International Journal of Research Studies in Computing
2014 April, Volume 3 Number 1, 3-12

Spanning tree with many leaves in quadratic graph

Rastogi, Akanksa
MIET, Meerut, India (Akanksharastogi.mca@gmail.com)
Singhal,Vinay
MIET, Meerut, India (Singhalmca04@gmail.com)

ISSN: 2243-772X
Online ISSN: 2243-7797

Received: 25 October 2013 **Revised**: 5 December 2013 **Accepted**: 6 December 2013
Available Online: 9 March 2014 **DOI:** 10.5861/ijrsc.2014.603

OPEN ACCESS

Abstract

Many problems arising in computer science can be viewed as a problem of, given a graph, finding a spanning tree that satisfies a specified property. Another large class of properties deals with the structure of the leaves of the spanning tree. Here, we wish to find a spanning tree with a Lower bound on maximum number of leaves (NP-complete problem). Applications of this problem include communications networks, circuit layout, and graph Theoretic problem. A polynomial algorithm for constructing full spanning trees for 4-regular (Quadratic) graph is presented. For a connected graph G let L(G) denote the maximum number of leaves in any spanning tree of G. We give a simple construction and a complete proof that if G is a connected quadratic graph on n vertices, then L (G) $\geq$ 2n/5 +2. The main idea is to count the number of "dead leaves" as the tree is being constructed.

Keywords: Maximum leaf spanning tree; 4-regular graph; (Quadratic graph) NP-complete; regular graph; dead leaves

Spanning tree with many leaves in quadratic graph

1. Introduction

The problem of finding spanning trees with many leaves has been thoroughly investigated (Bodlaender. H.L. (1989), Bonsma, P. S. (2003), Ding, G. et al. (2001), Fuzie, T. (2004), Griggs, J. R. et al. (1989), Griggs, J. R. et al. (1992), Kleitman, D. J. et al. (1991), Lemke, P. (1988), Lorys, K. et al. (2002), Lu, H. et al. (1992 & 1998), Oba, R. S. (1998)). It is known to be NP hard (Ding, G. et al. (2001)). Lu and Ravi (1996, 1998) provided 3-approximation algorithms and a 2-approximation algorithm was presented by Oba, R. S. (1998). It is known that the problem remains NP hard even if the input is restricted to d-regular graphs for any fixed $d \geq 3$.

A 7/4 approximation algorithm for cubic graphs is presented in Lorys, K. et al. (2002). Finding approximation algorithms with ratio less than 2 for d-regular graphs remains an open problem for $d \geq 4$.The NP hardness of the optimization problem leads to seek constructive proofs for related extremal problems. A constructive proof that all graphs in a particular class have spanning trees with at least m leaves becomes an algorithm to produce such a tree for graphs in this class. Let L(G) be the maximum number of leaves The value L(G) known for $d \leq 3$. Trivially $L(G) \geq 2$ for $d = 2$. Storer, J. A. (1981) proved that $L(G) \geq \lceil n/4 + 2 \rceil$ for $d = 3$. Now we are interested to find lower bound on maximum leaves for the restricted graph (Quadratic Graph).

In section 2, we describe the statements of our results for finding minimum spanning tree in 4-regular graph. In section 3, we defined the concept of dead leaves and presented the theorem. In section 4, various applications of minimum spanning tree are presented. Then in section 5, we present an algorithm that finds the minimum spanning tree for quadratic (4-regular) graph. In section 6, we show an example to implement the above algorithm and prove that the algorithm works well & produce minimum spanning tree possible. Finally in section 7, we present some open problems for future work.

2. Statement of results

We consider the problem of finding spanning trees in given graphs that contain many leaves (degree one vertices). All graphs are assumed to be simple (undirected, no loops or multiple edges). If G is a connected graph, let L (G) denote the maximum number of leaves in any spanning tree of G. We are interested here in L (G) for quadratic (4-regular) graphs G.

