

Finding Hamilton Circuit in a Graph

Ghulam Mustafa Babar
EnterpriseDB Corporation,
Software Technology Park-1,
Islamabad, Pakistan

Dr. Sikandar Hayat Khiyal
Dept. of Computer Science
Intl. Islamic University,
Islamabad, Pakistan

Abdul Saeed
Dept. of Computer Science
Intl. Islamic University,
Islamabad, Pakistan

Abstract - The purpose of this paper is to develop an algorithm to determine the Hamilton Circuit in a given graph of degree three. This algorithm will find Hamilton Circuit in polynomial steps. We got some properties; these properties are combined to develop the above mentioned algorithm. These principles and their use will be explained by different examples. More or less these principles are simple and obvious.

Keywords: Hamilton Circuits, Degree of a node, Virtual Single Unit, Active End Point, Passive End Point.

1. Introduction:

A path $X_0, X_1, \dots, X_{n-1}, X_n$ in the graph $G = (V, E)$ is called a Hamilton Path if $V = \{X_0, X_1, \dots, X_{n-1}, X_n\}$ and $X_i \neq X_j$ for $0 \leq i < j \leq n$. A circuit $X_0, X_1, \dots, X_{n-1}, X_n$ (with $n > 1$) in a graph $G = (V, E)$ is called a Hamilton Circuit if $X_0, X_1, \dots, X_{n-1}, X_n$ is a Hamilton Path.

Is there a simple way to determine whether a graph has a Hamilton Circuit? At first, it might seem that there should be an easy way to determine this. Surprisingly, there are no known simple necessary and sufficient criteria for the existence of Hamilton Circuit. However, many theorems are known that give sufficient conditions for the existence of Hamilton Circuit. Also certain properties can be used to show that graph has no Hamilton Circuit [1].

2. Terminology:

Here are the special terms used in the following discussion.

1. Active and Passive End Points:

While finding the Hamilton Circuit, we extend the path on both directions from starting node. One end point is referred as Active Endpoint and other is referred as Passive Endpoint. On each move, we extend Active Endpoint. Passive Endpoint is only extended when some Virtual Single Unit (VSU) is connected with it or it fulfills some special condition described in some rule.

2. Virtual Single Unit (VSU):

During the process of finding Hamilton Circuit, multiple independent paths are created. Each independent path is referred as Virtual Single Unit. In other words, VSU is simply a path which has been decided for Hamilton Circuit. These VSUs are connected and finally form the Hamilton Circuit.

3. Assigned Edge:

Any edge which is assigned some VSU ID is referred as Assigned Edge.

4. Selected Edge:

Any edge which is selected for Main Path or some VSU but not assigned ID until now is referred as Selected Edge.

5. Unselected Edge:

Any edge which is undecided for any VSU/Main Path is referred as Unselected Edge.

3. Basic Properties for Hamilton Circuit:

Following are some of the basic properties for Hamilton Circuit

- 1) If a graph has any vertex of degree one then the graph cannot have Hamilton Circuit.
- 2) If a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton Circuit.
- 3) If there is an edge in the graph which is when removed, divide the graph into two disjoint sub-graphs then the original graph cannot have Hamilton Circuit.
- 4) If there is a vertex in the graph which is when removed (connected/incident edges will also be removed) divide the graph into two disjoint sub-graphs, then the original graph cannot have Hamilton Circuit. (This property is not necessary for degree-3 graphs but useful for higher degree graphs therefore we are not including checks for this property in the following algorithm).
- 5) If edges of degree two vertices create a cycle and that cycle does not contain all vertices of graph, the graph cannot have any Hamilton Circuit.
- 6) When a Hamilton Circuit is being constructed this circuit passes through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed [5].

4. Useful Theorems:

There are two famous theorems regarding the existence of Hamilton Circuit.

- 1) **Dirac's Theorem:** If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$ then G has a Hamilton Circuit.[2]
- 2) **Ore's Theorem:** If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for every pair of non-adjacent vertices u and v in G , then G has Hamilton Circuit. [3]

But still these don't give enough information to find out Hamilton Circuit or we can't produce an algorithm just on the basis of these algorithms.

5. Special Cases:

Now we consider some special cases and try to solve each case independently and finally combine their results to produce the algorithm. Right now we shall stick to the graphs up to degree three. The graphs having degree higher than three have more probability to contain Hamilton Circuit.

5.1. When each vertex in a graph has degree two:

As mentioned earlier in property-2, if a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton Circuit. Since in this case, all vertices have degree two therefore all edges must be in the path. In this case, nothing is to be done, Hamilton Circuit is already present and this can be considered the best case.

