

Bachelors Project: Graph isomorphism problem

Eindhoven University of Technology  
Department of Industrial Applied Mathematics

Vincent Remie (495445)

September 5, 2003

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Invariances</b>	<b>3</b>
2.1	Degree Invariance . . . . .	3
2.2	Distance Multiplicity Invariance . . . . .	4
2.3	Subgraph Invariance . . . . .	5
2.4	Extended Subgraph Invariance . . . . .	6
<b>3</b>	<b>Description of the GAP code</b>	<b>9</b>
<b>4</b>	<b>Conclusion</b>	<b>11</b>

# 1 Introduction

This report discusses the graph isomorphism problem. The problem is about two connected graphs  $G$  and  $H$ , both consisting of  $n$  vertices, are isomorphic. Let  $E(G)$  and  $E(H)$  be the set of edges in  $G$  respectively  $H$ . Let  $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$  be a permutation which reorders the vertices of the graph  $G$  such that  $\overline{e_1 e_2} \in E(G) \Rightarrow \overline{\pi(e_1) \pi(e_2)} \in E(\pi(G))$ . The graph with such a reordering can be written as  $\pi(G)$ . We say that the graph  $G$  is isomorphic to the graph  $H$  (denoted as  $G \cong H$ ) if there exist a permutation  $\pi$  such that  $\pi(G) = H$ , i.e,  $G$  and  $H$  are really the same graph with a different labeling.

The isomorphism problem for  $G$  and  $H$  is easily reduced by determining the automorphism group of one graph. Each  $\pi \in Sym(n)$  gives a graph  $\pi(G)$  isomorphic to  $G$  defined on  $\{1, \dots, n\}$ . We say that  $\pi$  is an automorphism of  $G$  if  $\pi(G) = G$ . The set of all automorphisms of  $G$  is a subgroup of  $Sym(n)$ , called the automorphism group of  $G$ , and denoted by  $Aut(G)$ . The number of distinct graphs isomorphic to  $G$  is  $|Sym(n)/Aut(G)|$ .

The best available program for determining the isomorphism of two graph is nauty by Brendan McKay. If the graphs  $G$  and  $H$  are isomorphic, there is at least one element of the permutation group  $Sym(n)$  that maps  $G$  to  $H$ . This map is a certificate for the statement that  $G$  and  $H$  are isomorphic. If  $G$  and  $H$  are not isomorphic, this can be proved as a consequence of the fact that  $Sym(n)$  does not contain an element mapping  $G$  to  $H$ . However, the proof is very time-consuming, i.e. all  $\pi \in Sym(n)$  have to be checked.

For many pairs of graphs  $G$  and  $H$ , it can be easily seen that they are nonisomorphic. For instance, the number of vertices need to be the same for the two graphs to be isomorphic (we already assumed). However, this information is insufficient to distinguish all cases. Using this idea, proofs can be constructed to distinguish simple cases. This report discusses some possible proofs and their implementation in GAP.

## 2 Invariances

An effective way of proving that two graphs are nonisomorphic uses invariances. Before describing some invariances we first need to introduce the definition of an invariance and we need to introduce a notation for an arbitrary graph.

**Invariance** Let  $f$  be a function defined on graphs.  $f$  is called an invariant if  $G \cong H \Rightarrow f(G) = f(H)$ .

**Neighbour-graph** Suppose an arbitrary graph  $G$  with vertex set  $V(G) = \{g_1, \dots, g_k\}$  and with edges set  $E(G)$ . The neighbour-graph data is the list  $[N_1, \dots, N_k]$  where  $\forall i \in \{1, \dots, k\} \wedge \forall j \in N_i : \overline{ij} \in E(G)$ . Notice that  $N_i$  is a set.

All invariances described in this report are grounded on this neighbour-graph notation. In the next subsections some invariances will be worked out.

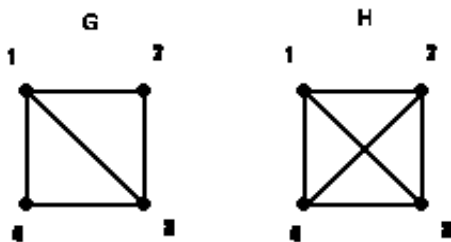
### 2.1 Degree Invariance

A simple first invariance could easily be derived. We call this invariance the degree invariance.

**Degree invariance** Let  $G$  be a connected graph with  $k$  vertices. Let  $[N_1, \dots, N_k]$  be the neighbour-graph notation for the graph  $G$ . Define the multiset  $dgr(G) = \{|N_1|, \dots, |N_k|\}$ . Then  $dgr$  is an invariant.

The evidence for this invariance holds because a permutation  $\pi \in Sym(n)$  only reorders the vertices and does not change the degrees of the vertices. We give an example which uses the degree criterion.

