Efficient Graph Automorphism by Vertex Partitioning

Glenn Fowler, Robert Haralick and F. Gail Gray
Department of Electrical Engineering, Virginia Tech, Blacksburg, VA, U.S.A.

Charles Feustel
Department of Mathematics, Virginia Tech, Blacksburg, VA, U.S.A.

Charles Grinstead
Department of Mathematics, Swarthmore University, Swarthmore, PA, U.S.A.

ABSTRACT

We describe a vertex partitioning method and squeeze tree search technique, which can be used to determine the automorphism partition of a graph in polynomial time for all graphs tested, including those which are strongly regular. The vertex partitioning procedure is based on first transforming the graph by the 1- or 2-subdivision transform or the 1- or 2-superline transform and then employing a distance signature coding technique on the vertices of the transformed graph. The resulting adjacency refinement partition of the transformed graph is reflected back to the original graph where it can be used as an initial vertex partition which is equal to or coarser than the desired automorphism partition.

The squeeze tree search technique begins with two partitions, one finer than the automorphism partition and one coarser than the automorphism partition. In essence, it searches through all automorphisms refining the coarser partition and coarsening the finer partition until the two are equal. At this point the result is the automorphism partition. The vertex partitioning method using the 2-superline graph transform preceeding the squeeze tree search is so powerful that for all the graphs in our catalog (random, regular, strongly regular, and balanced incomplete block designs) it produces the automorphism partition, thereby making the tree search nothing more than a verification that the initial partition is indeed the automorphism partition.

1. Introduction

The efficient determination of whether two graphs are isomorphic is of great interest to the artificial intelligence, scientific, and mathematical communities.

0004-3702/83/0000–0000/$03.00 © 1983 North-Holland

Graph isomorphism algorithms are currently used in all artificial intelligence applications requiring a matching between graph and relational structures. Such matches are required in object identification by structural techniques. Exact matching corresponds to finding isomorphisms. One technique for inexact matching corresponds to finding maximal domains on which monomorphisms can be established [3]. Additional examples occur in chemical structure matching [23], parallel computer network classification [18], and image scene analysis [10, 31], to name a few.

The dominant feature of all algorithms which either determine whether an isomorphism exists or construct an isomorphism if one exists is an exponential tree search, in the worst case. There is currently no known way of avoiding the tree search.

There have been published algorithms which make the tree search more efficient than simple backtracking. Among them are Gaschnig's [6] backmarking, Haralick and Elliott's [9] forward checking, the Schmidt and Druffel [20] use of distance matrices, the Bhat [1] use of vertex coding, and the discrete relaxation technique which was originally discovered by Ullmann [26, 27], used by Waltz [29], and analyzed as well as generalized by Haralick and Shapiro [10, 11].

An equivalent problem to the graph isomorphism problem is the graph automorphism partition problem. The automorphism partition of a graph consists of cells satisfying that two vertices are in the same cell if and only if there exists an isomorphism from the graph to itself mapping the first vertex to the second. Construction of the automorphism partition of a graph is computationally equivalent to the determination of whether there exists an isomorphism between two graphs [12]. To see this for the case of connected graphs, first construct a graph which is the disjoint union of the two given connected graphs which are to be tested for isomorphism. Then construct the graph automorphism partition for the disjoint union. The original two graphs are isomorphic if and only if there exists one cell of the automorphism partition containing a vertex from each of the original graphs. Of course, if one such cell is found, the connectedness of the original graph forces each cell to have such a pair.

The graph automorphism partition has other uses. Since it is in essence an encoding of the symmetries of the graph, having the automorphism partition available can be of definite aid in the tree search employed in determining graph monomorphisms, a problem often called the subgraph isomorphism problem.

Symmetry information contained in the automorphism partition can also be used to prune backtracking tree searches. The tree search can maintain the automorphism partition of the data structures searched thus far. Then, when new structures are generated, the symmetries in the partition can be used to detect and eliminate redundant branches in the search.

In this paper we consider the construction of the graph automorphism partition. We describe a vertex coding technique which appears to be more powerful than any previously existing one. This vertex coding is used to determine an initial partition which is guaranteed to be equal to or coarser than the automorphism partition. In fact, however, we have no regular graphs in our catalog for which the partition determined by the vertex coding scheme is not the automorphism partition. In any case, the coarser partition is employed along with forward checking in a new squeeze tree search technique for determination of the automorphism partition. The squeeze tree search is one which, in essence, begins with a partition guaranteed to be equal to or finer than the automorphism partition and another one which is guaranteed to be equal to or coarser than the automorphism partition. These two partitions bound the automorphism partition. In the squeeze tree search technique, as automorphisms are searched for, the finer partition is coarsened and the coarser partition is refined. The automorphism partition is produced when the finer partition and the coarser partition become identical.

Our results are all experimental and prove the efficiency of the forward checking squeeze tree search technique even when there is no preceeding vertex coding on either random graphs, where the tree search is typically trivial, or on large and highly symmetric regular graphs, where the tree search is more substantial. When the computational complexity criterion is the number of operations employed in the tree search as a function of graph size the computational complexity for the vertex coding and squeeze tree search is, on the average, polynomial time with the average order of the polynomial being approximately 6.

