**UNIVERSIDAD DEL NORTE**

DIVISIÓN DE INGENIERÍA

PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

ALGORITMOS Y COMPLEJIDAD

CÓDIGO: 200055832

NOMBRE: Luis Sebastian Caicedo Pimienta

FECHA : 04 / 02 / 2016 – 29 / 02 / 2016

1. Describa la formulación del problema NP-Hard Domatic Number, por enunciado y por parámetros.

Parametros: un grafo G= (V,E), entero positivo K ≤ |V|, D vértices dominantes de V.

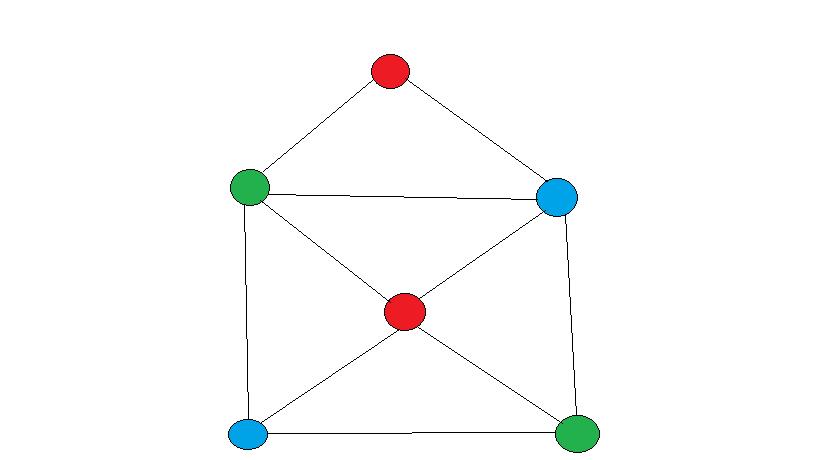
Enunciado: Numero domatico de un grafo es el número de conjuntos de vértices disjuntos K de un grafo G=(V,E) donde V puede ser dividido en K conjuntos disjuntos o tal que cada conjunto es un conjunto dominante de G.

1. Diseñe dos casos significativos propios de tu problema NP-H gráficamente y explíquelos.

* Upper Bounds

En un grafo G, sea el grado mínimo de los vértices d, K es como mucho d+1, ejemplo:

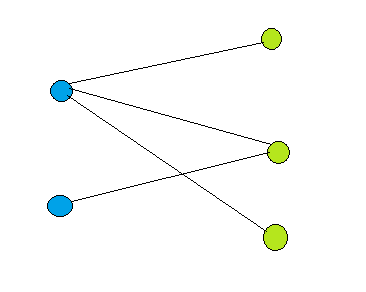
Ejemplo:



El número domatico del grafo anterior es 3, ya que se pueden formar tres grupos disjuntos de vértices o en particiones domaticas.

Ejemplo:

Si en un grafo no hay vértices aislados, en número de conjuntos de vértices disjuntos K como mucho es 2, esto se ve claramente en los grafos 2-Colored o bipartidos, ejemplo:



1. Investigar históricamente el avance del problema NP-Hard (Domatic Niumber) con relación a la resolución del problema teniendo en cuenta el tiempo y el espacio( utilice el referenciado ZOTERO con la metodología IEEE)

Se han trabajado este problema desde la década de los 50, sin embargo, no se ha llegado a un algoritmo eficiente que permita resolver este problema de la mejor manera. Sin embargo, se han presentado posibles soluciones. En busca de mayor eficiencia en la resolución del problema sin mucho éxito.

Las soluciones computarizadas para determinar el número domatico aun están siendo estudiadas matemáticamente, tal como se muestran en los artículos referenciados en este documento.

No obstante, pude desarrollar la aplicación que permite determinar el número domatico de un grafo ya sea conexo o no conexo.



domaticNumber(){

boolean sw=false;

boolean sw2=false

ArrayList adya

MQ(!sw2){

i=0;

sw=false;

Restablecer verG; (No reiniciar en cero, se debe volver a crear)

Limpiar adya;

MQ(sw = false && i < vertices.size){

SI(!adya.contains(vertices.get(i)) && !verG.contains(vertices.get(i)) && !verifEstaEnGrupo(vertices.get(i))){

Para(j = 0; j < aristas.size; j++)

SI(aristas.get(j).substring(0,1).equals(vertices.get(i))){

adya.add(aristas.get(j).substring(1,2))

SINO

SI(aristas.get(j).substring(1,2).equals(vertices.get(i))){

adya.add(aristas.get(j).substring(0,1))

