Restricted Backtracked Algorithm for Hamiltonian Circuit in Undirected Graph

Vinay Kumar

Abstract - While determining whether a graph is Hamiltonian, it is enough to show existence of a Hamiltonian cycle in it. An algorithm based on restricted backtracking is presented in the paper that uses tie breaking rules to reduce the possible number of backtrackings. If x is any intermediate node in HC then once its neighbour y has been visited from x, x is no longer required so drop it and process is continued on the remaining subgraph. Each node is visited exactly once in a HC except the start node. Adjacency matrix is used to encode the graph. Prevention of backtracking is achieved up to next node from start node. From third node onward, wherever it is not possible to break tie uniquely, a provision for backtracking is kept only for tied nodes. Time complexity of algorithm is $O(n^4)*B(n)$ in the worst case where B(n) is a factor due to possible backtracking. It returns $O(n^2)$ in the best case and $O(n^3)*B(n)$ on the average.

Index Terms - articulation point; complexity class; P; NP; Hamiltonian graph; connected graph; line sweeping; restricted backtracking

1. INTRODUCTION

The Icosian game [4], introduced by Sir William Hamilton is known as Hamiltonian Circuit (HC) problem [7]. The objective of the game is to visit all nodes of the graph exactly once before returning to the initial node. In graph theoretic world, a Hamiltonian circuit is defined as a simple cycle that contains every vertex of graph exactly once except the first one which is visited again at the end to complete the cycle [8]. A graph is said to be Hamiltonian if it contains a HC else it is nonhamiltonian. Although many graphs can be trivially determined as Hamiltonian or nonhamiltonian even then the problem is very complex in general. The problem of finding a Hamiltonian cycle in an undirected graph is studied for over a hundred years [36]. The problem "Does a graph G have a Hamiltonian cycle?" can be defined in formal language as

HAM_CYCLE = {<G>: G is a Hamiltonian graph}

Showing existence of one Hamiltonian cycle in G is sufficient to conclude that the graph is Hamiltonian. However, it is expected to test all possible n! permutations of vertices before concluding that G is nonhamiltonian. Basic properties of graph [5, 16, 38] are used in the introduced algorithm to restrict backtracking to the maximum possible extent and to avoid it if it can be. It is, therefore, not always required to explore all possible n! arrangements of vertices before concluding that a graph is nonhamiltonian. The following facts are taken into consideration while developing the algorithm.

1. One edge is sufficient to cross over from one node to its adjacent node [11]
2. Once a node is visited, it is no longer required (except the initial node), so drop it [1].
3. A node **y** to visit from **x** can be selected using some tie breaking rules in such a way that possibility of backtracking to explore other possible path from **x**. is drastically reduced [18, 19].
4. At any stage, if dropping of node **x** yields more that one dangle node [38], and if it is not avoidable (backtracking not possible), the graph can be concluded as nonhamiltonian.
5. If at the end only initial node is left then graph is Hamiltonian otherwise it is nonhamiltonian [22].

The core of the algorithm development process lies in the point 3 above. The detailed steps are outlined in section 2 of this paper. Section 3 contains proof for the correctness of the algorithm followed by two illustrative examples in section 4. Section 5 deals with computational analysis of the algorithm. Before stepping into section 2 let us see a basic concept that if graph contains an articulation point then graph is nonhamiltonian [24].

Let G = (V, E) be a connected undirected simple graph with $|V| = n \geq 3$, and $|E| = m$ where $m \geq n$. A graph is simple if it contains neither loop nor multi edge [3, 26]. A graph is connected if there is a path between every pair of nodes in it [3]. To maintain flow of presentation, few terms like node and vertex, edge and arc are used synonymously. In this paper, a graph implies a simple connected graph with no articulation point, unless otherwise stated. DFS (depth first traversal) algorithm is used to test connectivity and non-existence of articulation point in graph. DFS algorithm executes in polynomial time [2, 6, 27]. An articulation point in a graph G is a node **x** that, when removed from G, partitions set V into two (or more) non empty subsets U and W such that

- U and W are disjoined, and
- No node in U is adjacent to any node in W [28].

Theorem 1: A graph G = (V, E) with an articulation point has no Hamiltonian Circuit.

Proof: Let v be the articulation point and U and W be the two non empty subsets of V such that

- $V = U \cup W \cup \{v\}$,
- $v \notin U$, $v \notin W$ and
- $U \cap W \neq \phi$

Let us proof it by contradiction. Suppose G has a HC. Three possibilities about starting node x of HC in G are (a) x = v or (b) $x \in U$ or (c) $x \in W$.

Case (a) when x =v

Scientist 'D', NIC, Block A, C.G.O Complex, Lodhi Road New Delhi 110 003, India
E-mail: vinay.kumar@nic.in and vinay5861@gmail.com

Since G is connected, v has adjacent nodes in both U and W. Once a node in U is visited from v, there is no way to come to any node in W without visiting v. Similar case is faced when a node in W is visited first. Therefore there is no HC in G [10].

Case (b) when $x \in U$

Starting from x visit all nodes in U first, in the best case. Then visit v then a node in W. Once in W, there is no way to return to x because v is removed. Therefore there is no HC in G

Case (c) when $x \in W$

It can be proved in the same way as in the case (b). ♦

Corollary 1: A graph G containing a node of degree ≤ 1 is nonhamiltonian.

Proof: Any node y adjacent to the node x of degree one is an articulation point in G. A node of degree zero is unreachable. ♦

Converse of the theorem that "a graph having no articulation point is Hamiltonian" is not true. Many graphs can be presented in the support [13, 21]. However this theorem helps in early conclusion on the nonhamiltonian graph. Presence of an articulation point indicates that as and when it is dropped from the graph while traversing to find HC, it ensures that at least two nodes are left in the current subgraph when algorithm terminates its execution.

