COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO

Problems Polynomially Equivalent to Graph Isomorphism

Kellogg S. Booth
Charles J. Colbourn

CS-77-04

June, 1979

Problems Polynomially Equivalent to Graph Isomorphism

Kellogg S. Booth

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Charles J. Colbourn

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
M5S 1A7

ABSTRACT

A list of problems polynomially equivalent to graph isomorphism is given. A short description of each problem and the associated reductions is included. Some open problems involving graph isomorphism are also discussed.

1. *Preliminaries* 1

2. *Automorphisms of graphs* 2
2.1 *Connected graphs* 2
2.2 *Finding an isomorphism* 2
2.3 *Automorphism partition* 3
2.4 *The order of the automorphism group* 3
2.5 *Generators of the automorphism group* 4

3. *On proofs of isomorphism completeness* 4
3.1 *A sample proof* 5
3.2 *The general case* 6

4. *Edge replacement techniques* 6
4.1 *Labeled graphs and pseudographs* 7
4.2 *Directed graphs* 7
4.3 *Oriented graphs* 8
4.4 *Acyclic rooted digraphs* 8
4.5 *Bipartite graphs* 9
4.6 *Chordal graphs* 10
4.7 *Graphs with a forbidden induced subgraph* 10
4.8 *Transitively orientable graphs* 11
4.9 *Series-parallel digraphs* 12
4.10 *Undirected path graphs* 13

5. *Composition techniques* 14
5.1 *Compact graphs* 15
5.2 *Composite graphs* 16
5.3 *Self-complementary graphs and digraphs* 17
5.4 *Regular self-complementary graphs* 17
5.5 *Graphs with a unique centre* 18
5.6 *Cospectral graphs* 19

6. *Other techniques* 20
6.1 *Line graphs* 20
6.2 *Regular graphs* 21
6.3 *Regular bipartite graphs* 21
6.4 *Marked graphs and marked trees* 21
6.5 *K-trees* 23
6.6 *C-subgraph regular graphs* 23
6.7 *Eulerian graphs* 23
6.8 *Hamiltonian graphs* 24
6.9 *Hypergraphs* 25

7. *Isomorphism of algebraic structures* 25

7.1 Algebras 25
7.2 Lattices and posets 25
7.3 Semigroups 26
7.4 Finitely presented algebras 27
7.5 Automata and semiautomata 27

8. Combinatorial designs 27
8.1 Partially balanced incomplete block designs 27
8.2 Set packings 27
8.3 (r,λ)-systems 28
8.4 Exact set packings 28
8.5 Pairwise balanced designs 28

9. An equivalent clique problem 29

10. Other problems 30
10.1 Recognition of self-complementary graphs 30
10.2 Regular legitimate decks 30
10.3 Hadamard equivalence 31
10.4 The star-system problem 32

11. An NP-complete automorphism problem 32

12. Open problems 35
12.1 Canonizing or coding graphs 35
12.2 Strongly regular graphs 36
12.3 Transitive graphs 36
12.4 Rigid graphs 36
12.5 Block designs 36
12.6 Tournaments 36
12.7 Graphs of fixed genus 37
12.8 Degree bounded graphs 37
12.9 A good characterization 37
12.10 Modular lattices 37
12.11 Subgraph isomorphism 38
12.12 The complexity of graph isomorphism 38
12.13 Directed path graphs 38

13. Probabilistic Methods 38

Appendix 40

References 49

1. Preliminaries

The problem of deciding when two graphs are isomorphic has been the focus of much research in recent years [21,57]. The interest in the problem stems in part from its unresolved complexity. No polynomial time algorithm is known, nor is the problem known to be NP-complete.

The interest in graph isomorphism has resulted in research on many restrictions and generalizations of the problem. Many of these are now known to be polynomial time equivalent to graph isomorphism. Such problems are called *isomorphism complete.*

There are two important motivations for demonstrating the isomorphism completeness of a problem. The first is that a breakthrough on one isomorphism complete problem would result in a breakthrough on all. The second is that, since the isomorphism problem has occupied many researchers' efforts for years, an isomorphism complete problem is very likely a difficult one.

We will assume the usual definitions in graph theory [9,37], lattice theory [7], group theory [59], and design theory [64]. We will further assume familiarity with the notions of polynomial time reducibility [1,25,39], and we shall not distinguish between Cook's notion of polynomial time reducibility and Karp's notion of polynomial time transformability.

We introduce here notation concerned with graph isomorphism. Two graphs $G=<V,E>$ and $H=<W,F>$ are *isomorphic* if and only if there is an bijection I from V to W for which (v,w) is an edge in E if and only if $(I(v),I(w))$ is an edge in F; such a bijection is an *isomorphism* of G onto H. An *automorphism* of G is an isomorphism of G onto itself. The automorphisms of a graph form a group under composition; this group is the *automorphism group* of G. Two vertices v, w of G are *similar* if there is an automorphism of G mapping v to w. *Similarity* is an equivalence relation on the vertices of G; the partitioning induced by this equivalence relation is the *automorphism partition* of the graph. The *similarity classes* are the classes or blocks of the automorphism partition. A *set of generators* for the automorphism group is a collection of automorphisms whose closure under inverse and functional composition is the entire group.

In the remainder of the paper, we shall present a comprehensive list of problems which are currently known to be isomorphism complete. Where it is feasible to present a sketch of the proof quickly, we do so. Due to space limitations, we often shall present only the reductions involved, and refer the reader to source material for proofs that the reductions are correct. For isomorphism problems of restricted classes of graphs (sections 4-6) and algebras (section 7), we supplement the English description in the text with a formal specification of the reduction (except in some cases where the construction is easily accessible elsewhere); these are included in the appendix.

Many of the theorems presented have been in the 'folklore' for a long time, and typically have been independently rediscovered over and over again. In the case of folklore theorems, we do not cite any reference. In all other cases, a

reference to the first appearance (that we know of) is given.

2. Automorphisms of Graphs

In this section we consider problems dealing with isomorphisms and automorphisms of undirected graphs. In some reductions we allow the introduction of vertex labels; we postpone until section 4.1 a proof that equivalent reductions exist which do not use vertex labels. Following [50], we denote by $G_{v_1,\ldots,v_k}$ the labeled graph in which v_i is distinguishable as the unique i'th labeled vertex.

2.1 Connected Graphs

In many of the constructions which follow we shall require the following trivial result.

THEOREM: Connected graph isomorphism is isomorphism complete.

PROOF:

Since all connected graphs are graphs, we need only show how to solve graph isomorphism using an algorithm for isomorphism of connected graphs.

Given two n-vertex graphs G and H, find the connected components $G_1,\ldots,G_k$ of G and $H_1,\ldots,H_m$ of H. If $k \neq m$, G and H are nonisomorphic. Otherwise, find a component H_i of H which is isomorphic to G_1. If no such component exists, G and H are nonisomorphic. Otherwise, G and H are isomorphic if and only if $G-G_1$ and $H-H_i$ are. We proceed recursively to solve this smaller problem. □

ALTERNATE PROOF:

Given two graphs, determine whether G and H are connected. If both are connected, we use the connected graph isomorphism algorithm to decide whether they are isomorphic. If neither is connected, their complements are connected. Further, their complements are isomorphic if and only if G and H are. We then use the connected graph algorithm to decide whether their complements are isomorphic. Obviously, if one graph is connected and the other is not, the graphs are nonisomorphic. □

2.2 Finding an Isomorphism

The graph isomorphism problem is to decide, given two graphs, whether an isomorphism exists. A related problem is to find an isomorphism and explicitly present it, whenever one exists. The equivalence of these two problems has been known for many years; it is implicit in [27]. We follow the proof in [50].

THEOREM: Finding an isomorphism of two graphs is isomorphism complete.

PROOF:

A method for finding an isomorphism clearly will solve the graph isomorphism problem. We therefore need only show how to find an isomorphism given an algorithm to solve graph isomorphism.

Given two n-vertex graphs, G and H, one first ensures that they are isomorphic; if they are not, no isomorphism can be found. If they are, a vertex v of G is selected and labeled uniquely, the resulting graph being denoted by G_v. A search is then performed for a vertex a of H for which G_v and H_a are isomorphic. There is such a vertex since G and H are isomorphic. One now selects and uniquely labels a second vertex w of G, and a search for a vertex b of H is performed for which $G_{v,w}$ is isomorphic to $H_{a,b}$. This process is repeated until all vertices are labeled, at which point an isomorphism is the map carrying the i'th labeled vertex in G onto the i'th labeled vertex in H.

The method requires only $O(n^2)$ calls to the isomorphism algorithm. □

2.3 Automorphism Partition

Computing the automorphism partition of a graph has also long been known to be isomorphism complete. The equivalence seems to have been noted first by Karp; we follow the proof in [57]. This proof follows the same pattern as the proof in section 2.2.

THEOREM: Computing the automorphism partition of a graph is isomorphism complete.

PROOF:

Two connected graphs G and H are isomorphic iff there is a similarity class in $G+H$ containing vertices from each of G and H. Computing the automorphism partition of $G+H$ will thus automatically provide enough information to decide if G and H are isomorphic.

We need therefore only show that, given an isomorphism algorithm, we can find the automorphism partition of a graph in polynomial time. Given a graph G and a vertex v, define G^*v to be the result of first adding a complete graph on n vertices to G and then connecting v to each of the added vertices. Two vertices x and y are similar (in the same class of the automorphism partition) if and only if G^*x and G^*y are isomorphic. Thus, with $O(n^2)$ invocations of the isomorphism algorithm, we can find the automorphism partition of a graph. □

2.4 The Order of the Automorphism Group

Recently, the problem of finding the number of automorphisms of a graph has been shown to be isomorphism complete [3,50]. Valiant [65] proved that the counting analogues of many NP-complete problems are polynomially equivalent; such problems are called #P-complete. Angluin [2] observes that counting the number of automorphisms of a graph is unlikely to be #P-complete. Thus the Babai-Mathon result (given next) lends evidence to our opinion that graph isomorphism is not NP-complete (this is discussed further in [28]).

THEOREM: Computing the order of the automorphism group of a graph is isomorphism complete.

SKETCH OF PROOF:

Two connected graphs G and H are isomorphic if and only if $G+H$ has more automorphisms than the product of the number of automorphisms of G and that of H. Since connected graph isomorphism is isomorphism complete, this effectively reduces the graph isomorphism problem to the automorphism counting problem.

One counts automorphisms, given an isomorphism algorithm, by observing that the order of the group of $G_{v_1,\ldots,v_{k-1}}$ is exactly d times the order of the group of $G_{v_1,\ldots,v_k}$, where d is the size of the similarity class of v_k in $G_{v_1,\ldots,v_{k-1}}$.

This leads to a recursive algorithm which finds the order of the automorphism group and whose running time is polynomial in the time required to compute the automorphism partition of a graph. □

The reader should note that this last proof technique is essentially an application of Burnside's Lemma [15] which is used in combinatorial enumeration problems.

2.5 Generators of the Automorphism Group

Mathon showed that as well as finding the number of automorphisms we can also find a representation for the automorphism group at essentially no additional cost.

THEOREM: [50] Computing a set of generators for the automorphism group of a graph is isomorphism complete.

PROOF:

As in section 2.3, we note that two connected graphs G and H are isomorphic if and only if there is a generator in a set of generators for $G+H$ which carries a vertex in G to one in H, or vice versa.

Suppose we are given a graph G with vertex set $\{v_1,\ldots,v_n\}$. Applying the procedure for finding an isomorphism to $G_{v_1,\ldots,v_k}$ and $G_{v_1,\ldots,v_{k-1},v_m}$ for $k<m\leqslant n$, we obtain a set of automorphisms A_k of G. One set of generators for the automorphism group of G is the union of the A_i for $0\leqslant i<n$. □

3. On proofs of isomorphism completeness

In the next four sections we consider variants of the graph isomorphism problem; we show that these variants are isomorphism complete. The proofs of isomorphism completeness are usually straightforward and follow a simple pattern, which we illustrate here.