FINSI

Fin para

verG.add(vertices.get(i))

sw=verifEsGrupo(verG, adya);

SINO

SI (verticesSobrantes(verG) == 1 && verticeSobra(vertices.get(i)) && adya.contains(vertices.get(i))){

verG.add(vertices.get(i));

sw=verifEsGrupo(verG, adya);

FIN SI

I++

FIN MQ

Grupo g= new Grupo()

SI (sw == true) {

g.setVertices(verG)

grupos.add(g)

sw2=verifFin(verG);

SINO

SI (verticesSobrantes(verG) == 0)

sw2=true

FIN SI

FIN SI

FIN MQ

FIN

1. Diseñe un programa en Java con la implementación de la solución.
2. Escriba la documentación del programa de java añadiendo comentarios y marcándolos en azul para mayor comprensión.

Los datos de entrada deben ser:

* Para los vértices, se deben digitar cada vértice o nodo del grafo separados entre espacios; ejemplo a b c d e
* Para las aristas, se deben digitar las aristas separadas entre esapcios; ejemplo ab cd ea

//CLASE VENTANA (Es un jFrame)

//VARIABLES GLOBALES

ArrayList<Grupo> grupos =new ArrayList(); //Arraylist de tipo grupo que guarda todos los grupos encontrados.

ArrayList<String> vertices = new ArrayList(); //ArrayList de todos los vértices del grafo

ArrayList<String> aristas= new ArrayList(); //ArrayList de todas las aristas del Grafo.

//Subrutina que encuentra el numero domatico y los conjuntos dominantes del grafo

public void domaticNumber(){

boolean sw=false; //Swiche que controla si los vértices seleccionados hasta el momento es o no un grupo

boolean sw2=false; //Swiche que controla si ya se han encontrado o no todos los grupos

int i=0;

ArrayList<String> adya= new ArrayList(); //ArrayList que guarda los vértices adyacentes al vertice evaluado actualmente

while(!sw2){ //MQ no se hayan encontrado todos los grupos dominantes posibles

i=0;

sw=false;

ArrayList<String> verG= new ArrayList(); //Lista con los verices dominantes del grupo que se esta armando o creando

adya.clear(); //se limpia la lista de vertices adyacentes

while(sw==false && i<vertices.size()){ //MQ que los vértices agrupado actualemente no sean un grupo o no se hayan verficado todos los vertices

if(!adya.contains(vertices.get(i)) && !verG.contains(vertices.get(i)) && !verifEstaEnGrupo(vertices.get(i))){ // verigica que el vertice a tomar no haya sido seleciionado antes

for (int j = 0; j < aristas.size(); j++) {

if(aristas.get(j).substring(0,1).equals(vertices.get(i))){ //si el primer vertice de la arista es igual al vertice actaul

adya.add(aristas.get(j).substring(1,2)); //Agrega a la lista de vertices adyacentes el vertice dos de la arista actual

}else

if(aristas.get(j).substring(1,2).equals(vertices.get(i))){ //Si el seundo vertice de la arista es igual al vertice actual

adya.add(aristas.get(j).substring(0,1)); //Agrega a la lista de vertices adyacentes el vertice uno de la arista

}

}

verG.add(vertices.get(i)); //Se añade el vertice actual que no ha sido escogido antes y hacen parte del conjunto dominante

sw=verifEsGrupo(verG, adya); //Se verifica que los vertices seleccionados hasta el momento conforman o no un grupo dominante del grafo

}

else

if (verticesSobrantes(verG) == 1 && verticeSobra(vertices.get(i)) && adya.contains(vertices.get(i))){ //Vertice que fue seleccionado antes pero es el ultimo sobrante

verG.add(vertices.get(i)); //Se agrega al conjunto de vertices que no han sido escogidos antes y hacen parte del conjunto dominante

sw=verifEsGrupo(verG, adya); //Se verifica que los vertices seleccionados formen o no un grupo dominante

}

i++;//Aumento de i la variable que controla el vertice a evaluar.