Suppose T is a spanning tree for a connected quadratic graph G on n vertices. Necessarily, n is odd. Let di denote the number of vertices of degree i in T, i = 1, 2, 3, 4. Then the number of vertices $n = d1 + d2 + d3 + d4$, while the sum of the degrees $2n - 2 = d1 + 2d2 + 3d3 + 4d4$. Consequently, L (G) is maximized over such graphs G when it contains T with d2 and d3 as small as possible, that is, $d2 = d3 = 0$. Hence, $L(G) \leq 2n/3 + c$ where $0 \leq c \leq 2$. This bound is attained for all n by taking the caterpillar in which (n/3) vertices form a path, and two leg (leaf vertex) is joined to each interior vertex of the path, while two legs are joined to each end of the path. This is the desired tree T, which can be embedded in a suitable graph G by adding a cycle through the leaves of T (see Figure 1).

The more interesting question then is to minimize L (G) i. e., to obtain a lower bound on L (G) over all such graphs G in terms of n.

Theorem 1: If G is a connected quadratic graph on n vertices, then L (G) >= 2n/5 + 2

This bound is best possible for all (odd) n. For example, if n= 0 (mod 5), then take G to be a circular "necklace" of n/5 "beads", where each bead is K5 - e (meaning K5 with one edge deleted).So in this situation, we will get (2n/5) +2 leaves. If n=4(mod 5) then we get n - 4/5 beads of K5 - e and 4 vertices (i.e. K4) .So we

will get (2n/5) + 1 + 2 = (2n/5) + 3.Hence proved as claimed.

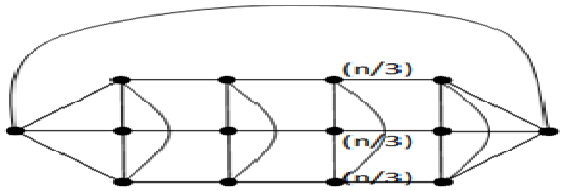


Figure 1: 4-regular graph with 14 vertices

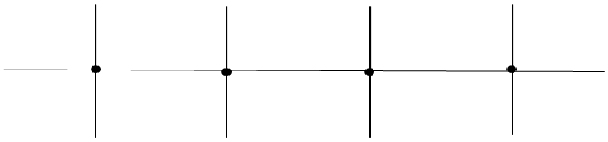


Figure 2: A Tree T of Quadratic Graph G with L (G) = (2n/3) + c

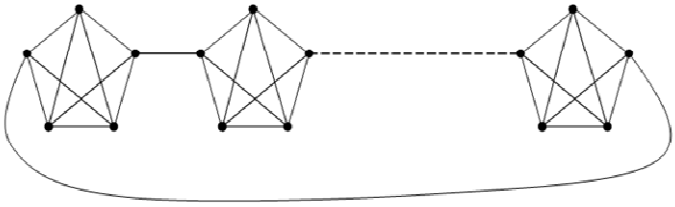


Figure 3: Extremal Graph for Theorem1

If one permits vertices of lower degree than 3 in G, then L(G) can drop dramatically, e.g., L(Pn) = 2, where Pn is a path on n vertices. Storer, J. A. (1981) works with graphs of maximum degree 3, rather than our more restricted setting of quadratic graphs. We consider the effect of introducing the stronger connectivity condition on G that it be 4-connected, i.e., the removal of any three vertices does not disconnect it. Such graphs cannot contain K5 – e, and our main result applies to this more general situation.

Our method of proof is to begin by finding a small tree in G with many leaves and then to grow the tree by adding several vertices in such a way that the number of leaves always grows enough to keep satisfying the theorem. A central idea is to keep track of number of dead leaf as well as the number of leaves. A dead leaf is a leaf in the tree under construction, all of whose neighbors are already in the tree. Once a leaf is dead, it remains a leaf during the rest of the construction. To illustrate the power of this approach, we offer a new proof of Theorem 1 in the next section.

