CIT. Journal of Computing and Information Technology, Vol. 28, No. 4, December 2020, 225–240
doi: 10.20532/cit.2020.1005136

Algorithms for Finding Diameter Cycles of Biconnected Graphs

Mehmet Hakan Karaata

Department of Computer Engineering, Kuwait University, Kuwait

In this paper, we first coin a new graph theoretic problem called the *diameter cycle problem* with numerous applications. A longest cycle in a graph $G = (V, E)$ is referred to as a diameter cycle of G iff the distance in G of every vertex on the cycle to the rest of the on-cycle vertices is maximal. We then present two algorithms for finding a diameter cycle of a biconnected graph. The first algorithm is an abstract intuitive algorithm that utilizes a brute-force mechanism for expanding an initial cycle by repeatedly replacing paths on the cycle with longer paths. The second algorithm is a concrete algorithm that uses fundamental cycles in the expansion process and has the time and space complexity of $O(n^6)$ and $O(n^2)$, respectively. To the best of our knowledge, this problem was neither defined nor addressed in the literature. The diameter cycle problem distinguishes itself from other cycle finding problems by identifying cycles that are maximally long while maximizing the distances between vertices in the cycle. Existing cycle finding algorithms such as fundamental and longest cycle algorithms do not discover cycles where the distances between vertices are maximized while also maximizing the length of the cycle.

ACM CCS (2012) Classification: Theory of computation → Graph algorithms analysis

Keywords: biconnected graphs, diameter cycle, fundamental cycles, graph algorithms

1. Introduction

Let $G = (V, E)$ be an undirected bi-connected graph with vertex set V and edge set E. Cycle C of $G = (V, E)$ is referred to as a *candidate diameter cycle* of G iff there does not exist a cycle C' in G such that for two vertices i and j that are on both C and C', there exist two paths $P(i, j)$ and $P'(i, j)$ disjoint except their endpoints i and j in G from vertex i to vertex j where C' is obtained by replacing path $P(i, j)$ on C by path $P'(i, j)$ satisfying

(i) $|P(i, j)| < |P'(i, j)|$ and

(ii) for every vertex v on $C \setminus P(i, j)$, and vertex z on $P(i, j)$ and w on $P'(i, j)$ such that z is a vertex that minimizes $d(v, z)$, w is a vertex that minimizes $d(v, w)$, and $d(v, z) \leq d(v, w)$ holds. (See Figure 1 illustrating the definition.)

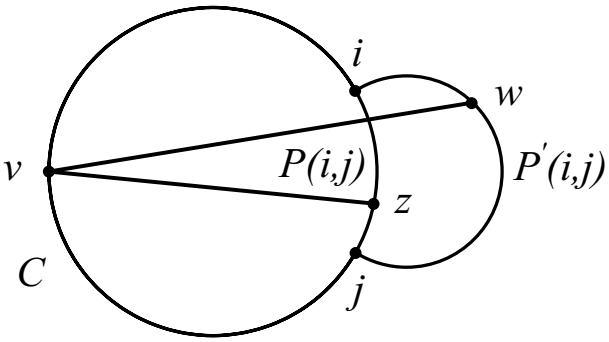

Figure 1. Graph showing path replacement.

Informally, the longest cycle in G on which the distance in G of every vertex on the cycle to the rest of the on-cycle vertices is maximal is referred to as a *diameter cycle* of G.

Consider Figure 2 where a graph is shown with a number of non-candidate diameter cycles and candidate diameter cycles. Observe that cycles 1, 3, 4, 1 and 1, 4, 5, 2, 1 are not candidate diameter cycles since they can be expanded satisfying both properties (*i*) and (*ii*). Notice that cycle 1, 3, 4, 1 can be expanded using cycle

1, 4, 5, 2, 1 and vice versa satisfying both the properties. Also notice that the resulting cycle 1, 3, 4, 5, 2, 1 of both the expansions is a candidate diameter cycle. In addition, there exists another diameter cycle 1, 3, 4, 5, 6, 1 that can be obtained through other expansions.

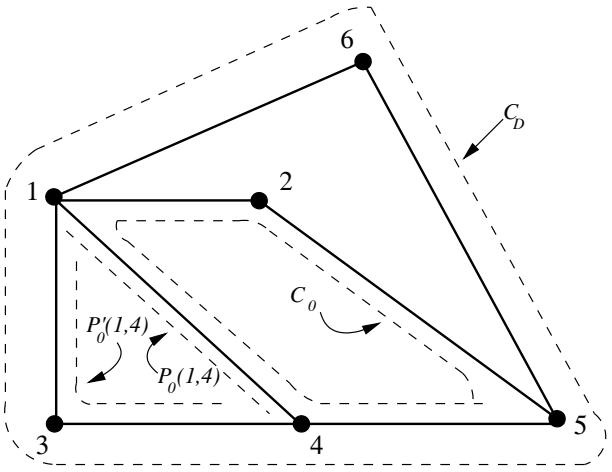

Figure 2. Initial cycle C_0, path $P(1, 4)$ and its corresponding expansion path $P'(1, 4)$ and diameter cycle C_D.

A diameter cycle of a graph resembles the perimeter of the graph in the sense that it encloses all the vertices and edges of the graph (or a subgraph). A diameter cycle represents different notions depending on what the graph represents. For instance, if the graph is used to represent a road network of a city, then this cycle is the outer-city ring road which is considered the least polluted and the one desired for heavy vehicles and trucks.

Many principal characteristics of a communication network are related to the global geometric shape and the topology of the network which can be discovered using the diameter cycle of the network. These characteristics identify the connectivity structure of a network which in turn facilitates improvement of network efficiency, optimizes network facilities placement, provides protection for path failures, detects boundaries, optimizes graph layout, assesses the vulnerability of a network, optimizes routing and scheduling (especially critical path planning), and minimizes the complexity of routing. For example, finding two vertex-disjoint paths between any two vertices in G is simplified through the use of a diameter cycle in G. Section 2 provides details on the applications of diameter cycles.

Various cycle detection algorithms have been proposed in the literature. For instance, [1] presents a diffusing computation-based cycle knot detection algorithm for general graphs. A *knot* is a directed graph in which every vertex is reachable from every other vertex via a directed path. It detects whether a given vertex is in a cycle or knot. To do so, the initiator launches a diffusion computation to detect whether a given vertex is in a cycle or a knot. An algorithm for detecting holes and antiholes in general undirected graphs is discussed in [1], where the existence of a cycle is checked by a special depth first search traversal that proceeds along a chordless path on four vertices of the input graph. Similarly, [2] introduces an adaptive approach (the Adaptive Bellman-Ford algorithm) to negative cycle detection in dynamically changing graphs. This technique explicitly addresses the common, practical scenario in which negative cycle detection is periodically performed after intervals in which a small number of changes are made to the graph. However, to the best of our knowledge, no algorithm for constructing a diameter cycle is available in the literature.

In this paper, we first introduce an entirely new problem of finding diameter cycles in graphs which has numerous applications. Then, we propose an abstract algorithm to find a candidate diameter cycle, and a concrete algorithm to find a diameter cycle in a bi-connected graph. The first algorithm utilizes a brute-force approach based on the expansion of paths on an initial arbitrary cycle of G to find a candidate cycle of G, while the second algorithm employs a novel approach to combine fundamental cycles of a given graph to find a diameter cycle in polynomial time. The second algorithm consists of three distinct phases. The first phase of the algorithm, as given in [3], finds all fundamental cycles in the graph. The second phase of the algorithm, as given in [3], marks each vertex with each fundamental cycle(s) it is part of. The third phase of the algorithm finds a diameter cycle in the graph by combining fundamental cycles.

The paper is organized as follows: In Section 2, applications of diameter cycles are presented. In Section 3, the brute-force algorithm is presented. In Section 4, we propose an algorithm

based on the use of fundamental cycles to find the diameter cycle of a given graph. In Section 5, we present the proof of correctness and show that diameter cycles possess a number of useful properties. Finally in Section 6, we provide some concluding remarks on work present and its applications.