Section 2 concisely defines all the relevant technical graph theoretic terms we use in the paper. Section 3 discusses some of the vertex invariants which have appeared in the literature and which are the basis of vertex coding. Section 4 discusses graph transforms and introduces the k-line graph transform and the k-superline graph transform which when employed along with the vertex coding of Section 3 produces the automorphism partition, without any tree search, for all strongly regular and balanced incomplete block design graphs in our catalog. Section 5 discusses the squeeze tree search technique. Finally, Section 6 presents an analysis of experimental results.

2. Definitions

A graph G consists of a nonempty set of *vertices* $V(G)$, and a (possibly empty) set of *edges* $E(G)$ containing unordered pairs of vertices from $V(G)$. Two vertices $u, v \in E(G)$ are said to be *neighboring* or *adjacent.* We let $p = |V(G)|$ and $q = |E(G)|$. The degree $d(v)$ of a vertex $v \in V(G)$ is the number of edges incident with v, i.e., $d(v) = |\{(u, v)\colon (u, v) \in E(G)\}|$. The *degree sequence* of a graph G is the ordered p-tuple $\langle d(v_1), d(v_2), \ldots, d(v_p)\rangle$, where $v_i \in V(G)$ and

$d(v_i) \leqslant d(v_j)$ if $i \leqslant j$. A *regular* graph is graph in which all vertex degrees are the same. A graph is *simple* if there are no multiple edges and if each edge connects two distinct vertices. A graph is *connected* if there exists a path of edges between any pair of vertices. We consider only simple, connected graphs.

Two graphs $G1$ and $G2$ are *isomorphic* (denoted $G1 \cong G2$) if there exists a bijection θ mapping the vertices of $G1$ to the vertices of $G2$ such that, for any $u, v \in V(G1)$,

$$(v, v) \in E(G1) \Leftrightarrow (\theta(u), \theta(v)) \in E(G2) .$$

In other words, the bijection θ *preserves adjacency*. An *automorphism* of a graph is an isomorphism of the graph onto itself. The set of all automorphisms of a graph G forms the automorphism group AUT(G) of the graph. AUT(G) induces a partition on the vertices of G, where two vertices $u, v \in V(G)$ are in the same class (or cell) if and only if there is an automorphism in AUT(G) that maps u to v. This partition is the *automorphism partition* of G, denoted by $\pi_A(G)$. Vertices in the same cell of $\pi_A(G)$ are said to be *indistinguisable* or *similar* (denoted $v \cong u$); vertices in different cells are *distinguishable*. A graph G is said to be *transitive* if its automorphism partition consists of one cell (that is, $|\pi_A(G)| = 1$) and *non-transitive* if $|\pi_A(G)| > 1$.

A *graph invariant* is a function F whose domain is the set of all graphs G^* and whose range is some set X such that for any two graphs $G1, G2 \in G^*$

$$G_1 \cong G_2 \Rightarrow F(G1) = F(G2) .$$

Two graphs $G1$ and $G2$ are distinguishable, and therefore not *isomorphic*, if there exists some graph invariant F such that

$$F(G1) \neq F(G2) .$$

Some simple graph invariants are: number-of-vertices(G), the function that assigns $|V(G)|$ to G; number-of-edges(G), the function that assigns $|E(G)|$ to G; degree-sequence(G), the function that assigns to G its degree sequence.

A vertex invariant is a function f whose domain is the cartesian product of the set of all graphs G^* and the set of all vertices, and whose range is some set X such that for any graph G and any pair of vertices $u, v \in V(G)$

$$u \cong v \Rightarrow f(G, u) = f(G, v) .$$

Two vertices $u, v \in V(G)$ are distinguishable, and therefore *not similar* if there exists some vertex invariant f such that

$$f(G, u) \neq f(G, v) .$$

Example vertex invariants are: degree(G, v), the function that assigns to $v \in V(G)$ its degree $d(v)$; number-of-N-clique(G, v), the function that assigns to $v \in V(G)$ the number of N-cliques in G that contain v.

A vertex invariant f can be used to induce a partition on the vertices of a graph G such that for any two vertices $u, v \in V(G)$, u, v are in the same cell if and only if $f(G, u) = f(G, v)$. We denote the vertex partition induced by a vertex invariant f as $\pi_f(G)$. Note that by definition of vertex invariants, for $u, v \in V(G)$ and any vertex invariant f,

$$u \cong v \Rightarrow u, v \text{ are in the same cell of } \pi_f(G).$$

In other words, $\pi_f(G)$ is as coarse or coarser than $\pi_A(G)$, the automorphism partition of G. We denote this coarsening relationship as

$$\pi_f(G) \geqslant \pi_A(G).$$

By using vertex invariants to induce vertex partitions, we are guaranteed that partitions finer than $\pi_A(G)$ will *never* be generated.