3. Dead leaves and theorem 1

A natural approach to proving a result such as Theorem 1 is as follows. Starting with a small tree in G that is nice, one tries to add on some number of vertices N in such a way that the tree gains at least 2N/5 leaves. If this can always be done, then the tree eventually constructed must have at least roughly 2n/5 leaves. There is one case that is especially difficult with this approach. Consider a vertex v outside the tree that has all four of its neighbors being leaves in the tree. Then adding v causes no gain in the number of leaves. What can we do if there is many such vertices v? We do gain something by adding v, which is that v itself would have all of its edges inside the tree and that several edges involving neighbors of v are likewise accounted for. So v itself and perhaps some of its neighbors become dead leaves. At any given stage in the construction, it is obvious that the number of dead leaves D is at most the number of leaves L. So if we aim to show that at the end, $L \geq (2n/5) + c$,

for some c, or equivalently, $5L \geq 2n + 5c$, then it suffices to show that $aL + bD \geq 2n + 5c$ ($aL + bD - 2n \geq 5c$) for some choice of a, $b \geq 0$ such that $a + b = 5$. We start off by constructing a tree on N vertices with L leaves and D dead leaves such that $\Delta (N, L, D) \geq 5c$, where $\Delta (N, L, D) = aL + bD - 2N$. It then suffices to show that for any constructed tree that does not yet span G there exists some set of vertices, say N of them, that can be added in such a way as to increase the number of leaves by L and the number of dead leaves by D, where $\Delta (N, L, D) \geq 0$.

We are ready to prove Theorem 1. Besides illustrating the value of the dead leaves approach, it may be useful to have a complete proof written down. Storer's approach is to start with a breadth – first spanning tree and then modify it to gain leaves. His approach is natural, but the proof is merely sketched, and we were unable to work out all of the details.

Proof of Theorem 1: Let G be a connected quadratic graph on n vertices.

Since n is odd and L(G) is integral, $L(G) \geq (2n/5) + c$ if and only if $L(G) > (2n/5) + (6/5)$. For our dead leaves approach, we seek a and b, $a + b = 5$, so that $aL + bD - 2N > 0$. It turns out that $a = 4.5$ and $b = 0.5$ are suitable choices, so we assume these values hereafter. Concerning the notation, vertices shall be denoted by lower case letters and $v \sim w$ means v and w are adjacent while $v \not\sim w$ means they are not adjacent. If a vertex v is outside a tree T but adjacent to some vertex in T, we write $v \sim T$. We denote the edge between two vertices v and w by vw.

To start off, select any vertex $v \in G$, and let $w, x, y, z \sim v$. Begin the tree T by taking the subsequent stages we show that a tree T that only partially spans G can be vx extended by some amount such that $\Delta > 0$. First suppose there exists $v \in T$, $v \sim x, y, z \in T$ such that $v \sim w$ T. Then add vw to T, and it gains one leaf, so we have $\Delta(N, L, D) = aL + bD - 2N = \Delta(1, 1, 0) = 4.5*1 + 0.5*0 - 2*1 = 2.5 > 0$.We can have three possibilities while we extend tree-(a) Vertex v is once adjacent to T.(b) Vertex v is twice adjacent to T.(c) Vertex v is thrice adjacent to T In case (a) we can have many possible cases some cases are: Let $v \sim t \in T$ and $v \sim x, y, z$ T, then by adding tv, vx, vy, vz we gain two leaves $\Delta(4, 3, 0) = 5.5 \geq 0$, now we can assume that y is splits into three parts say a, b, c such that $a \sim b$, $b \sim c$, and $a \sim c$, $b \sim T$, $c \sim z$, then by adding vx, vy, vz, yb, ya, yc, tv we gain three leaves and two dead leaves, $\Delta(7, 3, 2) \geq 0$. In case (b) ,we describe some cases that are: if we have vertex $v \sim T$ such that $v \sim t1, t2 \in T$ and $v \sim x, y$ T then by adding vx, vy we gain 2 leaves, $\Delta(3, 2, 0) = 3 \geq 0$. Again let vertices x & y are join at the single vertex say z, i.e. $x \sim y$, $x \sim z$, $x \sim t1$, $y \sim z$, then by adding t1x, xv, xy, xz we have $\Delta(4, 2, 1) = 1.5 \geq 0$.