2. Applications

There is a wide range of sensor network applications that require certain knowledge of the global geometry and topology of a network which can be discovered using the diameter cycle of a network. For instance, the identification of the network boundary can be facilitated through the discovery of the diameter cycle that represents the perimeter of the target area. The boundaries of the sensor field capture the global geometric shape of the network which identifies the densely monitored underlying space. The discovery of the global geometry and the topology of the sensor field is of great importance in the design of basic networking operations such as point to point routing, data gathering mechanism and sensor coverage verification. The identification of boundaries [4], *i.e.*, the outer edges/perimeter of the network, and holes [5], *i.e.*, regions without enough active vertices in a network, are useful in improving the overall performance of a network by adapting appropriate routing techniques. Detecting holes allows locally selecting an appropriate routing path. Hole detection also plays an important role in geographic multicasting where a message is delivered from a single source to a set of destination vertices in a geographic region and path migration by maintaining virtual connections among a set of moving objects.

The number of (inner boundaries) holes in a network and their sizes is the most important performance metric used to measure the quality of service a network can provide. This holds especially in dynamic settings, where sensor nodes can run out of power, fail or move forming holes and changing the sizes and the topologies of the inner and outer boundaries. Hence, holes and boundaries need to be dynamically detected in sensor networks. Diameter and candidate diameter cycles can be used to detect holes and boundaries in sensor networks.

Graph drawing is a crucial stage in the process of optimizing network or network overlay design via good graph representation that facilitates a better understanding of the system network [6]. There are multiple approaches to enhance graph layout such as tree layout, hierarchical layout, bus layout and circular layout. In a circular layout (*CL*), the graph is represented by its set of cycles and the edges connecting them. *CL* is best suited for graphs representing interconnected ring and/or star network topologies and has been applied to LAN diagrams, organization charts and web hyperlink neighbourhood representation. Circular graph layout of a graph, which is easier to understand and to analyze, can be constructed using candidate diameter cycles of the graph [6]. Using candidate diameter cycles in a graph, a desirable circular layout of the graph can be discovered. For a given graph G, the circular layout consists of candidate diameter cycles C_{D_1}, C_{D_2}, C_{D_3}, ... where C_{D_i} is a candidate diameter cycle of $G \setminus \{C_{D_1}, C_{D_2}, ..., C_{D_{i-1}}\}$, where $\setminus$ denotes the removal of set of graphs/cycles $\{C_{D_1}, C_{D_2}, ..., C_{D_{i-1}}\}$ from graph G. This results in a layout with a moderate number of cycles. Graph algorithms for layout normalization and enhancement continue to receive significant attention [7, 8].

Path based network protection is an efficient technique for network survival in the presence of edge failures where pre-assigned capacity is used for network restoration [9]. *Network restoration* is a process in which data is rerouted towards the destination in the event of edge failures. Many protection schemes including the *ring-based protection* [10] adopted this concept. In ring-based network protection, a cyclic path is used for network protection where the data is transmitted in the same direction (*e.g.*, clockwise) from source to destination through the cycle that contains them in the network, and simultaneously a copy of that data is transmitted in the opposite direction of the cycle between the source and destination [11]. Ring based protection paths provide fast route switching however they are inefficient in capacity usage [12]. One approach to solve this problem is to use the *p*-cycles (preconfigured protection cycle) in which the spare capacity is organized in cycles where the vertices on each cycle assemble and share a protection path for any on-cycle edge/link in addition to the straddling edges [12].

A diameter cycle provides a general protection path for the vertices on the cycle and other vertices [13], [14] in the network that provides fast path switching; for every vertex x in the network, x is either on a diameter cycle or x has a distance less than or equal to half the diameter of the graph to the diameter cycle as illustrated in Figure 3, where the diameter cycle is shown by a thick line and dashed arrows show the distances of nodes to the diameter paths. It is easy to see that the maximum distance of any vertex in the graph shown in Figure 3 to the closest vertex on the diameter path is $D/2$ which is facilitated by the properties of the diameter path where D is the diameter of graph G, *i.e.*, D is the longest distance (number of edges on the shortest path) between two vertices in G. Therefore a diameter cycle provides an eligible path from any source vertex to any destination vertex in the graph. The selection of a diameter cycle as a protection path balances the distance from an arbitrary vertex to the closest vertex on the cycle and the length of the cycle reducing the average and maximum length of the protection path when a single protection cycle is used. When such an approach is adapted to maintaining all paths between each pair of vertices, sizes of routing tables are also reduced.

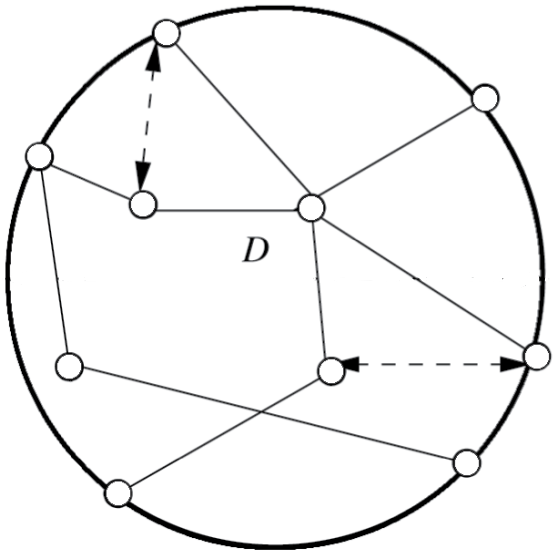

Figure 3. Distance between the vertices in a network and a diameter cycle.

Moreover, diameter cycles provide a solution to FIPP Failure-Independent Path-Protection p-cycle design problem; *FIPP* p-cycle is a concept that extends straddling edges protection as in typical p-cycles [15] to protect independent paths. Since graphs under consideration are bi-connected, each vertex that is not part of a diameter cycle is connected to two distinct vertices on a diameter cycle via two distinct paths allowing a diameter cycle to be used as a *FIPP* p-cycle. In addition, when a diameter cycle of G is used as a *FIPP* p-cycle in G, the number of independent paths protected by the cycle is increased. Figure 4 illustrates the concept of independent paths protected using a cycle. Protection paths and cycles are commonly used in optical networks to enhance performance and reliability [16, 17, 18, 19].

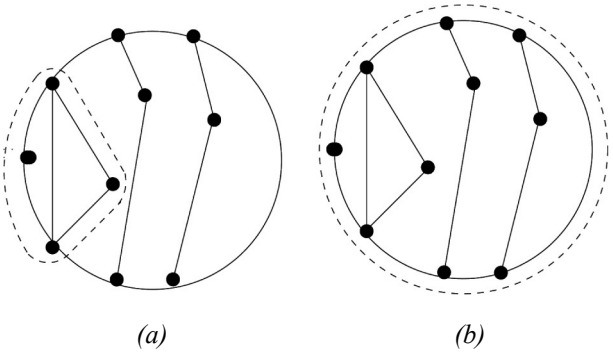

Figure 4. (a) Cycle in G protecting 1 independent path, *(b)* Diameter cycle of G protecting 4 independent paths.

In computer networks, communication between any two arbitrary endpoints requires the establishment of a routing path between them. In the presence of a diameter path, the establishment of such routing paths reduces to the discovery of a path from each vertex (not already on a diameter cycle) to a vertex on the cycle and identification of the next vertex on the cycle. In particular, each vertex not on the cycle maintains information about its neighbour which is the next vertex on the shortest path to the cycle. In addition, each vertex on the cycle maintains in its routing table the following items: next vertex on the cycle to form a directed cycle, neighbours not on the cycle through which a set of vertices not on the cycle can be reached, and the set of vertices accessible through the neighbours. Moreover, finding routes between each pair of vertices on a cycle reduces to selecting one of the two alternative paths available on the cycle for each pair. See Figure 5 where p_1 and p_2 are the two vertex disjoint paths from source to destination. It is easy to see that this scheme

reduces the amount of information to be kept in routing tables.