}

Grupo g= new Grupo(); //Se crea una instancia de Grupo

if (sw == true) {

g.setVertices(verG); //Al parametro de vertices que conforman el grupo de la instancia g se le asigna el grupo creado actualmente

grupos.add(g); //Se guarda en la lista global de los grupos encontrados hasta ahora

sw2=verifFin(verG); //Se verifica si se han encontrado todos los posibles grupos dminantes del grafo

}

else

{

if (verticesSobrantes(verG) == 0) { //Si no es un grupo dominante y ya no hay mas vértices por seguir verificando

sw2=true; //Se finaliza la búsqueda de grupos dominantes

}

}

}

}

boolean verticeSobra(String vertice){ //Función booleana que determina si el vertice no ha sido tenido en cuenta aun en los grupos dominantes anteriores

for (int i = 0; i < grupos.size(); i++) {

for (int j = 0; j < grupos.get(i).getVertices().size(); j++) {

if (grupos.get(i).getVertices().get(j).contains(vertice)) {

return false;

}

}

}

return true;

}

int verticesSobrantes(ArrayList<String> vertG){//Funcion que devuelve la cantidad de evrtices que aun no se han tenido en cuenta a la hora de formar los grupos y los ya formados

int cont=0;

boolean sw=false;

int j;

for (int i = 0; i < vertices.size(); i++) {

j=0;

sw=false;

if (vertG.contains(vertices.get(i))) {

cont++;

sw=true;

}

while(j < grupos.size() && sw == false) {

if (grupos.get(j).getVertices().contains(vertices.get(i))) {

cont++;

sw=true;

}

/\*if (vertG.contains(vertices.get(i)) && sw2==false) {

cont++;

sw=true;

}\*/

j++;

}

}

return vertices.size()-cont;//Devuelve la resta entre el total de vértices del gafo menos el numero de vértices usados hasta el mmento

}

boolean verifFin(ArrayList<String> vertG){ //Verificar si ya Econtro todos los posibles grupos

int cont=0;

int j=0;

boolean sw=true;

for (int i = 0; i < vertices.size(); i++) {

sw=false;

j=0;

if (vertG.contains(vertices.get(i))) {

cont++;

sw=true;

}

while (j < grupos.size() && sw==false){

if (grupos.get(j).getVertices().contains(vertices.get(i))) {

cont++;

sw=true;

}

j++;

}

}

if (cont == vertices.size()) {

return true;

}

return false;

}

boolean verifEstaEnGrupo(String vertice){//Verifica si el vertice está en algun grupo creado con anterioridad

//boolean sw=false;

for (int j = 0; j < grupos.size(); j++) {

if (grupos.get(j).getVertices().contains(vertice)) {

return true;

}

}

return false;

}

boolean verifEsGrupo(ArrayList<String> verG,ArrayList<String> adya ){ //Verifica si los vertices escogidos actualmente forman o no un grupo

boolean sw=false;

for (int i = 0; i < vertices.size(); i++) {

if (verG.contains(vertices.get(i)) || adya.contains(vertices.get(i))) {//Para poder formar un grupo es necesario que se hallan escogido Todos los vertces que generan al

grafo. es decir que todos los vértices Deben estar en alguno de los dos grupos los los dominantes o los adyacentes, de esa forma se sabe si el conjunto forma o no un grupo

sw=true;

}

else

{

return false;

}

}

return sw;

}

void mostrar(){ //ubrutina para mostrar los resultados obtenidos

DefaultListModel lista= new DefaultListModel();

lista.addElement("El numero Domatico del grafo es: "+grupos.size());

for (int i = 0; i < grupos.size(); i++) {

lista.addElement("Grupo V"+(i+1));

for (int j = 0; j < grupos.get(i).getVertices().size(); j++) {

lista.addElement(grupos.get(i).getVertices().get(j));

}

}

jList1.setModel(lista);

}

//CLASE GRUPO

public class Grupo {

ArrayList<String> vertices = new ArrayList(); //Para guardar los Vertices dominantes

public ArrayList<String> getVertices() {

return vertices;

}

public void setVertices(ArrayList<String> vertices) {

this.vertices=vertices;

}

}

Referencias

* V. KULLI and D. PATWARI, “TOTAL EFFICIENT DOMINATION IN GRAPHS ,” *International Research Journal of Pure Algebra*, vol. 6, no. 1, 2016.
* S. Sheikholeslamia and L. Volkmannb, “The signed Roman domatic number of a digraph,” *Electronic journal of graph theory and applications*, vol. 3, no. 1, pp. 85–93, Mar. 2015.
* H. Aram, S. Sheikholeslami, and L. Volkmann, “On the total k-domination an total k-domatic number of graphs,” Oct. 2010.
* G. J. Chang, “The domtic number problem,” *Discrete Mathematics*, vol. 125, pp. 115–122, Mar. 1992.
* P. Dankelmann and N. Calkin, “The domatic number of regular graphs,” *South African national research foundation*, Sep. 2008.

Diseñe una expresión matemática representativa del numero medio de sondajes en la búsqueda binaria (valor esperado).