3.1 A Sample Proof

Let us consider a specific problem. We are told to devise an isomorphism algorithm, and we are given the additional information that all of the graphs we consider will have diameter at least $n/2$, where n is the number of vertices in the graph. We call such graphs *thin* graphs.

We have looked for a polynomial time isomorphism algorithm for thin graphs, and failed to find one. Before giving up the search, we would like evidence that our "restricted" problem is as hard as graph isomorphism. How do we go about demonstrating that our problem is isomorphism complete? We must show that with polynomially many invocations of an isomorphism algorithm for thin graphs, we can test isomorphism of arbitrary graphs.

Given a pair of graphs (i.e., an instance of the graph isomorphism problem), one approach is to transform this to a pair of thin graphs. Consider, then, the following transformation.

Given an n-vertex graph G, add to G a path with $n+1$ new vertices. Let x be a leaf (endpoint) of this path. Define $THIN(G)$ to be the graph obtained from G by connecting x to each original vertex of G. We observe that

(1) $THIN(G)$ has $2n+1$ vertices, and it has diameter $n+1$, thus it is indeed a thin graph.

(2) $THIN(G)$ and $THIN(H)$ are isomorphic exactly when G and H are isomorphic. We can see this by noting that, given $THIN$(G), we can recover G as follows. As long as G has at least two vertices (i.e., as long as $THIN(G)$ has at least five vertices), there is a unique vertex of degree 1 which is adjacent to a vertex of degree 2. This is the unattached end of the added path. Using this fact, we can uniquely identify the added path. To recover G from $THIN(G)$, we simply delete this added path and all of its incident edges.

(3) Given G, we can compute $THIN(G)$ in time polynomial in the number of vertices of G.

We now know that isomorphism of thin graphs is isomorphism complete. Why? Because given an instance (G,H) of the graph isomorphism problem, we solve it by solving the thin graph isomorphism problem $(THIN(G),THIN(H))$. Observations (1) and (2) guarantee that the answer to the thin graph problem is also the answer to the original problem. Observation (3) guarantees that the reduction can be done in polynomial time.

3.2 The General Case

Let us consider a general isomorphism problem. The problem involves determining isomorphism of graphs in a class C. There will typically be two reductions involved, one in each direction. The first transforms an arbitrary graph into a graph in class C; this shows that graph isomorphism is polynomial time reducible to C-graph isomorphism. The second transforms a graph in C into a graph, reducing C-graph isomorphism to graph isomorphism.

Consider a construction T transforming arbitrary graphs into C-graphs where C is any class of graphs. The transformation must satisfy three properties:

(1) $T(G)$ is in C for each graph G.

(2) $T(G)$ and $T(H)$ are isomorphic iff G and H are.

(3) $T(G)$ can be computed in time polynomial in the number of vertices in G.

An equivalent version of (2) is often used:

(2′) $T(G)$ uniquely determines G up to isomorphism.

Almost all of the transformations satisfy the stronger condition that G can be uniquely recovered from $T(G)$ in polynomial time; this stronger condition is not necessary, however. The existence of two transformations satisfying the above properties, one representing graphs as C-graphs, and one representing C-graphs as graphs, demonstrates the isomorphism completeness of isomorphism testing of C-graphs. In fact, a somewhat weaker condition is acceptable; the transformations can fail to have one (or more) of the properties in finitely many instances and the same conclusion holds. This is because an algorithm can deal with a fixed number of special cases at no asymptotic increase in running time.

In many of the subsequent problems, one of the transformations is trivial. This happens when all graphs are C-graphs, or when all C-graphs are graphs.

It is important to note that once isomorphism of a class C' of graphs is known to be isomorphism complete, we may transform graphs to C-graphs by first transforming graphs to C'-graphs and then C'-graphs to C-graphs. In instances of this, we shall not repeat the transformation from graphs to C'-graphs.

4. Edge Replacement Techniques

In this section we consider transformations which transform graphs by replacing single edges with instances of a more general subgraph.

4.1 Labeled Graphs and Pseudographs

A *labeled graph* is a graph with labels associated with its vertices and edges. A *pseudograph* is a graph with (possibly) multiple edges and loops. We here reduce labeled graphs to pseudographs, and thence to graphs. One transforms a labeled graph into a pseudograph by first sorting the vertex labels on the graph. When checking isomorphism, a necessary condition for two labeled graphs to be isomorphic is that the sorted lists for the two graphs be identical. In the pseudograph, a vertex will have a number of loops determined by the rank of the vertex's label in the sorted list. One next sorts edge labels and in a similar manner the multiplicity of an edge in the pseudograph is the rank of the edge's label in the sorted list. We can therefore represent labeled graphs as pseudographs. We now reduce pseudographs to graphs.

THEOREM: Pseudograph isomorphism is isomorphism complete.

PROOF:

Observe that the *subdivision graph* of a pseudograph (every edge is split into two edges and a new vertex is inserted between them) is a graph without loops. The subdivision graph of this multigraph is a graph. The resulting graph uniquely represents the pseudograph (up to isomorphism), and is created in polynomial time from the pseudograph.

An alternate transformation is the transform *LONG* which replaces each edge (including loops) of an n-vertex pseudograph by a path of length $n+1$. □

Yet another proof of this result can be found in[34].

4.2 Directed Graphs

A *directed graph* (digraph) is a set V of vertices and a set E of arcs, where E is a subset of V^2 and loops are not allowed.

THEOREM: Digraph isomorphism is isomorphism complete.

PROOF:

Representing graphs by digraphs can be done by replacing each undirected edge $<x,y>$ of the graph by arcs $<x,y>$ and $<y,x>$

Representing digraphs by graphs can be done by replacing each arc $<x,y>$ of the digraph with the subgraph shown in Figure 4.2.1. □

A similar construction is given in[54].

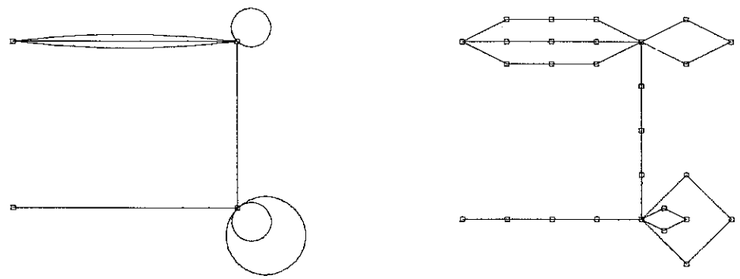

FIGURE 4.1: THE PSEUDOGRAPH CONSTRUCTION

4.3 Oriented Graphs

An *oriented graph* is a digraph in which the presence of the arc $\langle x,y\rangle$ precludes the presence of $\langle y,x\rangle$. In light of the isomorphism completeness of digraph isomorphism, we need only show

THEOREM: Graph isomorphism is polynomial time reducible to oriented graph isomorphism.

PROOF:

Replace each edge $\langle x,y\rangle$ of the given graph by the graph shown in Figure 4.3.1. The resulting digraph is oriented, and the transformation has the three desired properties. □

4.4 Acyclic Rooted Digraphs

An *acyclic rooted digraph* is a digraph having no directed cycles, with a distinguished vertex called the root. The oriented graphs constructed in the previous section are acyclic. Rooting the digraph uniquely is done by adding a new vertex s to the oriented graph and directing arcs from s to each source of the

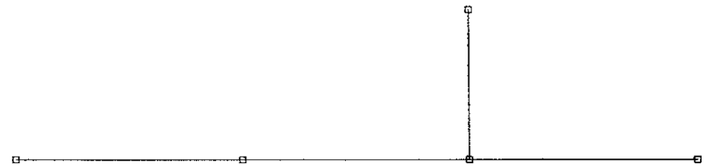

FIGURE 4.2.1

oriented graph. The resulting acyclic oriented graph is then rooted at s. We conclude the following.

THEOREM: [1, Problem 10.26] Acyclic rooted digraph isomorphism is isomorphism complete.

4.5 Bipartite Graphs

A *bipartite graph* is a graph whose vertex set can be partitioned into two classes, both being a set of pairwise nonadjacent vertices.

THEOREM: Bipartite graph isomorphism is isomorphism complete.

PROOF:

Given a graph G, we transform it to its subdivision graph $S(G)$. Subdivision graphs are bipartite and the transformation has the three desired properties. □

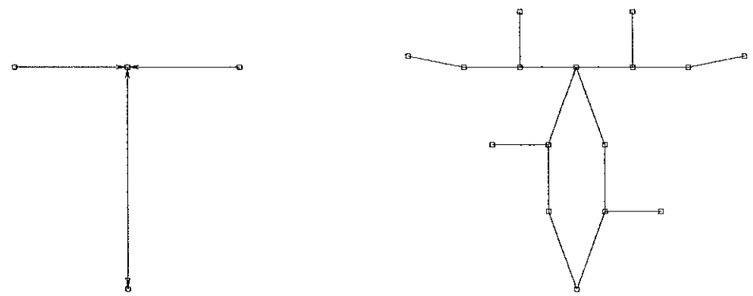

FIGURE 4.2.2: THE DIRECTED GRAPH CONSTRUCTION

4.6 Chordal Graphs

A *chordal graph* is a graph having no induced subgraph isomorphic to a cycle of length greater than three. Chordal graph isomorphism is isomorphism complete [49]. One transformation which demonstrates this maps a graph G to its subdivision graph, and then connects all of the vertices of $S(G)$ which were vertices in G. Stewart [63] notes that this construction also demonstrates the isomorphism completeness of split graph isomorphism. A *split graph* is a graph whose vertex set can be partitioned into two classes, one containing pairwise adjacent vertices, and the other containing pairwise nonadjacent vertices.

4.7 Graphs with a Forbidden Induced Subgraph

Let $F(H)$ be the family of graphs having no induced subgraph isomorphic to H.

THEOREM: [20] Testing isomorphism of graphs in $F(H)$ is isomorphism complete unless H is an induced subgraph of the path on four vertices.

PROOF:

FIGURE 4.3.1

Observe first that testing isomorphism in $F(H)$ is polynomially equivalent to testing isomorphism in $F(\overline{H})$. Observe next that either H or $\overline{H}$ is cyclic unless H is an induced subgraph of the path on four vertices. We therefore assume without loss of generality that H is cyclic.

Given a graph G, transform G by replacing each edge by a path of length $|V(H)|+1$, where $V(H)$ is the vertex set of H. The result has no cycles small enough to be cycles in H; thus the result contains no induced subgraph isomorphic to H. □

Testing isomorphism of graphs containing no induced paths on four vertices (cographs) can be done in polynomial time [45,63]. All other induced subgraphs of the path on four vertices result in trivial isomorphism problems.

4.8 Transitively Orientable Graphs

An oriented graph is *transitive* if the existence of arcs $<x,y>$ and $<y,z>$ necessitates the presence of $<x,z>$. A *transitively orientable graph* is a graph which is isomorphic to the underlying graph of some transitive oriented graph. Booth and Lueker [13] show that isomorphism testing for transitively orientable graphs is isomorphism complete. We give an alternate proof here.

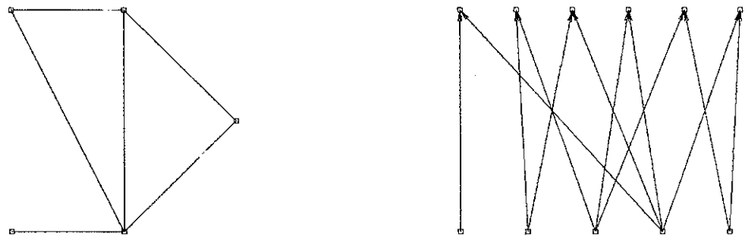

FIGURE 4.3.2: THE ORIENTED GRAPH CONSTRUCTION

THEOREM: Transitively orientable graph isomorphism is isomorphism complete.