2. ALGORITHM

The step by step algorithm determines existence of one cycle out of possible n! to conclude that G is Hamiltonian. Current node **x**, other than initial node, is dropped when its neighbour **y** is visited. While dropping **x** it is ensured that no backtracking to the node **x**, in due course can yield otherwise result. It is achieved by applying tie breaking rule whenever ∃ more than one options from **x**. If it is not possible to break a tie, the possible options available at that point is stored in array BACKTRACK []. The array BACKTRACK [] is indexed on the nodes as visited in the graph. List of currently visited nodes is denoted by π. And nodes are referred as $v_1, v_2, v_3, \ldots, v_n$ in the sequence they are visited. Before applying the algorithm, line-sweeping [37] algorithm is executed on the graph to merge all nodes in one linear component because all nodes in a line are visited one after other in a sequence as per this algorithm. For example if nodes from j to k are merged (visited) in sequence then merged (visited) nodes are referred as $<v_j, v_{j+1}, \ldots, v_{j+k}>$. List of articulation points is updated in the current subgraph when a node is dropped from graph. The list of current articulation point is referred as ARTPNT.

When a node v_2 or later visited node v_k is dropped from current subgraph, start node v_1 may become dangle. While counting number of dangle nodes at any stage in the algorithm, only intermediate nodes are taken into account but not node v_1.

Let G = (V, E) be a simple graph with |V| = n, |E| = m, m ≥ n. Initialize adjacency matrix M[n, n] as per adjacency in G. The degree spectrum [8, 9, 12] of G is stored in one dimensional array Degree[n].

Step 1: Select a node $\mathbf{v_1}$ from G such that $\mathbf{v_1}$ is of minimum degree. Resolve a tie by taking node from earliest row (or column) of matrix M. For example if nodes in rows 5 and 10 have same minimum degree then select node from row 5. Initialize path π to $\mathbf{v_1}$ and Start_node to $\mathbf{v_1}$.

Start_node ← $\mathbf{v_1}$; π: $\mathbf{v_1}$

Step 2: Find a node to be visited next from start node.

Step 2.1 Create a set of all nodes adjacent to $\mathbf{v_1}$ and call it NGBR – set of neighbours of Start_node.

Step 2.2 Select a node $\mathbf{v_2}$ from NGBR to visit next in the following way. Resolve any tie as in Step 1.

Step 2.2.1 Pick up a node of degree two. If such node is found then go to step 2.3 else continue to next step 2.2.2

Step 2.2.2 Find a node that does not yield any dangle node when dropped from graph G. If such node is found then go to step 2.3 else continue to next step 2.2.3.

Step 2.2.3 Find a node that yields only one dangle node when dropped from G. If such node is found then go to step 2.3 else skip to step 5.

Step 2.3 Initialize Current_node to $\mathbf{v_2}$ and extend path π up to $\mathbf{v_2}$.

Current_node ← $\mathbf{v_2}$; π: $\mathbf{v_1}$ $\mathbf{v_2}$

Update NGBR = NGBR – {$\mathbf{v_2}$}

Update set ARTPNT treating Current_node as dropped.

Step 3: Select a node $\mathbf{v_{j+1}}$ from adjacent nodes of Current_node $\mathbf{v_j}$ to visit next in the following way. Resolve a tie by ignoring the node that is in ARTPNT. Even then if there is tie then resolve as in Step 1, keep list of other candidate nodes at BACKTRACK [$\mathbf{v_j}$] and set flag BACKTRAK_possible as true.

Step 3.1 If number of adjacent node is one then return the node and go to step 3.6 else remove the Start_node from list of adjacent node, if it is in the list, and continue to step 3.2.

Step 3.2 If there are more than one adjacent node of degree two then go to step 5 else pick up the node of degree two. If such node is found then go to step 3.6 else continue to step 3.3.

Step 3.3 Find a node that is neither in NGBR nor equal to Start_node and that does not yield any dangle node when dropped from graph G. In case of tie resolve it. If such node is found then go to step 3.6 else continue to next step 3.4.

Step 3.4 Find a node that is neither in NGBR nor equal to Start_node and that yields only one dangle node

when dropped from G. In case of tie resolve it. If such node is found then go to step 3.6 else continue to step 3.5.

Step 3.5 Find a node from nodes not considered in step 3.3, 3.4 as below:

Step 3.5.1 Find a node that does not yield any dangle node when dropped from G. If such node is found then go to step 3.6 else go to step 3.5.2

Step 3.5.2 Select a node that yields one dangle node when dropped. If such node is found then go to step 3.6 else continue to step 5.

Step 3.6 Initialize
Prev_Cuurent_node←Current_node
Current_node← $\mathbf{v_{j+1}}$
Extend path π up to $\mathbf{v_{j+1}}$.
Drop Prev_Cuurent_node from graph and update the degree of all affected nodes accordingly in G
Update set ARTPNT for the current subgraph treating current node as dropped
Update NGBR, if required.

Step 4: Repeat Step 3 as long as visit to a neighbor is possible else go to step 5.

Step 5: **If** only Start_node is left at this stage, after successive removal of intermediate nodes, then G is Hamiltonian

Else If back track is possible (i.e. BACKTRAK_possible is true) **then**
Restore the matrix by adding nodes one by one from last visited node in $\boldsymbol{\pi}$ up to last index node $\mathbf{v_k}$ in array BACKTRACK. Then pick up first node from list of options available in BACKTRACK [$\mathbf{v_k}$] and initialize
Current_node← $\mathbf{v_k}$
Update set ARTPNT, NGBR and BACKTRAK_possible flag as applicable for the latest subgraph and Repeat Step 3 as long as visit to a neighbour is possible.

Else Graph G is nonhamiltonian.

The algorithm in steps 1 through 5 ensures two things: (1) it restricts backtracking by dropping the visited intermediate nodes, and (2) while dropping a node it ensures that no other path from that node shall yield different result in most of the circumstances by using tie breaking rules. While iterating in step 3, only remaining sub graph is taken. Algorithm terminates when no more visit is possible i.e. even backtracking is not feasible. A visit is not possible if there is no adjacent node to Current_node and BACKTRAK_possible falg is false. This case arises when there is only one node (i.e. Start_node) is left at the end or graph is detected as nonhamiltonian at an early stage. Two illustrations of the algorithm are given in the following section that deals with the situation (1) when no backtracking is required and (2) when it is really required.