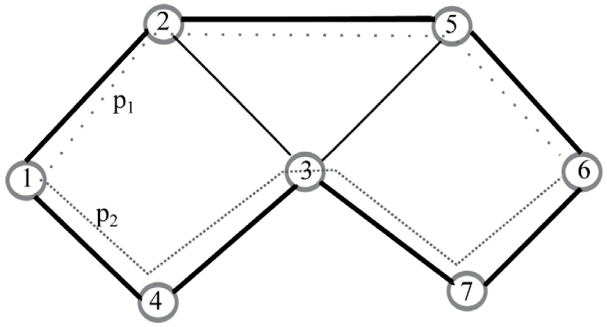

Figure 5. Diameter cycle of a biconnected graph. A vertex disjoint route from vertex 1 to 6 is easily found through the cycle.

The problem of finding two vertex disjoint paths has a large number of applications including increasing the security and reliability of communication in computer networks. Two disjoint paths between each pair of vertices could be built using diameter paths as follows. If both the vertices are on the cycle, obviously the cycle connects the vertices via two disjoint paths [20]. If one of the vertices x is on the cycle while the other vertex y is not on a diameter cycle, a path containing y with its distinct endpoints on the cycle is constructed. Clearly this path and a diameter cycle form a cycle containing both x and y providing two vertex disjoint paths connecting x and y. If both the vertices x and y are not on the cycle, two distinct paths containing x and y with endpoints on the cycle or a single path containing both x and y can be constructed. Obviously, in each case, a cycle containing both x and y can be readily obtained leading to the construction of two disjoint paths between vertices x and y.

3. Candidate Diameter Cycle Algorithm

In order to describe the approach employed by the algorithm, we need to define the following terms. A graph $G = (V, E)$ is *connected* iff a path exists between any two vertices in G [12]. A vertex in a connected graph is referred to as an *articulation* or *cut vertex* if its removal disconnects the graph, i.e, the graph is split into disjoint subgraphs [12]. A graph is *bi-connected* if it does not contain an articulation vertex [21]. A shortest path P of length D between two vertices in G is referred to as a *diameter path*. *Diameter endpoints* are the origin and the terminal vertices on P.

The algorithm is based on constructing a cycle of G between two diameter endpoints and then expanding the initial cycle repeatedly until a point where the obtained cycle can no longer be expanded and each expansion step satisfies a number of properties. It is easy to see that a diameter path can be found by first finding the distance between every pair of vertices and then identifying two vertices of maximal distance between them.

1. Select any cycle C_0 containing two diameter endpoints of a diameter path of G
2. $i = 1$
3. **while** true **do**
4. **if** *there exist path* $P(x, y)$ *on* C_{i-1} *with origin* x *and terminal* y *and another path* $P'(x, y)$ *disjoint from* C_{i-1}*, except for its origin* x *and terminal* y *such that* $|P'(x, y)| > |P(x, y)|$*, and for every vertex* v *on* $C \setminus P(x, y)$*, and vertex* z *on* $P(x, y)$ *and* v *on* $P'(x, y)$ *such that* z *is a vertex that minimizes* $d(v, z)$*,* v *is a vertex that minimizes* $d(v, w)$*, and* $d(v, z) \leq d(v, w)$ *hold*. **then**
5. Obtain cycle C_i by replacing its subpath $P(x, y)$ by $P'(x, y)$ in C_{i-1}
6. Increment i
7. **else**
8. exit
9. **end**
10. **end**
11. Identify the last cycle produced by the cycle expansion process as candidate diameter cycle C_D

Figure 6. Candidate diameter cycle algorithm.

Note that $|P'(x, y)|$ and $|P(x, y)|$ refer to the lengths of paths $P'(x, y)$ and $P(x, y)$, respectively, and $C_{i-1} \setminus P(x, y)$ refers to a path obtained from C_{i-1} by removing $P(x, y)$.

The algorithm starts by finding an initial cycle containing both the endpoints of a diameter path of G by forming two distinct vertex-disjoint paths of total minimal length between the diameter endpoints and combining them to form the initial cycle. Such a cycle can be constructed using an algorithm to find two vertex-disjoint paths between the diameter endpoints using the

algorithm in [22]. The reason for starting with such an initial cycle is because such a cycle can be shown to comply with the restrictions of a candidate diameter cycle. The process of cycle expansion continues by replacing a path on the initial cycle with one that is longer and satisfying the *cycle expansion condition*; this process is repeated on the resulting cycle until no further expansion is possible. The cycle expansion condition ensures that no vertex v on the cycle is replaced by another vertex v that is closer to an arbitrary vertex y on the cycle than v in G in a cycle expansion step. In this process, the length of the cycle is increased by replacing paths of the cycle with other paths in G that are longer in size. The cycle that can no longer be expanded is identified as a candidate diameter cycle C_D by the cycle expansion process.

We now formally describe the cycle expansion process. Let vertices i and j be two diameter endpoints of a diameter path in G. Let P_f and P_s be two vertex-disjoint paths of minimal total length connecting diameter endpoints i and j, and C_0, referred to as *initial cycle*, be the cycle formed by two vertex-disjoint paths P_f and P_s which can be found using the algorithm proposed in [23]. Also let C_0, C_1, C_2, ..., C_k be a sequence of cycles in G such that each cycle C_i for $0 < i \leq k$, is obtained through the expansion of cycle C_{i-1}. Path $P(x, y)$ for $0 < i < k$, refers to a path on C_{i-1} with origin x and terminal y. *Expansion path* $P'(x, y)$ of path $P(x, y)$ refers to a path on C_i in G for $0 < i \leq k$ where $P'(x, y)$ is disjoint from $P(x, y)$ except for its origin x and terminal y such that path $P'(x, y)$ is longer than path $P(x, y)$. Each expansion of cycle C_{i-1} to C_i is carried out by replacing a subpath $P(x, y)$ with a longer path $P'(x, y)$ such that $P(x, y)$ and $P'(x, y)$ satisfy the following cycle expansion conditions:

- $|P'(x, y)| > |P(x, y)|$
- for every vertex v on $C \setminus P(i, j)$, and vertex x on $P(i, j)$ and v on $P'(i, j)$ such that z is a vertex that minimizes $d(v, z)$, v is a vertex that minimizes $d(v, w)$, and $d(v, z) \leq d(v, w)$ holds; that is, for two vertices v, y on C_{i-1} at distance D in G, vertex v is not replaced by another vertex v on C_i at distance less than D to y in G by a cycle expansion step, where cycle in C_i is the cycle obtained from C_{i-1} in a single cycle expansion step.

After expanding an initial cycle to the point where no more cycle expansions satisfying the cycle expansion rules are possible, the resulting cycle is identified as a candidate diameter cycle C_D. The algorithm implementing the aforementioned approach is given in Figure 6.

The following example demonstrates the way a candidate diameter cycle is found by the algorithm: given the graph in Figure 2, the algorithm starts by computing the shortest distances between every pair of vertices and the diameter of the graph. The diameter of the graph in Figure 2 equals 2, thus vertices 1 and 5 are diameter endpoints; edges (1, 2), (2, 5), (5, 4), (4, 1) form an initial cycle C_0, and by performing a cycle expansion process we obtain the following candidate diameter cycle: (1, 3), (3, 4), (4, 5), (5, 2), (2, 1). In the next step, the initial cycle is expanded by identifying $P(1, 4)$ and $P'(1, 4)$ satisfying the cycle expansion rule and by replacing $P(1, 4)$ by $P'(1, 4)$ in C_0 to obtain cycle C_1 made up of edges (1, 3), (3, 4), (4, 5), (5, 2), (2, 1). Then the cycle expansion process is applied to C_1 which is not possible in this case; thus the cycle C_1 is identified as candidate diameter cycle C_D. The diameter cycle of a graph given in Figure 2 is not unique. Observe that edges (1, 4), (4, 5), (5, 6), (6, 1) form another initial cycle leading to the construction of another candidate diameter cycle (1, 3), (3, 4), (4, 5), (5, 6), (6, 1) of the graph. Notice that both cycles contain the same number of edges and are candidate diameter cycles of G.