PROOF:

Given a graph G, transform it to its subdivision graph $S(G)$. One transitive orientation of $S(G)$ orients edges from the original vertices to the added vertices (as in 4.3). □

4.9 Series-Parallel Digraphs

A digraph is *transitive series-parallel* (TSP) if it is a single vertex, or the series composition of two TSP digraphs, or the parallel composition of two TSP digraphs. The parallel composition is the usual disjoint union of the two digraphs. The series composition of D and F is their union followed by the addition of arcs from each vertex of D to each vertex of F.

A digraph is *series-parallel* if its transitive closure is TSP.

THEOREM: Series-parallel digraph isomorphism is isomorphism complete.

PROOF:

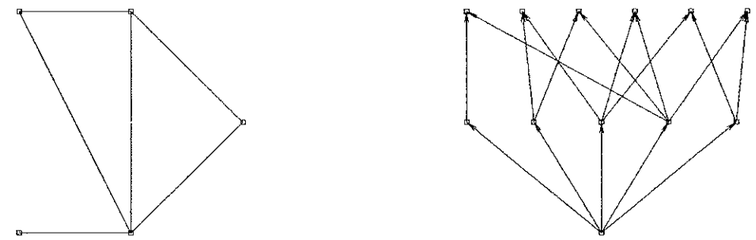

FIGURE 4.4: ROOTED ACYCLIC DIGRAPHS

Given a graph G, construct a series-parallel digraph by taking the oriented graph constructed in section 4.3 and adding a new vertex v. Arcs are added from each original vertex to v, and from v to each added vertex. The result is a series-parallel digraph, since its transitive closure is obviously transitive series-parallel. □

This result is especially interesting in light of the existence of a polynomial time isomorphism test for TSP digraphs [43].

4.10 Undirected Path Graphs

Consider a tree T and a set of undirected paths which are subgraphs of T. We represent each path by a vertex, and connect two vertices when the corresponding paths intersect in T. *Undirected path graphs* are graphs which can be represented in this manner by paths in some tree [58].

THEOREM: [12] Undirected path graph isomorphism is isomorphism complete.

PROOF:

Given a graph G, we construct an undirected path graph $UP(G)$ as follows. We first create the subdivision graph of G (as in section 4.5). We then connect

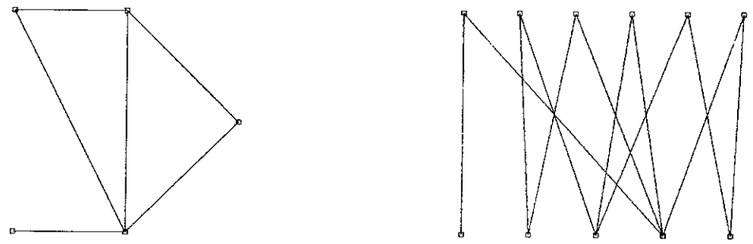

FIGURE 4.5: THE BIPARTITE GRAPH CONSTRUCTION

each pair of new vertices added in the construction.

The crucial step in the proof is to show that $UP(G)$ is an undirected path graph. Consider the dominant (maximal) cliques of $UP(G)$. The added vertices (representing the edges of G) form a dominant clique, and each original vertex together with the edge vertices adjacent to it form a dominant clique. A *clique tree* for $UP(G)$ is an n-star whose centre is the clique containing the edge vertices and whose leaves are the other cliques.

Each original vertex in $UP(G)$ lies in one dominant clique of $UP(G)$; each edge vertex lies in three dominant cliques. In either case, the cliques in the clique tree of $UP(G)$ containing a vertex of $UP(G)$ induce a path in the clique tree.□

5. Composition techniques

In section 4, we described transformations which substituted fixed graphs (e.g., paths) for edges in the given graph. In this section, we turn our attention to transformations which substitute the given graph into a fixed graph. A *substitution* for a vertex v by a graph S is defined to be the replacement of v by the vertices and edges of S, followed by the addition of edges connecting each

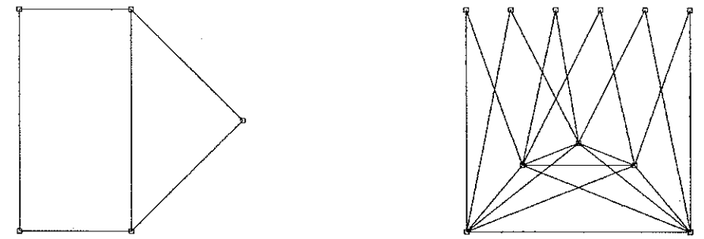

FIGURE 4.6: THE CHORDAL GRAPH CONSTRUCTION

vertex of S with each vertex which was adjacent to v.

5.1 Compact Graphs

A graph is *compact* when the distance between any two vertices is either one or two.

THEOREM: [30] Compact graph isomorphism is isomorphism complete.

PROOF:

Given a graph G, define $COMP(G) = C_5[G]$ (equivalently, C_5 with G substituted for each vertex). $COMP(G)$ is compact; it uniquely represents G as follows. A copy of G can be obtained by taking any vertex v and all vertices sharing at least $2n$ adjacencies with v. □

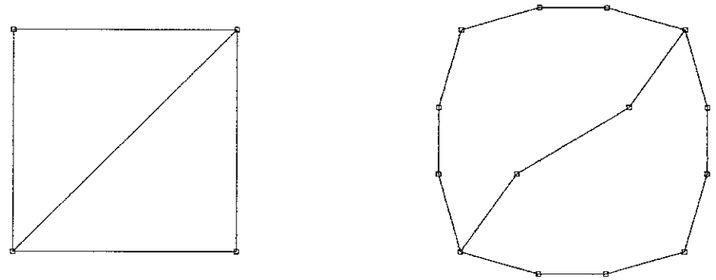

FIGURE 4.7: GRAPHS WITHOUT TRIANGLES

5.2 Composite Graphs

Corneil [29] observed that a more general proof is possible which does not rely on properties of C_5. A graph G is a *composite graph* if there exist graphs C and H for which G is isomorphic to C[H].

THEOREM: Isomorphism of composite graphs is isomorphism complete.

PROOF:

Consider an arbitrary fixed graph C with maximum degree d and any connected n-vertex graph G having no vertices of degree exceeding $(n\text{-}1)/2$. $C[G]$ is a composite graph and uniquely represents G as follows. Select a vertex v with maximum degree in $C[G]$. Consider the graph induced on the vertices sharing at least $2n$ adjacencies with v. Each connected component of this graph is a copy of G. □

It should be pointed out that the isomorphism completeness of composite graph isomorphism follows from the theorem in 5.1; we include the present result simply as an interesting proof technique.

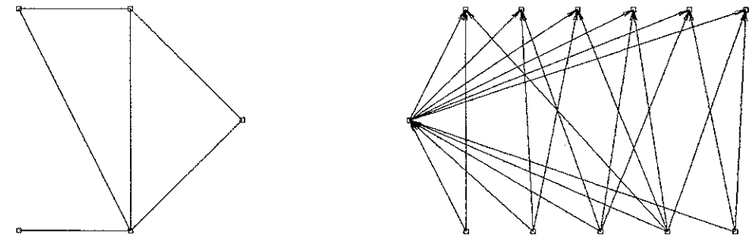

FIGURE 4.9: SERIES-PARALLEL DIGRAPHS

5.3 Self-complementary graphs and digraphs

A graph is *self-complementary* if it is isomorphic to its complement.

THEOREM: [18] Self-complementary graph isomorphism is isomorphism complete.

PROOF:

Define $SC(G)$ to be the result of substituting G for vertices 1, 4, 5, and 8, and substituting $\overline{G}$ for vertices 2, 3, 6, and 7 in the graph S displayed in Figure 5.3. $SC(G)$ is self-complementary and uniquely represents G. □

5.4 Regular self-complementary graphs and digraphs

We can state an even stronger result which subsumes the theorem in section 5.3.

THEOREM: [19] Regular self-complementary graph isomorphism is isomorphism complete. □

Corresponding isomorphism completeness results for digraphs are also

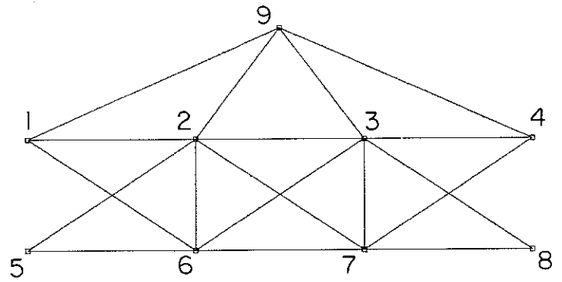

FIGURE 5.3: THE SELF-COMPLEMENTARY GRAPH S

proved in [19].

5.5 Graphs with a Unique Centre

A *centre* in a graph is a vertex whose maximal distance to any vertex is minimal (i.e., a vertex of minimal eccentricity [37]). Tree isomorphism algorithms operate by first rooting the tree at its unique centre. With this in mind, one might suspect that graphs with a unique centre would be easier for isomorphism testing (see, for example, [61]). This is not the case.

THEOREM: Isomorphism of graphs with a unique centre is isomorphism complete.

PROOF:

Given a graph G, substitute a copy of G for both leaves in the path on three vertices. The centre of the path is the unique centre of the resulting graph. □

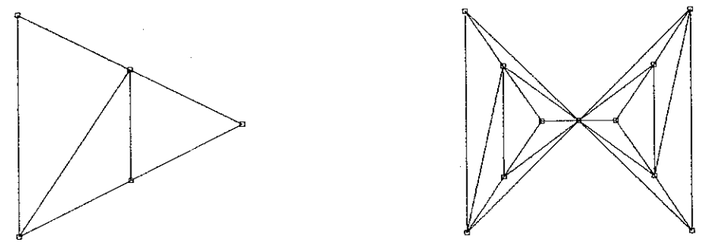

FIGURE 5.5: GRAPHS WITH A UNIQUE CENTRE

5.6 Cospectral Graphs

The *spectrum* of a graph is the set of eigenvalues of its adjacency matrix. Two graphs are *cospectral* if the characteristic polynomials of their adjacency matrices are identical.

We can immediately conclude that cospectral graph isomorphism is isomorphism complete, as follows. Given two graphs, we compute their spectra in polynomial time (via the usual algorithm using Gaussian elimination [1]). If their spectra differ, the graphs are nonisomorphic; otherwise, the graphs are cospectral and we may apply the tests for cospectral graph isomorphism.

This reduction is, in some ways, unsatisfying. It does not tell us that pairs of cospectral graphs occur frequently. We give here a construction which, given two arbitrary graphs, represents them by a cospectral pair of graphs which are isomorphic if and only if the given graphs are.

THEOREM: Cospectral graph isomorphism is isomorphism complete.

PROOF:

Let C be the graph shown in Figure 5.6. Given two graphs G and H, let L be the graph obtained by substituting G for vertices 1, 2, and 3 and H for vertices 4, 5, and 6 in C. Let R be the result of substituting H for vertices 1, 2, and 3 and G for vertices 4, 5, and 6 in C. The graphs L and R are cospectral for any choice of G and H [60].

Further, L and R are isomorphic if and only if G and H are isomorphic. □

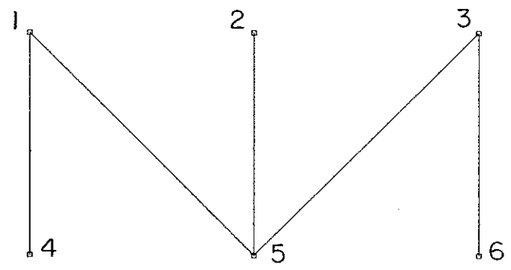

FIGURE 5.6: COSPECTRAL GRAPHS -- THE GRAPH C

6. Other Techniques

In this section we consider a pot-pourri of other transformation techniques used for restricted classes of graphs.