Let  $G = [\{2, 4\}, \{1, 3, 4\}, \{2, 4\}, \{1, 2, 3\}]$  and  $H = [\{2, 3, 4\}, \{1, 3, 4\}, \{1, 2, 4\}, \{1, 2, 3\}]$ .



By the figure it is easily seen that the graphs  $G$  and  $H$  are nonisomorphic. When using the invariance, we get the multisets of degrees  $dgr(G) = \{2, 3, 2, 3\}$  respectively  $dgr(H) = \{3, 3, 3, 3\}$  and we see that  $dgr(G) \neq dgr(H)$ . This means, that  $G$  and  $H$  are nonisomorphic. Using a function gives the following output:

- > G and H are nonisomorphic, because:
- >
- > The vertices of G have the following degrees:
- > 2 vertices with degree 2
- > 2 vertices with degree 3
- >
- > The vertices of H have the following degrees:
- > 4 vertices with degree 3

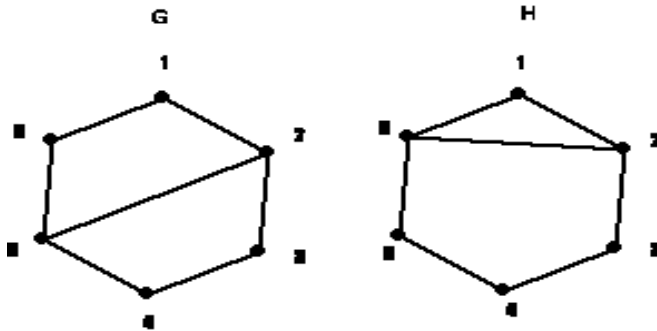
## 2.2 Distance Multiplicity Invariance

There are graphs which are nonisomorphic but fail the degree invariance. In such cases we need more invariances. One invariance we could use if the degree invariance has failed is the distance multiplicity invariance.

**Distance multiplicity Invariance** *Let  $G$  be a connected graph with  $n$  vertices. Define the distance of two vertices in a graph as the shortest path from one vertex to the other. The maximum distance is called the diameter. Let  $g = [[g_{1,1}, \dots, g_{1,k_1}], \dots, [g_{n,1}, \dots, g_{n,k_n}]]$  where  $g_{i,j}$  is the number of vertices at distance  $j$  from vertex  $i$  in  $G$ . Define the multiset  $dmp(G, j) = \{\{g_{i,j} | i \in \{1, \dots, n\}\}\}$  and define  $d$  as the diameter of  $G$ . Then the function  $f_i : G \rightarrow dmp(G, j)$  is invariant for all  $i \in \{1, \dots, d\}$ .*

The evidence for this invariance holds because a permutation  $\pi \in Sym(n)$  only reorders the vertices and does not change the distances between vertices. We give an example which uses the distance multiplicity invariance.

Let  $G = [\{2, 6\}, \{1, 3, 5\}, \{2, 4\}, \{3, 5\}, \{2, 4, 6\}, \{1, 5\}]$  and  $H = [\{2, 6\}, \{1, 3, 6\}, \{2, 4\}, \{3, 5\}, \{4, 6\}, \{1, 2, 5\}]$ .



By the figure it is easily seen that the graphs are nonisomorphic. When using the degree invariance we get the multisets of degrees  $dgr(G) = \{\{2, 3, 2, 2, 3, 2\}\}$  respectively  $dgr(H) = \{\{2, 3, 2, 2, 2, 3\}\}$  and we see that  $dgr(G) = dgr(H)$ . So, according to the degree invariance the graphs are isomorphic. When using the distance multiplicity invariance we get multisets representing the number of vertices at a given distance from a vertex. If distance 2 is chosen we get the sets  $dmp(G, 2) = \{\{2, 2, 2, 2, 2, 2\}\}$  respectively  $dmp(H, 2) = \{\{2, 2, 2, 2, 3, 3\}\}$  and we see that  $dmp(G, 2) \neq dmp(H, 2)$ . This means, according to the invariance, that  $G$  and  $H$  are nonisomorphic. Using a function gives the following output:

> G and H are nonisomorphic because:  
 >  
 > G has  
 > 6 vertices with 2 vertices at distance 2  
 >  
 > But H has  
 > 4 vertices with 2 vertices at distance 2  
 > 2 vertices with 3 vertices at distance 2

## 2.3 Subgraph Invariance

There are nonisomorphic graphs for which both the degree invariance and the distance multiplicity invariance fail. In such cases we still need more invariances. One invariance we use is the subgraph invariance. Before defining this invariance we first introduce the term subgraph and component.