The proposed algorithm finds a candidate diameter cycle of G. The same algorithm can be used to find other candidate cycles by executing the algorithm for initial cycles formed using different diameter paths than the ones used in earlier executions. Upon finding all candidate cycles, the longest candidate diameter cycle can be identified as a diameter cycle of G.

The proposed algorithm is referred to as a brute-force algorithm since it does not include an efficient mechanism to select the expansion path and the selection that results in the minimum number of path expansion steps to find a candidate diameter cycle. In addition, an efficient mechanism to select initial cycles that do not result in candidate diameter cycles found in its earlier iterations to find candidate diameter cycles is not included in the algorithm. The cor-

rectness of the algorithm follows from the fact that if a cycle is not a candidate diameter cycle, the proposed cycle expansion process expands the cycle. In the next section, we present a more efficient algorithm to find diameter cycles.

4. Diameter Cycle Algorithm

In this section, we first informally describe the proposed *Diameter Cycle* (*DC*) algorithm and introduce the notation used for the description of the algorithm. We then formally present the algorithm. Unlike the first algorithm, the *DC* algorithm is a concrete algorithm, hence, all implementation details are given.

4.1. Basis of the Algorithm

The algorithm consists of three phases. The first phase of the algorithm is given in [3] and is responsible for traversing a bi-connected graph G and identifying all the *fundamental cycles* in G. If T is a spanning tree of G and e is a *non-tree edge* (*i.e.*, an edge that is not included in T), then the fundamental cycle defined by e is the simple cycle consisting of e and the path in T that connects the endpoints of e. The second phase of the algorithm is executed after the first phase terminates and is responsible for marking every vertex in the graph with the unique id of the fundamental cycle it is part of. For the first two phases, we adapt the algorithm given in [3] to identify and mark the fundamental cycles of a graph. Note that although the algorithm given in [3] is distributed, it is straightforward to devise its sequential counterpart. Vertices that are part of two or more fundamental cycles are marked with multiple ids using the non-tree edge id's corresponding to the fundamental cycles in which they are included. Using these markings, we identify intersection paths between *neighbouring fundamental cycles* in the final phase of the algorithm. An intersection path refers to a path in G that two cycles share. Two fundamental cycles are referred to as neighbouring if they share an intersection path. We use this neighbourhood knowledge to combine fundamental (and other) cycles to obtain longer cycles.

The identification of fundamental cycles, the manner they are marked, and how the marking are used to identify neighbouring cycles are illustrated with the help of the example in Figure 7. In the figure, a graph with a BFS tree rooted at Vertex 11 is shown where each tree edge is shown by solid lines and each non-tree edge is shown by dashed lines. The figure also illustrates the basis of the algorithm in [3]. The graph contains four non-tree edges, namely {1, 2}, {4, 5}, {6, 7}, and {7, 8}. Observe that fundamental cycles $FC(\{1, 2\}) = \{1, 2, 10\}$, $FC(\{4, 5\}) = \{3, 4, 5, 6, 10\}$, $FC(\{6, 7\}) = \{6, 7, 9, 10\}$ and $FC(\{7, 8\}) = \{7, 8, 9, 10, 11\}$ exist in the graph. Notice that $FC(\{1, 2\})$ does not have a neighbouring FC, whereas, $FC(\{4, 5\})$ and $FC(\{6, 7\})$, and $FC(\{7, 8\})$ are neighbouring. In the algorithm given in [3], each nodal process maintains a variable b called b-set of node i and denoted by b_i containing a set of tuples. Informally, starting from the nodes at the largest depth of each biconnected component B, the b-set of each node i in B collects the set of descendants of i contained in the biconnected component containing i in a bottom-up fashion. Eventually, the b-set of the ancestor of each biconnected component B contains the set of its descendants in the biconnected component. If i is a node in biconnected component B such that each node in $B \setminus \{i\}$ is a descendant of i, then node i is referred to as the ancestor of biconnected component B. The fundamental data structure b-set of the algorithm is not a simple set, and therefore requires further explanation. We now describe b-sets in more detail. Each tuple in $b(i)$ is of the form $\langle x, y\rangle$, where $x = \{p, q\}$ denotes a non-tree edge incident on a descendant of i joining nodes p and q, and y denotes a set of descendants of i. It is easy to see that for any process $i \in V$, the first element of each tuple $\langle x, y\rangle \in b(i)$ is the non-tree edge id denoting that process i is contained in the fundamental cycle formed by the non-tree edge. For instance, both $b(6)$ and $b(10)$ include both non-tree edge ids {4, 5} and {6, 7} as the first element in their tuples denoting that path 6, 7 is an intersection path of cycles formed by non-tree edges {4, 5} and {6, 7}. Similarly, $b(7)$, $b(9)$ and $b(10)$ include both non-tree edges' ids {6, 7} and {7, 8} as the first element in their tuples indicating that path 7, 9, 10 is an intersection path of cycles formed by non-tree edges {6, 7} and {7, 8}. Observe that in this manner intersection paths are marked by the non-tree edges ids.

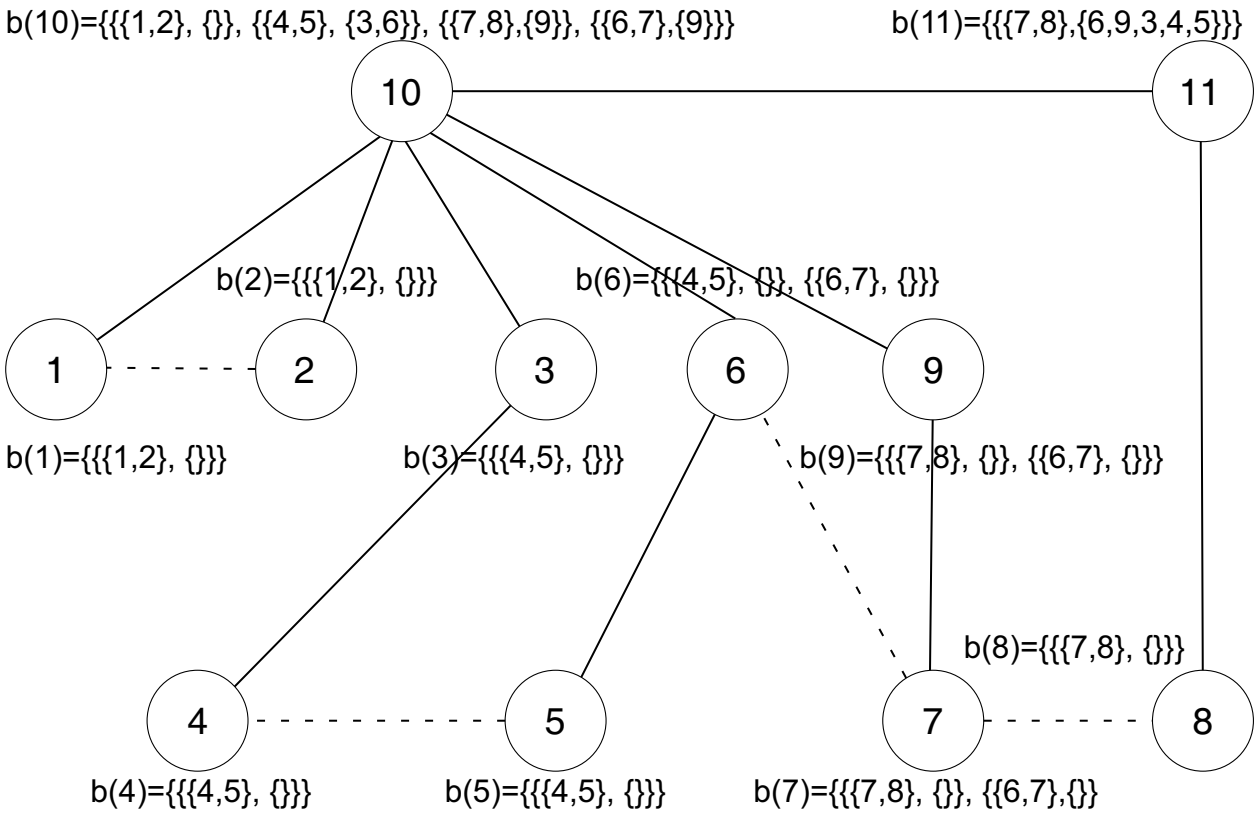

Figure 7. State of a system after termination.

