An approximation for the Vertex Cover Problem

Ted Gress

All vertices have at least one incident edge. Sometimes the edges for more than one vertex overlap. It is the overlapping that is the real core of this problem.

Given three vertices a,b, and c, and three connecting edges, circling a will activate edge ab, and ac. Go to vertex b activate ab, but it is already activated so we skip this vertex. Go to vertex c, activate edges bc and ac and there is no overlap.

This is at least an approximation algorithm.

For more thoroughness we have to iterate this process over every vertex and then verify which is the smallest vertex cover, which is trivial. The iteration would add another n term, resulting in O(n2k) or for worst case scenarios, O(n3).

**Proof**:

By Induction.

Base Case:

a

Easiest with a diagram.

f

e

b

c

g

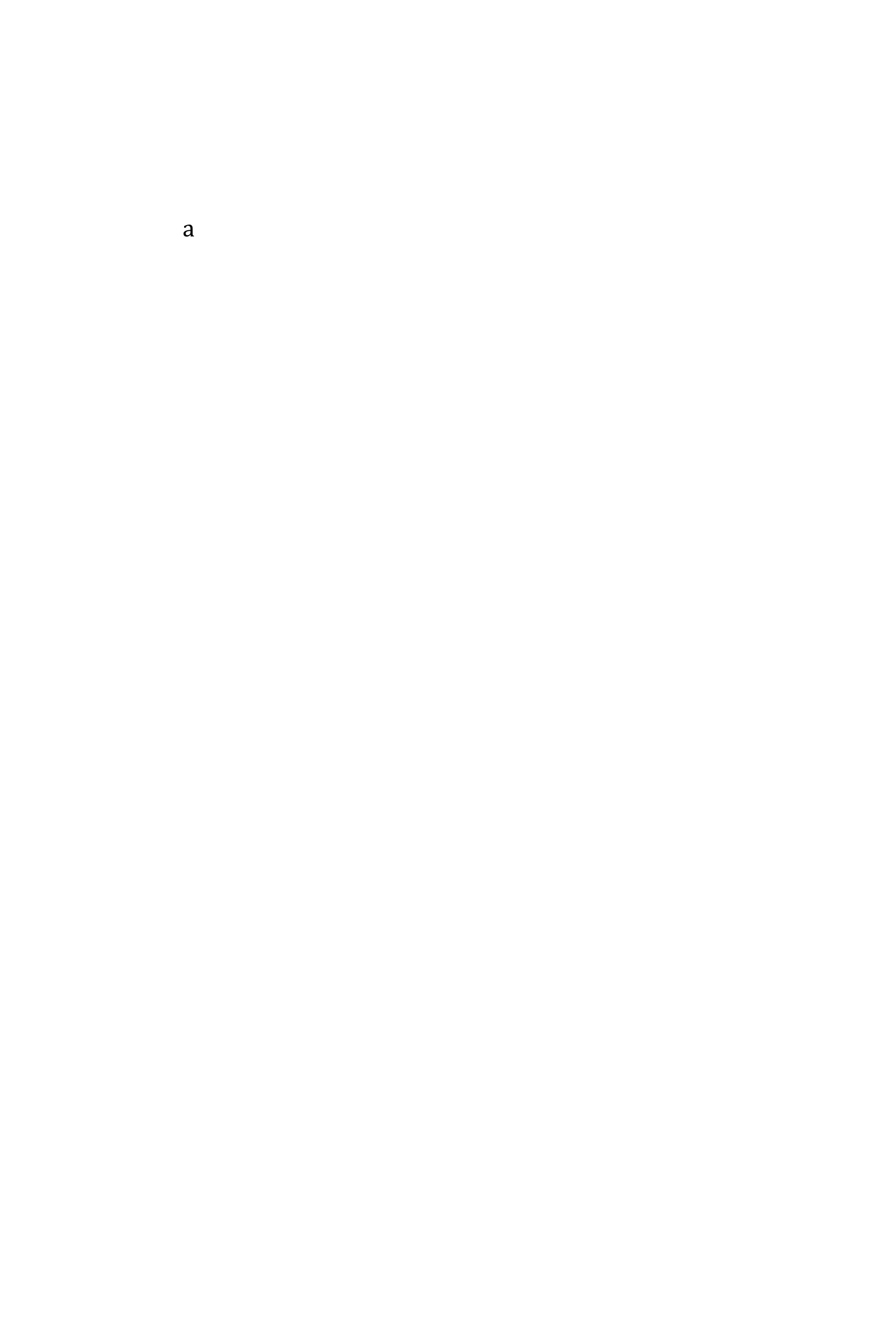
A triangle graph, a b and c vertices and e, f, and g edges. a is incident to e and f b is incident to e and g, and c is incident to g and f. Start with vertex a. Check to see if edge e is activated. Its not so, mark vertex a as activated and edge e as activated and continue to vertex b. b is incident to edge e and edge g. E is already activated so activate g. Mark b as activated. Proceed to vertex c. g is already activated so skip that edge. Vertex a has already been traversed and is activated so no need to mark f.