**Subgraph** Let  $G$  be a connected graph with vertices set  $V(G) = \{1, \dots, n\}$  and edges set  $E(G)$ . Define the subgraph  $G_M$  as a connected graph with vertices set  $M$  and edges set  $\{\overline{ij} | i, j \in M; \overline{ij} \in E(G)\}$

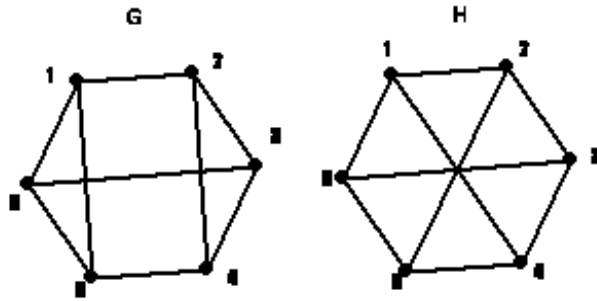
**Component** Let  $G$  be a graph. Define a component of  $G$  as a maximal connected subgraph. The size of the component represents the number of vertices of the maximal connected subgraph.

By the introduced definitions above we can think of a invariance that looks at the different possible subgraphs. Therefore we introduce the subgraph invariance. We define the size of a component as the size of the vertices set of the component.

**Subgraph Invariance** Let  $G$  be a connected graph with  $n$  vertices. Let  $G[j]$  be the set of vertices at distance 1 from vertex  $j$ . Define  $comp(G, i, j)$  as the number of components of size  $j$  in the subgraph  $G_{G[i]}$ . Define the multiset  $cms(G) = \{\{\sum_i comp(G, i, j) | i \in V(G); j \in \{1, \dots, |G[i]|\}\}\}$ . Then the function  $f : G \rightarrow cms(G)$  is invariant.

The evidence for this invariance holds because a permutation  $\pi \in Sym(n)$  only reorders the vertices and does not interchange the subgraphs without change the subgraphs themselves. We give an example which uses the subgraph criterion.

Let  $G = [\{2, 5, 6\}, \{1, 3, 4\}, \{2, 4, 6\}, \{2, 3, 5\}, \{1, 4, 6\}, \{1, 3, 5\}]$  and  
 $H = [\{2, 4, 6\}, \{1, 3, 5\}, \{2, 4, 6\}, \{1, 3, 5\}, \{2, 4, 6\}, \{1, 3, 5\}]$ .



By the figure it is easily seen that the graphs are nonisomorphic. When using the degree invariance we get the multisets of degrees  $dgr(G) = \{\{3, 3, 3, 3, 3, 3\}\}$  respectively  $dgr(H) = \{\{3, 3, 3, 3, 3, 3\}\}$  and we see that  $dgr(G) = dgr(H)$ . So, according to the degree invariance the graphs are isomorphic. When using the distance multiplicity invariance we get multisets representing the number of vertices at a given distance from a vertex. If distance 2 is chosen we get the sets  $dmp(G, 2) = \{\{2, 2, 2, 2, 2, 2\}\}$  respectively  $dmp(H, 2) = \{\{2, 2, 2, 2, 2, 2\}\}$  and we see that  $dmp(G, 2) = dmp(H, 2)$ . This means that  $G$  and  $H$  are isomorphic according to the distance multiplicity invariance. When using the subgraph invariance we get the multisets  $cms(G) = \{\{6, 6, 0\}\}$  and  $cms(H) = \{\{18, 0, 0\}\}$ . This means, according to the invariance, that  $G$  and  $H$  are nonisomorphic. Using a function gives the following output:

```
> G and H are nonisomorphic because:
>
> The Subgraphs of G have got
> 6 components of size 1
> 6 components of size 2
> 0 components of size 3
>
> But the subgraphs of H have got
> 18 components of size 1
> 0 components of size 2
> 0 components of size 3
```

## 2.4 Extended Subgraph Invariance

There are graphs which are nonisomorphic but where the degree invariance, the distance multiplicity invariance and even the subgraph invariance fail. In such cases we could extend the subgraph invariance, i.e., apply the subgraph invariance to other type of subgraphs. Therefore we'll introduce the distance-subgraph.

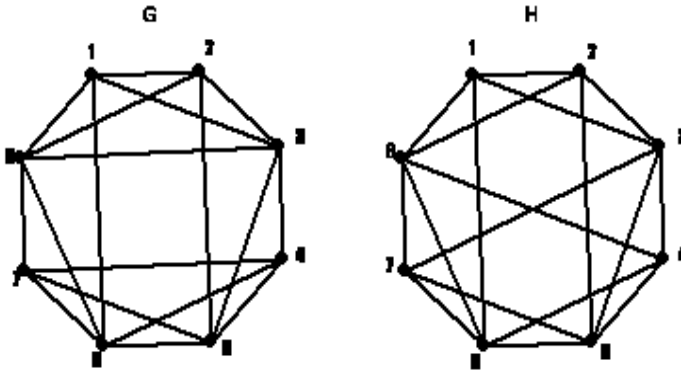
**Distance-subgraph** Let  $G$  be a connected graph with vertices set  $V(G) = \{1, \dots, n\}$  and edges set  $E(G)$ . Suppose  $G[i, j]$  is the set of vertices at distance  $j$  from vertex  $i$  in  $G$ . Now the distance subgraph  $G_{d,j}$  has vertices set  $G[j, d]$  and edges set  $\{\overline{ij} | i, j \in G[j, d]; \overline{ij} \in E(G)\}$ .