Upon termination of the second phase, the third (final) phase of the algorithm is entered in which a diameter cycle in the graph is identified. In this phase, a cycle referred to as the candidate cycle is considered for expansion and expanded in each step of the process. The proposed algorithm first chooses the longest fundamental cycle as the candidate cycle. Second, it marks the sequence of vertices that forms the candidate cycle by traversing it. While vertices are being marked, its neighbouring cycles and the intersection paths of the candidate cycle with neighbouring cycles are also identified. The algorithm also determines the number of *intersection paths* between the candidate cycle and each of the neighbouring cycles. An intersection path of two cycles refers to a maximal path that is common to both the cycles. Once the traversal of the candidate cycle is complete, the proposed algorithm chooses one of its neighbouring cycles, if any, in order to perform the *expansion*. This latter refers to the process where two neighbouring cycles are combined by removing their intersection paths, iff this results in a longer cycle satisfying the properties (*i*) and (*ii*) in a cycle expansion step. Combining two neighbouring cycles could potentially result in one or more disconnected paths/cycles as shown in Figure 9 where cycle c_2 is disconnected from the rest of the vertices when cycles c_0 and c_1 are combined. As a result of being disconnected from the current candidate cycle, these paths/cycles will not be considered for expansion of the current candidate cycle. Furthermore, they might also not be considered to become part of a candidate diameter cycle formed when building a different candidate cycle which is formed when the algorithm chooses a new initial cycle to expand. Therefore, each cycle which is disconnected in an expansion step is added to the set of non-expandable cycles so that it is considered in later expansion steps. The set of non-expandable cycles contains the set of cycles that cannot be used for expansion currently and are to be considered later for expansion. If such a neighbouring cycle is found and is combined with the candidate cycle, the combined cycle is identified as the new candidate cycle and it is marked. Subsequently, the neighbouring cycle is removed from the set of neighbouring cy-

```
1.  set of all not yet considered fundamental cycles ac = fc
2.  set of expandable cycles ex = ∅
3.  set of non-expandable cycles ne = ∅
4.  choose the longest fundamental cycle in fc and include in ne and remove from ac
5.  while ne ≠ ∅ ∧ ac ≠ ∅ do
6.    if ne ≠ ∅ then
7.      select a cycle cc from ne and remove it from ne
8.    else
9.      select a cycle cc from ac and remove it from ne
10.   end
11.   traverse cc and identify its set of neighbouring cycles ncs
12.   while ncs ≠ ∅ do
13.     select a neighbouring cycle nc from ncs and remove it from ncs
14.     if (removing the intersection path between nc and cc results in a longer cycle and no two vertices on
        either cycle get closer together after combining) then
15.       if nc ≠ ex then
16.         remove nc from ex}
17.       end
18.       combine cycles nc and cc, and make it the new candidate cycle cc
19.       traverse cc and update the neighbouring cycle set ncs
20.       add any neighbouring cycle that becomes disconnected as a result of the expansion to ne
21.     else
22.       if nc ∉ ex then
23.         remove nc from ac
24.         add nc to ne
25.       end
26.     end
27.   end
28.   add cc to ex
29. end
30. select the longest cycle in ex as DC
```

Figure 8. The DC Algorithm using Fundamental Cycles.

cles and the set of non-expanded cycles, if it exists. If removing the intersecting path does not result in a longer cycle, the neighbouring cycle is included in the set of non-expandable cycles for later consideration, provided that it is not already present in the set of *expandable cycles*. The set of expandable cycles contains cycles that could not be expanded at the time it is considered for expansion by removing its intersection path with the candidate cycle. Note that a neighbouring cycle that cannot be combined with the candidate diameter cycle in the current expansion can be used in a later expansion step. Afterwards, another neighbouring cycle of the candidate cycle is considered using the same steps starting from the second step as described above. The algorithm proceeds in this manner until the candidate cycle cannot be expanded with any of its neighbouring cycles. If the current candidate cycle cannot be expanded further, it is added to the set of expandable cycles. Before terminating, the algorithm repeats the expansion process for all the remaining cycles in the non-expanded set. This would result in one or more cycles remaining in the set of expandable cycles which cannot be combined

with any other cycle. The diameter cycle is identified as the longest one in the set of expandable cycles.

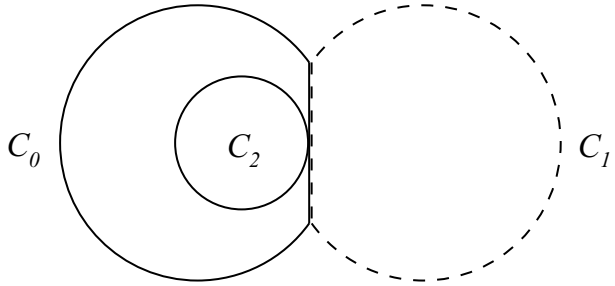

Figure 9. Forming a disconnected cycle by combining two cycles.

The last phase of the above approach, which consists of the algorithmic steps after the fundamental cycles are found, is summarized using the pseudocode given in Figure 8.

4.2. Algorithm Description

Now we describe the algorithm and provide implementation details.

The first phase of the algorithm identifies all the fundamental cycles in the bi-connected graph and stores them in a variable called *fc*. The variable *fc* is of the form $fc\{a, b\} = \{id_0, id_1, id_2, ..., id_n\}$, where $\{a, b\}$ refers to the non-tree edge of a fundamental cycle, and id_i, $0 \leq i \leq n$, refers to the vertex (id) contained in the fundamental cycle formed by non-tree edge $\{a, b\}$. Note that the non-tree edge of the fundamental cycle is used as the unique id for the cycle. Hence, the *fc* variable associated with each cycle stores the set of vertices that are part of the cycle. The second phase of the algorithm marks each vertex in the graph with the fundamental cycle(s) it is part of. For each vertex i, this is done by assigning the unique id of the cycle(s) containing vertex i to the s-set of each vertex i. For each vertex i, s-set $s(i)$ stores cycle ids in the form $s(i) = \{\{a, b\}, \{c, d\}, ...\}$, where $\{a, b\}, \{c, d\}, ...$ are non-tree edges associated with fundamental cycles containing vertex i. This allows the identification of intersections between two cycles by examining the s-sets of vertices in the cycles.

After computing the fundamental cycle set *fc* and the s-set of each vertex in phases one and two, in the third phase of the algorithm, the longest fundemental cycle, given by $max(fc)$, is designated as the candidate cycle *cc*. The sequence of vertices forming the candidate cycle is marked. A variable, called *ce*, is maintained for each vertex i. $c(i)$ contains the non-tree edge identifying the cycle, and the vertex id of the next vertex on the cycle.