Given any vertex partition $\pi_X(G) \geqslant \pi_A(G)$, $\pi_X(G)$ can be refined using vertex adjacencies. Assume that the classes of the partition are labeled from 1 through k consecutively and the class label for a vertex $u \in V(G)$ is given by CLASS[u]. Define $N(u)$ to be $\{v: (u, v) \in E(G)\}$, the *neighborhood* of u. For each vertex $u \in V(G)$ construct the *neighborhood class* vector of u

$$\langle \text{CLASS}[u], \text{CLASS}[v_1], \text{CLASS}[v_2], \ldots, \text{CLASS}[V_{d(u)}] \rangle$$

where the v_i consist of all of the vertices in $N(u)$ and are sorted from lowest to highest class number. New classes are formed by placing all vertices with the same new neighborhood class vector in one class. The new class numbers may be assigned arbitrarily or they may be assigned by a lexicographic sort on the neighborhood class vectors. The refinement process iterates until no more new classes are added. The process must eventually terminate since the partition can have no more than p classes. The termination partition is called the *adjacency partition* induced by π_X. Mathon [16] has shown that the adjacency partition formed by neighborhood class vectors is unique. For a graph G and a partition $\pi \geqslant \pi_A(G)$ the adjacency partition σ induced by π satisfies the following rules.

(1) Any vertex in a cell $C_j \in \sigma$ is adjacent to the same number P_{jk} of vertices belonging to $C_k \in \sigma$, $1 \leqslant j, k \leqslant |\sigma|$, where $|\sigma|$ is the number of cells in σ.

(2) Of all such partitions satisfying (1) the adjacency partition induced by π has the smallest number of cells.

Unless stated otherwise, we let $\pi_f(G)$ denote the *adjacency partition* induced by the vertex invariant f on graph G. By definition of vertex invariants and theorems presented by Mathon [16],

$$\pi_f(G) \geqslant \pi_A(G).$$

Thus, the adjacency partition induced by any vertex invariant f is as coarse or coarser than the automorphism partition.

3. Vertex Invariants

One of the simplest vertex invariants is degree(G, v), the function that assigns to each vertex $v \in V(G)$ its degree $d(v)$. The adjacency partition $\pi_{\text{degree}}(G)$ does not distinguish vertices in non-transitive regular graphs. Fig. 1 illustrates one such graph.

A stronger invariant utilizes a logical extension of vertex degrees. Define the distance $D(u, v)$ between two vertices $u, v \in V(G)$ to be the length of the shortest path connecting u and v, where $D(u, v) = p$ when u and v are not connected, and $D(u, v) = 0$ when $u = v$. Define the *diameter* Δ of a graph G to be MAX$\{D(u, v)$: $u, v \in V(G)\}$. We define the vertex *distance-k-degree* $d_k(v)$ of a vertex v to be the number of vertices at distance k from v. Some relationships satisfied by distance degrees are

(1) $d_0(v) = 1$,
(2) $d_1(v) = d(v)$, the degree of v,
(3) $d_j(v) = 0$, for $j > \Delta$,
(4) $\sum_{i=0}^{\Delta} d_i(v) = p$.

We define the *distance signature* DSIG(G, v) of a vertex $v \in V(G)$ to be the tuple whose kth component is the distance-k-degree of v

$$\text{DSIG}(G, v) = \langle d_1(v) d_2(v) \cdots d_\Delta(v) \rangle$$

DSIG(G, v) is a vertex invariant, and the adjacency partition induced by vertex distance signatures is called the *distance partition* $\pi_D(G)$ of G. Note that $\pi_{\text{degree}}(G) \geqslant \pi_D(G)$ since the first component of DSIG(G, v) is exactly the vertex degree invariant. Fig. 1 illustrates a graph in which $\pi_{\text{degree}}(G) \neq \pi_D(G)$. Distance functions form the basis of a number of vertex partitioning algorithms [1, 20].

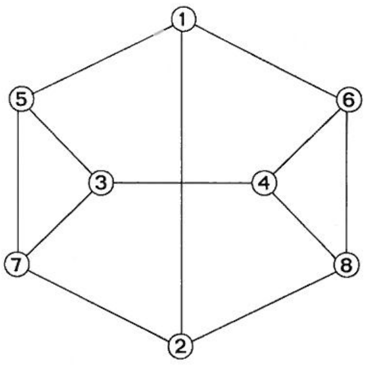


FIG. 1. Graph showing different distance partitions. $\pi_{\text{degree}}(G1) = \{(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8)\}$, $\pi_D(G1) = \{(1\ 2\ 3\ 4)\ (5\ 6\ 7\ 8)\}$, $(S^1)^{-1}(\pi_D(S^1(G1))) = \{(1\ 2)\ (3\ 4)\ (5\ 6\ 7\ 8)\}$, $\pi_A(G1) = \{(1\ 2)\ (3\ 4)\ (5\ 6\ 7\ 8)\}$.

We refer to the set of graphs examined during the course of our research as our *catalog*. There are many graphs in our catalog for which $\pi_D(G) > \pi_A(G)$ (see Fig. 1). Such graphs are most often members of a special class: *strongly regular* graphs (note that the graph in Fig. 1 is *not* strongly regular). A graph is strongly regular (SR) if each vertex has the same degree and any two adjacent vertices have the same number of common neighboring vertices and any two non-adjacent vertices have the same number of common neighboring vertices.