Now we introduce the extended subgraph invariance, a variant to the subgraph invariance.

**Extended Subgraph invariance** Let  $G$  be a connected graph with  $n$  vertices. Let  $G[j, d]$  be the set of vertices at distance  $d$  from vertex  $j$ . Define  $comp(G, i, j, d)$  as the number of components of size  $j$  in the subgraph  $G_{G[i, d]}$ . Define the multiset  $ecms(G, d) = \{\{\sum_i comp(G, i, j, d) | i \in V(G); j \in \{1, \dots, |G[i, d]|\}\}\}$ . Define  $d$  as the diameter of  $G$ . Then the function  $f_i : G \rightarrow ecms(G, i)$  is invariant for all  $i \in \{1, \dots, d\}$ .

The evidence for this invariance holds because a permutation  $\pi \in Sym(n)$  only reorders the vertices and does not interchange the distance subgraphs without change the distance subgraphs themselves. We give an example which uses the subgraph invariance.

Let  $G = [\{2, 3, 6, 8\}, \{1, 3, 5, 8\}, \{1, 2, 4, 5, 8\}, \{3, 5, 6, 7\}, \{2, 3, 4, 6, 7\}, \{1, 4, 5, 7, 8\}, \{4, 5, 6, 8\}, \{1, 2, 3, 6, 7\}]$   
and  
 $H = [\{2, 3, 6, 8\}, \{1, 3, 5, 8\}, \{1, 2, 4, 5, 7\}, \{3, 5, 6, 8\}, \{2, 3, 4, 6, 7\}, \{1, 4, 5, 7, 8\}, \{3, 5, 6, 8\}, \{1, 2, 4, 6, 7\}]$



By the figure it is seen that the graphs are nonisomorphic. When using the degree invariance we get the multisets of degrees  $dgr(G) = \{\{4, 4, 5, 4, 5, 5, 4, 5\}\}$  respectively  $dgr(H) = \{\{4, 4, 5, 4, 5, 5, 4, 5\}\}$  and we see that  $dgr(G) = dgr(H)$ . So, according to the degree invariance the graphs are isomorphic. When using the distance multiplicity invariance we get multisets representing the number of vertices at a given distance from a vertex. If distance 2 is chosen we get the multisets  $dmp(G) = \{\{3, 3, 2, 3, 2, 2, 3, 2\}\}$  respectively  $dmp(H) = \{\{3, 3, 2, 3, 2, 2, 3, 2\}\}$  and we see that  $dmp(G) = dmp(H)$ . This means, according to the invariance, that  $G$  and  $H$  are isomorphic. When using the subgraph invariance we get the multisets  $cms(G) = \{\{0, 6, 2, 2\}\}$  and  $cms(H) = \{\{0, 6, 6, 2\}\}$  and we see that  $cms(G) \neq cms(H)$ . This means that, according to the subgraph invariance,  $G$  and  $H$  are nonisomorphic. When using the extended version of the subgraph invariance we get the multisets  $ecms(G, 2) = \{\{0, 4, 4\}\}$  and  $ecms(H, 2) = \{\{2, 6, 2\}\}$ , which leads to  $ecms(G, 2) \neq ecms(H, 2)$ . This means that, according to the invariance,  $G$  and  $H$  are nonisomorphic. Using a function gives the following output:

> G and H are nonisomorphic because:  
>  
> The set of distance-subgraphs of size 2 of G contains  
> 0 components of size 1  
> 4 components of size 2  
> 4 components of size 3  
>  
> But the set of distance-subgraphs of size 2 of H contains  
> 2 components of size 1  
> 6 components of size 2  
> 2 components of size 3

### 3 Description of the GAP code

In this section the implemented GAP-functions will be described and explained. The functions that will be discussed are the functions NeighbourGraph, GraphIsomorphism, DegreeInvariance, DistanceGraph, DistanceMults, DistanceMultiplicityInvariance, Subgraph, Components, SubgraphInvariance, ExtendedSubGraphInvariance and GraphIsomorphismProblem.

**NeighbourGraph:** This function is created for the neighbourgraph notation: as input we take the graph in neighbourgraph notation and as output we get the graph in GAP's own notation. The necessity of this function is that for detecting an isomorphism between two graphs there is a function IsIsomorphicGraph(G,H) in GAP which requires the GAP graph notation as input.