3. ILLUSTRATIVE EXAMPLES

A primitive idea about working of the algorithm is shown using a visually very simple graph in figure1. This is the case when no backtracking is required. Represent the graph as adjacency matrix [15, 20]. Start from a node of minimum degree. All nodes in this graph are of equal degree 3. Without loss of generality, let A be in the earliest row (column) and select node A to start with. Here,

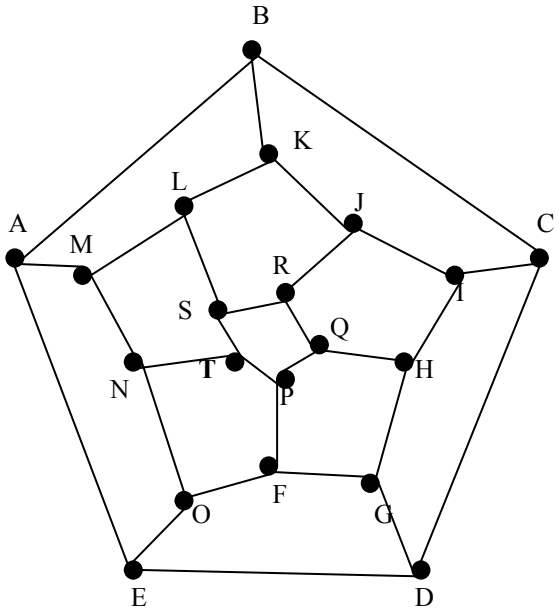

Figure 1

Start_node ← **A;**
π: **A**

A has three adjacent nodes B, E and M and all are of degree 3. None of them yields any dangle node when dropped from G, thus using step 2.2.2, we may select node B using tie breaking rule to proceed further. Here,

Current_node ← **B;**
π: **A B**
NGBR = {B, E, M} – {B} = {E, M}
ARTPNT = {}

Now node B has two adjacent nodes C and K. Using step 3.3 select node C to proceed and following updating is done.

Prev_Cuurent_node← **B**
Current_node← **C**
π: **A B C**.

Drop node **B** from graph and update the degree of all affected nodes accordingly in G. The set ARTPNT = {} for the current subgraph. There is no need to update NGBR. The step by step execution of algorithm is outlined in the table 1 below.

Algorithm steps	Node Selected	Current Path π:	Articulation Set ARTPNT	BACKTRACK [iteration]
1	A	A		
2.2.2	B	A B	{}	
3.3	C	AB C	{}	{K}
3.3	D	ABC D	{}	{I}
3.3*	G	ABCD G	{}	
3.3	F	ABCDG F		
3.3	P	ABCDGF P		
3.3	T	ABCDGFP T	{L}	
3.3	S	ABCDGFPT S	{L K, J}	
3.4*	R	ABCDGFPTS R	{L, K, J}	
3.2	Q	ABCDGFPTSR Q	{L, K, J}	
3.1	…	ABCDGFPTSR QHIJKL	{}	
3.1	M	ABCDGFPTSRQHIJKL M		
3.2	N	ABCDGFPTSRQHIJKLM N	{E}	
3.1	…A	ABCDGFPTSRQHIJKLMN OEA		

Table 1: Indicates that a tie was resolved between nodes G and E using NGBR

.At the end only one node A is left in the subgraph and hence G is Hamiltonian. Here numbering of node has no effect on the requirement of backtracking as long as tie is resolved as per the algorithm. The graph in figure 2 is Hamiltonian. Backtracking may be required in one case. Node A in this graph is of minimum degree 2. Select node A to start with. Here,

Start_node ← **A;**

π: **A**

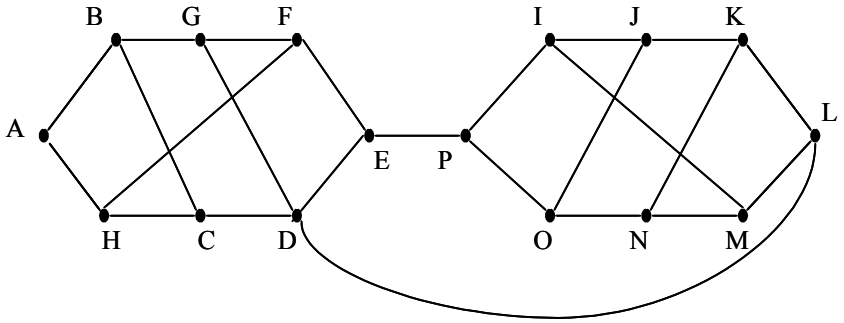

Figure 2

Node A has two adjacent nodes B and H and both are of degree 3. None of them yields any dangle node when dropped from G, thus using step 2.2.2, we may select node B using tie breaking rule to proceed further. Here,

Current_node ← **B;**

π: **A B**

NGBR = {B, H} – {B} = {H}

ARTPNT = {H}

Now node B has two adjacent nodes C and G. Using step 3.3, node C is selected to proceed further with following updating.

Prev_Cuurent_node← **B**

Current_node← **C**

π: **A B C**.

ARTPNT = {H, F}

Drop node **B** from graph and update the degree of all affected nodes accordingly in G. The set ARTPNT = {} for the current subgraph. There is no need to update NGBR. The step by step execution of algorithm is outlined in the table 2 below.

Algorithm steps	Node Selected	Current Path π:	Articulation Set ARTPNT	BACKTRACK [iteration]
1	A	A		
2.2.2	B	A B	{H}	
3.3	C	AB C	{H, F}	{G}
3.3	D	ABC D	{H, F, E, P}	
3.2	G	ABCD G	{H, F, E, P}	

Algorithm steps	Node Selected	Current Path π:	Articulation Set ARTPNT	BACKTRACK [iteration]
3.1	F*	ABCDG F	*	*
℘5	G	AB G	{H}	
3.3	F	ABG F	{H, C, D}	
3.3	E	ABGF E	{H, C, D, L}	
3.3	P	ABGFE P	{H, C, D, L}	
3.3	I	ABGFEP I	{H, C, D, L}	{O}
3.3	J	ABGFEPI J	{H, C, D, L}	
3.4	K	ABGFEPIJ K	{H, C, D, L, M, N}	{M}
3.2	O	ABGFEPIJ K	{H, C, D, L, M, N}	
3.1	…A	ABGGFEPIJK ONMLDCHA		

Table 2

* Dropping of the node F yields two dangle nodes E and H (other than start node).

℘ By step 5 backtracking is initiated up to node C and replacing C by G (available option at that level).

