http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (5): 851-859, 2006
ISSN 1812-5638

Finding Hamiltonian Cycle in Polynomial Time

Khadija Riaz and Malik Sikander Hayat Khiyal
Department of Computer Science, International Islamic University, Islamabad, Pakistan

Abstract: One of the big mysteries in contemporary computer science is whether P=NP. Finding a Hamiltonian cycle in a graph is one of the classical NP-complete problems. Complexity of the Hamiltonian problem in permutation graphs has been a well-known open problem. No solution exists to get HC in polynomial time and there are no such conditions to decide the probability of HC exists (Neapolitan and Naimipour, 1996). In this study the authors prove that Hamiltonian cycle in an undirected graph can be found in polynomial time, and thus the problem is a discrete problem. Authors present valid conditions to tell in advance, while entering the graph input, that HC does not exist. A polynomial time algorithm for constructing a Hamiltonian cycle is also presented.

Key words: Discrete structure, graphs, algorithms, Hamiltonian cycle

INTRODUCTION

Hamiltonian Cycle Problem is a problem on graphs formalized by Sir William Rowan Hamilton, a mathematician of 19th century in Ireland. Hamiltonian circuit for a graph G is a sequence of adjacent vertices and distinct edges in which every vertex of graph G appears exactly once (Fig. 1). A Hamiltonian Graph is a graph that has a Hamiltonian cycle. The problem is, whether there is a circuit passing all the points of a given graph or not. In the case of a graph whose number of points is N, the number of such circuits is N! at most (Nasu, 1999). The simplest way to examine whether there is a circuit satisfying the request is to check all the paths thoroughly in round robin and its calculation cost amounts to the order N! of exponential time algorithms. This method has been used in Brute Force Algorithm (Neapolitan and Naimipour, 1996).

On the time complexity bases, there are two types of problems. Polynomial time P and non deterministic polynomial time NP problems. P is a set of all decision problems that can be solved by polynomial time algorithms .For example, the problem of determining whether a key is present in an array. A polynomial time non deterministic algorithm (NP) is a nondeterministic algorithm whose verification stage is a polynomial time algorithm. So the NP (non-deterministic polynomial) decision problems can be solved by polynomial time nondeterministic algorithms. There is a group of problems which is not certified whether any polynomial time algorithms exist or not and they are called NP –Complete problems (Neapolitan and Naimipour, 1996). Finding a Hamiltonian cycle in a graph is one of the classical NP-complete problems. Now the question is, whether there exist a deterministic algorithm which finds HC in polynomial time or not. This problem is a challenge for mathematicians for a long time of one century. A problem is NP if it is easy to check the correctness of a claimed

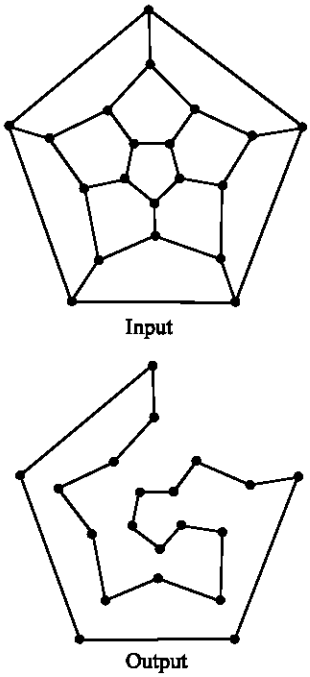


Fig. 1: Input and output of Hamiltonian cycle

Corresponding Author: Malik Sikander Hayat Khiyal, Department of Computer Science, International Islamic University, Islamabad, Pakistan

solution. In other words the solution can be checked in polynomial time. This does not say it is easy to find a solution. Thus a problem which is both NP (verifiable in nondeterministic polynomial time) and NP-Hard (any other NP-problem can be translated into this problem) is NP complete problem In the current study we present an algorithm which runs in polynomial time. We have imposed certain conditions; these conditions make this problem a pure discrete problem.