6.1 Line Graphs

The *line graph* of a graph $G(V,E)$ is the graph $H=(E,F)$ in which Whitney [66] examined the question of when an isomorphism of the edges of two graphs necessitated that the graphs be isomorphic. He proved that the only two graphs which are edge-isomorphic but not (vertex) isomorphic are the 3-star and the triangle. His result has as a corollary that

THEOREM: Line graph isomorphism is isomorphism complete. □

6.2 Regular Graphs

A *regular graph* is a graph in which every vertex has the same degree. Booth established that

THEOREM: [11] Regular graph isomorphism is isomorphism complete. □

Booth's construction for regular graphs significantly increases the maximum degree of the graph. Miller described a construction which does not have this behaviour.

THEOREM: [54] Isomorphism of graphs with maximum degree d is polynomial time reducible to isomorphism of regular graphs of degree d, whenever d is odd. □

Miller's result was later improved:

THEOREM: [30,56] Isomorphism of graphs with maximum degree d is polynomial time reducible to isomorphism of regular graphs of degree d.

PROOF:

A graph $T(d)$ is defined; $T(d)$ has all vertices of degree d except two, s and t, which have degree $d-1$. Further, s and t must be similar in $T(d)$.

Given a graph G, we take two copies of it, G_1 and G_2. Each vertex x of G corresponds to a vertex x_1 of G_1 and a vertex x_2 of G_2. For each vertex x in G, we take $d-deg(x)$ copies of $T(d)$ and connect x_1 to the 's' in each, and x_2 to the 't' in each. The result is regular of degree d, and uniquely determines G. □

6.3 Regular Bipartite Graphs

THEOREM: [56] Isomorphism of graphs with maximum degree d is polynomial time reducible to isomorphism of bipartite graphs which are regular of degree d. □

6.4 Marked Graphs and Marked Trees

A *marked graph* is a graph together with a partition of its vertices. An isomorphism of marked graphs is an isomorphism of the underlying graphs which preserves the partitioning.

THEOREM: [31] Marked graph isomorphism is isomorphism complete.

PROOF:

A graph is a marked graph in which every vertex belongs to its own class. We therefore need simply represent marked graphs as graphs. We will in fact create a labeled graph, which we have shown is equivalent (section 4.1). This labeled graph contains a vertex labeled '0' for each vertex in the marked graph, and a vertex labeled '1' for each class in the partitioning of the marked graph.

Edges connect the '0' vertices as in the marked graph. Additional edges connect a '1' vertex to the vertices contained in the corresponding class of the marked graph. □

THEOREM: [31] Marked tree isomorphism is isomorphism complete.

PROOF:

In light of the previous result, we need only show how to represent graphs as marked trees. Given a directed graph G, we number its vertices 1 through n. Each arc is assigned a number equal to the number of the vertex towards which it is directed. Each arc is then replaced by a vertex adjacent to the source of the arc; this vertex is assigned the arc's number. Finally, a vertex x is added and connected to each of the original vertices; x is given the number $n+1$. The result is a tree, and the vertex numbering induces a partition of its vertex set. Thus the result is a marked tree; further, G can be recovered by deleting the only vertex in a singleton class and then identifying all vertices in the same class. □

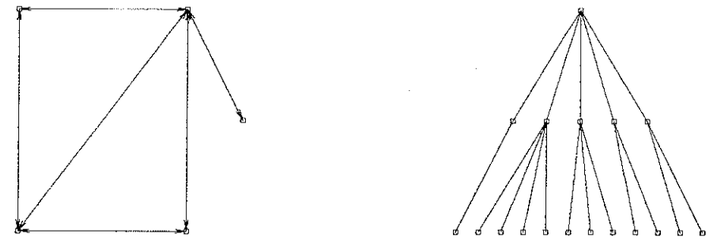

FIGURE 6.4: THE MARKED TREE CONSTRUCTION

6.5 K-trees

A *k-tree* is recursively defined as follows. K_k is a k-tree. If G is a k-tree, the result of adding a vertex adjacent to every vertex in a k-clique of G is a k-tree.

THEOREM: [31] k-tree isomorphism is isomorphism complete. □

This result is especially interesting when one considers that if k is a fixed constant, k-tree isomorphism can be tested in polynomial time [31].

6.6 C-Subgraph Regular Graphs

A graph G is c-subgraph regular if, for any S with at most c vertices, the number of ways of embedding S in G for which a particular vertex p of S is mapped to a particular vertex v of G depends only on p and S.

THEOREM: [30] For fixed constant c, c-subgraph regular graph isomorphism is isomorphism complete. □

A *Zykov regular graph* [14] is a graph having the property that the neighbours of a vertex induce a graph which is independant of the particular vertex chosen. The c-subgraph regular graphs constructed by Corneil and Kirkpatrick are Zykov regular; the induced graph is void. Corneil [29] observed that even if one excludes void and complete neighbourhoods, Zykov regular graph isomorphism is isomorphism complete. Given a graph G and the c-subgraph regular graph H which represents it, $C_5[H]$ is Zykov regular and the neighbourhood is neither void nor complete; further, it uniquely represents G.

6.7 Eulerian Graphs

A graph is *Eulerian* if one can start at a vertex, walk along every edge of the graph and return to the starting point never walking along an edge twice.

THEOREM: Eulerian graph isomorphism is isomorphism complete.

PROOF:

Given a connected graph G, take two copies of G, G_1 and G_2. Each vertex x of G is associated with a vertex x_1 in G_1 and a vertex x_2 in G_2. When x has odd degree, add a new vertex connected to both x_1 and x_2. When the degree of x is even, add two new vertices, both connected to x_1 and both to x_2. The resulting graph is connected and all degrees are even − thus the resulting graph is Eulerian. □

An easier proof technique is to use previous constructions to transform graphs into connected graphs (section 2.1) and thence to connected graphs which are regular of even degree (section 6.2).

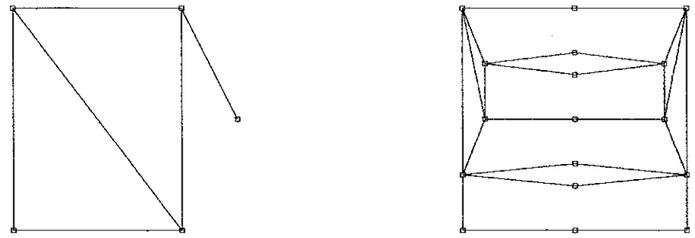

FIGURE 6.7: THE EULERIAN GRAPH CONSTRUCTION

6.8 Hamiltonian Graphs

An n-vertex graph is *Hamiltonian* if it has as a subgraph the cycle on n vertices.

THEOREM: Hamiltonian graph isomorphism is isomorphism complete.

PROOF:

Given a graph, construct an Eulerian graph from it, and then take the line graph of this Eulerian graph. The result is Hamiltonian [37]. □

ALTERNATE PROOF:

Given any graph G, observe that $G+G$ has $K_{n,n}$ as a subgraph; thus, $G+G$ is Hamiltonian. In addition, $G+G$ uniquely represents G. □

6.9 Hypergraphs

A *hypergraph* is a set V of vertices together with a subset E of the powerset of V.

THEOREM: Isomorphism of hypergraphs is isomorphism complete.

PROOF:

All graphs are hypergraphs; we need thus only show that we can uniquely encode a hypergraph as a graph. We construct a labeled graph with a vertex labeled 'v' for every vertex of the hypergraph and a vertex labeled 'e' for every edge. A 'v' vertex is connected to an 'e' vertex if and only if the vertex corresponding to the 'v' vertex belongs to the edge corresponding to the 'e' vertex. □

7. Isomorphism of Algebraic Structures

7.1 Algebras

An *algebra* is a set A together with some fixed constant number of relations. A relation is a subset of A^A. Miller and Monk [54] proved the following

THEOREM: Algebra isomorphism is isomorphism complete.

PROOF:

Since graphs are algebras, it suffices to uniquely represent an algebra as a digraph. The digraph has a vertex for each element in the algebra. Each tuple in each defining relation is encoded; a tuple $<x_1, \cdots, x_k>$ in the i'th defining relation is represented by a subgraph involving the k vertices representing the elements, as shown in Figure 7.1. □

7.2 Lattices and Posets

A *lattice* is a set of elements together with a partial order $<$, with a unique maximal element and a unique minimal element. Element e dominates f if $f < e$. The smallest element dominating any two (their *join)* and the largest element dominated by any two (their *meet)* are each unique.

In 1950, Frucht [35] demonstrated that any abstract group is isomorphic to the group of some lattice. In so doing, he proved that

THEOREM: Lattice isomorphism is isomorphism complete.

PROOF:

It suffices to represent a graph uniquely as a lattice. Given a graph G with n vertices and m edges, we define a lattice with an element for each vertex, an element for each edge, and two additional elements '0' and '1'. Element '1'

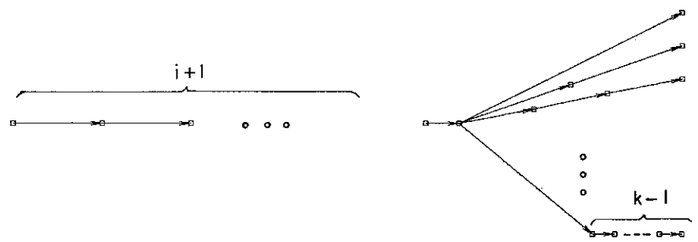

FIGURE 7.1: THE ALGEBRA CONSTRUCTION

dominates all others (the *supremum),* and element '0' is dominated by all other elements (the *infimum*). An edge dominates exactly those vertices which are its endpoints. The result is a lattice which uniquely represents G. □

7.3 Semigroups

A *semigroup* is a set of elements together with a binary operation. This result, proved by Booth [11], is proved in a simpler manner here.

THEOREM: Semigroup isomorphism is isomorphism complete.

PROOF:

Lattices are semigroups under the binary meet operation. Thus lattice isomorphism is reducible to semigroup isomorphism, so the result is established. □

In fact, lattices are commutative semigroups, as are the semigroups constructed in [11].

7.4 Finitely Presented Algebras

A *finitely presented algebra* is, roughly speaking, an algebra with a finite number of generators, a finite number of operators, and a finite set of equivalences among the applications of the operators to the generators.

THEOREM: [41] Isomorphism of finitely presented algebras is isomorphism complete. □

7.5 Automata and Semiautomata

An *automaton* is a 5-tuple comprised of a set of states, a start state, a set of final states, an input alphabet, and a transition function (see [1] for details). A *semiautomaton* is a 3-tuple comprised of an set of states, an input alphabet, and a transition function.

THEOREM: [11] Automaton isomorphism is isomorphism complete. □

THEOREM: [11] Semiautomaton isomorphism is isomorphism complete even if the alphabet is binary. □

8. Combinatorial Designs

8.1 Partially Balanced Incomplete Block Designs

A *partially balanced incomplete block design* (PBIBD) with parameters (v,b,r,k,L) is a collection of b k-sets called blocks chosen from a v-set; each element appears in exactly r blocks and each pair of elements appear in exactly l blocks, for some l in L.

THEOREM: [24,29] Isomorphism of PBIBD's is isomorphism complete.

PROOF:

Given a graph, we construct a regular graph G with n vertices and e edges representing it. From G we construct a PBIBD D with e elements, the edges of G. For each vertex v of G, the edges incident with v form a block of D.

Since G is regular, all blocks have the same number of elements and each element of D appears the same number of times. Finally, each pair of elements appears either 0 or 1 times; thus $L=\{0,1\}$. □

8.2 Set Packings

A *set packing* with parameters (v,k,λ) is a collection of k-subsets (blocks) of a v-set for which any two blocks intersect in at most λ elements [17].