A property of a graph $G \in \text{SR}$ is, for any $u, v \in V(G)$,

$$d_k(u) = d_k(v), \quad k = 0, 1, \ldots, p,$$

For this reason algorithms employing $\pi_D(G)$, as well as many other distance partitioning algorithms, cannot distinguish between distinct vertices in strongly regular graphs. To avoid total tree searches in analyzing such graphs, stronger invariant functions are required. The next section discusses two approaches to obtaining stronger invariants by first transforming the original graph to a new graph and then employing the distance signature invariants discussed in this section.

4. Graph Transforms

We define a *graph transform* (graph-valued function) to be a function with domain and range consisting of the set of all graphs G^*. A graph transform projects sets of vertices in the domain graph to sets of vertices in the transform (range) graph. Note that some vertex sets may not have an image in the transform graph. Each graph transform may be divided into two parts: a *vertex transform* and an *edge transform*. The vertex transform determines how vertices are formed in the transform graph, and the edge transform determines how edges are formed. For example, in the complement graph transform $C(G)$, the vertex transform is

$$V(C(G)) = V(G)$$

and the edge transform is

$$E(C(G)) = \{(u, v): u \neq v, (u, v) \neq E(G)\}.$$

The line graph $L(G)$ transforms are

$$V(L(G)) = \{(u, v): (u, v) \in E(G)\},$$

$$E(L(G)) = \{((u, v), (v, w)): u \neq w, (u, v) \in E(G), (v, w) \in E(G)\}.$$

The edge-deleted-subgraph $e\backslash(G)$ is obtained by deleting edge e from $E(G)$. The corresponding vertex and edge transforms are

$$V(e\backslash(G)) = V(G),$$

$$E(e\backslash(G)) = \{(u, v): (u, v) \neq e, (u, v) \in E(G)\}.$$

Figs. 3, 4, and 5 illustrate these graph transforms on the graph in Fig. 2. We observe that vertices and edges in the transform graph are somehow related to vertices and edges in the original graph. In the complement graph transform for example, the relationship between $V(C(G))$ and $V(G)$ is simply the bijection of the vertices of $C(G)$ to the vertices of G.

Our main interest lies in determining the relationship between *vertices and edges* in the transform graph to *vertices* in the original graph. In particular, given a vertex or edge partition in the transform graph $H(G)$, we wish to induce a vertex partition in the original graph G using the vertex-edge to vertex relationship defined by the graph transform H. We denote this induced vertex partition by $H^{-1}(\pi(H(G))$.

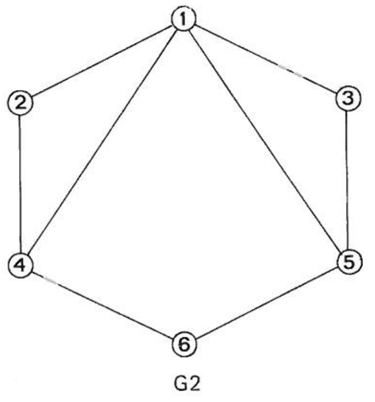


FIG. 2. Example graph for transformations.

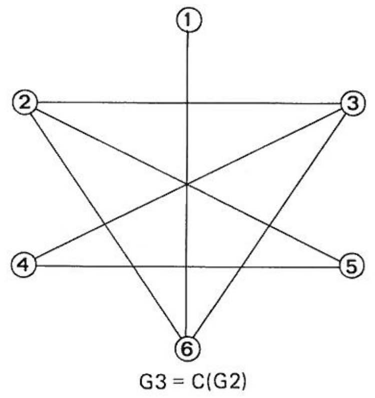


FIG. 3. Complement graph transform example.

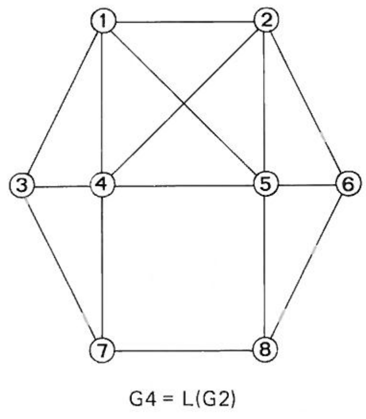


FIG. 4. Line graph transform example.

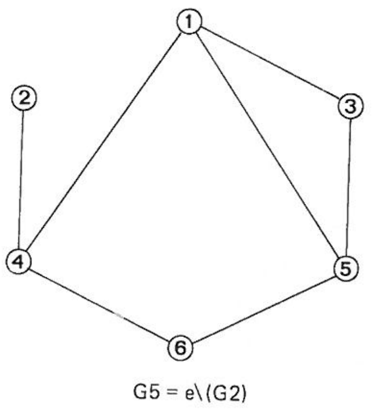


FIG. 5. Edge-deleted graph transform example.

In order to simplify the process of inducing a partition on the vertices of the original graph, the original vertices can be directly represented in the transform graph. Thus the class label of a vertex in the original graph is exactly the class label of its direct counterpart in the transform graph.

