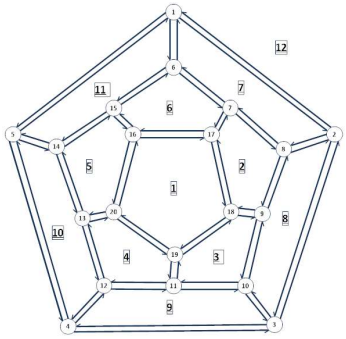


Fig.8 Arrows group graph

Table 1 Cycles-arrows description

Cycle	Arrows
1	$\overrightarrow{(16,17)}$,$\overrightarrow{(17,18)}$, $\overrightarrow{(18,19)}$, $\overrightarrow{(19,20)}$, $\overrightarrow{(20,16)}$
2	$\overrightarrow{(18,17)}$, $\overrightarrow{(17,7)}$, $\overrightarrow{(7,8)}$, $\overrightarrow{(8,9)}$, $\overrightarrow{(9,18)}$
3	$\overrightarrow{(18,9)}$, $\overrightarrow{(9,10)}$, $\overrightarrow{(10,11)}$, $\overrightarrow{(11,19)}$, $\overrightarrow{(19,18)}$
4	$\overrightarrow{(19,11)}$, $\overrightarrow{(11,12)}$, $\overrightarrow{(12,13)}$, $\overrightarrow{(13,20)}$, $\overrightarrow{(20,19)}$
5	$\overrightarrow{(20,13)}$, $\overrightarrow{(13,14)}$, $\overrightarrow{(14,15)}$, $\overrightarrow{(15,16)}$, $\overrightarrow{(16,20)}$
6	$\overrightarrow{(15,6)}$, $\overrightarrow{(6,7)}$, $\overrightarrow{(7,17)}$, $\overrightarrow{(17,16)}$, $\overrightarrow{(16,15)}$
7	$\overrightarrow{(1,2)}$, $\overrightarrow{(2,8)}$, $\overrightarrow{(8,7)}$, $\overrightarrow{(7,6)}$, $\overrightarrow{(6,1)}$
8	$\overrightarrow{(2,3)}$, $\overrightarrow{(3,10)}$, $\overrightarrow{(10,9)}$, $\overrightarrow{(9,8)}$, $\overrightarrow{(8,2)}$
9	$\overrightarrow{(3,4)}$, $\overrightarrow{(4,12)}$, $\overrightarrow{(12,11)}$, $\overrightarrow{(11,10)}$, $\overrightarrow{(10,3)}$
10	$\overrightarrow{(4,5)}$, $\overrightarrow{(5,14)}$, $\overrightarrow{(14,13)}$, $\overrightarrow{(13,12)}$, $\overrightarrow{(12,4)}$
11	$\overrightarrow{(5,1)}$, $\overrightarrow{(1,6)}$, $\overrightarrow{(6,15)}$, $\overrightarrow{(15,14)}$, $\overrightarrow{(14,5)}$
12	$\overrightarrow{(1,5)}$, $\overrightarrow{(5,4)}$, $\overrightarrow{(4,3)}$, $\overrightarrow{(3,2)}$, $\overrightarrow{(2,1)}$

the case of the dodecahedron, this incidence matrix has 30 entries per row (the number of links) and will be not be represented here because of lack of space. However, the non-zero entries, corresponding to the links found in each cycle, are found in Table 1. As the nodes-links and cycle-links incidence matrices are orthogonal, there is no direct adjacency matrix between the nodes and cycles of a graph. However, we can use matrix **B** to find paths in the graph, as we shall show subsequently.

V. ROUTING ALGORITHM USING ARROWS

As discussed in Section I, our graph-arrow addition method will be applied to a specific path-finding algorithm, namely one based on cycle-merger.

In this method, in order to find a route between an initial node i and a final node f, we must first identify which cycles comprise those nodes. Recall that if a node has a degree d then

the respective node can be found in d cycles [1]. Once all such cycles have been identified, one creates a Laplacian of the cycles, and we perform the cycle-merger algorithm until the two nodes are within the same larger cycle [2,3,11]. At this moment we know which cycles have participated in merger. Furthermore, because the result of the program is a larger cycle, a direct path and a return paths exist. The program can be extended to provide all possible routing paths connecting the starting node and the final node, but this will be more time consuming.

The next step should be identifying the routing paths. The result of the algorithm of [10] was a list of the travelled cycles. The complexity of translating this list to usable routes was not taken into account into the complexity analysis, and is indeed costly. In the following we will optimize the format of the results towards a possible use in routing.

Suppose that in our example of the dodecahedron we want to find a path between the start node $i = 1$ and the final node $f = 20$. In the original paper, cycle merger is done starting from rows of the cycle-links incidence matrix. We do the same here, but we depart from the cycle-arrow incidence matric. Each time we are meant to merge two cycles, we add the corresponding rows by using the rules of the addition operation described in Definition 1.