THEOREM: [24] Isomorphism of set packings is isomorphism complete.

PROOF:

Given a graph we construct a PBIBD from it as in section 8.1. We then construct the *dual* of the PBIBD as follows. The elements of the dual are the blocks of the PBIBD. For each element e of the PBIBD, the blocks containing e form a block of the dual.

The dual of a PBIBD is a set packing. □

8.3 (r,λ)-Systems

An *(r,λ)-system* on v elements is a collection of subsets (blocks) of any size of a v-set for which every element appears in r blocks and every pair of elements appear in λ.

THEOREM: [24] Isomorphism of (r,λ)-systems is isomorphism complete.

PROOF:

Given a graph we construct an (r,λ)-system S as follows. The elements of S are the vertices of the graph and one additional element a. For each edge (x,y) of the graph, (a,x,y) is a block of S. One adds sufficient blocks of size 2 to ensure that every pair appears the same number of times and finally adds sufficient blocks of size 1 to ensure that every element appears the same number of times. □

8.4 Exact Set Packings

An *exact set packing* with parameters (v,k,λ) is a collection of k-subsets (blocks) of a v-set for which any two blocks intersect in exactly λ elements.

THEOREM: [24] Isomorphism of exact set packings is isomorphism complete.

PROOF:

The dual of an (r,λ)-system is an exact set packing. □

8.5 Pairwise Balanced Designs

A *pairwise balanced design* (PBD) is an (r,λ)-system in which blocks of size 1 are forbidden.

THEOREM: [20] Isomorphism of PBD's is isomorphism complete.

PROOF:

We will construct a PBD from a regular self-complementary graph which we have shown is equivalent (section 5.4). Given an n-vertex sc graph $G=(V,E)$ which has q edges and is regular of degree d, we construct a PBD D as follows. The set of elements of D consists of V and an n-set X disjoint from V. For each edge $\{v,w\}$ of E and each pair of distinct elements $<x,y>$ of X^2 the set $\{v,w,x,y\}$ is a block of D. Each nonedge $\{v,w\}$ of G $(v \neq w)$ forms a block of D; it is repeated $2q$ times. Each pair of distinct elements in X^2 is a block of D; they are repeated q times. Finally, for each element x of X and each vertex v of G, (v,x) is a block of G; it is repeated $2q - 2d^2$ times.

D is a PBD. It uniquely determines G as follows: when all blocks of size 2 are discarded, the remaining pairs appearing $2q$ times are precisely the edges of G. □

9. An Equivalent Clique Problem

Until this point, all of the problems considered have been isomorphism problems. It is of considerable interest to determine problems which are isomorphism complete and yet not transparently isomorphism problems.

One such problem was described first by Levi [46], and later by Barrow and Burstall [6]. Kozen [42] first observed the importance of this construction as it concerns the complexity of graph isomorphism, answering a question posed in [11]. We follow Kozen's presentation here.

An *M-graph* of order n^2 is any graph G with n^2 vertices $\{v_{i,j} \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant n\}$, satisfying

(1) $(x_{i,j},x_{k,l})$ is an edge of G only if $i \neq k$ and $j \neq l$.

(2) $(x_{i,j},x_{k,l})$ is an edge of G only if $(x_{k,j},x_{i,l})$ is also an edge.

(3) Any two quadrilaterals of G are connected by an even number of edges.

THEOREM: [42] Finding an n-clique in an M-graph with n^2 vertices is isomorphism complete.

PROOF:

We first reduce graph isomorphism to finding an n-clique. Given two n-vertex graphs G and H define a graph M for which $V(M) = V(G)*V(H)$, and $E(M) = \{\{<v,a>,<w,b>\} \mid (v,w)$ is an edge in G if and only if (a,b) is an edge in $H\}$. M is an M-graph, and has an $n$clique if and only if G and H are isomorphic.

We now reduce the clique problem for M-graphs to graph isomorphism. Given an M-graph K with n^2 vertices $x_{i,j}$, we define two graphs, G and H (it is not clear to the present authors how one can find the indexing of the $x_{i,j}$ in polynomial time, unless it is given). G has vertex set v_i, and H has vertex set w_j. The edges of G are as follows – v_i is connected to v_j if and only if $(w_{i,1}, w_{j,2})$ is an edge of K. The edges of H are given by the following rule – w_i is connected to w_j if and only if either $(x_{1,i}, x_{2,2})$ is an edge of K and $(x_{1,j}, x_{2,j})$ is an edge of K, or neither are edges of K.

G and H are isomorphic if and only if K has a n-clique. □

An interesting aspect of this result, noted in [42], is that finding a $(1-\epsilon)n$ clique in an n^2 vertex M-graph is NP-complete for ϵ arbitrarily small.

Further, one should note that the Babai-Mathon result from section 2.4 demonstrates that we can *count* the number of n-cliques in an M-graph using an isomorphism algorithm. The problem of counting cliques in graphs in general is #P-complete [65]. This constitutes further evidence that graph isomorphism is not NP-complete.

10. Other Problems

In this section we describe three other problems which are isomorphism complete, but are not isomorphism problems in the restrictive sense used in sections 4 through 7.

10.1 Recognition of Self-complementary Graphs

THEOREM: [18,19] Deciding whether a graph is self-complementary is isomorphism complete.

PROOF:

Recognizing self-complementary graphs is reducible to graph isomorphism; one simply asks whether the graph is isomorphic to its complement.

Graph isomorphism is reducible to recognition of self-complementary graphs as follows. Given graphs G and H, take the graph S in Figure 5.3. Substitute G for vertices 1 and 4, H for 6 and 7, $\overline{G}$ for 2 and 3, and $\overline{H}$ for 5 and 8. The resulting graph is self-complementary if and only if G and H are isomorphic. □

10.2 Regular Legitimate Decks

The *deck* of a graph G is the set of one-vertex deleted subgraphs of G. A deck of n graphs each having $n-1$ vertices is *legitimate* if and only if it is the deck of some graph. A *regular deck* is a deck in which every graph has the same degree sequence. Necessary and sufficient conditions for a regular deck to be legitimate have been the objective of much research in the area of reconstruction theory [8].

THEOREM: [62] Deciding whether a regular deck is legitimate is isomorphism complete.

PROOF:

Given a regular deck $D = \langle G_1, \ldots, G_n \rangle$, create n graphs $H_1, \ldots, H_n$ as follows. H_i is created from G_i by adding a vertex to G_i and connecting the added vertex to all vertices not having maximum degree in G_i. Then D is a legitimate deck if and only if H_i and H_j are isomorphic for all i and j.

Given two regular graphs G and H, let $D = \langle H_1, \ldots, H_n \rangle$ be the deck of H. Let $E = \langle H_1+G, \cdots, H_n+G \rangle$. Finally, let F be two copies of E. F is a regular deck, and is legitimate if and only if G and H are isomorphic. □

10.3 Hadamard Equivalence

We define four matrix operations:

(1) interchanging two rows,

(2) interchanging two columns,

(3) multiplying each entry in a row by -1,

(4) multiplying each entry in a column by -1.

Two $(-1,1)$ matrices are *Hadamard equivalent* if one can be obtained from the other by a finite sequence of operations of types (1)-(4).

THEOREM: [53] Hadamard equivalence of $(1,-1)$ matrices is polynomial time reducible to graph isomorphism.

PROOF:

Given a $(1,-1)$ matrix $H = (h_{i,j})$, we construct a graph $HAD(H)$ as follows. There are two vertices v_i and v'_i for the i'th column and two vertices w_j and w'_j for the j'th row. The edges of $HAD(H)$ are

(v_i, w_j) and (v'_i, w'_j) if $h_{i,j} = 1$

(v_i, w'_j) and (v'_i, w_j) if $h_{i,j} = -1$

Two $(1,-1)$ matrices X and Y are Hadamard equivalent if and only if $HAD(X)$ and $HAD(Y)$ are isomorphic. □

The converse also holds.

THEOREM: [22] Graph isomorphism is polynomial time reducible to Hadamard equivalence of $(1,-1)$ matrices.

PROOF:

Given a graph G, let E be the n by m incidence matrix of G, with 1 replaced by -1 and 0 replaced by 1. Define the matrix $N(G)$ as follows. $N(G)$ is $2n$ by $2m$, and

$$n_{i,j} = \begin{cases} 1 \text{ if } i>n \text{ or } j>m \\ e_{i,j} \text{ otherwise} \end{cases}$$

If $N(G)$ and $N(H)$ are Hadamard equivalent, there is some sequence of operations of type (1)-(4) mapping $N(G)$ into $N(H)$. If a row is multiplied by -1 in the mapping, it must either be multiplied by -1 again (in which case both operations can be removed), or at least $2n-2$ columns must be multiplied by -1.

This must be done in order to preserve the number of 1's in the row. It is clear that multiplying $2n-2$ columns by -1 cannot be required, since the matrix has a majority of 1 entries.

A similar argument demonstrates that multiplication of columns by -1 can be ignored. □

10.4 The Star-System Problem

The *star* of a vertex v, denoted by $*v$, consists of v together with its neighbours, and the edges connecting v to its neighbours. The *star-system* of a graph is the family set $\{*v \mid v \text{ is in } V(G)\}$. The star-system problem is to decide, given a collection of subsets (a set-system), whether it is the star-system of some graph.

THEOREM: [5] The star-system problem is isomorphism complete.

PROOF:

Given a set-system S, let $I(S)$ be its incidence graph. $I(S)$ is bipartite, and has an automorphism interchanging the two classes of the bipartition which carries each vertex to a vertex adjacent to it whenever S is a star-system. We omit the remaining details.

Given a pair of graphs, G and H, assume both have at least two vertices. Let $G'=S(COMP(G))$ and $H'=S(COMP(H))$. Now $G'(H')$ is a bipartite graph with vertices of degree $\geqslant 3$ in one class of the bipartition and vertices of degree 2 in the other class. Let X be the disjoint union of G' and H' with additional edges connecting each vertex of degree 2 in G' to each vertex of degree $\geqslant 3$ in H', and similarly for H' and G'.

X has an automorphism interchanging the two classes of its bipartition which carries each vertex onto an adjacent vertex if and only if G' and H' are isomorphic (and thus, G and H are isomorphic).

Consider the stars obtained from one class of the bipartition of X. These stars form a star system if and only if there is an automorphism of X which is of the desired type. □

11. An NP-complete Automorphism Problem

An automorphism is *fixed point free* (fpf) if it fixes no vertex. Recently, Lubiw [48] proved that deciding whether a graph has a fixed point free automorphism is NP-complete. This constitutes the first example of an NP-complete problem which involves the automorphisms of a graph.

THEOREM: [48] Deciding whether a graph has an fpf automorphism is NP-complete.

PROOF:

The problem is in NP. One simply "guesses" an fpf automorphism; in polynomial time, it can be verified that the guess is an automorphism and that it fixes no vertex.

Completeness is shown by reducing 3-CNF satisfiability to the fpf problem. Given a formula, a graph is constructed which has an fpf automorphism if and only if the formula is satisfiable. We are given a formula C. C is the conjunction of m clauses $\{c_i \mid 1 \leqslant i \leqslant m\}$, and contains n variables $\{u_i \mid 1 \leqslant i \leqslant n\}$. Each clause is the disjunction of three variables or their negations. We assume without loss of generality that no clause is repeated.

We construct a labeled graph G as follows. G has vertex set consisting of two parts:

(1) $\{x_i, y_i, u_{i,1}, u_{i,2}, u_{i,2}{}^t, u_{i,1}{}^t, u'_{i,1}, u'_{i,2}, u'_{i,1}{}^t, u'_{i,2}{}^t \mid 1 \leqslant i \leqslant n\}$

(2) $\{c_{j,k} \mid 1 \leqslant j \leqslant m, 1 \leqslant k \leqslant 8\}$.