Of course, since our goal is to compute the automorphism partition, we are not interested in transforms that induce partitions finer than the automorphism partition in the original graph. For this reason we define an *invariant graph transform* to be a graph transform H that, for any graph G and any vertex invariant f,

$$H^{-1}(\pi_f(H(G)) \geqslant \pi_A(G)$$

That is, a partition on the vertices of G, induced by a partition on the components of $H(G)$ (induced by the vertex invariant f) must be as coarse or coarser than the automorphism partition of G in order for H to be an invariant graph transform. The complement and line graph transforms are invariant, whereas the edge-deleted-subgraph transform is not.

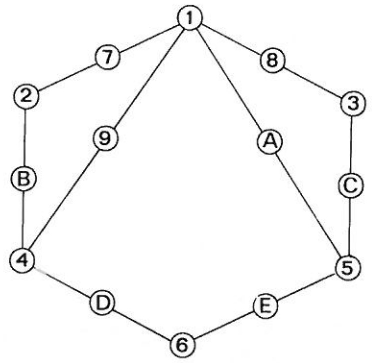
4.1. The k-subdivision transform

The k-subdivision graph $S^k(G)$, $k \geqslant 0$, is another example of an invariant graph transform. $S^k(G)$ is obtained by placing k new vertices on each edge of G. We note that $S^0(G)$ is G itself. The vertex and edge transforms of $S^1(G)$ are

$$V(S^1(G)) = V(G) \cup \{(u, v): \ (u, v) \in E(G)\},$$
$$E(S^1(G)) = \{(u, (u, v)): \ (u, v) \in E(G)\}.$$

Fig. 6 illustrates the 1-subdivision graph of the graph in Fig. 2. It follows that

$$|V(S^k(G))| = p + kq, \qquad |E(S^k(G))| = (k+1)q.$$



G6 = S^1(G2)

FIG. 6. 1-subdivision graph example.

Computation of the induced partition $(S^k)^{-1}(\pi(S^k(G)))$ is straightforward: the partition cell label for vertex $u \in V(G)$ is just the partition cell label for $u \in V(S^k(G))$, where u corresponds to an element of the first set in the vertex transformation above. It is an advantage to let $V(G)$ be a subset of the vertices in the transform graph since the class of a vertex in $V(G)$ is directly derivable from the class of the corresponding vertex in the transform graph. This is the case for the complement and k-subdivision transforms but it is not the case for the line graph transform discussed in Section 4.2.

Initial experiments using vertex distance signatures in the 1-subdivision graph to induce partitions in strongly regular graphs indicate that the induced vertex partition is exactly the automorphism partition for all 13 strongly regular graphs in our catalog. Refer to Weisfeiler [30] for a list of these graphs. This seems to be a significant result since published algorithms (not employing tree searches) have not been successful in distinguishing between vertices in strongly regular graphs, and in most cases these algorithms produce partitions consisting of one cell when $|\pi_A| > 1$ [1, 20]. It should be noted that the Bhat algorithm is capable of distinguishing between the four non-isomorphic strongly regular graphs on 28 vertices, even though it fails to produce the automorphism partitions of these graphs.

One explanation for the increased strength of $(S^1)^{-1}$ $(\pi_D(S^1(G)))$ is that the subdivision vertices in $S^1(G)$ (vertices in the second set of the vertex transform) effectively label the edges of G, allowing both edges *and* vertices to be accounted for. Define the *edge distance degree* $e_k(x)$ to be the number of edges at distance k from x, where x is either a vertex or an edge. The induced partition $(S^1)^{-1}$ $(\pi(S^1(G)))$ transforms the distance signature of each vertex $v \in V(G)$ into

$$\langle e_1(v)d_1(v)\cdots e_\Delta(v)d_\Delta(v)e_{\Delta+1}(v)\rangle$$

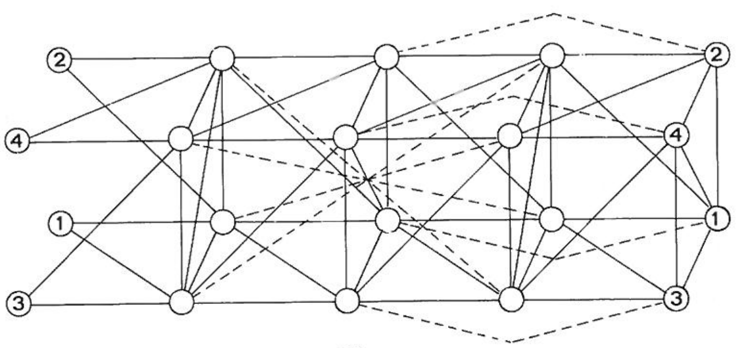
where Δ is the diameter of G. In addition, the edges of G are also partitioned

$$\langle d_1(x)e_1(x)\cdots d_\Delta(x)e_\Delta(x)d_{\Delta+1}(x)\rangle$$

where $x \in E(G)$. The combination of edge and vertex distance degree signatures induces partitions on the original vertices of G that are at least as fine as $\pi_D(G)$, and produces $\pi_A(G)$ in most graphs in our catalog, including strongly regular graphs. For example, in Fig. 1 $(S^1)^{-1}(\pi_D(S^1(G1))) = \pi_A(G1)$.

We know of only one graph for which $(S^1)^{-1}(\pi_D(S^1(G))) > (S^2)^{-1}(\pi_D(S^2(G)))$ (see Fig. 7). However, for this graph the 2-subdivision transform induces the automorphism partition.