Repeat graph for every other vertex as a start point. Return the minimum set.

Higher Case:

Because of the complexity of this theorem, I want to demonstrate a graph one dimension higher.

e

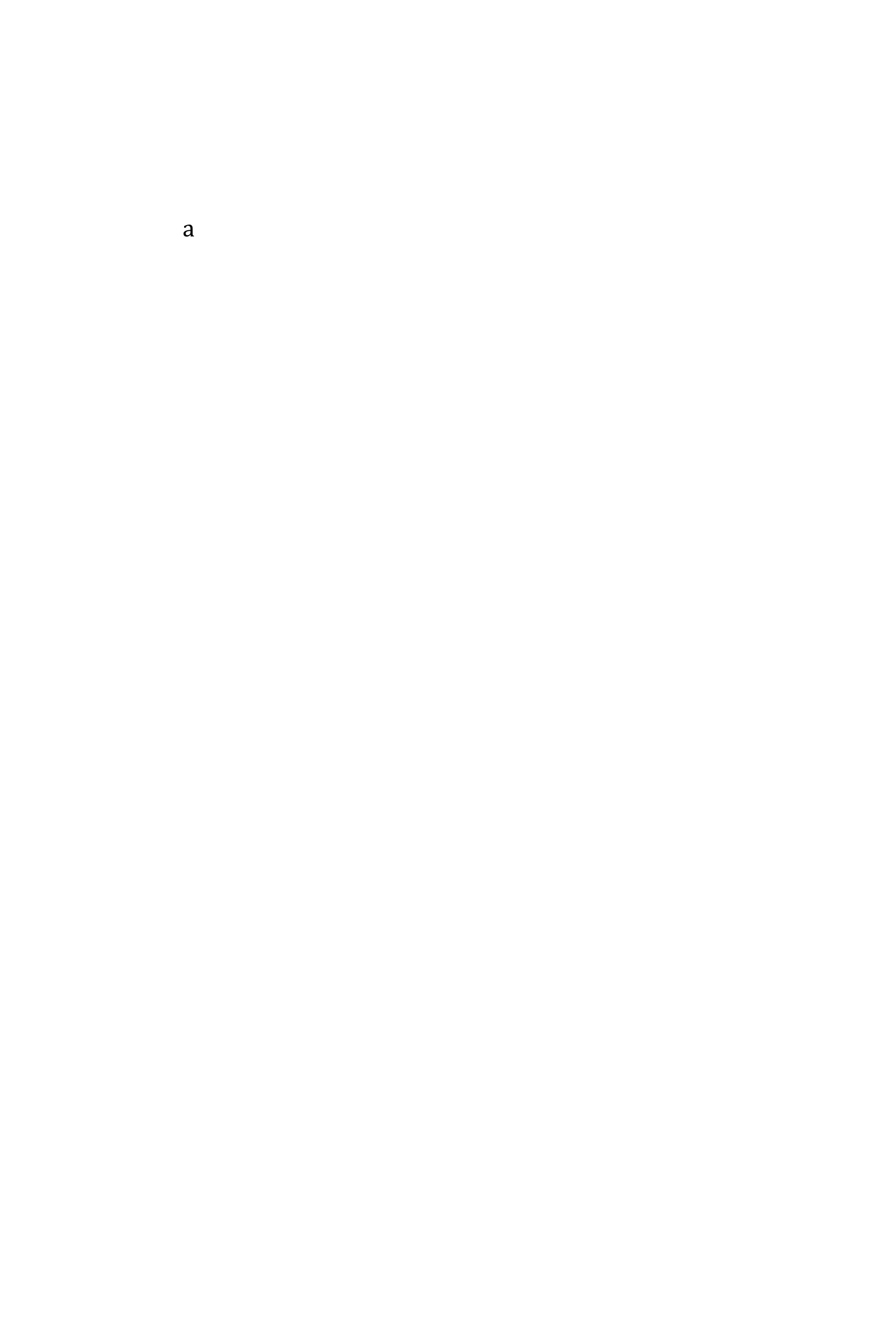
b

a

f

h

c

g

d

We begin with an arbitrary vertex a. h and e are adjacent so we activate them and activate a. And move on to b. e is already activated but f is not so we activate f. We move on to c. f is activated but g is not. Activate c. But we do not need to have g marked, as we already have a minimum cover. Finally we tour to d. h has already been activated and c has already been activated, so we can’t activate g.

Part of the problem is knowing when to stop. Here c stops because d has already been activated, along with g.

One more problem, then the inductive proof.

a

b

c

d

e

f

**j**

**k**

**l**

**m**

**n**

**p**

Ok. Starting at an arbitrary vertex a. a activates itself and edge i. Next is vertex b. b activates edges k and m. Next is vertex e. m is already activated so e activates itself and edge p. Vertex f is adjacent to both p and n. p has already been activated so it activates edge n and itself. Next is vertex c. n is already activated so we skip that. K and n are already activated so we activate the last, l.

Note that in this configuration the vertex cover was the whole graph. We can do better than that.

For a more sparse vertex cover we can make it so that no edges can be created on nodes that are already activated.

So let’s try again.

Starting with a, we activate the vertices a, b, and the edge i. We skip b since it has already been activated and go to e. The edge m can’t be activated because it is adjacent to b. But the edge p can be. So p and e are activated. We move on to f and we can’t activate p since it was already activated, so we activate f and c and edge n. We can’t activate k since b is already activated. So we activate l and d and finish with a much sparser graph. But is it accurate?

i, p,n, l are the edges we activated. It looks like a minimum cover to me. Now for a proof. This is the tough part.

Proof by Induction

Inductive Case:

Let = number of vertices

Let = number of edges

Let = max amount of out connections per vertex

Let G = the graph in question

Let E = the set of edges

Let V = the set of vertices

Let v be any vertex

Let e be any edge

Let the number of covered vertices