Labels are assigned to some of the vertices from a set of distinct labels $\{a_1, \cdots, a_n, b_1, \cdots, b_m\}$. The vertices x_i and y_i are given label a_i, for $1 \leqslant i \leqslant n$. The vertices $\{c_{j,k}\}$ are given label b_j, for $1 \leqslant j \leqslant m, 1 \leqslant k \leqslant 8$.

The edges of G are of two types, the first between vertices in (1), the second connecting vertices in (1) with vertices in (2). We build a gadget (subgraph) for each variable, and a gadget for each clause. The gadget for variable u_i is shown in Figure 11.1. Suppose (for instance) c_l is the disjunction of u_i, u_j, and u'_k. Figure 11.2 shows how a gadget for this clause is constructed.

The edge set of G consists of the union of the edge sets of the gadgets for the variables and the edge sets of the gadgets for the clauses.

Suppose G has an fpf automorphism F. It must then be the case that F moves each $u_{i,1}$. Since x_i and y_i are the only two vertices with label a_i, $u_{i,1}$ must map to one of $u_{i,2}$, $u_{i,1}{}^t$, or $u_{i,2}{}^t$. Clearly the latter is not a candidate since F would then fix x_i. Thus $u_{i,1}$ maps to either $u_{i,1}$ (an assignment of false to u_i) or to $u_{i,1}{}^t$ (an assignment of true to u_i). The reader can easily verify that the gadget for a variable ensures that if the variable is assigned true, its negation is assigned false. We conclude that F induces an assignment of truth values to the variables. We must now show that the assignment satisfies the formula. Consider a clause c_i of C. If none of the three variables in c_i is assigned true by F, then F must fix $c_{i,0}$. Thus some variable in the clause must be assigned true; thus the clause is satisfied. Therefore, each clause evaluates to true under the truth assignment induced by F; thus the formula is satisfiable.

We leave to the reader the proof that (i) a satisfying assignment to the variables of the formula induces an fpf automorphism of G, and (ii) the vertex labeling applied is not essential. □

COROLLARY: Counting the number of fpf automorphisms is #P-complete.

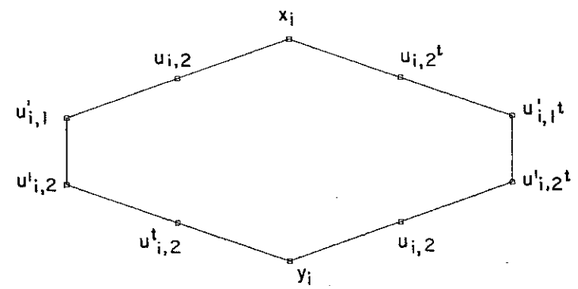

FIGURE 11.1: GADGET FOR VARIABLES

PROOF:

The satisying assignments to the variables of C are in 1-1 correspondence with the fpf automorphisms of G. □

One might suspect at first that a minor modification to Lubiw's construction would demonstrate that graph isomorphism is NP-complete. This does not appear to be the case. The problem of deciding whether a graph has an fpf automorphism remains NP-complete even given a set of generators for the automorphism group of the graph. Lubiw's construction can easily be modified to show that deciding whether a permutation group A contains a fixed point free permutation (a derangement), given a set of generators for A, is NP-complete. This latter problem remains NP-complete in the case when A is a 2-group (i.e., the set of generators contains only transpositions).

An isomorphism algorithm seems to be little help in deciding whether a graph has an fpf automorphism; these observations suggest that the difficult step is not finding a set of generators for the automorphism group, but rather deciding the existence of an fpf automorphism given this set.

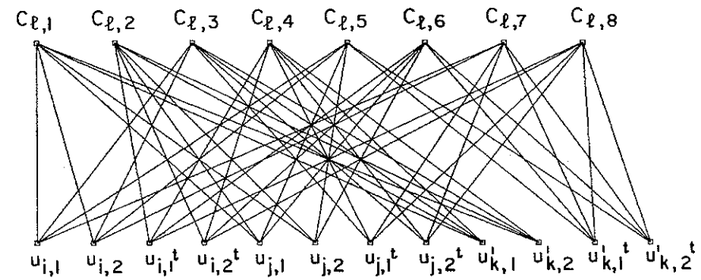

FIGURE 11.2: GADGET FOR CLAUSES

12. Open Problems

In this section we briefly describe some open problems concerning theoretical aspects of the graph isomorphism problem.

12.1 Canonizing or Coding Graphs

A *coding* or *certificate* for a graph is a function f for which $f(G)=f(h)$ if and only if G and H are isomorphic. It is clear that the existence of a polynomial time computable coding function would provide an efficient isomorphism test. The converse is an open problem. Does the existence of an efficient algorithm for graph isomorphism necessitate the existence of polynomial time computable certificates? (See [54].)

Many of the obvious candidates for coding functions are not suitable, as shown in [38].

12.2 Strongly Regular Graphs

Strongly regular graph isomorphism was early identified as an important subcase of the graph isomorphism problem [27]. Many hard graphs for practical isomorphism programs are strongly regular [51]. On the other hand, Babai has described an algorithm for strongly regular graph isomorphism which is a small improvement over the best current algorithms for graph isomorphism [4].

12.3 Transitive Graphs

Transitive graphs have automorphisms carrying each vertex onto every other vertex. Because of this, an algorithm operating by vertex refinement techniques (as many do, see [57]) will fail to reduce the number of candidate isomorphisms. No proof of isomorphism completeness for isomorphism of transitive graphs is known, however.

12.4 Rigid Graphs

At the other end of the scale from transitive graphs, rigid graphs are automorphism free. Nevertheless, the lack of symmetry does not seem to be of assistance. The hard graphs given in [51] are rigid. Moreover, it appears that new techniques are required to demonstrate isomorphism completeness in this case [19].

12.5 Block Designs

Isomorphism of block designs is intimately related to strongly regular graph isomorphism. Although some success in determining block design isomorphism has been achieved by counting subdesigns [36], the problem appears to be difficult [32].

Isomorphism testing in some cases does appear to be easier, however. Miller [55] gives a subexponential algorithm for isomorphism of Steiner triple systems (i.e, 2-$(v,3,1)$ designs), Latin squares, and projective planes (i.e., 2-$(n^2+n+1,n+1,1)$ designs). Marlene Colbourn [24] recently extended Miller's result to obtain a subexponential algorithm for t-$(v,t+1,1)$ designs; one important instance of designs with these parameters is Steiner quadruple systems.

Restrictions of the isomorphism problem for block designs have also been considered; of note in this area is the problem of equivalence of Hadamard matrices [26].

12.6 Tournaments

A *tournament* is an oriented complete graph. Tournament isomorphism appears to be as hard as graph isomorphism, but is not known to be isomorphism complete. The problem is discussed in [19].

12.7 Graphs of Fixed Genus

Planar graph isomorphism can be tested in polynomial time. In 1966, Lehman [44] conjectured that this behaviour extends to graphs of fixed constant genus.

12.8 Degree Bounded Graphs

Chemical "graphs", or molecules, have a fixed constant bound on the maximum degree of the graph. If we restrict our input graphs to have maximum degree at most some fixed constant k, does the problem remain isomorphism complete?

Common reduction techniques used to prove isomorphism completeness fail, as proved by Miller [54] and Kucera [3].

12.9 A Good Characterization

Edmonds introduced the concept of a *good characterization* of a problem; the following anecdote of Edmonds' brings home the importance of the concept. We paraphrase from Chvatal [16]:

"A rich and powerful King, driven by his aristocratic whims, desires to know whether a certain graph (with an awful lot of vertices and edges) has a 1-factor. He hires a team of mathematicians and they set to work. If they eventually find a 1-factor, they can just show it to the King; it won't take him long to see that he wasn't cheated. However, what happens if there is no 1-factor in the graph? Will the King take their word for it? Won't he suspect this bunch of intellectuals of doing nothing, only pretending to work? Spending their time at wild orgies? No. If G has no 1-factor, then there is a set S of vertices such that G-S has more than $|S|$ odd components; this is Tutte's theorem. If they cannot find a 1-factor, they had better find that set S. Needless to say, they may have a hard time finding it; however, it will be easy – for the King – to check that they did their job. And this is exactly what we mean when we say that Tutte's theorem provides a good characterization of graphs without 1-factors."

No good characterization of graph nonisomorphism is known.

A good characterization would imply that graph nonisomorphism is in NP; equivalently, it would imply that graph isomorphism is in co-NP. No NP-complete problem is known to be in co-NP. Thus, a good characterization of graph nonisomorphism would constitute strong evidence that graph isomorphism is not NP-complete [28].

12.10 Modular Lattices

Isomorphism of lattices is isomorphism complete [35], but isomorphism of distributive lattices is polynomial [23]. The class of modular lattices is properly contained in the set of all lattices, and properly contains the class of distributive lattices.

12.11 Subgraph Isomorphism

The subgraph isomorphism problem is to determine, given two graphs G and H, whether H is isomorphic to a subgraph of G. It has long been known to be NP-complete [25]. The only nontrivial restriction of subgraph isomorphism for which a polynomial time algorithm is known occurs when G and H are both trees [52]. When H is a tree and G is planar, the problem is NP-complete [67]; thus, extensions of the subtree isomorphism algorithm seem quite difficult.

However, an elementary extension of Tarjan's group isomorphism algorithm demonstrates that subgroup isomorphism can be tested in $O(n^{\log n})$ time. In fact, given two balanced incomplete block designs B_1 and B_2 with parameters (v,k,λ), determining whether B_2 is isomorphic to a subdesign of B_1 can be done in $O(D(v)\ v^{\log v})$, where $D(v)$ is the time required to test isomorphism of (v,k,λ) designs [24]. Thus a subexponential algorithm for block design isomorphism gives a subexponential algorithm for subdesign isomorphism.

Do there exist other nontrivial classes of structures also having this close relation between testing isomorphism of the structures and testing substructure isomorphism?

12.12 The Complexity of Graph Isomorphism

Lipton [47] observes that no asymptotic improvement on the $O(n!)$ brute force algorithm for graph isomorphism has appeared in the literature. Recently, an $O(c^n)$ algorithm for graph isomorphism has appeared [23]. Can this be improved?

12.13 Directed Path Graphs

Given a rooted directed tree T and a set of directed paths in T, we define a graph as follows. Each directed path is represented by a vertex; two vertices are adjacent if the corresponding directed paths share at least one vertex in T. A graph is a *directed path graph* if it represents directed paths in some rooted directed tree.

The class of directed path graphs properly contains the class of interval graphs and is properly contained in the class of undirected path graphs. Interval graph isomorphism can be tested in linear time [49] and undirected path graph isomorphism is isomorphism complete. How difficult is testing isomorphism of directed path graphs?

13. Probabilistic Methods

There has been much recent interest in the development of algorithms which yield an efficient isomorphism test for almost all graphs (in the asymptotic sense) [33,40,47].

These methods confirm our real world experience that testing isomorphism of garden variety graphs is usually easy. In practice, however, hard graphs for isomorphism are regular, or even strongly regular. Is there an efficient

isomorphism test for almost all regular graphs? strongly regular graphs?

Acknowledgements

We would like to thank Marlene Colbourn, Derek Corneil, and Anna Lubiw for their thorough reading and helpful comments on a preliminary version of this report. The research of the first author was supported by NRC Grant A4307; the research of the second author was partially supported by an NSERC Postgraduate Scholarship.

Appendix

In this appendix, we give a formal specification of the reductions described in the text of the paper. This is done for two reasons: to present reductions not presented in the text due to the difficulty in giving a concise English description of the reduction, and to help the reader if the English description in the text is found to be confusing or ambiguous.