The graph in Fig. 7 can be visualized as follows: the solid edges form a symmetric torus in which all vertices are similar, and the dashed edges remove some of the symmetries by distinguishing groups of vertices. As Fig. 7 shows, $S^2(G)$ potentially provides more information than $S^1(G)$. Intuitively, by placing two vertics on each edge, we can assign *directed* labels to edges, whereas the



G7

FIG. 7. Adjacency list for 2-subdivision example. $(S^1)^{-1}(\pi_D(S^1(G7))) > \pi_A(G7)$, $(S^2)^{-1}(\pi_D(S^2(G7))) = \pi_A(G7)$. Identify the labelled vertices to form a torus.

1-subdivision graph only assigns *undirected* labels to each edge. In other words, two labels per edge orients the edge in relation to the labels of its endpoints. It is not known if there exists a graph G such that

$$(S^2)^{-1}(\pi_D(S^2(G))) > (S^k)^{-1}(\pi_D(S^k(G))), \quad k > 2$$

but we conjecture that no further distance information is gained for $k > 2$.

4.2. *k*-line and *k*-superline graph transforms

Observing that the k-subdivision transform somehow enhances *edge* distance information, we now determine the effects of the line (edge) graph transform on vertex partitioning.

First note that the line graph $L(G)$ of some graph G may be empty, that is, $|V(G)| = 0$, $|E(G)| = 0$. Traditional graph theory requires $V(G)$ to be non-empty, but if needed, we assume the empty graph to be acceptable. We take this approach in defining the k-line graph $L^k(G)$:

(1) $L^0(G) = G$,
(2) $L^1(G) = L(G)$, the line graph of G,
(3) $L^k(G) = L^1(L^{k-1}(G))$, $K \geq 1$.

It follows by definition that

$$(S^0)^{-1}(\pi_D(S^0(G))) = (L^0)^{-1}(\pi_D(S^0(G))) .$$

Experiments also indicate

$$(S^1)^{-1}(\pi_D(S^1(G))) = (L^1)^{-1}(\pi_D(L^1(G))) ,$$

that is, the distance adjacency partition induced by $L^1(G)$ is the same as the distance adjacency partition induced by $S^1(G)$ for all graphs G in our catalog. The reason for this similarity is related to the assignment of undirected labels to the edges of the original graph by both transforms. In the case of the

1-subdivision graph, the edge labels in the original graph correspond to the subdivision vertices in the transform graph, whereas in the 1-line graph each edge label corresponds to a vertex in the transform graph. There is a direct correspondence between vertices in the 1-line graph and subdivision vertices in the 1-subdivision graph. Fig. 8 illustrates this correspondence by superimposing a graph and its 1-line graph.

The vertices labelled X and the dashed edges form the 1-line graph of the original graph. We note that the 1-subdivision graph is obtained by simply deleting the dashed edges from the superimposed 1-line graph.

The k-line graph transform does not preserve the original vertices in the transform graph. This makes the computation of the induced vertex partition in the graph difficult. As noted earlier, computing the induced partition in the original graph is trivial when the original vertices are represented in the transform graph. The line graph transform can be modified to preserve the original vertices by superimposing the line graph on the original graph, producing the *k-superline* invariant graph transform. The k-superline graph $\mathrm{SL}^k(G)$ of a graph G is defined as follows:

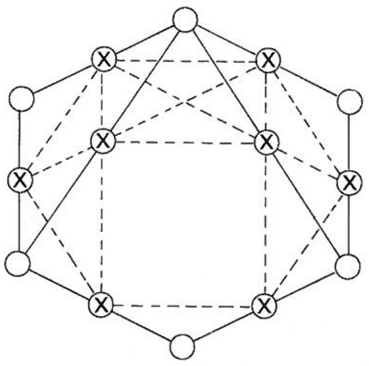
$$V(\mathrm{SL}^k(G)) = \bigcup_{j=0}^{k} V(L^j(G)),$$

$$E(\mathrm{SL}^k(G)) = E(L^k(G)) \cup \bigcup_{j=0}^{k-1} E(S^1(L^j(G))).$$

The k-superline transform preserves the original vertices, and the class label of a vertex in the original graph is just the class label of the corresponding vertex in the k-superline graph. Experiments show that

$$(\mathrm{SL}^k)^{-1}(\pi_D(\mathrm{SL}^k(G))) = (L^k)^{-1}(\pi_D(L^k(G))),$$

so we assume the k-line and k-superline partitions to be equivalent.



G8′

FIG. 8. 1-Line graph superimposed on original graph.

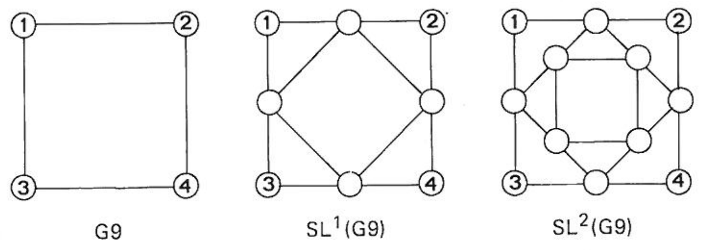


FIG. 9. A graph with 1-superline and 2-superline transforms.