Algorithm is along the following lines:

When we are standing at a node, we select the next node by using some conditions. These selection conditions make the algorithm polynomial. We use backtracking process when some blocking conditions occur during processing. But the selection conditions (selection of the next node) are such that they have minimized the probability of backtracking. In general HC in a graph is found without backtracking because of these selection conditions. The backtracking process mostly occurs in graphs where no HC exists or in symmetric graphs. Even in these graphs some specific nodes, called the junction nodes, participate in backtracking process. We use the term 'junction'to refer a node from which an adjacent node of least degree is selected as next node, while there are more than one nodes of least degree adjacent to that node. And by 'degree' we mean the number of nodes adjacent to a node. The degree of a node varies during processing. All nodes of a symmetric graph are not junction nodes because the degree of each node varies, and hence they will not participate in backtracking process. Preprocessing conditions tell in advance that HC does not exist, but do not tell about its presence. If a graph has no HC, our algorithm will accurately report this in fast polynomial time. If all the preprocessing conditions are true and processing starts while the HC in the graph does not exist, in this case soon we will encounter one of the blocking conditions. This condition will start the backtracking from the last junction node. If HC does not exist then the junction storage will become empty and no node would be there to bear backtracking.

So we find the solution of following basic problems related to Hamiltonian cycle.

- Polynomial time algorithm for finding Hamiltonian cycle.
- Valid conditions to decide the probability of HC exist.

The problem of HC was invented by Sir William Rowan Hamilton in 1859 as a game. Since 1936, some progress has been made in this field. For certain special graphs several efficient algorithms exists e.g. Polynomial algorithm for 4-connected planar graphs (Nishizeki and Chiba, 1988), Polynomial algorithm for quasi-transitive digraphs (Gutin, 1994). But for general graphs, no efficient algorithm exists (Neapolitan and Naimipour, 1996). The problem is still NP-complete for perfect graphs, planar bipartite graphs, grid graphs, 3-connected planar graphs. Some sufficient and necessary conditions for a graph to have HC have been given. Some of these conditions are listed below (Balakrishnan, 1997).

- A necessary condition for a graph to be Hamiltonian If G=(V,E) is Hamiltonian and if W is any nonempty proper subset of V, the graph G-W has at most | W| components.
- Ore's Theorem: A sufficient condition for a graph to be Hamiltonian (Ore, 1960). A simple graph with n vertices (where n>2) is Hamiltonian if the sum of the degrees of every pair of non adjacent vertices is at least n.
- Dirac's Theorem: A sufficient condition for a graph to be Hamiltonian (Chartran and Oellermann, 1993). A simple graph with n vertices (where n>2) is Hamiltonian if the degree of every vertex is at least n/2.
- If G is a 2-connected graph such that for every pair of nonadjacent nodes u and v, |d(u)+d(v)|>=(2n-1)/3 then G is Hamiltonian (Fraudee, Gould, Jacobsen and Schelp, 1989).
- All Hamiltonian graphs are biconnected, although the converse is not true (Skiena, 1990).

Classification of algorithms: There are two main classes of algorithms according to Hamiltonian problem:

- Heuristic algorithms (Pósa, UHC, DHC, HAM, etc)
- Backtrack algorithms (595HAM, KTC, MultiPath)

Heuristic algorithm VS our algorithm: An improved version of heuristic Algorithm for HC is given in (Altschuler, 2000). This algorithm runs in at most N^3 time, if there is no HC, the given algorithm will report this in polynomial time. If there is a Hamiltonian cycle and the algorithm finds it, it will do so in polynomial time. The only danger is if there is a Hamiltonian cycle but the algorithm does not find it, reporting the incorrect result that there is no HC. How could this happen? It might be that the Hamiltonian cycle has so low probability that it is not found even upon iterating the simulated annealing algorithm. Alternatively, as more roads are excluded, the probability of finding a Hamiltonian might be reduced because a subset of the excluded roads is necessary as "intermediate states" toward progressing to an Hamiltonian cycle. So the advantage of Heuristic