In case (c), we describe some cases that are: if we have vertex v T such that $v \sim t1, t2, t3 \in T$ and $v \sim x$, T then by adding vx we gain 1 leaf, $\Delta(2, 1, 0) = 0.5 \geq 0$.Thus in every case we can add to T so that $\Delta \geq 0$.By induction, we can eventually add (2n/5)+2.

The proof above is essentially a polynomial-time algorithm for constructing a tree with at least (2n/5) + 2 leaves.

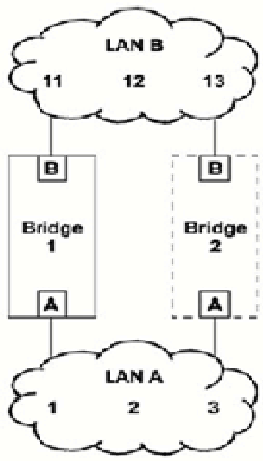
4. Applications of spanning tree with many leaves

Spanning tree concept is applied in various areas, some important application area of this concept are given below:

A. Industrial Automation:

In an industrial automation application that relies heavily on the health of the Ethernet network that attaches all the controllers and computer together, a concern exists about what would happen if the network fails. If the result is the loss of production or loss of processed batcher endangerment of people and equipment, redundancy schemes are examined .Since cable failure is the most likely mishap, cable redundancy is suggested by configuring the network in either a ring or by carrying parallel branches. If one of the segments is lost, then communication will continue down a parallel path or around the unbroken portion of the ring. Cable redundancy

introduces loops in the topology and, as we will see, these loops must be disabled. An industrial automation user may want loops to guard against a primary cable failure while anoffic automation user may want to guard against an inadvertent loop.



Looking at the same Figure 1, Bridge 2 is now added to parallel Bridge 1. This gives us a redundant path, but it also creates a loop with the following adverse results. When Station 1 initiates a message to Station 11, this message is forwarded by Bridge 1 and appears on LAN B. Bridge 2 interprets this message as originating on LAN B so it forwards the message to LAN A while incorrectly noting that Station 1 is located on LAN B. When Station 1 initiates a second message, Bridge 2 interprets this action as if Station 1 has now moved to LAN A from LAN B and resets its filtering database accordingly. Now assume that Station 1 sends out a broadcast message. Bridge 1 will forward the message to LAN B. Bridge 2 will observe the message on LAN B and forward it to LAN A. Bridge 1 will observe this message on LAN A as a new message and forward it to LAN B again initiating an endless cycle, totally consuming the bandwidth of both bridges and rendering both LANs To maintain the integrity of our network, we must guard against the formation of loops.

B. Spanning Tree Topology:

To avoid loops we need a tree topology consisting of a root, a succession of branches and then leaves. The leaves represent end stations, and there is one and only one path from a leaf to another leaf. Therefore, the tree is free of loops that can cause havoc in a network. The other requirement is that all leaves are connected. There are no isolated segments. Another term for this topology is distributed star. Within our tree structure will be a series of bridges used to connect the branches and leaves. There are two types. The root bridge is the main one of interest because it has a special assignment and there is only one within a network. The other bridges (that are to be used) are all designated bridges and there could be many within the network. To have a tree topology, you need bridges with more than two ports.

