Determine all Edge-Disjoint pairs of Hamiltonian Circuit

Mohd. Abdullah
Indian Institute of Information Technology, Allahabad
Ism2014004@iiita.ac.in

Jatin Goel
Indian Institute of Information Technology, Allahabad
Icm2014503@iiita.ac.in

Rohit Raj
Indian Institute of Information Technology, Allahabad
Ism2014003@iiita.ac.in

***Abstract*— A Hamiltonian cycle, is a graph cycle through a graph that visits each node exactly once. We are given an undirected graph from which we have to determine all the edge-disjoint Hamiltonian circuits. In this paper, we propose two approaches to determine edge-disjoint Hamiltonian circuit. In Approach 1, we apply permutation to determine the Hamiltonian cycles and in Approach 2 we apply backtracking to determine Hamiltonian cycles. After all valid Hamiltonian cycles are generated we determine all the edge-disjoint pairs.**

Keywords—Hamiltonian, Edge-Disjoint, Bitset

I. Introduction

A Hamiltonian cycle, also called a Hamiltonian circuit, is a graph cycle through a graph that visits each node exactly once. In general, the problem of finding a Hamiltonian cycle is NP-complete so the only known way to determine whether a given general graph has a Hamiltonian cycle is to undertake an exhaustive search. A pair of Hamiltonian Circuit T_1,T_2 is an edge-disjoint Hamiltonian circuit if T_1, T_2 don't have any common edges.

II. Proposed Approach

In order to determine the edge-disjoint Hamiltonian circuits from the given undirected graph we have proposed two different approaches, Approach 1 and Approach 2.

In the Approach 1, we first generate all the permutations which are possible for given set of vertices, then we checked whether the particular permutation form Hamiltonian circuit or not. If the particular permutation satisfies the property of Hamiltonian circuit then we store this Hamiltonian cycle. After this we check all the pairs of Hamiltonian cycles whether they are edge disjoint or not. For two edge-disjoint Hamiltonian cycles there must not be any common edge between them. The pseudo code for Approach 1 is as follows:

```
Edge_disjoint_hamiltonian( ){
        //Initialize path vector
        for i=1 to V
                path[i] = i
        //Generate all Permutation
        while(next_permutation(path))
                // To check given path vector is hamiltonian
        circuit or not
                 check_hamiltonian(path)
                        if(true)
                                hamiltonian_cycles.appen
                        d(path)

        // To Check Disjoint
        for each pair in hamiltonian_cycles
                check_if_disjoint();
                if(true)
                Edge_disjoint_hamiltonin.append(pair);
}
check_hamiltonian()
{
        for i=1 to V
                if(graph[path[i]][path[(i+1)%V]] == 0)
                        return false;
                return true;
}

check_if_disjoint()
{
        if(cycle pair have common edges)
                return false;
        return true;
}
```

Our Approach 2 algorithm is divided into two parts. The first part of our solution aims at finding all the unique Hamiltonian circuits and the second part of our solution aims at finding all pairs of edge disjoint Hamiltonian circuit.

Part 1:

As we know that in a circuit, any node can be a start node, but in our approach, we fix our start node to be 1. Now from this start node, we backtrack for rest of the nodes which can be part of circuit. Suppose we have considered nodes 1, a, b, c...m till now and nodes adjacent to m were p, q, r, s. Now we pick a node adjacent to m say p which is not yet included in path, and then we include it in path so our path becomes: 1, a, b, c....m, p and recur to find other nodes of path which are adjacent to p and so on. Later when we backtrack back again to p, we remove it from our path and insert q such that q is not yet included and update the path so now our path becomes: 1, a, b, c...m, q, and we recur for q and so on.

At a moment if we have included all the nodes of the graph in path such that they form a circuit then we get a Hamiltonian circuit and we store it. So after this algorithm is terminated we will have all the possible unique Hamiltonian circuits present in the graph.

Part 2:

In order to find all pairs of edge-disjoint Hamiltonian circuit, we are using a data structure called Bitset. Bitset is an array like linear data structure where each index can store only 0 or 1. Initially all the edges of the graph are numbered. So, if a particular edge say i^{th} edge is present in Hamiltonian circuit, we set the i^{th} bit of the Bitset to 1.

```
Bitset B[]
{
    B[i]=1: i^th edge is present in circuit
    B[i]=0: i^th edge is not present in circuit
}
```

Use of Bitset here is completely justified by the fact that it uses very less space and we can perform several bitwise-operations like xor, and, or, etc. very efficiently on Bitset. In order to check whether two circuits are edge disjoint or not we calculate the bitwise-and of their edge bitset. Let's say we have two Hamiltonian circuits H1 and H2, and their edge bitset be B1 and B2 respectively. We calculate a new bitset, B = B1 & B2. Now if i^{th} bit is set 1 in bitset B, this means that i^{th} edge is present in both the circuits H1 and H2. So, if there exists at least one bit which is set 1 in B then it means that H1 and H2 is not edge disjoint. Henceforth, in order to be an edge-disjoint Hamiltonian circuit, all the bits should be zero in our bitset B. We do this for every pair of Hamiltonian circuits to find out edge disjoint pairs. The pseudo code for our proposed solution is as follows:

```
// To find all the unique Hamiltonian circuits
backtrack(current_node,taken) {
    if taken == no_of_nodes
        e = edge(current_node,start_node)
        if e == -1
            return
        edge_bitset.set(e)
        all_circuit.insert(circuit,edge_bitset)
        edge_bitset.unset(e)
        return
    for v : adjacency_list[current_node]
        if visited[v.node] is false
            taken + = 1
            circuit.insert(v.node)
            edge_bitset.set(v.edge)
            visited[v.node] = true
            backtrack(v.node)
            taken - = 1
            visited[v.node] = false
            circuit.pop()
            edge_bitset.unset(v.edge)
}

main() {
    circuit.insert(1)
    taken = 1
    start_node = 1
    visited[1] = true
    backtrack(1)
    // Find all pairs of edge disjoint Hamiltonian circuit
    for(i ← 0 to all_circuit.size())
        for(j ← i+1 to all_circuit.size())
            if(((all_circuit[i].edge_bitset & all_circuit[j].edge_bitset).count()) == 0)
                print(i,j)
}
```