Following the algorithm ultimately results in (for instance) the following vector whose non-zero entries are given in Equation 1.

$$[\overrightarrow{(5,1)}, \overrightarrow{(1,2)}, \overrightarrow{(3,4)}, \overrightarrow{(6,5)}, \overrightarrow{(7,6)}, \overrightarrow{(8,7)}, \overrightarrow{(9,8)}, \overrightarrow{(10,11)}, \overrightarrow{(13,12)}, \overrightarrow{(14,15)}, \overrightarrow{(2,10)}, \overrightarrow{(12,3)}, \overrightarrow{(4,14)}, \overrightarrow{(14,16)}, \overrightarrow{(18,9)}, \overrightarrow{(11,19)}, \overrightarrow{(20,13)}, \overrightarrow{(16,17)}, \overrightarrow{(17,18)}, \overrightarrow{(19,20)}\ ] \quad (1)$$

We note that each entry of this vector represents a tuple of values, each in the set of nodes N. As a first step, we create two node lists: a list denoted T which will contain the first nodes (tails) of each arrow, and a list denoted H which will contain the second nodes (heads) of each arrow. Our goal is to find a path between our chosen nodes, 1 and 20.

We first apply the Knuth-Morris-Pratt (KMP) algorithm [4] to the tail list T to find node 1. Subsequently we find the node that is at the same index in the head list H as the node 1 was in the list T; denote this node by j. We now remove node i from list T and j from H. We subsequently search for node j in list T (using the KMP algorithm), and then continue in the same way until we reach node f. This will give us a path from i to f.

For our example we start from node 1 in list T: we find it on the second position. To this corresponds the value 2 in list H. We remove the two values and continue. We find node 2 in T and the corresponding value is node 10 in list H, etc.

The overall algorithm can be described.

Program Path Finding using graph arrows;

Input: the starting node n_s and the end node n_e;
Determine the cycles containing n_s and n_e;
Determine the cycles paths comprising both n_s and n_e using cycles mergers;
If there is no path, then STOP and print "No path found";
Once a path is found, translate the list of graph arrows to sets T and H as discussed.
By successive applications of the KMP algorithm as described above, find a direct and a return path between the nodes;
Output : the path discovered.

The following observations apply (see [10]):

1. Each time, the number of cycles limits the iterations to less than C.
2. The end criteria of the program can be flexible such that it can be stopped whenever a first path is found or alternatively it can be run until a path with a specific length is obtained.

Because of the linear costs of the KMP algorithm, the overall cost of finding the longest path becomes $O(N^2)$ -- the algorithm has linear complexity and it must be run at most N times. Obviously, the additional finding that the Knuth-Morris-Pratt algorithm adds nothing to the complexity encourages us to say that there are very good prospective that the algorithm may be used in real time systems.

VI. Conclusion and Future Work

In this paper we have presented an alternative method to processing the results of the routing algorithm of [10]. A new graph model has been introduced based on graph-arrows, which in turn provides us with a very efficient implementation of the algorithm. It allows the use of Knuth-Morris-Pratt algorithm, which in turn provides limited overhead in execution time. This encourages us to use this approach in other problems related to graphs.

It is our belief that an arrow-based description of the graph is a very powerful tool for describing graphs and it is worthwhile using it in other graph related problems.

References

[1] A. Bondy, U.S.R. Murty, "Graph Theory", Springer Graduate Texts in Mathematics, 2008

[2] C.E.Onete, M.C.C. Onete, "Building hamiltonian networks using the laplacian of the underlying graph", ISCAS 2015, pp. 145-148.

[3] C.E.Onete, M.C.C. Onete, "Finding the Hamiltonian circuits in an undirected graph using the mesh-links incidence", 19th IEEE International Conference Electronica, Circuits and Systems (ICECS), 2012, pp. 472-475.

[4] https://www.ics.uci.edu/~eppstein/161/960227.html

[5]Balabanian, N., Bickart, T.A.: "Electrical Network Theory", John Wiley and Sons, Inc., 1969.

[6]A. P. Bhondekar, H. Kaur, Routing Protocols in Zigbee Based networks: A Survey, https://www.researchgate.net/publication/275637579

[7]A. Narmada, P. Sudhakara Rao, Performance Comparison Of Routing Protocols For Zigbee Wpan, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 6, No 2, November 2011 ISSN (Online): 1694-0814.

[8]Prativa P.Saraswala, A Survey on Routing Protocols in ZigBee Network, International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 1, January 2013,pp.471-476.

[9]P. S. Shiju Kumar, A. Ramesh Babu, A Survey on Routing Techniques in ZigBee Wireless Networks in Contrast with Other Wireless Networks, Indian Journal of Science and Technology, *Vol 10(42), DOI:10.17485/ijst/2017/v10i42/120345,* November *2017,pp.1-8.*

[10] Onete, C.E., Onete, M.C.C. "An Alternative to Zigbee Routing Using a Cycles Description Of a Planar Graph", MOCAST2019, Thessaloniki 2019, pp. 29 – 32.

[11] Kavitha, Telikepalli & Liebchen, Christian & Mehlhorn, Kurt & Michail, Dimitrios & Rizzi, Romeo & Ueckerdt, Torsten & Zweig, Katharina. (2009). Cycle Bases in Graphs -- Characterization, Algorithms, Complexity, and Applications. Computer Science Review. 3. 199-243. 10.1016/j.cosrev.2009.08.001.