Examples demonstrate the working of the algorithm. At the end only one node A is left in the subgraph and hence G is Hamiltonian. The following graph in figure 3 is a nonhamiltonian. To show this there is no need to explore possibly all 9! permutations of 9 nodes. Just two runs are enough to say that the graph is nonhamiltonian.

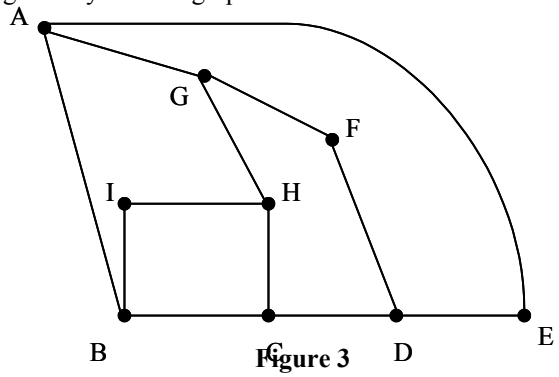

Figure 3

Algorithm correctness is proved in the following section. Related theorems, lemma, propositions and definitions are described as and when required. Obvious results are taken as axioms without any proof.

4. PROOF OF CORRECTNESS

An adjacent node y is visited from x in such a way that a cycle of length less than |V| does not form in the graph. The algorithm takes a biconnected graph (connected graph without articulation point) [14, 17] G = (V, E) as input (precondition) and outputs (post condition) a Hamiltonian (or nonhamiltonian path) π and a subgraph H of G with following properties:

If G is Hamiltonian

then $H = (V_H, E_H)$ with $|V_H| = 1$ and $|E_H| = 0$

Else $H = (V_H, E_H)$ with $|V_H| \geq 2$ and $|E_H| \geq 0$

Here V_H is set of nodes in H and E_H is set of edges in H. The proof of correctness has two parts:

(i) Partial correctness: If the algorithm will terminate then it will give the right result i.e. the result will satisfy the post condition.

(ii) Termination: Proof that the algorithm terminates [24].

To prove the correctness of the algorithm, it is required to prove the following postulates:

(a) Algorithm always finds a ***correct*** start node,

(b) It always finds a node adjacent to start node in correct way, if available, to initiate the process of finding HC in G,

(c) In every iteration, next node from the current node is found, if a visit is possible otherwise program terminates,

(d) Tie breaking rules restrict (in fact reduces number of possible) backtracking i.e. if G is found to be nonhamiltonian at k^{th} node, then backtracking to any of the previously ignored node does not yield any otherwise result.

(e) If only Start_node is left at the end then graph is hamiltonian else it is nonhamiltonian, and

(f) Finally algorithm terminates.

In general, if G is Hamiltonian then a HC may start from any node [35] and if G is nonhamiltonian then a cycle cannot be completed starting from any nodes in G. There is no loss of generality in selecting a start node based on some criteria. Thus the proposition,

"In a Hamiltonian graph a HC begins from a node x of minimum degree"

is true. And step 1 of algorithm selects a node of minimum degree from G to start with. Further, a start node has at least two adjacent nodes.

Lemma 1: $\forall x \deg(x) \geq 2$, where x is a node in the input graph G.

Proof: Let y be any node in G. The graph G is biconnected so there are at least two node disjoint paths between x and y. It implies that ∃ distinct nodes u, v adjacent to x such that one path from x to y goes through u and another through v.

$\therefore \deg(x) \geq 2$. ♦

Corollary: The start node $\mathbf{v_1}$ has $m \geq 2$ adjacent nodes.

Let L(x) denote the set of all adjacent nodes of x then $L(\mathbf{v_1})$ = {y | y is adjacent to $\mathbf{v_1}$}. We refer $L(\mathbf{v_1})$ as NGBR. Among m (≥ 2) adjacent nodes to start node $\mathbf{v_1}$, the different possibilities are:

(i) all are of degree two, or
(ii) some are of degree two and other are of degree > 2, or
(iii) all are of degree > 2.

One node from NGBR is taken to leave the start node and one other will be required to complete HC if G is Hamiltonian. In the case of (i) and (ii), it is STEP 2.2.1 that picks up a node $\mathbf{v_2}$ of degree 2 to start with. However in case of (iii), the algorithm looks one step further to make sure that once the node (to be selected) is removed from graph, it yields not more than one dangle node (excluding start node). The algorithm prefers in step 2.2.2 over 2.2.3, to select a node that does not yield any dangle node. Thus in order of precedence of steps 2.2.1, 2.2.2 and 2.2.3 (from left to right) the algorithm finds a node next to start node yielding the post condition as below:

π : $\mathbf{v_1v_2}$
H = G
NGBR = $L(\mathbf{v_1})$ – $\{\mathbf{v_2}\}$
ARTPNT = As determined.

It is obvious that algorithm finds a node next to start node. It resolves a tie as per the criteria described in the algorithm as and when it arises. Backtracking is restricted at this stage. It is proved in the lemma 2 that no backtracking is required at this level. Correctness of this step is implied from the lemma 2 and theorem2.

Lemma 2: If G is Hamiltonian then $\forall \mathbf{x} \in L(\mathbf{v_1})$ pre Hamiltonian $\mathbf{v_1x}$ leads to a Hamiltonian cycle. ***Proof:*** Let us prove it using mathematical induction on the degree m of start node $\mathbf{v_1}$. It is to be noted that start node is of minimum degree in G.

Basis Step: For m = 2, the result is obviously true.

Inductive Step: Let the result be true for m = k i.e. $\forall \mathbf{x} \in L(\mathbf{v_1})$ pre Hamiltonian $\mathbf{v_1x}$ leads to a Hamiltonian cycle. Let one of the Hamiltonian cycle be

$$\mathbf{v_1}x\ldots z\ldots y\ \mathbf{v_1}$$

where $\mathbf{x, y} \in L(\mathbf{v_1})$. See the figure 4(a) for conceptual visualization of Hamiltonian cycle from graph shown in figure 5(a). It is in no way to trivialize the general proof of the lemma.