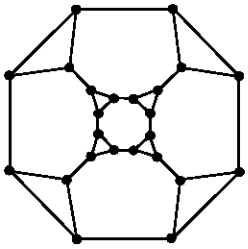


Fig. 2: Symmetric graph with junction nodes

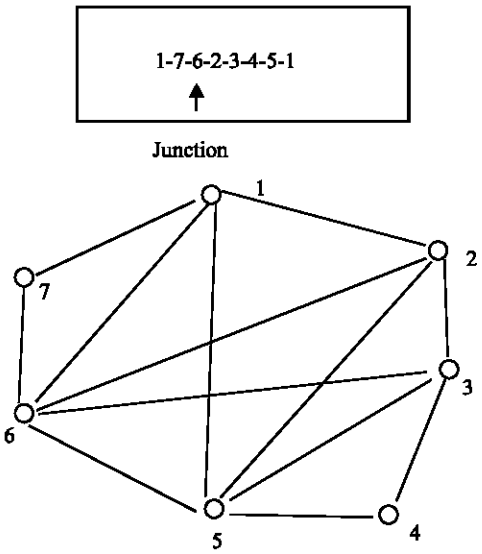


Fig. 3: Graph and its Hamiltonian Cycle

algorithm is that it is Fast, Linear or low-order polynomial time algorithm but the disadvantage is that, it does not always find the HC. Our Algorithm presented here finds HC in polynomial time but with no such danger. As we are not excluding edges from our graph, we only mark the edges where we do not want to go, and then in backtracking process we can demark those edges. Our algorithm is restricted in selecting the next node during processing.

Backtracking algorithms VS our algorithm: Backtracking algorithms search all the potential solutions and then see whether a HC exists or not. The main advantage of Backtracking algorithms is that it finds all solutions and can decide about its existence but at the same time there is a severe disadvantage that it is a worst case, it needs exponential time. Also it takes a long time to process (Neapolitan and Naimipour, 1996).

In our algorithm, only junction nodes may participate in backtracking. From a junction node, selection of next node depends on how the graph has been entered as input. During the processing when a blocking condition encounters, it causes backtracking. These blocking conditions normally occur in the graphs where no HC exist and may be in the symmetric graphs. Because of the selection conditions, HC is often found without backtracking. In the case when no HC exists or as in the symmetric graphs, not all nodes are junction nodes. As an example, consider Fig. 2, here we have 24 nodes and according to our algorithm, only 5 nodes are junction nodes.

In non symmetric graphs, there can be junction nodes, but backtracking does not normally occur. See graph of Fig. 3.

Thus we have reduced the chances of backtracking, by implementing the selection conditions (selection of next node) and blocking conditions. The selection conditions are such that if at a junction node we select a wrong path, soon a blocking condition will encounter. And we will have to move back to that junction node to select the right path. It implies that we are using backtracking with limitations i.e., we are finding HC accurately, but avoiding its disadvantage i.e long processing time.

OUR ALGORITHM

We have implemented our algorithm in C language. It gives correct results for all types of graphs including perfect graphs, planar bipartite graphs, grid graphs, 3-connected planar graphs. The algorithm is presented here.

Preprocessing conditions: Our algorithm simplifies the given graph by removing parallel edges and self-loops before looking for Hamiltonian circuit. Then preprocessing conditions are checked. If any of these conditions is met, the graph will not have Hamiltonian circuit.

- No node should have degree 1.
- No node should have more than two adjacent nodes having degree 2.

Other wise the processing will precede.

Processing:

Step 1: Select a node of highest degree from the graph, to start traveling. We call this node the Starting node. Store

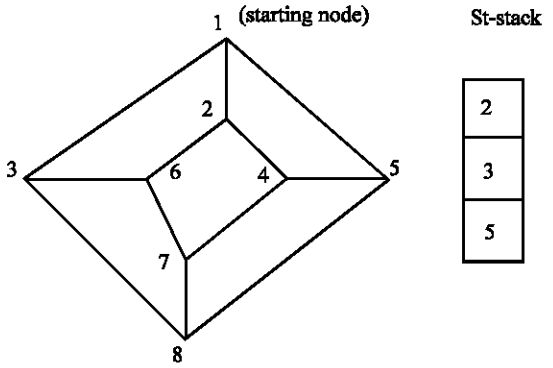


Fig. 4: Input of the graph and St-stack nodes

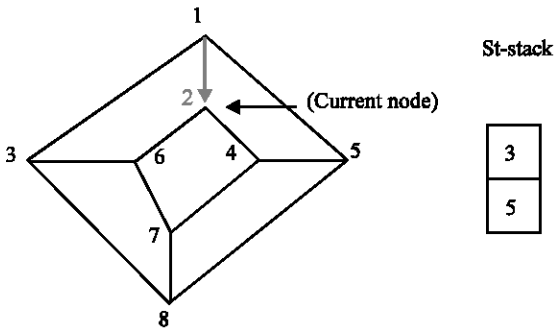


Fig. 5: Processing on the graph

the adjacent nodes of Starting node in a stack, we call this St-stack (Fig. 4).