We formally specify reductions as follows. We are *given* a structure to be transformed. When this structure is a graph, we assume it has vertex set $V=\{v_1, \ldots, v_n\}$ and edge set $E=\{e_1, \ldots, e_m\}$. The *result* is another structure, which we will specify in terms of its constituent parts. For example, a digraph is specified in terms of its vertices and arcs.

3.1 A sample proof

Given: an arbitrary graph with $n>2$
Result: a thin graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{w_i \mid 1 \leqslant i \leqslant n+2\}$
Edges:
(1) $\{e_1, \ldots, e_m\}$
(2) $\{(w_1, v_i) \mid 1 \leqslant i \leqslant n\}$
(3) $\{(w_i, w_{i+1}) \mid 1 \leqslant i \leqslant n+1\}$

4.1 Labeled Graphs and Pseudographs

Given: an arbitrary pseudograph
Result: a graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{w_{i,j} \mid 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n-1\}$
Edges:
(1) $\{(w_{i,j}, w_{i,j+1}) \mid 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n-2\}$
(2) $\{(v_i, w_{k,1}), (v_j, w_{k,n-1}) \mid e_k = (v_i, v_j)\}$

4.2 Directed graphs

Given: a directed graph
Result: a graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{w_i \mid 1 \leqslant i \leqslant m\}$
(3) $\{y_i \mid 1 \leqslant i \leqslant m\}$
(4) $\{z_i \mid 1 \leqslant i \leqslant m\}$
Edges:
(1) $\{(v_i, w_j) \mid e_j = (v_i, v_k)\}$
(2) $\{(z_j, v_k) \mid e_j = (v_i, v_k)\}$
(3) $\{(w_j, z_k) \mid 1 \leqslant j \leqslant m\}$
(4) $\{(z_j, y_j) \mid 1 \leqslant j \leqslant m\}$
Given: a graph
Result: a digraph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
Arcs:
(1) $\{(v_i, v_j), (v_j, v_i) \mid (v_i, v_j)$ is in $E\}$

4.3 Oriented graphs

Given: a graph
Result: an oriented graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
Arcs:
(1) $\{(v_i, e_j) \mid v_i$ is in $e_j\}$

4.4 Acyclic rooted digraphs

Given: a graph
Result: an acyclic rooted digraph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
(3) $\{s\}$
Arcs:
(1) $\{(v_i, e_j) \mid v_i$ is in $e_j\}$
(2) $\{(s, v_i) \mid 1 \leqslant i \leqslant n\}$
Root: s

4.5 Bipartite graphs

Given: a graph
Result: a bipartite graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
Edges:
(1) $\{(v_i,e_j) \mid v_i \text{ is in } e_j\}$

4.6 Chordal graphs

Given: a graph
Result: a chordal graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
Edges:
(1) $\{(v_i,e_j) \mid v_i \text{ is in } e_j\}$
(2) $\{(v_i,v_j) \mid i \neq j\}$

4.7 Graphs with a forbidden induced subgraph

Given: a graph, and a forbidden cyclic graph H with p vertices.
Result: a graph having no induced subgraph isomorphic to H
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{w_{i,j} \mid 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant p-1\}$
Edges:
(1) $\{(w_{i,j}, w_{i,j+1}) \mid 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant p-2\}$
(2) $\{(v_i,w_{j,1}), (v_k, w_{j,p-1}) \mid e_j = (v_i,v_k)\}$

4.8 Transitively orientable graphs

The construction is the same as that in 4.5

4.9 Series-parallel digraphs

Given: a graph
Result: a series-parallel digraph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
(3) {new}

Arcs:
(1) $\{(v_i,e_j) \mid v_i \text{ is in } e_j\}$
(2) $\{(v_i,\text{new}) \mid 1 \leqslant i \leqslant n\}$
(3) $\{(\text{new},e_j) \mid 1 \leqslant j \leqslant m\}$

4.10 Undirected path graphs

Given: a graph
Result: an undirected path graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
Edges:
(1) $\{(v_i,e_j) \mid v_i \text{ is in } e_j\}$
(2) $\{(e_i,e_j) \mid i \neq j\}$

5.1 Compact graphs

Given: a graph
Result: a compact graph
Vertices:
(1) $\{w_{i,j} \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 5\}$
Edges:
(1) $\{(w_{j,i},w_{k,i}) \mid (v_j,v_k) \text{ is in } E,\ 1 \leqslant i \leqslant 5\}$
(2) $\{(w_{i,j},w_{k,l}) \mid j \neq l,\ 1 \leqslant i,k \leqslant n\}$

5.3 Self-complementary graphs and digraphs

Given: a graph
Result: a self-complementary graph
Vertices:
(1) $\{z_{i,j} \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 8\}$
(2) {new}
Edges:
(1) $\{(\text{new},z_{i,j}) \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 4\}$
(2) $\{(z_{i,j},z_{k,j}) \mid (v_k,v_j) \text{ is in } E \text{ and } j = 1,4,6, \text{ or } 7\}$
(3) $\{(z_{i,j},z_{k,j}) \mid (v_k,v_j) \text{ is not in } E \text{ and } j = 2,3,5, \text{ or } 8\}$
(4) $\{(z_{i,j},z_{k,l}) \mid 1 \leqslant i,k \leqslant n,\ (j,l) \text{ is one of:}$
(1,2), (1,6), (2,3), (2,5), (2,6), (2,7), (3,4),
(3,6), (3,7), (3,8), (4,7), (5,6), (6,7), or (7,8)

5.4 Regular self-complementary graphs

Given: a self-complementary graph
Result: a regular self-complementary graph
Vertices:
(1) $\{w_{i,j} \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 4\}$
(2) {new}
Edges:
(1) $\{(\text{new}, w_{i,j}) \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant 2\}$
(2) $\{(w_{i,j}, w_{k,l}) \mid 1 \leqslant i,k \leqslant n,\ (j,l) = (1,3) \text{ or } (2,4)\}$
(3) $\{(w_{i,1}, w_{j,2}) \mid (v_i, v_j) \text{ is not in } E \text{ and } i \neq j\}$
(4) $\{(w_{i,3}, w_{j,4}) \mid (v_i, v_j) \text{ is not in } E \text{ or } i = j\}$

5.5 Graphs with a unique centre

Given: a graph
Result: a graph with a unique centre
Vertices:
(1) $\{w_i \mid 1 \leqslant i \leqslant n\}$
(2) $\{z_i \mid 1 \leqslant i \leqslant n\}$
(3) {centre}
Edges:
(1) $\{(\text{centre}, w_i) \mid 1 \leqslant i \leqslant n\}$
(2) $\{(\text{centre}, z_i) \mid 1 \leqslant i \leqslant n\}$
(3) $\{(w_i, w_j) \mid (v_i, v_j) \text{ is in } E\}$
(4) $\{(z_i, z_j) \mid (v_i, v_j) \text{ is in } E\}$

5.6 Cospectral graphs

Given: two graphs, G with vertex set $\{v_1, \ldots, v_n\}$ and edge set E, and H with vertex set $\{w_1, \ldots, w_p\}$ and edge set F.
Result: a cospectral pair of graphs
Graph 1, vertices:
(1) $\{\xi, y_i, z_i \mid 1 \leqslant i \leqslant n\}$
(2) $\{r_i, s_i, t_i \mid 1 \leqslant i \leqslant p\}$
Graph 1, edges:
(1) $\{(\xi, x_j), (y_i, y_j), (z_i, z_j) \mid (v_i, v_j) \text{ is in } E\}$
(2) $\{(r_i, r_j), (s_i, s_j), (t_i, t_j) \mid (w_i, w_j) \text{ is in } F\}$
(3) $\{(\xi, r_j) \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant p\}$
(4) $\{(z_i, t_j) \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant p\}$
(5) $\{(y_i, r_j), (y_i, s_j), (y_i, t_j) \mid 1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant p\}$
Graph 2, vertices: as Graph 1
Graph 2, edges: as Graph 1, except
(5) $\{(s_i, x_j), (s_i, y_j), (\xi, z_j) \mid 1 \leqslant i \leqslant p,\ 1 \leqslant j \leqslant n\}$

6.1 Line graphs

Given: a graph, not isomorphic to K_3 or $K_{1,3}$
Result: a line graph
Vertices:
(1) $\{e_1, \ldots, e_m\}$
Edges:
(2) $\{(e_i, e_j) \mid i \neq j,\ e_i \text{ and } e_j \text{ meet at a vertex}\}$

6.2 Regular graphs – Booth

Given: a graph with $m - n > 2$ and no isolated vertices
Result: a regular graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{e_1, \ldots, e_m\}$
(3) $\{f_i \mid 1 \leqslant i \leqslant m\}$
(4) $\{g_i \mid 1 \leqslant i \leqslant m-2\}$
(5) $\{h_i \mid 1 \leqslant i \leqslant m-n+2\}$
Edges:
(1) $\{(v_i, e_j) \mid v_i \text{ is in } e_j\}$
(2) $\{(v_i, f_j) \mid v_i \text{ is not in } e_j\}$
(3) $\{(e_j, g_k) \mid 1 \leqslant j \leqslant m,\ 1 \leqslant k \leqslant m-2\}$
(4) $\{(f_j, h_k) \mid 1 \leqslant j \leqslant m,\ 1 \leqslant k \leqslant m-n+2\}$

6.2 Regular graphs – Miller

The construction can be found in [30,56,54]

6.3 Regular bipartite graphs

The construction can be found in [56]

6.4 Marked graphs and marked trees

Given: a marked graph with partitioning $P = \{p_1, \ldots, p_k\}$
Result: a labeled graph
Vertices:
(1) $\{v_1, \ldots, v_n\}$
(2) $\{p_1, \ldots, p_k\}$
Edges:
(1) $\{e_1, \ldots, e_m\}$
(2) $\{(v_i, p_j) \mid v_i \text{ is in class } p_j \text{ of } P\}$
Vertex labels:

(1) $\{v_1, ..., v_n\}$ have label '0'
(2) $\{p_1, ..., p_k\}$ have label '1'
Given: a directed graph
Result: a marked tree
Vertices:
(1) $\{v_1, ..., v_n\}$
(2) $\{f_i \mid 1 \leqslant i \leqslant m\}$
(3) $\{g_i \mid 1 \leqslant i \leqslant m\}$
(4) {root}
Edges:
(1) $\{(\text{root}, v_i) \mid 1 \leqslant i \leqslant n\}$
(2) $\{(v_i, f_k), (v_j, g_k) \mid e_k = (v_i, v_j)\}$
Partitioning:
(1) $p_{n+1} = \{\text{root}\}$
(2) $p_i = \{v_i\} \bigcup \{g_k \mid e_k = (x, v_i)\} \bigcup \{f_k \mid e_k = (v_i, x)\}$

6.5 K-trees

Given: a directed graph with no sources or sinks
Result: a k-tree (with $k = n - 1$)
Vertices:
(1) $\{v_1, ..., v_n\}$
(2) $\{w_i \mid 1 \leqslant i \leqslant n\}$
(3) $\{z_i \mid 1 \leqslant i \leqslant m\}$
Edges:
(1) $\{(v_i, v_j) \mid i \neq j\}$
(2) $\{(v_i, w_j) \mid i \neq j\}$
(3) $\{(z_i, v_l) \mid e_i = (v_j, v_k), l \neq j, l \neq k\}$
(4) $\{(w_i, z_j) \mid e_j = (v_i, v_k)\}$

6.6 C-subgraph regular graphs

The construction can be found in [30]

6.7 Eulerian graphs

Given: a connected graph G with degree$(v_i) = d_i$
Result: an Eulerian graph
Vertices:
(1) $\{w_i \mid 1 \leqslant i \leqslant n\}$
(2) $\{\xi \mid 1 \leqslant i \leqslant n\}$
(3) $\{z_{i,j} \mid d_i \text{ odd}, j = 1\}$
(4) $\{z_{i,j} \mid d_i \text{ even}, 1 \leqslant j \leqslant 2\}$
Edges:
(1) $\{(w_i, w_j) \mid (v_i, v_j)$ is in $E\}$

(2) $\{(\xi, xj) \mid (v_i, v_j)$ is in $E\}$.
(3) $\{(\xi, z_{i,1}), (w_i, z_{i,1}) \mid d_i$ odd$\}$
(4) $\{(\xi, z_{i,j}), (w_i, z_{i,j}) \mid d_i$ even, $1 \leqslant j \leqslant 2\}$

6.8 Hamiltonian graphs

Given: a connected graph
Result: a Hamiltonian graph
Method: take the line graph of the Eulerian graph constructed above.