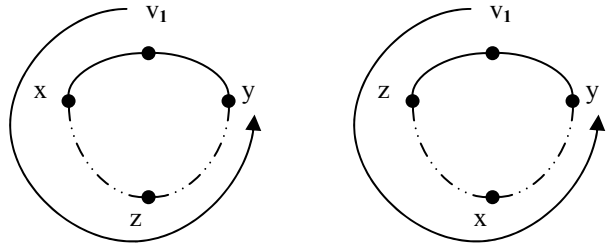

Figure 4

Let us add an edge from $\mathbf{v_1}$ to a node z in G in such a way that degree ($\mathbf{v_1}$) remains minimum in G and $\mathbf{v_1}$ remains the start node. Also $\forall x \in L(\mathbf{v_1})$ degree(x) > k otherwise $\mathbf{v_1}$ can no longer remain minimum degree node if an edge is added from $\mathbf{v_1}$ to any other node $z \notin L(\mathbf{v_1})$ in G. Further, addition of an edge in graph may make a nonhamiltonian graph Hamiltonian but not the reverse.

Let us now remove the edge between $\mathbf{v_1}$ and x. Even then $\mathbf{v_1}$ remains one of the minimum degree node and hence the start node. For conceptual visualization see figure 5(b).

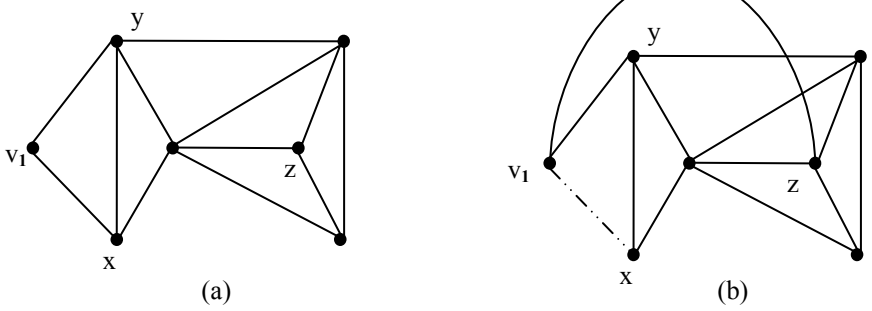

Figure 5

Now again m = k and hence the result is true from the assumption. See figure 4(b) for Hamiltonian cycle from graph 5(b). Presence of edge ($\mathbf{v_1}$, x) does not alter the result but only increases the number of possible Hamiltonian cycles in G. ♦

Theorem 2: If G is found to be nonhamiltonian at k^{th} node, then backtracking to any node x in NGBR = $L(\mathbf{v_1})$ – $\{\mathbf{v_2}\}$ does not yield any otherwise result.

Proof: Let $\deg(\mathbf{v_1})$ = m (≥ 2). When m = 2, $\mathbf{v_1}$ has two adjacent nodes i.e. $|L(\mathbf{v_1})|$ = 2 and any of the three cases outlined above may be applicable.

When m > 2, only case (iii) is applicable. Algorithmic step 2.2.2 or 2.2.3 finds a node $\mathbf{v_2}$ because step 2.2.1 is not relevant.

Case 1: When m = 2. Let the two adjacent nodes be **x** and **y** and rest of the graph be **H**. If both x and y are of degree two then any one can be used to leave the start node and other is

used to arrive at. No backtracking to y (in case x is selected) or to x (if y is selected) can yield otherwise result.

Suppose, without loss of generality, that deg(x) = 2 and deg(y) > 2. Instead of selecting x, refer figure 6, let node y be selected to start with and at the k^{th} stage it is found that x is an adjacent node of $\mathbf{v_k}$ then it leaves no alternative but to backtrack to the earliest available option from $\mathbf{v_{k-1}}$ otherwise the visit to x shall form a cycle of length < n. On the other hand if x is selected then y can always be ignored as it is in NGBR and alternative node to move ahead is available. Dotted lines in the figure 4 indicate the adjacency to y from k^{th} node (current subgraph H).

When deg(x) > 2 and deg(y) > 2 and both yield no dangle node when dropped then any one can be selected to leave the start node and other to arrive at. Same is true when both yield single dangle node when dropped from G. When one yields no dangle and other yields one dangle node then the first is selected to keep wider option available at the next step and hence reducing the number of possible backtracking later on to nodes $\mathbf{v_3}$ or any node visited thereafter. It is in no way contradictory to previous one when a node with degree 2 is preferred over other one.

Case 2: when m > 2. Because $\mathbf{v_1}$ is of minimum degree therefore $\forall x \in L(\mathbf{v_1})$, deg(x) > 2 and algorithmic step 2.2.2 is applicable to select a node $\mathbf{v_2}$. Obviously at this stage no node x can yield any dangle node. Tie is broken as per the coding of adjacency matrix M for G. Correctness follows from lemma 2.♦

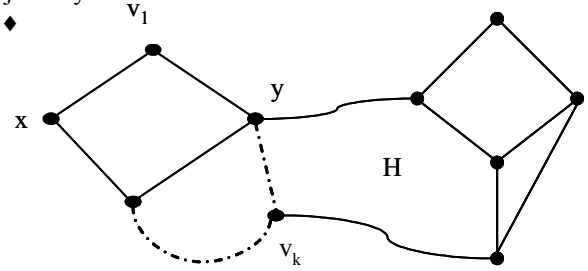

Figure 6

After proving the correctness of step 2, it is turn to show the correctness of step 3 and 4 of algorithm. Let $\mathbf{v_k}$ be the current node visited at the K^{th} iteration. It is essential to establish that in a Hamiltonian graph ∃ no valid current node $\mathbf{v_k}$ such that it is adjacent to more than one node of degree 2, and, backtracking from it is not possible. This excludes the start node. A current node is taken in context of the present subgraph H of G after successive removal of the visited nodes. A current node is said to be valid if and only if it either leads to a Hamiltonian cycle (possibly with backtracking) or helps in concluding that graph is nonhamiltonian at that stage itself. The hypothesis is proved in Lemma 3 and the hypothesis that a valid current node has at least one adjacent node in a Hamiltonian graph is proved in Lemma 4.

Lemma 3: In a Hamiltonian graph a valid current node cannot be adjacent to n > 1 nodes of degree 2, excluding start node.