**GraphIsomorphism:** This function checks whether two graphs are isomorphic. In the case they are the function will return the permutation of the automorphismgroup of G which maps the vertices of G to the vertices of H.

**DegreeInvariance:** This function is represents the degree invariance. The input is two graphs in neighbourgraph notation. In GAP these neighbourgraphs can be seen as a list of sublists. Therefore to check if their degrees of their vertices are the same the function calculates the length of each sublist and puts the results into two new lists. After sorting these two lists will be compared. In the case that they are not equal the function will return that the graphs are nonisomorphic.

**DistanceGraph:** This function will convert the neighbour-graph notation to the distance-graph notation. In the distance-graph notation we have, just like in neighbour-graph notation, a list of sublist. The sublists look like  $[[g_{j,1,1}, \dots, g_{j,1,k_{j,1}}], \dots, [g_{j,m,1}, \dots, g_{j,m,k_{j,m}}]]$  where the  $p^{th}$  subsublist in this sublist represents the vertices at distance  $p$  from the vertex  $j$ .

**DistanceMults:** This function returns a list of the length of the subsublists of the distance-graph. In fact the list looks like  $[[k_{1,1}, \dots, k_{1,m}], \dots, [k_{n,1}, \dots, k_{n,m}]]$  where  $k_{i,j}$  represents the number of vertices at distance  $j$  from the vertex  $i$ .

**DistanceMultiplicityInvariance:** This function represents the distance multiplicity invariance. For each possible distance, it compares the DistanceMults() function applied on the graph  $G$  with the DistanceMults() function applied on the graph  $H$  for every possible distance. In the case that a given distance results into two different lists the function returns that the graphs are nonisomorphic.

**Subgraph:** This function will return the neighbour-graph notation of a subgraph of the vertices given as input.

**Components:** This function splits up the input graph  $G$  into components and returns a list with sublists of the vertices of different components.

**SubgraphInvariance:** This function represents the subgraph invariance. The function contains the subfunction NumberOfComponents(). When applying this subfunction to a

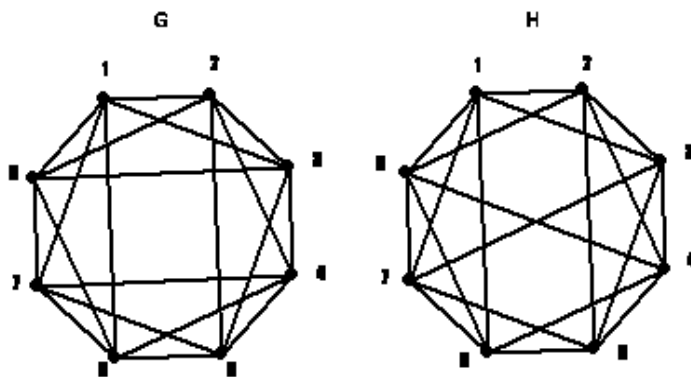
list returned by `Components()` we get a list that looks like  $\{a_1, \dots, a_k\}$  where  $a_j$  represents the number of components of size  $j$ . This subfunction will be applied to all subgraphs. Finally the results will be putted in a list which looks like a list returned by `NumberOfComponents()` and the list produced by the input  $G$  will be compared with the list produced by the input  $H$ . In the case that the lists aren't equal the function returns that the graphs are nonisomorphic.

**ExtendedSubgraphInvariance:** This function represents the extended subgraph invariance. The function contains the subfunction `DistanceSubgraph`. This subfunction returns a subgraph consist of vertices at an entered distance from an entered vertex. The edges in this reduced graph are the edges of the complete graph that connect the vertices in the reduced graph. The subfunction will be applied to all possible distances (say  $d$ ) and for all possible distances there will be generated lists of components. Subsequently lists representing the number of components of given sizes can be derived from those generated lists. Finally the  $d$  lists produced by the input  $G$  will be compared to the relating  $d$  lists produced by the input  $H$ . In the case that the lists aren't equal the function returns that the graphs are nonisomorphic.

**GraphIsomorphismProblem:** This function is the main function. The input is two graphs. The main function runs `GraphIsomorphism()` if the two graphs are isomorphic. Otherwise all functions representing the criteria will be runned till one of those functions returns that the two graphs are nonisomorphic. In the case that the functions return nothing the main function will return that the graphs are nonisomorphic, but that it can't be proved with the existing criteria.

## 4 Conclusion

Using invariants to prove that two graphs are non-isomorphic is very handy, because trying all permutations from the symmetric group to solve the problem is very time-consuming. However, the invariances described in this report will not work for all graphs. An example is the pair of graphs in the figure below



The invariances don't prove their nonisomorphism.