Let = the number of covered edges

Let the minimum vertex cover be the set min(k()) where k() is the covering function.

Covering function k for any subgraph:

k()

Given a graph of |G| = vertices, perform the covering function for all vertices.

Proposition:  
Since is the number of vertices, and is the number of edges, the most times that k() must run at maximum is .

Must mathematically prove the k() cover function

First let set boundaries.

> Number of vertices will always be smaller than the number of edges

For every v k(v) is run. Let m be a counter. If k() activates an edge e, then

V=V-v and E=E-e. The number of edges evaluated increases the m counter. Once m = , all edges have been accounted for. Not all vertices have been covered, but that is ok, we do not want them to be.

Assume we want to add a vertex to the graph G above. Let’s add two.

Not counting all of the operations required in adding a vertex, let us assume it is “just there”.

Then

Let the two new vertices be labeled . Then k(, and k( is run. Neither of the vertices are already activated. Let us assume two cases. For , the vertices and edges are already activated. For , there is a free vertex and edge. For both the m counter is increased. k(, result is to simply skip on to the next vertex. There are no more so we are done. k( activates the new vertices and edges and is done.

So we have vertices and edges being covered by k(). The big question is, how do we know we have the minimum vertex cover. I mean, that is the NP-Complete problem.

Earlier we had mentioned a function, min(k()). So, what ‘s the function min() look like?

Let the number of covered vertices

Let = the number of covered edges

min() look like this:

min(min(min(e)))…= minxe)

min() is recursive. It takes the minimum cover and tries to reduce it to another minimum cover, that reduces again, and so on until it arrives at a base case.

So what does min(k()) mean? It means that we are choosing the minimum choice for the covering function.

So without the explanations, here is the proof:

Base Case:

V = { a, b, c }

E = { e, f, g }

k(a) covers a; e

k(b) covers b; g

k(c) covers c; skip

min(k(a))

min(k(b))

min(k(c))

Inductive Case:

V = { V0, V1, V2, …, V }

E = { E0, E1, E2, …, E }

- descriptor operator, stands for iterates over, similar to a for loop

minx {

n

k(vi) covers vi, vi+1 , ei}

i=0

}