Proof: Let the current node be $\mathbf{v_k}$ and y and z be two adjacent nodes of $\mathbf{v_k}$ such that both are of degree 2 none is equal to $\mathbf{v_1}$. Node $\mathbf{v_k}$ is dropped once its neighbour is visited. It causes the degree of y and z reduced to one. While arriving at $\mathbf{v_1}$ in order to complete the HC either y or z is left out. This is contradiction to the assumption that graph is Hamiltonian.♦

Lemma 4: A valid current node has at least one adjacent node in the current sub graph in a Hamiltonian graph.

Proof: Let $\mathbf{v_k}$ be the current node in the current subgraph H of G. Lemma 1 and lemma 3 imply that every node is of minimum degree two. Let y and z be two such adjacent nodes to $\mathbf{v_k}$. If $\mathbf{v_k}$ is visited before visiting both y and z, or $\mathbf{v_k}$ is visited after y but before z or vice versa then the result is obvious. In the case when $\mathbf{v_k}$ is visited possibly after visiting y and z both, then in order to complete HC in G, there must be another arc as exit route from $\mathbf{v_k}$ and hence an adjacent node.♦

Let $L(\mathbf{v_k}) = \{y \mid y \text{ is adjacent to current node } \mathbf{v_k}\}$. A node $y = \mathbf{v_{k+1}} \in L(\mathbf{v_k})$ is taken to visit next using step 3 and step 4 of the algorithm. Step 3.1 does not leave any option whereas step 3.2 is preferred because of reason proved in case 1 of theorem 2. Backtracking at 3.2 is restricted because selection of any node

$$x \in L(\mathbf{v_k}) - \{y \mid y \in L(\mathbf{v_k}) \text{ and degree}(y) = 2\}$$

would make y dangle. Correctness of the step 3.1 and 3.2 is implied from lemma 3. Again from second part of case 1 of theorem 2, step 3.3 is preferred over 3.4. Since current node may not satisfy the condition of minimum degree node of original graph G, a provision of possible backtracking is kept wherever tie breaking is not possible. Precedence and correctness of the steps is implied from lemma 2 and 3.

Step 3.5 finds node from NGBR which are possibly available and not considered in step 3.3 and 3.4 in order to ensure that a cycle of length < n is not formed prematurely. This step is executed if it is not possible to find a node in 3.3 or 3.4. Correctness of precedence of steps in order of 3.1, 3.2, 3.3, 3.4, 3.5.1 and 3.5.2 is derived from Lemma 2, 3 and theorem 2. After selection of a node in step 3, the post condition is:

$\pi : \mathbf{v_1 v_2 v_3} \ldots \mathbf{v_{k+1}}$
$H = H - \{\mathbf{v_k}\}$
$NGBR = L(\mathbf{v_1}) - \{\mathbf{v_{k+1}}\}$
ARTPNT = As determined.
BACKTRACK $[\mathbf{v_{k+1}}] = \{x \mid x \in L(\mathbf{v_k})$ and x satisfies criteria of 3.3 or 3.4 based on which $\mathbf{v_{k+1}}$ has been selected to move on.$\}$

A node **y** selected in step 3.5 is from NGBR only. It implies that current node $\mathbf{v_k}$ has more than one adjacent nodes and degree(y) > 2 otherwise it would have been selected in either step 3.1 or step 3.2. It further implies that there is no non NGBR node adjacent to $\mathbf{v_k}$ that satisfies 3.3 and 3.4. If there exists more than one nodes satisfying 3.5.1 or 3.5.2 then a node is selected without noting the tied node for backtracking later on. It is implied from lemma 2 that backtracking to any

node latter on does not yield any otherwise result. This provides the proof for postulate (d). The proof of the next postulate (e) is given in theorem 3.

Theorem 3: A graph G is Hamiltonian, if and only if only start node is left at the end of the execution of the algorithm.

Proof: *If only start node* $\mathbf{v_1}$ *is left at the end of the execution of the algorithm then G is Hamiltonian:*

If the algorithm terminates with one node $\mathbf{v_1}$ left subgraph H, it ensures that all intermediate nodes have been visited and subsequently removed. It proves that all nodes starting from the start node have been visited exactly once before finally arriving at start node, therefore forming a HC.

If G is Hamiltonian then only start node is left at the end of the execution of the algorithm:

From lemmas 3 and 4, it is always possible to find an adjacent node to currently visited node in a Hamiltonian graph using the algorithm. Thus at the end only start node $\mathbf{v_1}$ is left when algorithm successfully terminates.♦

Corollary: If more than one node is left when algorithm terminates then G is nonhamiltonian.

Proof: This can be proved by method of contra positive. The 'only if' part of theorem 3 may be stated contra positively as "If not 'only start node is left at the end of the execution of algorithm' then graph is not Hamiltonian'". It can be further stated in simplified language as "if more than one node is left when algorithm terminates then G is nonhamiltonian". Proof is obvious from second part of theorem 3.♦

The important thing for any algorithm is not only to find a correct solution when it exists but to terminate after a finite number of steps in every case. The loop due to step 4 terminates when either or both of the following conditions are met.

i) Number of nodes in the leftover current sub graph is reduced to 1, or/and
ii) No suitable node is found to visit next.

In case (ii) the loop due to step 4 is possibly restarted subject to availability of a node to backtrack. Now it remains to prove that the algorithm terminates in all cases. It is implied from lemma 3 and 4 that the algorithm always returns a node to visit next from the current node if a graph is Hamiltonian. Whenever a valid node is returned to visit next, the number of nodes in the graph is reduced by 1. At the end, the subgraph contains start node only. The step 4 terminates and algorithm goes to step 5. 'If' part of this step ensures the termination.

On the other hand, if a graph is nonhamiltonian then the algorithm goes to step 5 either from 2.2.3 or from 3.5.2. In case of step 2.2.3 there is no possibility of backtracking and hence number of nodes left at that time is > 2. Whereas if control is transferred to step 5 from 3.5.2 then either backtracking is possible to step 3 or it is not possible. In former case, the loop at step 3 and 4 is restarted from a node stored at step 3.3 or 3.4. The node considered once is not considered again for backtracking and therefore ultimately ensures termination of algorithm. This is ensured by step 5. In the latter case the algorithm terminates there.

5. ANALYSIS OF THE ALGORITHM

Determining time complexity of an algorithm requires derivation of an expression that finds number of steps needed to complete the task as a function of the problem size **n** and is to be considered modulo a multiplicative constant [25, 29]. Objective of this section is to compute time complexity of the algorithm in best, worst and average cases [32, 33, 34]. The computation takes into consideration possibility of backtracking in case of tie breaking rules become insufficient.