III. Result

Notation:

N: Number of Nodes
T: Number of Hamiltonian Circuits
E: Number of Edges

The time complexity for the **Approach 1** is as follows:

When the given graph is complete, we have all edges between all pair of vertices then we have n! Hamiltonian Cycles. Therefore,

T(Permutation Generation) = $O(N*N!)$

In order to check for any pair whether they are edge disjoint or not, we have to check pairwise. So the time complexity for this step is $T*(T-1)/2*O(N \log N)$.

Therefore, the total time complexity is,

$O(N*N!) + O(T^2 N \log N)$.

The time complexity for **Approach 2** is as follows:

For our first part of our solution, where we find all the unique Hamiltonian circuits, its time complexity is,

$O((N+E)*(N-1)!)$

For the second part of our solution, where we find all pairs of edge disjoint Hamiltonian circuit, its time complexity is,

$|T * (T-1)/2 *E/32|$

Therefore, total time complexity of our proposed solution is,

$O((N+E)*(N-1)!) + O(|T^2 *E/32|)$

Table 1 Comparison Between Approach 1 Vs Approach 2 on Complete Graph

No of Nodes	Approach 1(sec.)	Approach 2(sec.)
4	0.004802	0.000036
5	0.006858	0.000087
6	0.008036	0.000643
7	0.121176	0.016820
8	0.246874	0.761507
9	0.519827	47.201474

Table 2 Time in seconds to process Sparse Graph using Approach 2

No of Nodes	Approach 2(sec.)
4	0.000099
5	0.000071
6	0.000188
7	0.001871
8	0.186956
9	0.009599
10	0.000361
11	0.000056
12	0.002942
13	113.816938

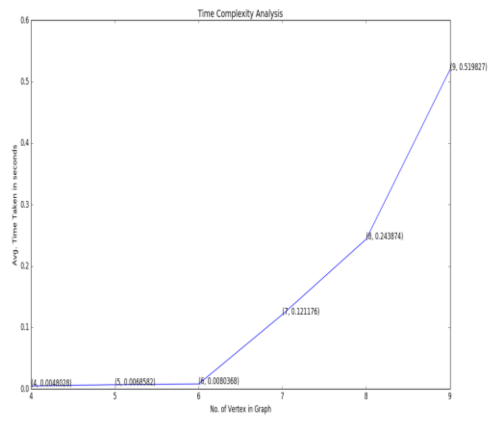

Figure 1 Time(sec.) Vs No of Nodes Using Approach 1 for Complete Graph

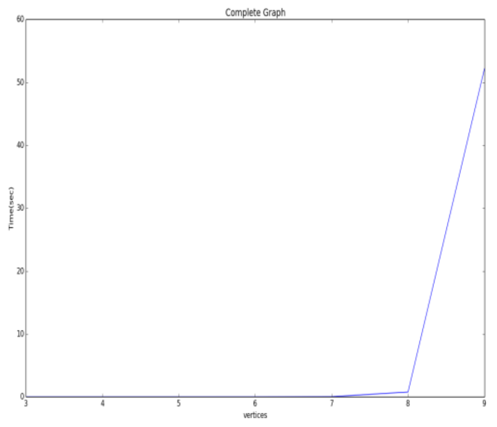

Figure 2 Time(sec.) Vs No of Nodes Using Approach 2 for Complete Graph

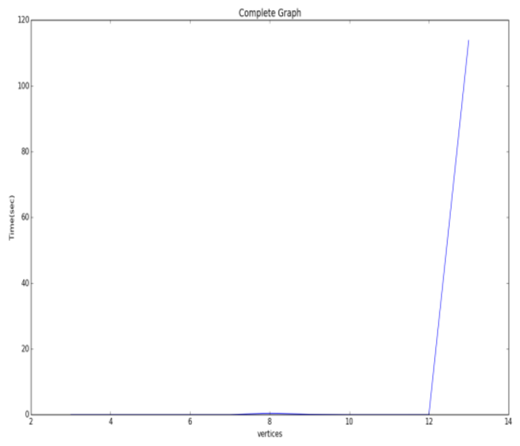

Figure 3 Time(sec.) Vs No of Nodes Using Approach 2 for Sparse Graph

IV. Conclusion

We proposed two approaches to calculate the edge disjoint pair of Hamiltonian circuits, while both methods correctly calculate the edge-disjoint pair of Hamiltonian circuits but their execution time differs greatly depending on sparsity of graph. The Approach 1 works independent of the fact that whether the graph is sparse or dense, whereas in Approach 2 execution time is inversely related to the sparsity. In Approach 1, firstly one Hamiltonian is generated then it is checked for whether it is valid or circuit or not, whereas in Approach 2 a Hamiltonian circuit is generated only if is valid. So, we can conclude that the first approach suites well for denser graph (example complete graph), while for a mixed kind of graph second approach is better.