5.2. When each vertex in a graph has degree three:

Following are the rules to find Hamilton Circuit in a graph having degree three:

- (i) Select a starting node at random (since every vertex has same probability).
- (ii) Select any two incident edges to the vertex. In other words, start extending path on both directions from starting node.
- (iii) Remove the third edge.
- (iv) Since each vertex has degree three therefore the vertex on the opposite side of removed edge is left with two edges and according to the property-2, both edges must be in the path, therefore mark both edges as Virtual Single Unit that must be in the path/circuit.

- (v) In this algorithm, we will proceed on both sides from starting vertex, we name one endpoint (vertex) as ACTIVE ENDPOINT, and the other endpoint (vertex) as PASSIVE ENDPOINT. The ACTIVE ENDPOINT will take decision for further move at each step and PASSIVE ENDPOINT will only move/extended further when it gets some VIRTUAL SINGLE UNIT (V.S.U) incident to it or found an edge which can create a smaller cycle (property-5). In this case the other end of VIRTUAL SINGLE UNIT is marked as PASSIVE ENDPOINT after joining of original path with V.S.U.
- (vi) On every move on ACTIVE ENDPOINT, the last vertex in the extended path is marked as ACTIVE ENDPOINT and same with PASSIVE ENDPOINT.
- (vii) At each extension on either endpoint of main path or some V.S.U, the unused edges are removed and opposite ends (vertices) of removed edges are reconsidered for V.S.U creation, extension or joining with another V.S.U or ACTIVE/PASSIVE ENDPOINT'S. [This step has highest priority after every move/extension]
- (viii) On each extension, check the concerning V.S.U'S for the edge which can create smaller cycle, removed it & extend V.S.U. Cycle can only be created if endpoints of a V.S.U or ACTIVE/PASSIVE ENPOINTS share the same edge and we adopt that particular edge. Thus simply check whether endpoints of V.S.U or ACTIVE/ PASSIVE ENDPOINTS are sharing the same edge, if so, remove that edge to avoid smaller cycle. [This step is second highest priority item after step-(vii) on every move/extension]
- (ix) On joining two V.S.U'S, always check the unused edges, if pair of unused edges, on either node of connecting edge, is leading to same vertex, adopt the alternate path other than joining VSUs, as we cannot remove two edges from any vertex. (it leads to property-1 of section-3)
- (x) On each extension, check if the ACTIVE ENDPOINT reach at certain vertex where remaining two edges are connecting it with two (or endpoints of a single) VSU(s) then the move is wrong. Adopt the alternate path right. If we don't have any choice or the other path isn't possible or result in wrong move then the graph doesn't have Hamilton Circuit.
- (xi) Search an edge (unselected i.e. not part of any VSU) in the graph which is when removed, divide the graph into two disjoint sub-graphs. If found such edge then the original graph does not have Hamilton Circuit.
- (xii) For extension on ACTIVE ENDPOINT, select either of two edges incident to ACTIVE ENDPOINT for next move.
- (xiii) Whenever two VSUs are combined, both result one larger VSU, containing all the vertices of both the smaller VSUs. If one of the VSUs is main path, move the ACTIVE/PASSIVE ENDPOINTS to the new endpoint of main path.
- (xiv) Both endpoints, ACTIVE & PASSIVE, can only be joined when all vertices are traversed otherwise, whenever both endpoints share the same edge that will be removed (property-5) considering it unused and extend both endpoints according to property-2.

6. The Algorithm:

Here is the actual algorithm to find out the Hamilton Circuit in a degree three graph.

Procedure FindHC ()

Begin

If any node has degree < 2, FAILED

If FindCycle () fails for any edge of graph

FAILED (No Hamilton Circuit)

End

ReturnValue: = 0

While ReturnValue is zero

ReturnValue: = MoveNext ()

End

If ReturnValue = -1

```

    FAILED
Else
    SUCCESS
End
End

```

Function FindCycle (e: edge)

There should not be any edge which is when removed, divide the graph into two disjoint sub-graphs. Any such edge cannot be part of any cycle. So this function tries to find closest/smallest cycle containing the given edge. Function returns TRUE if found some cycle else returns FALSE.