6.9 Hypergraphs

Given: a hypergraph
Result: a graph
Vertices:
(1) $\{v_1, ..., v_n\}$
(2) $\{e_1, ..., e_m\}$
Edges:
(1) $\{(v_i, e_j) \mid v_i$ is in $e_j\}$

7.1 Algebras

The construction can be found in [54]

7.2 Lattices

Given: a graph with no isolated vertices
Result: a lattice
Elements:
(1) $\{v_1, ..., v_n\}$
(2) $\{e_1, ..., e_m\}$
(3) $\{0, 1\}$
Domination relation **R**:
(1) 1 **R** e_i, $1 \leqslant i \leqslant m$
(2) e_j **R** v_i, when v_i is in e_j
(3) v_i **R** 0, $1 \leqslant i \leqslant n$

7.3 Semigroups

The construction is identical to that in 7.2

7.4 Finitely presented algebras

The construction can be found in [41]

7.5 Automata and semiautomata

Given: a graph
Result: an automaton
States:
(1) $\{v_1, ..., v_n\}$
(2) {start,stop}
Input alphabet: $\{v_1, ..., v_n\}$
Start state: start
Final states: $\{v_1, ..., v_n\}$
Transition function:
(1) $D(\text{start},v_i) = v_i$
(2) $D(\text{stop},v_i) = \text{stop}$
(3) $D(v_i,v_j) = v_i$ if $i=j$
(4) $D(v_i,v_j) = v_j$ if v_i,v_j is in E
(5) $D(v_i,v_j) = \text{stop}$ if $i \neq j$ and (v_i,v_j) is not in E
Given: a digraph
Result: a semiautomaton with binary input alphabet
States:
(1) $\{v_1, ..., v_n\}$
(2) $\{e_1, ..., e_m\}$
(3) {dead}
Input alphabet: {in,out}
Transition function:
(1) $D(q,\text{in}) = v_i$ if $q = v_i$,
$= v_j$ if $q = (v_i,v_j)$ is an arc in E,
$=$ dead otherwise.
(2) $D(q,\text{out}) = v_i$ if $q = (v_i,v_j)$ is an arc in E, dead otherwise.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974)

[2] D. Angluin, "On counting problems and the polynomial time hierarchy", submitted for publication

[3] L. Babai, "On the isomorphism problem", *Proceedings of the Foundations of Computation Theory Conference*, Poznan-Kornak (1977)

[4] L. Babai, "On the complexity of canonical labeling for strongly regular graphs", submitted for publication

[5] L. Babai, "The star-system problem is at least as hard as the graph isomorphism problem", *Proceedings of the Combinatorics Colloquium of the Bolyai Math. Society*, Kesthely (1976)

[6] H.G. Barrow and R.M. Burstall, "Subgraph isomorphism, matching relational structures, and maximal cliques", *Information Processing Letters 4* (1976) 83-84

[7] G. Birkhoff, *Lattice Theory*, American Mathematics Society (1948)

[8] J.A. Bondy and R.L. Hemminger, "Graph reconstruction - a survey", *Journal of Graph Theory 1* (1977) 227-268

[9] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications* , Macmillan (1976)

[10] K.S. Booth, *PQ-tree algorithms*, Ph.D. thesis, University of California at Berkeley (1975)

[11] K.S. Booth, "Isomorphism testing for graphs, semigroups, and finite automata are polynomially equivalent problems", *SIAM Journal on Computing 7* (1978) 273-279

[12] K.S. Booth, "Problems polynomially equivalent to graph isomorphism (abstract)", in *Algorithms and Complexity* (J.F. Traub, editor), Academic Press (1976) 435

[13] K.S. Booth and G.S. Lueker, "Linear algorithms to recognize interval graphs and test for the consecutive ones property", *Proceedings of the Seventh Annual ACM Symposium on the Theory of Computing (1975)* 255-265

[14] M. Brown and R. Conolly, "On graphs with constant link", in *New Directions in the Theory of Graphs* (F. Harary, editor), Academic Press (1973) 19-51

[15] W. Burnside, *Theory of Groups of Finite Order*, Cambridge University Press (1911)

[16] V. Chvatal, "New directions in Hamiltonian graph theory", in *New Directions in the Theory of Graphs* (F. Harary, editor), Academic Press (1973) 65-95

[17] M.J. Colbourn, "Analytic and computer techniques for set packings", M.Sc. thesis, Department of Computer Science, University of Toronto (1977)

[18] C.J. Colbourn and M.J. Colbourn, "Graph isomorphism and self-complementary graphs", *SIGACT News 10*, 1 (1978) 25-29

[19] C.J. Colbourn and M.J. Colbourn, "Isomorphism problems involving self-complementary graphs and tournaments",.I Proceedings of the Eighth Manitoba Conference on Numerical Mathematics and Computing (1978)

[20] C.J. Colbourn and M.J. Colbourn, unpublished

[21] C.J. Colbourn, "A bibliography of the graph isomorphism problem", *Technical Report 123/78, Department of Computer Science, University of Toronto (1978)*

[22] C.J. Colbourn, unpublished

[23] C.J. Colbourn, "A polynomial time algorithm for isomorphism of distributive lattices", submitted for publication

[24] M.J. Colbourn, private communications

[25] S.A. Cook, "The complexity of theorem-proving procedures", *Proceedings of the Third Annual ACM Symposium on the Theory of Computing (1971)* 151-158

[26] J. Cooper, J. Milas, and W.D. Wallis, "Hadamard equivalence", in *Combinatorial Mathematics* (D.A. Holton and J. Seberry, editors), Springer Verlag (1978) 126-135

[27] D.G. Corneil, *Graph Isomorphism*, Ph.D. thesis, University of Toronto (1968)

[28] D.G. Corneil, "Recent results on the graph isomorphism problem", *Proceedings of the Eighth Manitoba Conference on Numerical Mathematics and Computing (1978)*

[29] D.G. Corneil, private communications (1979)

[30] D.G. Corneil and D.G. Kirkpatrick, "A theoretical analysis of various heuristics for the graph isomorphism problem", to appear in *SIAM Journal on Computing*

[31] D.G. Corneil, M. Klawe, and A. Proskurowski, "Marked trees and the graph isomorphism problem", in preparation

[32] D.G. Corneil and R.A. Mathon, "Algorithmic techniques for the construction and analysis of strongly regular graphs and other combinatorial configurations", *Annals of Discrete Mathematics 2* (1978) 1-32

[33] P. Erdos and L. Babai, "On random graph isomorphism", submitted for publication

[34] M. Fontet, "Automorphismes de graphes et planarite", *Asterisque 38-39* (1976) 73-90

[35] R. Frucht, "Lattices with a given abstract group of automorphisms", *Canadian Journal of Mathematics 2* (1950) 417-419

[36] P.B. Gibbons, *Computing techniques for the construction and analysis of block designs*,. Ph.D. thesis, University of Toronto (1976)

[37] F. Harary, *Graph Theory*, Addison Wesley (1969)

[38] D. Hirschberg and M. Edelberg, "On the complexity of computing graph isomorphism", Technical Report 130, Department of Electrical Engineering, Princeton University (1973)

[39] R.M. Karp, "Reducibilities among combinatorial problems", in *Complexity of Computer Computations* (R.E. Miller and J.W. Thatcher, editors), Plenum (1972) 85-104

[40] R.M. Karp, "Probabilistic analysis of a canonical numbering algorithm for graphs", *Proceedings of the AMS Symposium on the Relation Between Combinatorics and Other Branches of Mathematics* (1978)

[41] D. Kozen, "Complexity of finitely presented algebras", *Proceedings of the Ninth Annual ACM Symposium on the Theory of Computing (1977)* 164-177

[42] D. Kozen, "A clique problem equivalent to graph isomorphism", *SIGACT News 10*, 2 (1978) 50-52

[43] E.L. Lawler, "Graphical algorithms and their complexity", *Math. Centre Tracts 81* (1976) 3-32

[44] A.B. Lehman, private communications

[45] H. Lerchs, "Cographs and their applications", unpublished manuscript, Department of Computer Science, University of Toronto (1972)

[46] G. Levi, "A note on the derivation of the maximal common subgraphs of two directed or undirected graphs", *Calcolo 9* (1972) 341-354

[47] R.J. Lipton, "The beacon set approach to graph isomorphism", to appear in *SIAM Journal on Computing*

[48] A. Lubiw, private communications (1979)

[49] G.S. Lueker and K.S. Booth, "A linear algorithm for deciding interval graph isomorphism", to appear in *Journal of the ACM*

[50] R.A. Mathon, "A note on the graph isomorphism counting problem", submitted for publication

[51] R.A. Mathon, "Sample graphs for graph isomorphism testing", *Proceedings of the Ninth Southeast Conference on Combinatorics, Graph Theory, and Computing (1978)*

[52] D.W. Matula, "Subtree isomorphism in $O(n^{5/2})$ time", *Annals of Discrete Mathematics 2* (1978)

[53] B.D. McKay, "Hadamard equivalence via graph isomorphism", submitted for publication

[54] G.L. Miller, "Graph isomorphism: general remarks", to appear in the *Journal of Computer and System Sciences*

[55] G.L. Miller, "On the $n^{\log n}$ isomorphism technique", *Proceedings of the Tenth Annual ACM Symposium on the Theory of Computing (1978) 51-58*

[56] G.M. Prabhu, "Graph isomorphism: a heuristic approach", M.Tech. thesis, Indian Institute of Technology Kanpur (1978)

[57] R.C. Read and D.G. Corneil, "The graph isomorphism disease", *Journal of Graph Theory 1* (1977) 339-363

[58] P.L. Renz, "Intersection representation of graphs by arcs", *Pacific Journal of Mathematics 34* (1970) 501-510

[59] J.J. Rotman, *The Theory of Groups*, Allyn and Bacon (1965)

[60] A.J. Schwenk, W.C. Herndon, and M.L. Ellzey, "The construction of cospectral composite graphs", *Proceedings of the Second International Conference on Combinatorial Mathematics*, New York (1978)

[61] C.R. Snow and H.I. Scoins, "Towards the unique decomposition of graphs", in *Machine Intelligence 4* (D. Michie, editor), Edinburgh University Press (1969) 45-55

[62] R. Statman, "The graph isomorphism problem is equivalent to the legitimate deck problem for regular graphs", unpublished manuscript (1978)

[63] L. Stewart, "Cographs - a class of tree representable graphs", M.Sc. thesis, University of Toronto (1978); also Technical Report 126/78, Department of Computer Science, University of Toronto (1978)

[64] A.P. Street and W.D. Wallis, *Combinatorial Theory: An Introduction*, Charles Babbage Research Centre (1977)

[65] L. Valiant, "The complexity of computing the permanent", to appear in *Theoretical Computer Science*

[66] H. Whitney, "Congruent graphs and the connectivity of graphs", *American Journal of Mathematics 54* (1932) 150-168

[67] J.K. Wong, *Isomorphism problems involving planar graphs*, Ph.D. thesis, Cornell University (1975)