Step 2: Select one of the St-stack nodes and name it as next node according to the following priorities:

- The St-stack node having least degree.
- If nodes having least degree are more than one, select anyone of them as next node. And add current node in the stack junction.

Step 3: Go to next node and delete it from the St-stack. Mark the connecting edge between the starting node and

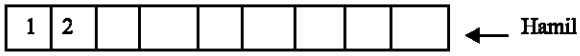


Fig. 6: Hamil

next node, from the graph. Now the next node becomes the current node (Fig. 5) .

Step 4: As we proceed from one node to the next, add all visited nodes in a stack called hamil. Up to now in our example two nodes have been visited. Hamil stores the nodes of complete HC (Fig. 6).

Step 5: *While* (Next node ≠ Starting Node)

If (we are standing at a St-stack node and some other nodes are left which have not been processed yet) *Then*

{

Mark the connecting edge between St-stack node and the staring node (we do not select the starting node as next node).

}

Note: Marking of an edge is similar to deleting the edge, only the difference is, we can restore a marked edge in backtracking. "Adjacent" means adjacent to the current node.

/*Selection Conditions : In section selecting condition below, we select next node from the adjacent nodes of the current node, in order of the following conditions. After a next node has been selected in one of the following steps, remaining steps will not work */

Selecting condition:

1. *If* (we are standing at St-stack node while there is no node (including St-stack nodes) left which has not been processed) *Then*

{

select starting node as next node

}

2. *If* (There is an adjacent node of degree 2, to the current node) *then*

{

```
        If  (There are more than one adjacent node having degree 2)  then
                Call Backtrack( )
        Else  If  (There is only one adjacent node of degree 2)  then
        {
                Select that node as next node
                If  ( This selected node is St-stack node)     then
                {
                   If (One St-stack node left while all other nodes of the graph have been processed )  then
                       The selected node is next node
                   If  (One St-stack node left while some other nodes of the graph are also there which have not been
                      processed yet)  then
                      Call Backtrack( ).
                }
        }
    }
3. If (There is one St-stack node adjacent to the current node)  then
    {
       If (The stack (St-stack) have more than one nodes)  then
         pick this node as next node.  And delete it from the stack (St-stack)
       Else If ( The stack (St-stack) have only one node left)  then
       {
            // there are two possibilities
            If  (All other nodes of the graph have been processed) then
            Choose this node as next node and delete it from the
            stack St-stack
            If (Some  nodes of the graph  are left which have not been
                processed yet)  then
            Mark this connecting edge (b/w  current node and  St-stack
            node( we are not taking it  as next node)
       }
    }
else if (there are more than one St-stack nodes adjacent to the current node)

       select the St-stack node with least degree
    //if there are more than one St-stack nodes of least degree then do not add this
    // current node in the stack "junction".
    // Fourth Condition, select a node with the least degree
    If ( next node has not been selected yet in the previous steps) then
    {
            select a node  with least degree
       If (there are more than one nodes having least degree) then
            place  this Current node in a stack called "junction"
    }
4. If (no next node has been selected because there is no appropriate adjacent node)  then
            call   Backtrack().
4.2.2.    /* A next node has been selected in previous step.  next node will become current
          node, after following changes */
```

- All the edges from current node (on which we are standing) to all its adjacent nodes are marked , this path in future will not be taken . It means there is no path from any vertex to come back at this node.
- Decrement the degree of all adjacent nodes by one (except the previous node from which we have shifted at this node).
- Add Current node to the stack hamil

- Now rename next node as current node.

End while

Procedure Backtrack ( )
{
If (The stack " JUNCTION" is empty) *then*
Stop processing and prompt that it is not Hamiltonian circuit
// Other wise all the following reverse processing will be carried out:

- Take the last node from the stack junction as current node
- Select this current node from the stack hamil.
  /* From this node till the last stored node in the stack hamil, the reverse changes occur, these changes are as described below. we should keep in mind that if here the current node is Starting node then we will neither increment the degree of St-stack nodes nor demark the edges. We increment the degree of only those nodes which we have decremented in previous processing and demark only those edges which we have been marked */
  - In the graph start from current node and start demarking all previously marked edges, to the last node of hamil.
  - Increment degree of adjacent nodes by one
- After all the reverse changes have occurred, the next alternative node adjacent to the current node is chosen as next node
- If there is no more alternative path after selecting next node, this node is deleted from the stack, junction.
- If current node is staring node then after selecting next node delete current node from the stack junction(As we are checking only two paths from start, if blocking conditions occur on both of the paths then HC will not exist)
- The new next node selected is sent back for further processing

}

End of Algorithm

OUR ALGORITHM IS POLYNOMIAL TIME ALGORITHM

All NP-Complete problems appear to be difficult. We are not aware of any polynomial time algorithms to solve such problems. However, there is no proof that polynomial time algorithms do not exist! (Standish, 1994).