Begin

```

Mark one endpoint (node) as RED node & Add to RED list
Mark other endpoint (node) as BLUE node & Add to BLUE list
Remove the given edge
LOOP
    Get each node one by one from RED list & Get all connected nodes.
    Add all connected node to temporary list
    Remove all edges incident to nodes in RED list & Empty the RED list
    Copy all nodes from temporary list to RED list
    Remove all edges connecting one node in RED list to any other RED node

    If RED list is empty Return FALSE    (loop is broken here)

    If any node in RED list is connected to BLUE list by some edge in graph
        Return TRUE    (loop is broken here)
    End

    Get each node one by one from BLUE list & Get all connected nodes to them. Add all
    connected nodes to temporary list.
    Remove all edges incident to nodes in BLUE list.
    Empty the BLUE list.
    Copy all nodes from temporary list to BLUE list

    Remove all edges connecting one node in BLUE list to any other BLUE node

    If RED list is empty Return FAILURE    (loop is broken here)

    If any node in RED list is connected to BLUE list by some edge in graph
        Return SUCCESS    (loop is broken here)
    End

```

End

End

Function MoveNext ()

Begin

```

If any node has degree < 2 Return -1
If any node has degree = 2 Call MarkVSU () for that node
If Main Path/VSU exist, Active Endpoint:= First node of (first) path
Else Active Endpoint := First node of Node List
If no incident edge selected
    Mark any two edges as selected
    Call PruneNode (ActiveEP)
    Call PrunePath ()
    If FindCycle() fails for any unselected edge
        Return -1

```

End

```

Else if one edge is selected
  Select any of unselected edge as selected edge
  If selected edge is connecting two VSUs
    Get unselected edges on either node of connecting edge
    If both unselected edges have same other end node
      Mark selected edge as unselected & select the alternate edge on ActiveEP
    End
  Call PruneNode (ActiveEP) and then PrunePath ()
  If VSULinking() fails for any VSU
    Mark selected edge as unselected & select the alternate edge on ActiveEP
    Call PruneNode (ActiveEP) and then PrunePath ()
    If VSULinking() fails for any VSU
      Return -1
    Else
      Return 0
    End
  Else
    Return 0
  End
End
Else If both incident edges are assigned different VSU ID
  Call PruneNode (ActiveEP)
  If Main VSU covers all nodes
    Return 1 ( SUCCESS )
  End
End
End
End

```

Function VSULinking (id: VSUID)

Since in each step, we prune the unused edges, this may create two subgraphs connected by less than two edges, which is failure case for Hamilton circuit.

```

Begin
  Add Endpoints of main VSU to RED list & Endpoints of provided (argument) VSU to BLUE list.
  Remove all edges of both VSU's to simplify calculations
  Counter:=0
  Loop
    Extend RED side to adjacent nodes & remove old edges from RED list
    If any common edge found between two sides, increment 'Counter'
    If Counter = 2
      Return SUCCESS
    Else if any side is left with only one node
      Return FAILED
    End
    Extend BLUE side to adjacent nodes & remove old edges from BLUE list
    If any common edge found between two sides, increment 'Counter'
    If Counter = 2
      Return SUCCESS
    Else if any side is left with only one node
      Return FAILED
    End
  End
End
End

```

Function PruneNode (n: node)

```

Begin
  Remove the unselected edge at 'n' & Call MarkVSUs () on other end node of removed edge
  If both incident edges are assigned different VSU ID Call MarkVSU (n)
  Else If both incident edges are not assigned VSU ID Call MarkVSU(n)

```

```

    Else If one edge is assigned and other is unassigned.
        Copy VSU ID from assigned to unassigned edge
        Call MarkVSU() on other end node of previously unassigned edge
    End
End

```

Function PrunePath ()

```

Begin
    If endpoints of any VSU are sharing some edge
        Remove edge & Call MarkVSU() on either nodes of removed edge
    End
End