The algorithm consists of the following major components.

(i) Step 1 and Step 2 are executed once and in sequence.
(ii) Step 3 and step 4 are repeated (n – 2) times.
(iii) Step 5 may force back tracking.

Time complexity of algorithm depends upon number of nodes **n** and edges **e** in G [30, 31]. Step 1 finds start node and it executes in $O(n^2)$ times. Step 2 is executed once and in sequence with step 1. Let time complexity of the step 2 be f(n) to be computed later. Next step 3 and 4 are repeated n times. Let its time complexity be h(n) to be computed later. Hence, the time complexity H(n), of the algorithm is

$$H(n) = O(n^2) + f(n) + O(n * (h(n)) * B(n) \quad \text{.....(1)}$$

Here B(n) is a time complexity function due to possible backtrack. There are three sub steps within step 2. Step 2.1 computes set NGBR. The set union operation is of O(1) as it simply appends a node adjacent to Start node to set NGBR. Append operations may be executed at the most (n – 1) times when all other nodes are adjacent to start node i.e. when graph is complete. Matrix encoding of the graph ensures that no checking of prior presence of a node is required before putting the node in NGBR. Thus step 2.1 if of O(n).

Within step 2.2, 2.2.1 is of O(n). Step 2.2.2 and 2.2.3 can be put within a loop that executes in $O(n^2)$ time complexity. Thus step 2.2 is of $O(n^2)$. In step 2.3, set difference operation

NGBR = NGBR – {Current_node}

is performed in O(n) times. Existence of an articulation point in a graph is determined by applying depth first traversal (DFS) algorithm, which of order of number of edges in the graph i.e. $O(n^2)$ in the worst case. Therefore,

$$f(n) = O(n) + O(n^2) + O(n^2)$$

$$= O(n^2) \quad \text{.....(2)}$$

Turning to step 3, it is repeated n, in fact (n – 1) times due to step 4. It contains six sub steps. In order to initiate the task a list of adjacent nodes is constructed which is of O(n). Then from 3.1 to 3.6 steps are executed in sequence. Step 3.1 and 3.2 are of O(1) and O(n) respectively. Step 3.3 and 3.4 are of $O(n^3)$. Tie, if any, is broken in O(n) and BACKTRACK list is updated in O(1) and that too in sequence, therefore step 3.3 and 3.4 remains of $O(n^3)$.

Next Step 3.5 is of O(n) average case as number of such node shall be minimum. However in worst case it may be of $O(n^2)$. Step 3.6 performs all tasks that step 2.3 performs. In addition to that step 3.6 drops the node just previously visited. The task is performed in O(n) time. Thus, step 3.6 is of $O(n^2)$. Time complexity h(n) of step 3 and 4 therefore can be written as

$$h(n) = O(n) + O(n^3) + O(n^2)$$

$$= O(n^3) \quad \text{.....(3)}$$

Substituting values (2) and (3) in equation (1), the complexity **H(n) is O(n^4 *B(n))**. The B(n) factor is applicable when backtracking is unavoidable from third node onward. Though backtracking is required to a few (one or two) nodes in a few graphs, statistically, even then it can not be overlooked in the worst case.

In the best case, if every node is adjacent to a few nodes e.g. two or three, then f(n) and h(n) are of order n and hence H(n) is of O(n^2). In general, number of adjacent nodes decreases as the algorithm progresses. This happens because of removal of intermediate nodes. In a complete graph, it will be (n – 1), then (n – 2), then (n – 3) and so on. And towards the end, it will be 4, 3, 2 and finally left with one. Therefore while computing f(n) and h(n), steps 3.3 and 3.4 are considered for (n – 1)(n – 2) then for (n – 2)(n – 3) and so on up to 3*2, 2*1 times. The average number of time then steps 3.3 and 3.4 are executed is

$$= \frac{(n-1)(n-2) + (n-2)(n-3) + ... + 3*2 + 2*1}{n}$$

$$= \frac{1}{n}\left(\sum_{i=2}^{n-1} i^2 + \sum_{i=2}^{n-1} i \right) = O(n^2)$$

Considering call to set membership function, h(n) shall return a complexity of O(n^3) and f(n) of O(n^2). If gradual removal of nodes are taken into consideration then algorithm may not enter into nested loop like construct of 3.3 and 3.4 all the times. In this case h(n) and f(n) shall return complexity of O(n) for some node. If about 50% call enter into nested loop and remaining return without entering into it then amortized analysis yields time complexity of O(n^2) for h(n). It means H(n), the complexity of the algorithm, is of O(n^3*B(n)) from equation (1) in average case.

6. CONCLUSION

The algorithm presented in the paper has been tested on large number of graphs varying from simple to very complex up to the tune of 300 nodes. The algorithm is programmed in C language and adjacency matrix is used as data structure to store graph. A graph is pre-processed using line-sweeping [37] algorithm, to merge all nodes in a linear component. It is found that the algorithm executes in polynomial time in most of the time. Number of backtracking required is almost negligible. But unless the algorithm is improved to completely prevent the backtracking from third node onward, it can not be claimed of the polynomial time.

The gist of algorithm is to visit next node, prune the graph by dropping the visited intermediate node as and when its neighbour is visited. While selecting the neighbour to visit next, it is ensured that no backtracking to the current node will be needed in many cases to confirm the result. Wherever there is absolutely no way to break the tie, provision to keep the option open for backtracking is made. Number of nodes to be explored, next, is reduced at every step due to pruning. This reduces the complexity of the algorithm from n! to polynomial of degree 3. In worst case, when backtracking is required, the complexity calculation is generalized to non polynomial.

The presented algorithm may be further improved to evolve tie breaking rule(s) to prevent backtracking in steps 3.3 and 3.4. If it so happens, it might be a breakthrough in the field of graph algorithm and theory of algorithmic complexity. The algorithm presented may find its application in many areas. The author of this paper is using it in steganographic technique using graph theoretic approach.