We now describe the computation of variable *ce* of vertices on *cc* to mark *cc*. Variable *ce* of a vertex denotes the sequence of vertices in a cycle, where $ce(i) \in \{t_0, t_1, ...\} = \emptyset$ initially holds, i is a vertex on the cycle, and t_j, $0 < j$, is of the form $\{c, v\}$ where c is a cycle id and v is the vertex id i is pointing to. Starting from one of the vertices on *cc*, the algorithm traverses all the vertices on *cc* to identify its neighbouring cycles and stores the set of neighbouring cycles in variable *ncs*. Vertices that contain multiple cycles in their s-sets are vertices that are shared between two or more fundamental cycles. In the traversal, the next vertex is identified as a vertex containing the id of *cc* in its s-set. Before moving to the identified neighbour, the variable *ce* of the current vertex is assigned the id of the identified neighbour so that the variable *ce* of each vertex points to the next vertex on the cycle. Then, the same is repeated for the neighbouring vertex, and so on. In this manner, *cc* is marked where *ce* variables of vertices on *cc* form a cycle. While the cycle is marked, the algorithm also identifies the neighbouring cycles of *cc* and stores them in variable *ncs*.

After all the vertices on the candidate cycle are traversed in this manner, a neighbouring cycle is chosen from *ncs* and is assigned to the variable *nc*. The candidate cycle *cc* and cycle *nc* are examined. If removing the intersecting path and combining the two cycles results in a longer cycle, and no vertex v on the cycle is replaced by another vertex v that is closer to an arbitrary node y on the cycle than v in G in a cycle expansion step, then the candidate cycle and cycle *nc* are combined and this new cycle is designated as the new candidate cycle. In addition, any neighbouring cycle of the candidate cycle that is disconnected is added to the non-expanded set *ne*. Then, it is traversed again to update *ce* values and the neighbouring cycle set *ncs*. Note that if *nc* was present in the set of expandable cycles *ex*, *nc* is removed from *ex* before the cycles are combined. This is done since cycle *nc* is

used in the expansion process to obtain a longer cycle. The set *ex* is maintained by our algorithm to keep track of the candidate cycles such that a candidate cycle ζ is added to set *ex* if ζ currently cannot be combined with any of its neighboring cycles. However, cycles in set *ex* could potentially be combined with other cycles after the other cycles are expanded and become longer in later stages.

During the expansion process, a candidate cycle may not be expanded with any of its neighbours and the cycles to consider in *ne* may have been exhausted though there are fundamental cycles that have not been considered for expansion. To ensure that all fundamental cycles are considered, the set of all fundamental cycles is kept in variable *ac* and any fundamental cycle not considered before is selected from *ac* when the candidate cycle cannot be expanded and set *ne* is empty.

Subsequently, the next neighbouring cycle in *ncs* is considered. If removing the intersecting path does not result in a longer cycle, then the algorithm adds *nc* to *ne* for further consideration later, provided *nc* is not present in *ex*. If *nc* is already present in *ex*, there is no need to consider it later as it has already been considered for expansion. This process continues until we have no more neighbouring cycles to look at and the candidate cycle is eventually added to *ex*. The algorithm repeats the process for the remaining cycles in *ne* and then in *ac*. Once they have all been considered, the longest remaining cycle in *ex* is marked as the *DC*.

We need the following notation to facilitate the description of the algorithm.

Tuple $\langle c, v_{\text{next}} \rangle$ denotes a tuple where the first element is cycle *c* and second element is vertex v_{next}. Tuple $\langle x, - \rangle$ denotes a tuple that has cycle (id) *x* as its first element and the second element does not matter (*i.e.*, it is irrelevant). For a tuple whose first element is a cycle id and the second element is an integer, tuple $\langle x, a{+}{+} \rangle$ denotes a tuple whose first element is *x* while the second element is *a* plus one. Using the above descriptions, the rest of the similar notation can readily be understood. The algorithm described above is given in Figure 10.

Parameters

$N(i) \in \{1, 2, ..., n-1\}$:	denotes the set of neighbours of vertex *i*, where *n* represents the set of vertices in the network
fc:	denotes the set of all fundamental cycles in *G* (computed in earlier phases)
$s(i) \subset E$:	denotes the non-tree edge(s) assigned to each vertex *i*, revealing which fundamental cycle(s) it is part of (computed in earlier phases)

Variables

$ncs \in \{p_0, p_1, ...\}$	denotes a set of neighbouring cycles $\{p_0, p_1, ...\}$ where p_j, $0 < j$, is of the form id_{fc} where id_{fc} is a cycle id
$ne \subset E$:	denotes the set of non-expandable cycles that are yet to be considered
$ex \subset E$:	denotes the set of expandable cycles
$ce(i) \in \{t_0, t_1, ...\}$:	denotes the sequence of vertices in a cycle, where $ce(i) \in \{t_0, t_1, ...\} = \varnothing$ initially holds, so that *i* is a vertex on the cycle, and t_j, $0 < j$, is of the form $\{c, v\}$ where *c* is a cycle id and *v* is the vertex id *i* is pointing to
ac:	set of all fundamental cycles
nc:	temporary set of all neighbouring cycles *x*

Functions

$lenInt(c_1, c_2)$:	takes two cycle id's as parameters and returns the length of their intersection paths
$lenNInt(c_1, c_2)$:	takes two cycle id's as parameters and returns the length of cycle c_1 after subtracting the length of the intersection path between the two cycles.
$max(s)$:	returns the longest cycle among set of cycles in *s*

Figure 10. Algorithm to find Diameter Cycles in an Arbitrary Biconnected Graph.

mark(c) ≡
$v := x \mid x \in c$;
do ($\exists_{v_{next}} \in N(v)\{\{c, v_{next}\} \notin ce(v)) \wedge c \in s(v_{next})\}$)
if $\{\exists_{x \in ce(v)}\ \{\{x, -\} \notin ncs\}\} \rightarrow ncs := ncs \cup \{x, 0\}$;
if $\{\exists_{x \notin ce(v)}\ \{\{\{x, a\} \in ncs\} \wedge \{x \in ce(v_{next})\}\} \rightarrow ncs := ncs \setminus \{x, a\} \cup \{x, a{+}{+}\}$;
$ce(v) = ce(v) \cup \{c, v_{next}\}$; $ce(v_{next}) = ce(v_{next}) \cup \{c, v\}$;
od

combine(cc, nc) ≡
$i = 0$;
do ($\exists_{v \in V}(\{nc, -\} \in ce(v))$)
$v = x \mid \{nc, -\} \in ce(x)$;
if ($\{cc, -\} \in ce(v) \wedge \{nc, -\} \in ce(v)) \vee (\{cc, -\} \notin ce(\mathrm{v}) \wedge \{nc, -\} \in ce(v))$)
if ($\exists_{v_{next}} \in N(v)\ \{nc, v_{next}\} \in ce(v)$)
$ce(v) := ce(v) \setminus \{nc, v_{next}\} \cup \{cc, v_{next}\}$;
$ce(v_{next}) := ce(v_{next}) \cup \{cc, v\}$;
else if ($i = 0$)
$first := v$; $i := 1$;
od

Actions
$nc = \emptyset$;
$ex = \emptyset$;
$ne = ne \cup$ **max**(fc)};
$ac = fc \setminus$ **max**(fc);
do ($ne \neq \emptyset \wedge ac \neq \emptyset$)
if $ne \neq \emptyset$
$cc := \{\ \{x\} \mid x \in ne\}$;
else
$cc := \{x\} \mid x \in ac\}$;
$ne := ne \setminus cc$;
mark(cc);
do ($ncs \neq \emptyset$)
$nc = x \mid \{x, 0\} \in ncs$
$len_{int} = lenInt(cc, nc)$;
if ($len_{int} < LenNInt(nc, cc) \wedge len_{int} < LenNInt(cc, nc) \wedge \neg\exists_{(a \in cc \wedge b \in cc \cap nc)}\ (d(a, b) < d(a, c))$)
if $nc \in ex$
$ex = ex \setminus nc$
combine(cc, nc);
mark(cc);
else
if ($nc \notin ex$)
$ac := ac \setminus nc$
$ne = ne \cup nc$
od
$ex = ex \cup cc$;
od
$output \rightarrow max(ex)$

Figure 10 (cont.)