Even though the partitions may be equivalent, computation and space requirements for each transform are drastically different. The k-line transform requires k vertex to edge to vertex class label transformations to induce a partition in the original graph, but uses considerably less vertices than the k-superline graph. In contrast, the induced vertex partition of the original graph can be computed directly from the k-superline graph, but requires more vertices than the k-line graph. Fig. 9 illustrates a graph and its 1-superline and 2-superline transforms with only the original vertices labelled.

A *balanced incomplete block design* BIBD(p, k, λ) is a collection of k-subsets, called blocks, of a p-element set V such that every 2-subset of V is contained in exactly λ blocks. Relating to a graph G, the set V corresponds to $V(G)$ and the 2-subsets correspond to $E(G)$. For the class of non-transitive graphs BIBD derived from balanced incomplete block designs, the partition induced by the 2-subdivision graph partition is coarser than the automorphism partition $((S^2)^{-1}(\pi_D(S^2(G))) \geqslant \pi_A(G),\ G \in \mathrm{BIBD})$. However, experiments on BIBD's in our catalog indicate that

$$(\mathrm{SL}^2)^{-1}(\pi_D(\mathrm{SL}^2(G))) = \pi_A(G)\,, \quad G \in \mathrm{BIBD}$$

In fact, no graphs have been found wherein the distance adjacency partition induced by the 2-superline graph is *not exactly* the automorphism partition, and it seems that such graphs may be very large. We conjecture that for the class of 3-level regular graphs [16] (strongly regular graphs are 1-level regular),

$$(\mathrm{SL}^2)^{-1}(\pi_D(\mathrm{SL}^2(G))) \geqslant \pi_A(G)\,.$$

The only known non-transitive 3-level regular graph has 139300 vertices [16], which is too large to be studied on our current graph analysis system. The increased strength of the 2-superline transform over the 2-subdivision transform indicates that the 2-superline transform does more than directed edge label assignment in the original graph.

To conclude this section, we summarize the experimental results on distance adjacency partitions induced by several invariant graph transforms.

$$F^{-1}(\pi_D(F(G))) \geqslant \pi_A(G),$$
$$(\mathrm{SL}^0)^{-1}(\pi_D(\mathrm{SL}^0(G))) = (S^0)^{-1}(\pi_D(S^0(G))),$$

$$(\mathrm{SL}^1)^{-1}(\pi_D(\mathrm{SL}^1(G))) = (S^1)^{-1}(\pi_D(S^1(G))),$$
$$(\mathrm{SL}^2)^{-1}(\pi_D(\mathrm{SL}^2(G))) \leq (S^2)^{-1}(\pi_D(S^2(G))),$$
$$(S^k)^{-1}(\pi_D(S^k(G))) \leq (S^{k-1})^{-1}(\pi_D(S^{k-1}(G))), \quad k \geq 1,$$
$$(\mathrm{SL}^k)^{-1}(\pi_D(\mathrm{SL}^k(G))) \leq (\mathrm{SL}^{k-1})^{-1}(\pi_D(\mathrm{SL}^{k-1}(G))), \quad k \geq 1,$$
$$(\mathrm{SL}^2)^{-1}(\pi_D(\mathrm{SL}^2(G))) = \pi_A(G)$$

for graphs in our catalog, including SR and BIBD graphs

where

$F(G)$: any invariant graph transform,
$S^k(G)$: k-subdivision graph transform,
$\mathrm{SL}^k(G)$: k-superline graph transform.

5. The Squeeze Tree Search

A few basic definitions are required in order to describe the squeeze tree search for the automorphism partition. We assume that the vertices in $V(G)$ are numbered consecutively from 1 to p. Given a permutation X of $V(G)$, we denote the ith vertex element of X as $X[i]$. The *identity permutation I* is defined to be the permutation whose ith element is the vertex i ($I[i] = i$). A permutation X of $V(G)$ is an automorphism of G, denoted $X \in \mathrm{AUT}(G)$, if

$$(X[i], X[j]) \in E(G) \Leftrightarrow (i, j) \in E(G), \quad 1 \leq i, j \leq p. \tag{1}$$

We refer to (1) as a *consistency check.*

First we describe the basic search for the automorphism partition of a graph. The basic search is inherently exponential since all $p!$ permutations are generated, making this method impractical even for small values of p. It proceeds as follows. A vertex relation

$$R : V(G) \times V(G) \to \{\mathrm{SAME, DIFF, UNKNOWN}\}$$

is initialized to $R(u, u) = \mathrm{SAME}$, $u \in V(G)$, else $R(u, v) = \mathrm{UNKNOWN}$. During the basic tree search R is updated and has the following meaning:

$$R(u, v) = \mathrm{SAME} \Rightarrow u \cong v,$$

$$R(u, v) = \mathrm{DIFF} \Rightarrow u \ncong v,$$

$$R(u, v) = \mathrm{UNKNOWN} \Rightarrow u \cong v \text{ or } u \ncong v.$$

The tree search successively generates all permutations of $V(G)$ and each permutation X is checked for membership in $\mathrm{AUT}(G)$ using (1) above. If $X \in \mathrm{AUT}(G)$, then R is updated as follows.