C. Spanning tree construction in wireless ad hoc networks:

The importance of spanning trees lies in the fact that they help removing cycles and establishing short paths between any node and the rest of the nodes in the network. This work addressed the problem of constructing such a structure with a protocol that tolerates faults and adapts itself to dynamic topology changes that often occur in mobile ad hoc networks. This approach uses parallel random walks. These multiple random walks collapse into a final one that defines the final territory and provides the random spanning tree. The advantages of our scheme are that it generates a random spanning tree structure that is less sensitive to failures compared to a deterministically predetermined spanning tree. The solution is adaptive and deals with topology changes, and is therefore suitable for ad hoc wireless networks. The technique is self-stabilizing, resilient to transient faults of wireless networks, inherently parallel and uses the whole power of the distributed resources to achieve a good

average running time.

D. Landscape connectivity:

A graph-theoretic perspective: Ecologists are familiar with two data structures commonly used to represent landscapes. Vector-based maps delineate land cover types as polygons, while raster lattices represent the landscape as a grid. Here we adopt a third lattice data structure, the graph. A graph represents a landscape as a set of nodes (e.g., habitat patches) connected to some degree by edges that join pairs of nodes functionally (e.g., via dispersal). Graph theory is well developed in other fields, including geography (transportation networks, routing applications, sitting problems) and computer science (circuitry and network optimization). We present an overview of basic elements of graph theory as it might be applied to issues of connectivity in heterogeneous landscapes, focusing especially on applications of meta population theory in conservation biology. We develop a general set of analyses using a hypothetical landscape mosaic of habitat patches in a non-habitat matrix. Our results suggest that a simple graph construct, the minimum spanning tree, can serve as a powerful guide to decisions about the relative importance of individual patches to overall landscape connectivity.

E. Spanning tree method for link state aggregation in large communication networks:

We consider a communication network in which dynamic routing is used for establishing connections that support information transfer between end-users. Link state information is exchanged and maintained up-to-date among network nodes for path computation and network resource allocation. When the population of users is large, the amount of link state information can be overwhelming. A common solution is to use a hierarchical structure. We present a method for aggregating link state information in a hierarchical network. We assume that each link state parameter associated with a link is symmetrical in both directions of the link. The key idea for the method is to first reduce the original sub network topology to a full-mesh representation that consists of a logical link for each pair of border nodes in the sub network, and then encode the link state information associated with the full-mesh representation with an appropriate spanning tree.

F. Spanning-tree based coverage of continuous areas by a mobile robot:

Considers the problem of covering a continuous planar area by a square-shaped tool attached to a mobile robot. Using a tool-based approximation of the work-area, we present an algorithm that covers every point of the approximate area for tasks such as floor cleaning, lawn mowing, and field defining. The algorithm, called Spanning Tree Covering (STC), subdivides the work-area into disjoint cells corresponding to the square-shaped tool, then follows a spanning tree of the graph induced by the cells, while covering every point precisely once. We present and analyze three versions of the STC algorithm. The first version is off-line, where the robot has perfect a priori knowledge of its environment. The off-line STC algorithm computes an optimal covering path in linear time O (N), where N is the number of cells comprising the approximate area. The second version of STC is on-line, where the robot uses its sensors to detect obstacles and construct a spanning tree of the environment while covering the work-area. The on-line STC algorithm completes an optimal covering path in time O (N), but requires O (N) memory for its implementation. The third version of STC is “ant”-like. In this version, too, the robot has no a priori knowledge of the environment, but it may leave pheromone-like markers during the coverage process.

5. Algorithm for finding spanning tree with many leaves in quadratic graph

In this section, we present an algorithm that find spanning tree with many leaves in 4-regular graph.