In this direction it seems that construct new invariances is necessary. It will decrease the existence of graphs for which the invariances will not work and finally there will be constructed so many invariances such that every nonisomorphism can be proved by the invariances.

## Attachment

The GAP-code from the last section

```
NeighbourGraph := function( L )
  local Grp, list, act, f;
  Grp := Group( ( ) );
  list := [1..Length( L )];
  act := OnPoints;
  f := function( x, y )
    return x in L[ y ];
  end;
  return Graph( Grp, list, act, f );
end;

GraphIsomorphism := function( G, H )
  if IsIsomorphicGraph( G, H ) then
    return G.canonicalLabelling * ( H.canonicalLabelling )^-1;
  else
    return "";
  fi;
end;

DegreeInvariance := function( G, H )
  local G1, H1, index, count, index2, answer;
  G1 := List( [1..Length( G )], x -> Length( G[ x ] ) );
  H1 := List( [1..Length( H )], x -> Length( H[ x ] ) );
  Sort( G1 );
  Sort( H1 );
  answer := "";
  if G1 <> H1 then
    answer := Concatenation( answer, "\n");
    answer := Concatenation( answer, "G and H are nonisomorphic, because: \n \n");
    answer := Concatenation( answer, "The vertices of G have the following degrees: \n");
    count := 1;
    index2 := 1;
    for index in [2..Length( G1 )] do
      if G1[ index ] = G1[ index2 ] then
        count := count + 1;
        if index = Length( G1 ) then
          answer := Concatenation( answer, String( count ), " vertices with degree ",
            String( G1[ index ] ), "\n");
        fi;
      else
        answer := Concatenation( answer, String( count ) );
        if count = 1 then
```

```

        answer := Concatenation( answer, " vertex with degree ",
                                String( G1[ index2 ] ), "\n");
    else
        answer := Concatenation( answer, " vertices with degree ",
                                String( G1[ index2 ] ), "\n");
    fi;
    if index = Length( G1 ) then
        answer := Concatenation( answer, "1 vertex with degree ",
                                String( G1[ index ] ), "\n");
    else
        index2 := index;
        count := 1;
    fi;
fi;
od;
answer := Concatenation( answer, "\n");
answer := Concatenation( answer, "But the vertices of H have the following
                        degrees: \n");

count := 1;
index2 := 1;
for index in [2..Length( H1 )] do
    if H1[ index ] = H1[ index2 ] then
        count := count + 1;
        if index = Length( H1 ) then
            answer := Concatenation( answer, String( count ), " vertices with degree ",
                                    String( H1[ index ] ), "\n");
        fi;
    else
        answer := Concatenation( answer, String( count ) );
        if count = 1 then
            answer := Concatenation( answer, " vertex with degree ",
                                    String( H1[ index2 ] ), "\n");
        else
            answer := Concatenation( answer, " vertices with degree ",
                                    String( H1[ index2 ] ), "\n");
        fi;
        if index = Length( H1 ) then
            answer := Concatenation( answer, "1 vertex with degree ",
                                    String( H1[ index ] ), "\n");
        else
            index2 := index;
            count := 1;
        fi;
    fi;
od;
answer := Concatenation( answer, "\n \n");
fi;

```

```

    return answer;
end;

DistanceGraph := function( L )
    local index, list, a, b;
    list := List( [1..Length(L)], x -> [L[x]] );
    for index in [1..Length( L )] do
        a := L[ index ]; # exact distance
        b := Set( Concatenation( [ [index], a ] ) ); # ball
        repeat
            a := Difference( Set( Concatenation( L{ a } ) ), b );
            Add( list[ index ], a );
            b := Union( a, b );
        until IsEmpty( a );
        Unbind( list[index][Length(list[index])] );
    od;
    return list;
end;

DistanceMults := function( M, d)
    local f;
    f := function( x )
        if IsBound(M[x][d]) then
            return Length(M[x][d]);
        else
            return 0;
        fi;
    end;
    return List( [1..Length(M)], f );
end;

DistanceMultiplicityInvariance := function( G, H)
    local G1, H1, count, g1, h1, number, index, index2, answer;
    G1 := DistanceGraph( G );
    H1 := DistanceGraph( H );
    count := 0;
    answer := "";
    repeat
        count := count + 1;
        g1 := DistanceMults( G1, count);
        h1 := DistanceMults( H1, count);
        Sort( g1 ); Sort( h1 );
    until g1 <> h1 or Sum(g1)+Sum(h1) = 0;
    if g1 <> h1 then
        answer := Concatenation( answer, "\n", "G and H are nonisomorphic because: \n \n",
            "G has \n");
        number := 1;
    fi;
end;