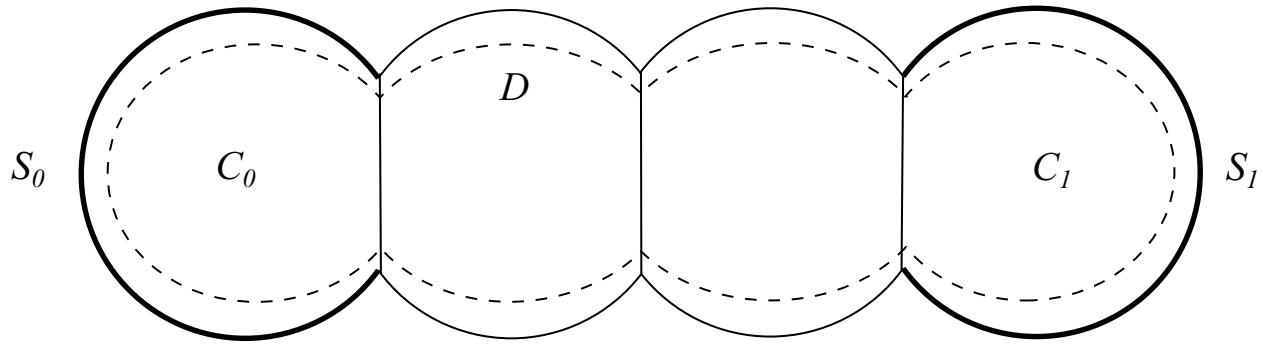

Figure 11. Diameter cycle D for Proof of Correctness.

5. Proof of Correctness

In this section, we provide a correctness proof for the proposed diameter cycle algorithm.

Lemma 1. The cycle expansion process eventually considers every fundamental cycle in the graph.

Proof. The algorithm maintains a set of fundamental cycles contained in the graph. The algorithm starts by picking the largest fundamental cycle from this set, upon which the expansion process is applied. Then, the algorithm considers the neighbours of the selected fundamental cycle for expansion. Subsequently, the neighbours of the neighbouring cycle are considered that are used for expansion, and so on. If a fundamental cycle is not reached in the expansion process, it remains in set *ac* and is eventually considered (when the current cannot be expanded further). Since each fundamental cycle is a neighbour of another in a biconnected graph, and the number of fundamental cycles in a particular graph G is finite, the algorithm eventually visits and considers every fundamental cycle either as a neighbour of a cycle under consideration or as a cycle not considered in set *ac*. Hence the proof follows. □

Consider the graph given in Figure 11, where cycles C_0, C_1, subcycle S_0 of C_0, subcycle S_1 of C_1 are clearly shown. The figure also shows diameter cycle D using dashed lines. The following lemma shows that if there exists cycles C_0 and C_1 with subcycles S_0 and S_1, respectively such that S_0 and S_1 are part of a diameter cycle of the graph, cycles C_0 and C_1 are eventually combined to include S_0 and S_1 in D.

Lemma 2. Let C_0 be a candidate cycle and C_1 be another cycle in graph G such that both cycles contain segments that are part of diameter cycle D. When the proposed expansion process is applied to C_0, cycles C_0 and C_1 are eventually combined and the segments are included as part of D.

Proof. Consider the diameter cycle D where segments of C_0 and C_1 are part of diameter cycle D. Since G is a biconnected graph and segments of C_0 and C_1 are part of diameter cycle D, either C_0 and C_1 are neighbouring cycles or there exists a set of one or more fundamental cycles connecting C_0 and C_1.

Case 1: Cycles C_0 and C_1 are neighbours. In this case, clearly cycles C_0 and C_1 are combined and the segments of C_0 and C_1 are included in the candidate cycle.

Case 2: Cycles C_0 and C_1 are not neighbours. It is easy to show by contradiction that there exists a sequence of cycles $\Gamma_0 = C_0, \Gamma_1, ..., \Gamma_k = C_1$ in G such that there exists two consecutive cycles Γ_i and Γ_{i+1}, $0 \leq i < k$, that can be combined leading to a sequence of cycles of length $k - 1$. Since the proposed algorithm eventually selects the cycles that can be combined in the expansion process by Lemma 1, it can be shown inductively that cycles C_0 and C_1 are eventually combined.

In addition, it is easy to show by contradiction that upon its inclusion in the candidate cycles, since no segment/path of longer length connecting its endpoints can be found (due to the segment being in diameter cycle D), the segments of C_0 and C_1 in D remain as part of the candidate cycle during the expansion process. Hence the proof follows. □

Lemma 3. If two neighbouring cycles C_1 and C_2 are expanded by removing the common path P between them, P will not be included in the diameter cycle found or in any other cycle obtained in intermediate expansion steps.

Proof. Recall that in each step of the expansion process, an intersection path of two cycles is replaced by a longer path. Let C_0 be a cycle which expanded using cycle C_1 by removing the intersection of the two cycles. Let path P with endpoints of i and j be the intersection of two initial cycles to obtain the resulting cycle. Also, let P' and P'' be the remainder of cycle C_0 and C_1, after the removal of path P from cycle C_0 and C_1, respectively. Observe that if P is included again in the candidate cycle, one of P', P'', or a longer path that replaced one of them needs to be excluded. This contradicts the definition of a candidate diameter cycle and the expansion process, as no two vertices on the graph should get closer upon an expansion step. This is a contradiction. Hence, the proof follows. □

Lemma 4. The state-space and time complexity of the algorithm are $O(n^2)$ and $O(n^6)$ respectively.

Proof. We require each vertex in the graph to store its set of neighbours variable $N()$ and the cycles it is part of variable *ce*. For n vertices, each of these stored values have a state-space of $O(n)$ and hence, the algorithm requires state-space of $O(n^2)$ in total for these variables. In addition, it is easy to observe that each of the variables *fc*, *ncs*, *ne*, *ex*, and *ac* contribute to the state-space complexity by $O(n^2)$.

Consider Figure 8 and the algorithm given in Figure 10. The first loop in the algorithm has a time complexity of $O(n^2)$ as it depends on the number of non-expandable cycles (marked using edges), which is $O(n^2)$ in the worst case. Traversing and marking a cycle takes $O(n)$ steps inside the outer loop. The inner loop depends on the size of the neighbouring cycle set *ncs* which also has an upper bound of $O(n^2)$. Inside the inner loop, we have two steps that contribute to the time complexity. First, combining two cycles would depend on the length of the cycles, which has time complexity of $O(n)$. Second, traversing and marking the new candidate cycle takes $O(n)$ steps. Hence, the time complexity of the algorithm is $O(n^6)$. Hence the proof follows. □

Theorem 1. The proposed diameter cycle algorithm finds a diameter cycle of a given graph G in $O(n^6)$ time with space complexity of $O(n^2)$.

Proof. By Lemma 1, we know the algorithm eventually considers every fundamental cycle in the graph.

By Lemmas 1 and 2, every fundamental cycle is to be considered in the expansion process, the segments of the fundamental cycles that are contained in a diameter cycle is included in a candidate cycle, and they remain as part of the candidate cycle. Therefore, each candidate cycle is expanded until a point where it can no longer be expanded.

Also observe that after a candidate cycle is expanded into a candidate diameter cycle, a cycle/fundamental cycle in the non-expanded set or a fundamental cycle unconsidered, if any, is considered for expansion. If the newly considered candidate cycle is not combined with previously discovered candidate cycles, a new candidate cycle is formed, which leads to the discovery of a new candidate cycle. When a fundamental cycle is considered, its segments in the diameter/candidate cycle are added to the candidate cycle and they remain as part of the candidate cycle by Lemma 3. In this manner, all candidate diameter cycles are found. The longest candidate cycle among them is identified as the diameter cycle of the graph.