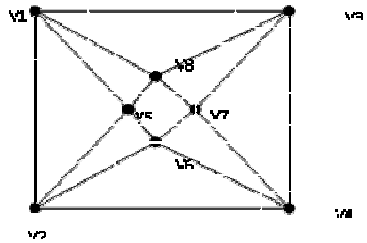
Firstly start with tree T, finding the vertices vi ~ T, added these vertices to T if that vertices satisfy the inequality aL + bD - 2N >= 0 (i.e. described in previous section). As we know tree is a connected graph, in the last steps of this algorithm we connect T and the isolated vertices (which does not satisfy inequality) and return the connected spanning tree TA. Concerning the notation, set of vertices of a graph G is denoted by V[G], set of

edges of a graph is denoted by E[G], Queue in which we insert vertices or removing the vertices, is denoted by Q. N is the number of vertices to be added to T to gain L leaves and D dead leaves. Enqueue () is a function that insert a vertex into Q. Dequeue () is function for retrieving the element from the Q.V is the set of vertices (does not belongs to T) adjacent to T. V = {v1, v2…….vi} where $1 \leq i \leq 4$ and n[V] is the number of vertices in set V.

Initialization:
a=4.5
b=0.5
N=Number of vertices added to extend the tree
L=Number of leaves gained
D=Number of dead leaves
Step 1: T (tree) ←ϕ
Step 2: Select any vertex S from V[G]
V[G] = V[G] - S
V[T] = {S}
Enqueue(Q, S)
Step 3: Repeat V[G] ≠ ϕ and Q ≠ ϕ
u= Dequeue(Q)
Find V = v € (Adj(u) ∉ T)
If(n[V] > 0 and aL + bD – 2N > 0)
E[T] = E[I] U {(u, v_i) such that all v_i € V}
V[T] = V[T] U {v_i € V}
V[G] = V[G] - v_i for all i
Enqueue(Q, v_i) for all i.
(End of step 3)
Step 4: Build T_A by connecting T and all nodes in V[G] - V[T]
Step 5: return T_A

6. Sample run of the above algorithm on 4 –Antiprism Graph

Now we take antiprism graph as a sample to run this algorithm.



Step 1: T ←ϕ V[G]={V1,V2,V3,V4,V5,V6,V7,V8}
Step 2: Select V1 from V[G]
V[T] = {V1}
Q = V1
Step 3: Repeat steps

Iteration-1
u = V1 //retrieving vertex from the Queue
V = {V2, V5, V3, V8} //adjacent vertices of u that does not belongs to T
N = 4, L = 4, D = 0
Compute aL + bD – 2N = 4.5*4 + 0.5*0 – 2*4 which is greater than 0
E[T]={(V1, V2),(V1, V5),(V1, V3),(V1, V8)}

V[T]={ V1, V2, V3, V5, V8}
V[G]={V4, V6, V7}
Q=V2, V3, V5, V8

Iteration-2
u = V2
V = {V4, V6}
N = 2, L = 2, D = 0
Which satisfies inequality aL + bD – 2N > 0
E[T]={(V1, V2),(V1, V5),(V1, V3),(V1, V8) ,(V2, V4),(V2, V6)}
V[T]={ V1, V2, V3, V5, V8, V4, V6}
V[G]={V7}
Q=V3, V5, V8, V4, V6

Iteration-3
u = V3
V = {V7}
N = 1, D = 1, L = 0
Which does not satisfy inequality aL + bD – 2N > 0

Iteration-4
u = V5
V = ϕ
N = 0, D = 0, L = 0
Which does not satisfy inequality aL + bD – 2N > 0

Iteration-5
u = V8
V = {V7}
N = 1, D = 1, L = 0
Which does not satisfy inequality aL + bD – 2N > 0

Iteration-6
u = V4
V = {V7}
N = 1, D = 1, L = 0
Which does not satisfy inequality aL + bD – 2N > 0

Iteration-6
u = V6
V = {V7}
N = 1, D = 1, L = 0
Which does not satisfy inequality aL + bD – 2N > 0
Now Q is empty // Exit from the loop

Step 4: Build TA by connecting T and node V7 (V[G] - V [T])

Step 5: return TA
In this way, we got n (number of leaves) = 5 which satisfy our theorem

i.e. $L(G) \geq 2n/5 + 2 = 2*8/5 + 2 = 5$

7. Conclusion and open problem

In this paper we presented a polynomial algorithm for finding lower bound on maximum number of leaf which is the NP-Complete problem. But we find the solution for only restricted graphs (i.e. 4-regular graph). One can find polynomial algorithm for d>4, where d is the degree of the vertices. In future, we can sharpen the upper and lower bounds for L(G) for any $d \geq 4$? We can find solution for the d-regular planar graph.