```

```

index2 := 1;
for index in [2..Length( g1 )] do
  if g1[ index ] = g1[ index2 ] then
    number := number + 1;
    if index = Length( g1 ) then
      answer := Concatenation( answer, String( number ), " vertices with ");
      answer := Concatenation( answer, String( g1[ index ] ),
        " vertices at distance ");
      answer := Concatenation( answer, String( count ), "\n");
    fi;
  else
    answer := Concatenation( answer, String( number ) );
    if number = 1 then
      answer := Concatenation( answer, " vertex with ", String( g1[ index2 ] ) );
      answer := Concatenation( answer, " vertices at distance ",
        String( count ), "\n");
    else
      answer := Concatenation( answer, " vertices with ",
        String( g1[ index2 ] ) );
      answer := Concatenation( answer, " vertices at distance ",
        String( count ), "\n");
    fi;
    if index = Length( g1 ) then
      answer := Concatenation( answer, "1 vertex with ", String( g1[ index ] ) );
      answer := Concatenation( answer, " vertices at distance ",
        String( count ), "\n");
    else
      index2 := index;
      number := 1;
    fi;
  fi;
od;
answer := Concatenation( answer, "\n", "But H has \n");
number:= 1;
index2 := 1;
for index in [2..Length( h1 )] do
  if h1[ index ] = h1[ index2 ] then
    number := number + 1;
    if index = Length( h1 ) then
      answer := Concatenation( answer, String( number ), " vertices with ");
      answer := Concatenation( answer, String( h1[ index ] ),
        " vertices at distance ");
      answer := Concatenation( answer, String( count ), "\n");
    fi;
  else
    answer := Concatenation( answer, String( number ) );
    if number = 1 then

```

```

        answer := Concatenation( answer, " vertex with ", String( h1[ index2 ] ) );
        answer := Concatenation( answer, " vertices at distance ",
                                String( count ), "\n");
    else
        answer := Concatenation( answer, " vertices with ",
                                String( h1[ index2 ] ) );
        answer := Concatenation( answer, " vertices at distance ",
                                String( count ), "\n");
    fi;
if index = Length( h1 ) then
    answer := Concatenation( answer, "1 vertex with ", String( h1[ index ] ) );
    answer := Concatenation( answer, " vertices at distance ",
                            String( count ), "\n");
else
    index2 := index;
    number := 1;
fi;
fi;
od;
answer := Concatenation( answer, "\n \n");
fi;
return answer;
end;

Subgraph := function( G, L )
    local index, list, relabel;
    list := List( [1..Length( L )], x -> [] );
    relabel := function( H )
        local h, i, j;
        for i in [1..Length( H )] do
            for j in [1..Length( H[ i ] )] do
                H[ i ][ j ] := Position( L, H[ i ][ j ] );
            od;
        od;
    end;
    for index in [1..Length( L )] do
        list[ index ] := Intersection( G[ L[ index ] ], L );
    od;
    relabel( list );
    return list;
end;

Components := function( G )
    local L, C, D, u, v, index;
    L := [1..Length( G )];
    D := [];
    repeat

```

```

C := [ L[ 1 ] ];
for u in C do
  for v in G[ u ] do
    if not v in C then
      Add( C, v );
    fi;
  od;
od;
Add(D, C);
L := Difference( L, C );
until Length( L ) = 0;
return D;
end;

SubgraphInvariance := function( G, H )
local g1, h1, g2, h2, NumberOfComponents, index, answer;
NumberOfComponents := function( L )
local index, index2, index3, count, list;
list := [];
for index in [1.. Maximum( List( [1..Length( L )], x ->
Length( Concatenation( L[ x ] ) ) ) ) ] do
count := 0;
for index2 in [1..Length( L )] do
for index3 in [1..Length( L[ index2 ] )] do
if Length( L[ index2 ][ index3 ] ) = index then
count := count + 1;
fi;
od;
od;
Add( list, count );
od;
return list;
end;
g1 := NumberOfComponents( List( [1..Length( G )], x ->
Components( Subgraph( G, G[ x ] ) ) ) );
g2 := NumberOfComponents( List( [1..Length( G )], x ->
Components( Subgraph( G, G[ x ] ) ) ) );
h1 := NumberOfComponents( List( [1..Length( H )], x ->
Components( Subgraph( H, H[ x ] ) ) ) );
h2 := NumberOfComponents( List( [1..Length( H )], x ->
Components( Subgraph( H, H[ x ] ) ) ) );
Sort( g1 );
Sort( h1 );
answer := "";
if g1 <> h1 then
answer := Concatenation( answer, "\n", "G and H are nonisomorphic because: \n \n");
answer := Concatenation( answer, "The subgraphs of G have got \n");