7. ACKNOWLEDGEMENT

I am thankful to my friends and students who occasionally come to me for guidance in Discrete Mathematics, Theory of Computation and Analysis & Design of Algorithms. I am grateful to all those who constantly encouraged me to go for such academic work besides the work which I am doing in NIC. I remain indebted to my teachers and colleagues who have supported in many ways in completing the work. At the last but not the least, I place my sincere thanks to all anonymous referees/reviewers who gave very critical judgement and suggestions to improve upon algorithm and overall presentation of the paper.

REFERENCES

[1]. A. M. Frieze, "Limit Distribution for the Existence of Hamiltonian Cycles in a Random Bipartite Graph", European Journal of Combinatorics (1985) 6, 327-334.

[2]. Akiyama, Takanori, Nishizeki, Takao and Saito, Nobuji, NP-completeness of the Hamiltonian cycle problem for bipartite graphs. J. Inf. Process. v3 i2. 73-76.

[3]. Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, 1974, The Design and Analysis of Computer Algorithm, Addison-Wesley.

[4]. Berge C. 1962, Theory of Graphs and its Application, Methuen Press.

[5]. Berge, Graphs and Hypergraphs, Elsevier Science Ltd, 1985.

[6]. Bertossi, Alan A., Finding Hamiltonian circuits in proper interval graphs. Inform. Process. Lett. v17 i2. 97-101.

[7]. Bertossi, Alan A., The edge Hamiltonian path problem is NP-complete. Inform. Process. Lett. v13 i4-5. 157-159.

[8]. Bollob~s, 'Almost all regular graphs are Hamiltonian', European Journal of Combinatorics 4 (1983) 311-316.

[9]. Chvátal, V., Hamiltonian cycles. In: Wiley-Intersci. Ser. Discrete Math, Wiley, Chichester. pp. 403-429.

[10]. Corneil, D.G., Lerchs, H. and Stewart Burlingham, L., Complement reducible graphs. Discrete Appl. Math. v3. 163-174.

[11]. E. Shamir, 'How many edges are needed to make a random graph Hamiltonian?' Combinatorica (1983).

[12]. G. A. Dirac, 1952, Some theorems on abstract graphs, Proc. London. Math. Soc.2.

[13]. Garey, M.R., Johnson, D.S. and Endre Tarjan, R., The planar Hamiltonian circuit problem is NP-complete. SIAM J. Comput. v5 i4. 704-714.
[14]. Haiko Müller, Hamiltonian circuits in chordal bipartite graphs, Discrete Mathematics, v.156 n.1-3, p.291-298, Sept. 1, 1996
[15]. Itai, Alon, Papadimitriou, Christos H. and Szwarcfiter, Jayme Luiz, Hamilton paths in grid graphs. SIAM J. Comput. v11 i4. 676-686.
[16]. J. A. Bondy and U.S.R. Murty, 1976, Graph Theory with Applications, North-Holland.
[17]. J. Blazewicz , A. Hertz , D. Kobler , D. de Werra, On some properties of DNA graphs, Discrete Applied Mathematics, v.98 n.1-2, p.1-19, Nov. 15, 1999.
[18]. Jacek Błazewicz , Marta Kasprzak, Complexity of DNA sequencing by hybridization, Theoretical Computer Science, v.290 n.3, p.1459-1473, 3 January 2003.
[19]. Jacek Blazewicz , Marta Kasprzak, Computational complexity of isothermic DNA sequencing by hybridization, Discrete Applied Mathematics, v.154 n.5, p.718-729, 1 April 2006.
[20]. Jan van Leeuwen, 1990, Handbook of theoretical computer science, MIT Press.
[21]. Jitender S. Deogun , George Steiner, Polynomial Algorithms for Hamiltonian Cycle in Cocomparability Graphs, SIAM Journal on Computing, v.23 n.3, p.520-552, June 1994 [doi>10.1137/S0097539791200375].
[22]. Karp, Richard M., Reducibility among combinatorial problems. In: Complexity of Computer Computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y, 1972), Plenum, New York. pp. 85-103.
[23]. Knuth, D. E., 1973, The art of Computer Programming, vol 1, Fundamental algorithm, Addison-Wesley Publishing Company.
[24]. Lawler, Eugene L., Combinatorial Optimization: Networks and Matroids. 1976. Holt, Rinehart and Winston, New York.
[25]. Lewis, Harry and Christos Papadimitriou, 1978, "The efficiency of algorithms, Scientific American", 238:96-109.
[26]. M. Held and R. M. Karp, ' A dynamic programming approach to sequencing problems', SIAM Journal on Applied Mathematics 10 (1962) 196-210.
[27]. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. Proceedings of the sixth annual ACM symposium on Theory of computing, p.47-63. 1974.
[28]. Mark Keil, J., Finding Hamiltonian circuits in interval graphs. Inform. Process. Lett. v20 i4. 201-206.
[29]. Michael R. Garey and David S. Johnson (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman. ISBN 0-7167-1045-5. A1.3: GT37–39, pp.199–200.
[30]. Karp, R. M., & Steele, J. M. (1985). Probabilistic analysis of heuristics. In The Traveling Salesman Problem, pp. 181-205. John Wiley & Sons, Essex, England.
[31]. Rubin, Frank, "A Search Procedure for Hamilton Paths and Circuits'". Journal of the ACM, Volume 21, Issue 4. October 1974.
[32]. Selmer Bringsjord & Joshua Taylor, P = NP, Department of Cognitive Science, Department of Computer Science, RAIR Lab, RPI, Troy, NY
[33]. Sipser, M. 1992, The history and status of the P verses NP question, in Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, pp 603-618.
[34]. Stephen Cook, 2000, The P versus NP Problem, Official Problem Description, Millennium Problems, Clay Mathematics Institute.
[35]. T.I. Fenner and A.M. Frieze, 'on the Existence of Hamiltonian Cycles in a Class of Random Graphs', Discrete Mathematics 45(1983) 301-305.
[36]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, 1998, Introduction to Algorithms, PHI.
[37]. Vinay Kumar, and Sharma, V. (2006) 'Overcoming 64kb data size limit in handling large spatial data in GISNIC while cleaning and building topology', Int. J. Information Technology and Management, Vol. 5, No. 1, pp.77–86.
[38]. Vinay Kumar, 2002, Graph Theory, Chapter 7, Discrete Mathematics, Ed 1, BPB Publication, New Delhi, India.