```

Function MarkVSU (n: node)

```

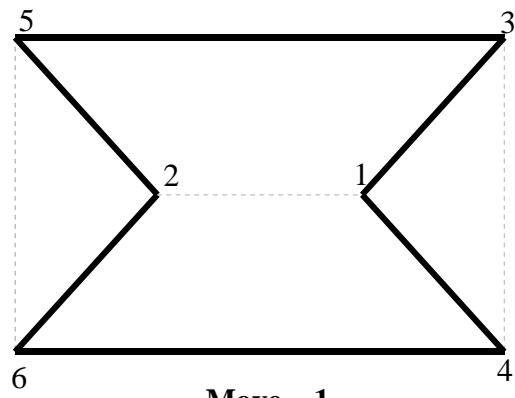
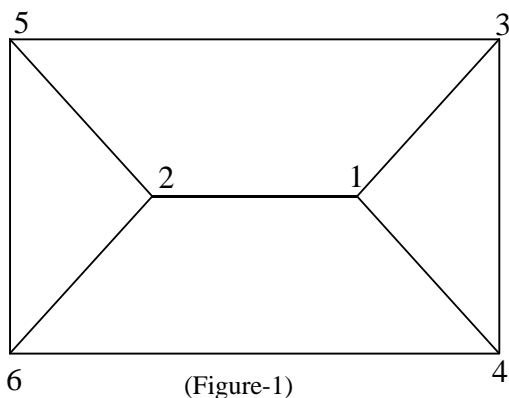
Begin
    If Degree of node = 2
        If any edge is not marked as selected, mark it as selected
        If Different VSU ID is set on both edges, Merge VSUs
        Else If VSU ID is set on one edge
            Copy VSU ID from assigned to unassigned edge
            Call MarkVSU () on otherend node of previously unassigned edge
        Else If both edges are unassigned
            If other end of first selected edge (Say m) is endpoint of some VSU
                Copy VSU ID from assigned to unassigned edge on 'm'. Call PruneNode(m).
            End

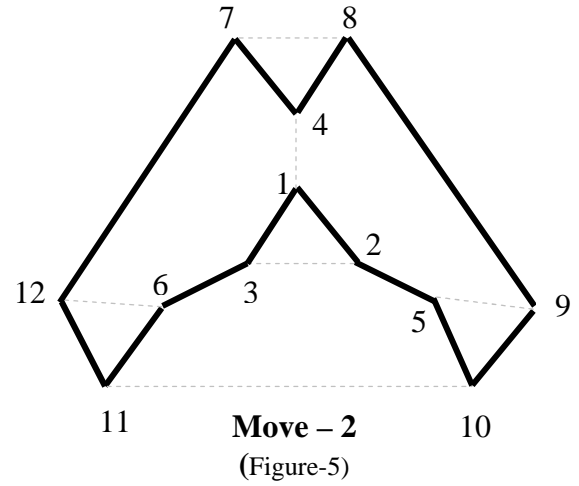
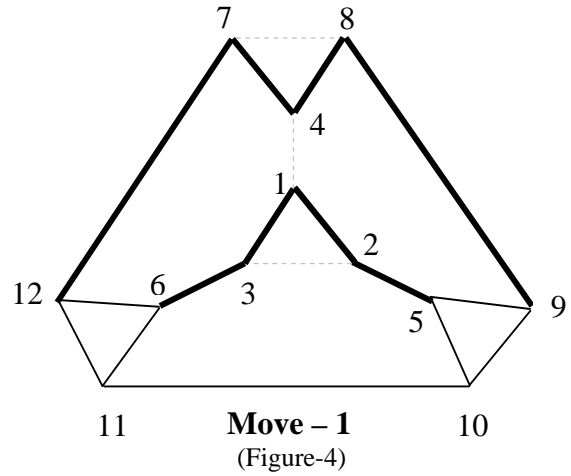
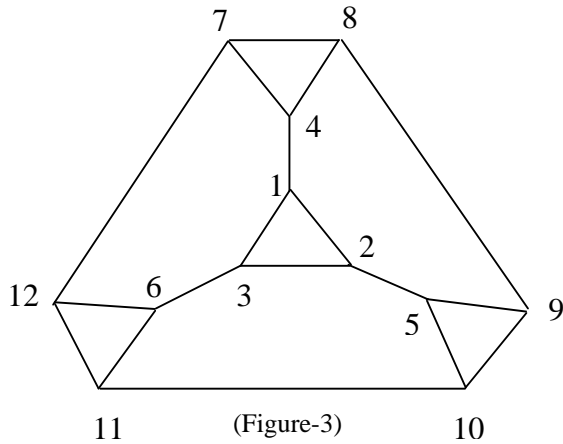
            If other end of second selected edge (Say k) is endpoint of some VSU
                Copy VSU ID from assigned to unassigned edge on 'k'. Call PruneNode(k)
            End

            If both selected edges on given node are assigned different ID,
                Merge VSUs.
            Else If one incident edge is assigned
                Copy VSU ID from assigned to unassigned
            Else
                Set new ID on either edge and add Path to VSUList
            End
        End
    End
    If Degree of node > 2 and any two edges are marked selected
        Call PruneNode(NodeNum)
    End
End

```

7. Experiments:





8. Conclusion:

We have tried graphs up to 30 nodes, having degree 3 and successfully solved all these graphs using the above mentioned algorithm. Graphs that do not have Hamilton Circuit are mostly rejected in six basic properties given in section 2. There is no back tracking involved in this algorithm therefore Hamilton Circuit in a graph of degree three can be found in polynomial steps.

9. References:

- [1] Kenneth H. Rosen, "Discrete Mathematics and Its Application", 4th Edition, p. 481-483
- [2] Eric W. Weisstein. "Dirac's Theorem." From *MathWorld*--A Wolfram Web Resource. <http://mathworld.wolfram.com/DiracsTheorem.html>
- [3] Ore, O. "A Note on Hamilton Circuits." *Amer. Math. Monthly* **67**, 55, 1960
- [4] Rubin, F. "A Search Procedure for Hamilton Paths and Circuits." *J. ACM* **21**, 576-580, 1974
- [5] Ralph P. Grimaldi, "Discrete and Combinatorial Mathematics", 4th Edition, p.523-528