```

```

for index in [1..Length( g2 )] do
    answer := Concatenation( answer, String( g2[ index ] ), " components of size ");
    answer := Concatenation( answer, String( index ), "\n");
od;
answer := Concatenation( answer, "\n", "But the subgraphs of H have got \n");
for index in [1..Length( h2 )] do
    answer := Concatenation( answer, String( h2[ index ] ), " components of size ");
    answer := Concatenation( answer, String( index ), "\n");
od;
answer := Concatenation( answer, "\n \n");
fi;
return answer;
end;

ExtendedSubgraphInvariance := function( G, H )
    local g1, h1, g2, h2, DistanceSubgraph, NumberOfComponents,
        value, dist, index, dg1, dh1, answer, IsBoundDistanceGraphs;
    DistanceSubgraph := function( K, x, distance )
        local j, k;
        j := DistanceGraph( K );
        k := Subgraph( K, j[ x ][ distance ] );
        return k;
    end;
    NumberOfComponents := function( L )
        local index, index2, index3, count, list;
        list := [];
        for index in [1.. Maximum( List( [1..Length( L )], x ->
            Length( Concatenation( L[ x ] ) ) ) )] do
            count := 0;
            for index2 in [1..Length( L )] do
                for index3 in [1..Length( L[ index2 ] )] do
                    if Length( L[ index2 ][ index3 ] ) = index then
                        count := count + 1;
                    fi;
                od;
            od;
            Add( list, count );
        od;
        return list;
    end;
    IsBoundDistanceGraphs := function( dg, dh, distance )
        local i, bounded;
        bounded := true;
        for i in [1..Length( dg )] do
            if IsBound( dg[ i ][ distance ] ) = false then bounded := false; fi;
            if IsBound( dh[ i ][ distance ] ) = false then bounded := false; fi;
        od;
    end;
end;

```

```

    return bounded;
end;
dist := 0;
dg1 := DistanceGraph( G );
dh1 := DistanceGraph( H );
repeat
    dist := dist + 1;
    g1 := NumberOfComponents( List( [1..Length( G )], x ->
        Components( DistanceSubgraph( G, x, dist ) ) ) );
    g2 := NumberOfComponents( List( [1..Length( G )], x ->
        Components( DistanceSubgraph( G, x, dist ) ) ) );
    h1 := NumberOfComponents( List( [1..Length( H )], x ->
        Components( DistanceSubgraph( H, x, dist ) ) ) );
    h2 := NumberOfComponents( List( [1..Length( H )], x ->
        Components( DistanceSubgraph( H, x, dist ) ) ) );
    Sort( g1 );
    Sort( h1 );
until g1 <> h1 or IsBoundDistanceGraphs( dg1, dh1, dist + 1 ) = false;
answer := "";
if g1 <> h1 then
    answer := Concatenation( answer, "\n", "G and H are nonisomorphic because: \n \n");
    answer := Concatenation( answer, "The set of distance-subgraphs of size ");
    answer := Concatenation( answer, String( dist ), " of G contains \n");
    for index in [1..Length( g2 )] do
        answer := Concatenation( answer, String( g2[ index ]), " components of size ");
        answer := Concatenation( answer, String( index ), "\n");
    od;
    answer := Concatenation( answer, "\n", "But the set of distance-subgraphs ");
    answer := Concatenation( answer, "of size ", String( dist ), " of H contains \n");
    for index in [1..Length( h2 )] do
        answer := Concatenation( answer, String( h2[ index ] ), " components of size ");
        answer := Concatenation( answer, String( index ), "\n");
    od;
    answer := Concatenation( answer, "\n \n");
fi;
return answer;
end;

GraphIsomorphismProblem := function(G, H)
    local proved;
    if GraphIsomorphism( NeighbourGraph( G ), NeighbourGraph( H ) ) <> "" then
        Print("\n");
        Print("G and H are isomorphic. The permutation ");
        Print( GraphIsomorphism( NeighbourGraph( G ), NeighbourGraph( H ) ), " ");
        Print("maps the vertices of G to H \n \n");
    else
        proved := false;
    end;
end;

```

```

if DegreeInvariance( G, H ) <> "" then
  Print( DegreeInvariance( G, H ) );
  proved := true;
fi;
if DistanceMultiplicityInvariance( G, H ) <> "" then
  if proved = false then
    Print( DistanceMultiplicityInvariance( G, H ) );
    proved := true;
  fi;
fi;
if SubgraphInvariance( G, H ) <> "" then
  if proved = false then
    Print( SubgraphInvariance( G, H ) );
    proved := true;
  fi;
fi;
if ExtendedSubgraphInvariance( G, H ) <> "" then
  if proved = false then
    Print( ExtendedSubgraphInvariance( G, H ) );
    proved := true;
  fi;
fi;
if proved = false then
  Print("\n","G and H are nonisomorphic but it can't
        be proved by using the criteria \n \n");
fi;
fi;
end;

```