8. References:

Bodlaender, H. L. (1989). On linear time minor tests and depth first search. In K. Frank, H. A. Dehne, J. R. Sack, & N. Santoro (Eds.), *WADS, Lecture Notes in Computer Science* (vol. 382, pp. 577–590). Springer Press.

Bonsma P. S., Bruggemann T., & Woeginger G. J. (2003). A faster fpt algorithm for finding spanning trees with many leaves. In B. Rovan & P. Vojtas (Eds.), MFCS, *Lecture Notes in Computer Science* (vol. 2747, pp. 259–268). Springer Press.

Caro, Y., West, D. B., & Yuster, R. (2000). Connected domination and spanning trees with many leaves. *SIAM Journal on Discrete Mathematics, 13*(2), 202–211. http://dx.doi.org/10.1137/S0895480199353780

Ding, G., Johnson, T., & Seymour, P. (2001). Spanning trees with many leaves. *Journal of Graph Theory, 37*, 189–197. http://dx.doi.org/10.1002/jgt.1013

Fujie, T. (2004). The maximum-leaf spanning tree problem: Formulations and facets. *Networks: An International Journal, 43*(4), 212–223. http://dx.doi.org/10.1002/net.20001

Griggs, J. R., Kleitman, D. J., & Shastri, A. (1989). Spanning trees with many leaves in cubic graphs. *Journal of Graph Theory, 13*(6), 669–695. http://dx.doi.org/10.1002/jgt.3190130604

Griggs, J. R., & Wu, M. (1992). Spanning trees in graphs of minimum degree 4 or 5. *Discrete Mathematics, 104*(2), 167–183. http://dx.doi.org/10.1016/0012-365X(92)90331-9

Kleitman, D. J., & West, D. B. (1991). Spanning trees with many leaves. *SIAM Journal on Discrete Mathematics, 4*(1), 99–106. http://dx.doi.org/10.1137/0404010

Lemke, P. (1988). *The maximum leaf spanning tree problem for cubic graphs is NP- complete*. Technical Report IMA Preprint Series # 428, University of Minnesota.

Singhal, V. K., & Rastogi, A. (2012). Strong fault-tolerant conflict-free coloring (Strong FTCFColoring) for intervals. *International Journal of Research Science & Computing, 1*(2). http://dx.doi.org/10.5861/ijrsc.2012.137

Lorys, K., & Zwozniak, G. (2002). Approximation algorithm for the maximum leaf spanning tree problem for cubic graphs. In R. H. Mohring & R. Raman (Eds.), *ESA, Lecture Notes in Computer Science* (vol. 2461, pp. 686–697). Springer Press.

Lu, H., & Ravi, R. (1992). *The power of local optimizations: Approximation algorithms for maximum-leaf spanning tree*. In Proceedings Thirtieth Annual Allerton Conference on Communication, Control and Computing. CS-96-05.

Lu, H., & Ravi, R. (1998). Approximating Maximum Leaf Spanning Trees in Almost Linear Time. *Journal of Algorithms, 29*(1), 132–141. http://dx.doi.org/10.1006/jagm.1998.0944

Oba, R. S. (1998). 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In G. Bilardi, G. F. Italiano, A. Pietracaprina, & G. Pucci (Eds.), ESA, *Lecture Notes in Computer Science* (vol. 1461, pp. 441–452). Springer Press.

Storer, J.A. (1981). Constructing full spanning trees for cubic graphs. *Information Processing Letters, 13*(1), 8–11. http://dx.doi.org/10.1016/0020-0190(81)90141-1