The time and space complexites of the algorithm follow from Lemma 4. Hence the proof follows. □

6. Conclusion

This paper first presents a brute-force abstract algorithm for finding a candidate diameter cycle based on constructing a cycle of G between two diameter endpoints and a cycle expansion process satisfying a number of rules. The process of cycle expansion starts by replacing each sub-path in the initial cycle with one that contains more edges and this process is repeated on the resulting cycle till no further expansion is possible. The cycle produced by the cycle expansion process that can no longer be expanded is identified as a candidate diameter cycle. The second algorithm employs a novel implemen-

tation using fundamental cycles in the expansion process.

Finding a diameter cycle in the graph is useful in improving cycle layout representation of graphs where it minimizes link crossings and reduces the graph area. In addition, diameter cycle provides a ring-based graph protection for all the vertices in the graph. Detection of the global geometric shape of the network allows the connectivity structure of a network to be discovered and improves network efficiency, optimizes certain parameters of the network, protects path failures and minimizes the complexity in routing. Identification of diameter cycles is also useful in network design; which serves as a survival mechanism against vertex failures and provides failure independent path protection for the vertices on the cycle and other vertices. The identification of a diameter cycle of a graph is useful in graph layout enhancement, failure independent path protection, ring-based graph protection and various path constructions.

Acknowledgment

The author would like to thank the anonymous reviewers for their helpful comments that greatly contributed to improving the quality of the paper. This work is supported by the Kuwait University Research Administration Grant EO 01/17.

References

[1] S. D. Nikolopoulos and L. Palios, "Detecting Holes and Antiholes in Graphs", *Algorithmica*, vol. 47, no. 2, pp. 119–138, 2007. http://dx.doi.org/10.1007/s00453-006-1225-y

[2] N. Chandrachoodan *et al.*, "Adaptive Negative Cycle Detection in Dynamic Graphs", in *Proc. of the 2001 IEEE International Symposium on Circuits and Systems*, 2001, pp. 163–166. http://dx.doi.org/10.1109/ISCAS.2001.922010

[3] M. H. Karaata, "A Stabilizing Algorithm for Nding Biconnected Components", *Journal of Parallel and Distributed Computing*, vol. 62, no. 5, pp. 982–999, 2002. http://dx.doi.org/10.1006/jpdc.2001.1833

[4] J. S. Deogun *et al.*, "An Algorithm for Boundary Discovery in Wireless Sensor Networks", High Performance Computing–HiPC 2005, Springer, 2005, pp. 343–352. http://dx.doi.org/10.1007/11602569_37

[5] I. Khan *et al.*, "An Overview of Holes in Wireless Sensor Networks", in *Proc. of the 11th Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, 2010.

[6] J. M. Six and I. G. Tollis, "Circular Drawings of Biconnected Graphs", In: Workshop on Algorithm Engineering and Experimentation, Springer, 1999, pp. 57–73. http://dx.doi.org/10.1007/3-540-48518-X_4

[7] M. Klammler *et al.*, "Aesthetic Discrimination of Graph Layouts", International Symposium on Graph Drawing and Network Visualization, Springer, 2018, pp. 169–184. http://dx.doi.org/10.1007/978-3-030-04414-5_12

[8] A. Arleo *et al.*, "A Distributed Multilevel Force-Directed Algorithm", *IEEE Transactions on Parallel and Distributed Systems*, 2018. http://dx.doi.org/10.1109/TPDS.2018.2869805

[9] W. He and A. K. Somani, "Path-Based Protection for Surviving Double-Link Failures in Mesh-Restorable Optical Networks", in *Proc. of the IEEE Global Telecommunications Conference*, 2003, pp. 2558–2563. http://dx.doi.org/10.1109/GLOCOM.2003.1258699

[10] R. Asthana *et al.*, "p-Cycles: An Overview", *IEEE communications surveys & tutorials*, vol. 12, no. 1, pp. 97–111, 2010. http://dx.doi.org/10.1109/SURV.2010.020110.00066

[11] G. D. Morley and W. D. Grover, "Current Approaches in the Design of Ring-Based Optical Networks", in *Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering*, 1999, pp. 220–225. http://dx.doi.org/10.1109/CCECE.1999.807199

[12] C. Mauz, "p-Cycle Protection in Wavelength Routed Networks", in *Proc. of the Seventh Working Conference on Optical Network Design and Modelling (ONDM'03)*, 2003.

[13] G. Shen and W. Grover, "Extending the p-Cycle Concept to Path Segment Protection for Span and Nnode Failure Recovery", *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 8, pp. 1306–1319, 2003. http://dx.doi.org/10.1109/JSAC.2003.816598

[14] B. [14] *et al.*, "Surviving Multiple Network Failures Using Shared Backup Path Protection", in *Proc. of the Eighth IEEE Symposium on Computers and Communications*, 2003, pp. 1333–1340. http://dx.doi.org/10.1109/ISCC.2003.1214298

[15] A. Kodian and W. Grover, "Failure-Independent Path-Protecting p-Cycles: Ecient and Simple Ful-

ly Preconnected Optical-Path Protection", *Journal of Lightwave Technology*, vol. 23, no. 10, pp. 3241–3259, 2005. http://dx.doi.org/10.1109/JLT.2005.855697

[16] X. Chen *et al.*, "Optimizing FIPP-p-Cycle Protection Design to Realize Availability-Aware Elastic Optical Networks", *IEEE Communications Letters*, vol. 22, no. 1, pp. 65–68, 2018. http://dx.doi.org/10.1109/LCOMM.2017.2763621

[17] H. Dao *et al.*, "An Ecient Network-Side Path Protection Scheme in OFDM-Based Elastic Optical Networks", *International Journal of Communication Systems*, vol. 31, no. 1, 2018. http://dx.doi.org/10.1002/dac.3410

[18] P. D. Choudhury *et al.*, "A Brief Review of Protection Based Routing and Spectrum Assignment in Elastic Optical Networks and a Novel p-Cycle Based Protection Approach for Multicast Trac Demands", *Optical Switching and Networking*, 2018. http://dx.doi.org/10.1016/j.osn.2018.12.001

[19] H. M. Singh and R. S. Yadav, "Ecient Algorithm for Removal of Loopbacks in p-Cycle-Based Survivable WDM Networks", *IET Communications*, vol. 12, no. 18, pp. 2366–2373, 2018. http://dx.doi.org/10.1049/iet-com.2018.5391

[20] R. Merris, "Graph Theory", Wiley Series in Discrete Mathematics and Optimization, Wiley. http://dx.doi.org/10.1002/9781118033043

[21] E. W. Weisstein, "Connected Graph", Accessed Aug. 29, 2021, MathWorld – A Wolfram Web Resource. [Online]. Available: https://mathworld.wolfram.com/ConnectedGraph.html

[22] R. Hadid *et al.*, "A Stabilizing Algorithm for Finding Two Node-Disjoint Paths in Arbitrary Networks", *International Journal of Foundations of Computer Science*, vol. 28, no. 4, pp. 411–435 2017. http://dx.doi.org/10.1142/S0129054117500253

[23] J. Suurballe, "Disjoint Paths in a Network", *Networks*, vol. 4, no. 2, pp. 125–145, 1974. http://dx.doi.org/10.1002/net.3230040204

Contact addresses:

Mehmet Hakan Karaata
Department of Computer Engineering
Kuwait University
Kuwait
e-mail: mehmet.karaata@ku.edu.kw

Mehmet Hakan Karaata received his PhD degree in Computer Science in 1995 from the University of Iowa. He joined Bilkent University, Ankara, Turkey as an Assistant Professor in 1995. He is currently working as a Professor in the Department of Computer Engineering, Kuwait University. His research interests include mobile computing, distributed computing, fault tolerant and autonomus computing.

Received: December 2020
Revised: July 2021
Accepted: July 2021