Backtracking on each node is the reason for HC problem to have exponential time complexity.

Without using backtracking finding HC in polynomial time is not possible. But problem can be made polynomial if we reduce the chances of backtracking to such an extent that only small number of nodes involve in backtracking. This can happen by selecting proper nodes as next node to find the complete cycle.

In our algorithm, we do not select a next node randomly we impose certain selection conditions which helps us in selecting exact node in the cycle. In our algorithm backtracking may occur only on the junction nodes. Junction nodes are those nodes where we select a next node by using fourth selection condition i.e a next node having least degree, while there are some other adjacent nodes having least degree. This node is stored in stack "junction", it may involve in backtracking. At a junction node to proceed further, if we make a wrong decision some blocking condition will encounter. And there would be backtracking up to that junction node. If we encounter a problem where there is a need to backtrack, while the stack junction is empty, we will prompt a message that no HC exists. It also can happen that the stack junction has many nodes, but no backtracking occurs, it usually happens in non symmetric graphs. And it also happens that from the nodes stored in junction only one or two last nodes bear backtracking while all other nodes do not.

If all the preprocessing conditions are true and processing starts while the HC in the graph does not exist, in this case soon we will encounter one of the following problems

- No two adjacent nodes of a node (other then starting node) should have degree 2
- Stack node with degree 2 while some other nodes are left which have not processed yet.

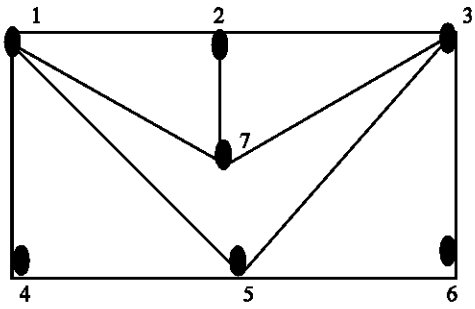


Fig. 7: Input graph

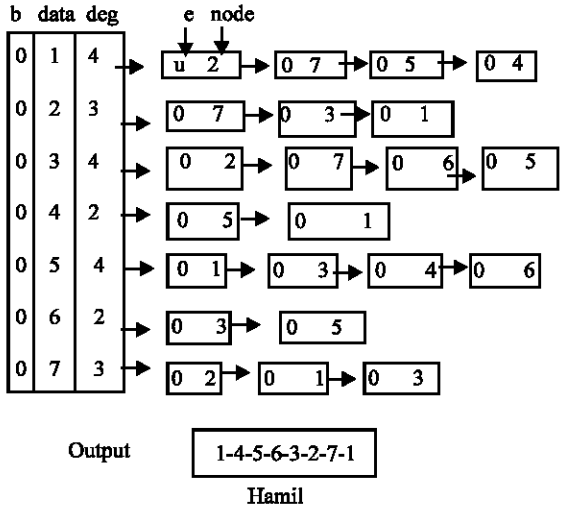


Fig. 8: List representation of graph input

These problems cause backtracking on the junction nodes. If HC does not exist then the junction stack will become empty and no node would be there for backtracking . It will be reported that HC does not exist. For the following reasons our algorithm is polynomial time:

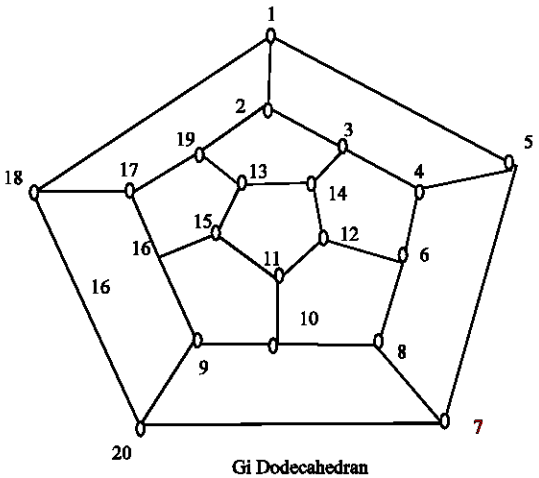
- Preprocessing conditions tell in advance that HC does not exist.
- At a node, where we have more than one paths to go further, if we select each adjacent node one by one to check whether it is suitable or not, then it would be exponential time algorithm. Its time complexity would be $\Theta = (n!)$ or (2^n) or (n^n). In our algorithm, at a node we select next node to find HC by using selection conditions. We do not need to process each adjacent node one by one and we directly select a proper next node.
- Few nodes are junction nodes so backtracking is limited. If there is need to backtrack then only junction nodes involve in this process. It usually happens that there are junction nodes but no backtracking occurs or it occurs only on few last junction nodes.
- We can say it is a discrete problem as we have valid selection conditions for the selection of next node.
- "If any of the preprocessing conditions is true, there are 100% chances that it is not Hamiltonian circuit. If no preprocessing condition is fulfilled then it may or may not be HC
- "It can find HC for any number of vertices, depending on implementation of this algorithm.

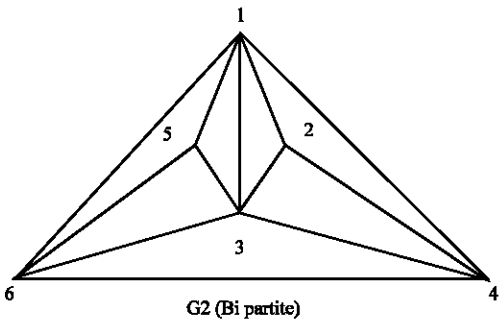
Example:

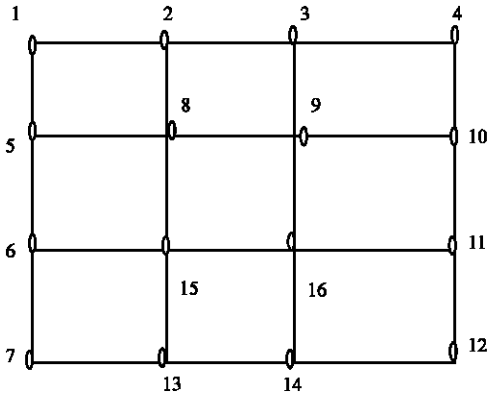
We have implemented our algorithm using link list. As an example consider a graph in figure 7. Fig. 8 shows List representation of input graph in deterministic machine. In this figure fields "b" and "e" tell whether this node has been passed or not.

By running our algorithm the output of HC is shown in hamil. All this has been find without backtracking. From node 3 there are two adjacent nodes of degree three but both are St-stack nodes. So we do not add node 3 in stack "junction". In this example junction stack is empty.

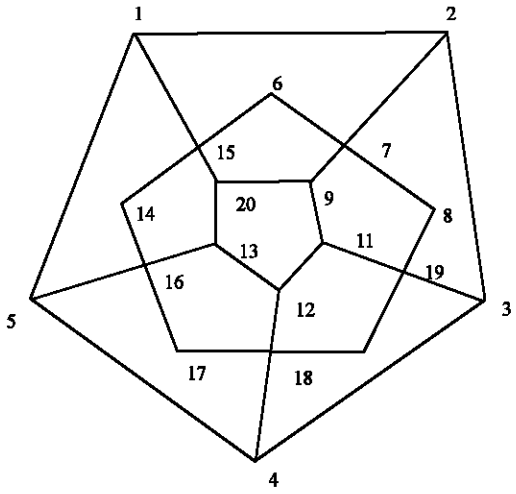
RESULTS



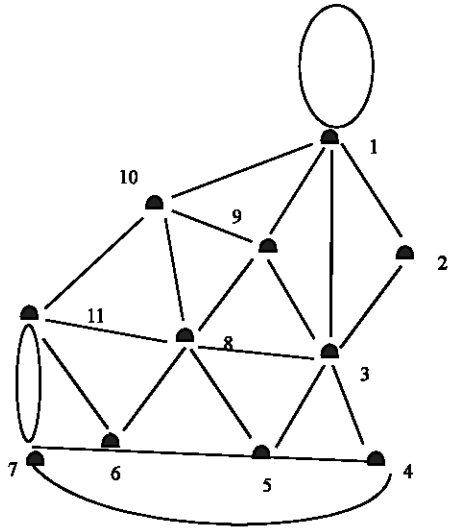




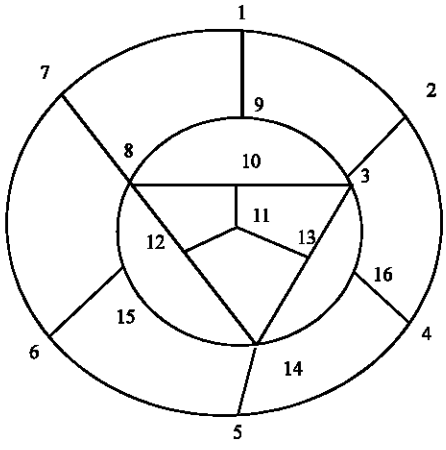
G3 (Grid graph)



G6



G4



G5

Output:

Graph		Results
G1	HC	1-18-20-7-5-4-6-8-10-9-16-17-19-13-15-11-12-14-3-2-1
	Junction nodes	1-18-20-4
	Backtracking	No Backtracking
G2	HC	1-2-4-6-3-5-1
	Junction nodes	1-4-6
	Backtracking	No Backtracking
G3	HC	8-2-1-5-6-7-13-15-16-14-12-11-10-4-3-9-8
	Junction nodes	8
	Backtracking	No Backtracking
G4	HC	3-2-1-9-8-10-11-6-7-4-5-3
	Junction nodes	11
	Backtracking	No Backtracking
G5	HC	HC does not exist
	Junction nodes	3 (Starting node)
	Backtracking	Backtracking occurs only on the starting node i.e., 3, two paths are checked from starting node. If both are ended by blocking conditions then we prompt that HC not exist and we do not further check any path.
G6	HC	HC does not exist
	Junction nodes	7 (Starting node)
	Backtracking	Backtracking occurs only on the starting node i.e., 7, two paths are checked from starting node. If both are blocked by blocking conditions then we prompt that HC not exist and we do not further check any path.

CONCLUSIONS

Our algorithm provides a successful and reliable mechanism for finding HC in polynomial time.

ACKNOWLEDGMENTS

The authors wish to thank Almighty Allah who enabled them to do such type of study. We are thankful for the inspiration, wisdom and constructive criticism of Dr. Zaigham Mehmood. We are grateful to Mr. Muhammad Nadeem, for his technical help in writing the program in C++. We are also grateful to Dr. Khalid Rashid for his moral support and encouragement.

REFERENCES

Altschuler, E.L., 2000. Suggestion for a fast Heuristic for the Hamiltonian cycle problem. Arxiv: physics/ 0007098 v1 30.

Balakrishnan, V.K., 1997. Graph Theory Including Hundreds of Solved Problems. Schaum Series, McGraw Hill Publishing Co.

Chartran, G. and O.R. Oellermann, 1993. Applied and Algorithmic Graph Theory. McGraw Hill, New York.

Fraudee, R.J., R.S. Goulb, M.S. Jacobsen and R.H. Schelp, 1989. Neighborhood unions and Hamiltonian properties in graphs. J. Combin. Theory, 47: 1-9.

Gutin, G., 1994. Polynomial algorithms for finding Hamiltonian paths and cycles in quasi-transitive digraphs. Aust. J. Combin., 10: 231-236.

Nasu, M., 1999. A study of Hamiltonian Circuit problem, the fourth draft, April 4, 1996 to August 18, 1999, Baba Laboratory Inc. Ltd. (www.geocities. com/ babalabo)

Neapolitan, R.E. and K. Naimipour, 1996. Foundations Of Algorithms using C++ psedocode. 2nd Ed. Jones and Barlett Publishers, Sudbury, UK.

Nishizeki, T. and N. Chiba, 1988. Plannar Graphs: Theory and Algorithms. North Holland Mathematics Studies 140, Amsterdam.

Skiena, S., 1990. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Addison Wesley, Reading city, CA.

Standish, T.A., 1994. Data Structure, Algorithms, and Software Principles. Addison-Wesley, Reading.

Ore, O., 1960. A note on Hamiltonian Circuits. Amer. Math. Monthly, 67: 55.