If $R(X[i], i) = \mathrm{UNKNOWN}$,
then $R(X[i], i) = \mathrm{SAME}$, $1 \leq i \leq p$.

After all permutations have been generated, R represents the automorphism partition of G, i.e.,

$$R(u, v) = \text{SAME} \Leftrightarrow u \cong v.$$

So far the R relation is only updated at level p by adding the SAME relation between vertices when there exists an automorphism mapping one vertex to the other. R may also be updated at level 1 by noting that if the tree search discovers no automorphism mapping level 1 vertex u to v, then $u \not\cong v$. Hence R may be updated by making $R(u, v) = \text{DIFF}$. This information constrains all subsequent vertex selections at each level of the tree search by eliminating all partial permutations in which $R(X[i], i) = \text{DIFF}$ for some i less than or equal to the current level. In this modified tree search the consistency check (1) becomes

$$R(X[i], i) \neq \text{DIFF and } (X[i], X[j]) \in E(G) \Leftrightarrow (i, k) \in E(G), \quad 1 \leq i, j \leq p. \tag{2}$$

This consistency check potentially eliminates more and more branches as the tree search progresses.

Tree searches involving R can be viewed as two separate but cooperating searches. The first search starts with a vertex partition $\sigma_C(G)$ which is coarser than $\pi_A(G)$, refining $\sigma_C(G)$ when new DIFF information is added to R. The second search starts with a partition $\pi_F(G)$ consisting of p cells, coarsening $\sigma_F(G)$ when new SAME information is added to R. The tree search terminates when $\sigma_F(G) = \sigma_C(G)$, which is exactly the automorphism partition.

The first improvement to the basic tree search is to convert it into a *backtracking tree search* [2]. In the backtracking search the permutations are generated by adding one vertex at a time, performing a consistency check on the partial permutation before any new vertices are added. The first vertex is added at level 1, the second at level 2, and so on, so that at level k in order for the assignment k to $X[k]$ to remain it must satisfy the consistency checks

$$R(X[k], k) \neq \text{DIFF and } (X[i], X[k]) \in E(G) \Leftrightarrow (i, k) \in E(G), \quad 1 \leq i \leq k. \tag{3}$$

At each level k one new vertex is added at $X[k]$. If (3) is not satisfied then there is no automorphism in which the first k elements exactly match the first k elements of X, so it is not necessary to search any deeper. If (3) is satisfied at a level less than p then the search continues on the next level. When all choices at the current level have been exhausted the tree search *backtracks* to the previous level where the selection process resumes. If (3) is satisfied at level p, then $X \in \text{AUT}(G)$ and R is updated as follows.

If $R(X[i], i) = \text{UNKNOWN}$,
then $R(X[i], i) = \text{SAME}$, $1 \leq i \leq p$.

The tree search terminates when an attempt is made to backtrack prior to level 1. Upon termination R represents the automorphism partition as in the basic search. The backtracking tree search in general eliminates many non-automorphic permutations that would have otherwise been generated by the basic search.

By modifying the consistency check (3) R can be used to decrease the number of automorphisms needed to determine $\pi_A(G)$. Consider the complete graph K_5 on 5 vertices. $\pi_A(K_5) = \{(1\,2\,3\,4\,5)\}$ and there are $5! = 120$ possible permutations of the 5 vertices, all of which are automorphisms. However, note that only two automorphisms are required to exactly determine the automorphism partition! For example, for the mapping

original vertex: 1 2 3 4 5
assigned vertex: 2 3 4 5 1

we have by transitivity,

$$1 \cong 2 \cong 3 \cong 4 \cong 5$$

from which we conclude that $\pi_A(K_5) = \{(1\,2\,3\,4\,5)\}$. Thus by taking a transitive closure on R, permutations that would provide no further updates to R can be eliminated from the tree search. A vertex u which is similar to a vertex v which is similar to w must imply that vertex u is similar to w. Likewise a vertex u which is similar to vertex v which is different than vertex w must imply that vertex u is different than vertex w.

The transitive closure of R is computed by repeating until no change occurs for each vertex pair (u, v):

(1) $R(u, v) = \text{SAME}$ and $R(v, w) = \text{SAME} \Rightarrow R(u, w) \leftarrow \text{SAME}$;

(2) $R(u, v) = \text{SAME}$ and $R(v, w) = \text{DIFF} \Rightarrow R(u, w) \leftarrow \text{DIFF}$.

The transitive closure of R efficiently combines the information gained at level 1 with level p. Experiments show that on graphs with $|\pi_A(G)| > 1$, backtracking tree searches employing the transitive closure of R require less consistency checks and generate less nodes than basic backtracking. When $|\pi_A(G)| = 1$, R provides no additional information since DIFF can never be added.

Now consider the example in Fig. 10 of permutations generated by a backtracking tree search on K_5.

Using a natural vertex selection order for generating automorphisms, only 4 out of 120 automorphisms actually modify R. Before the tree search starts $\sigma_F(G) = \{(1)\,(2)\,(3)\,(4)\,(5)\}$. The first permutation, $P1$, is the identity and does not modify $\sigma_F(G)$. $P2$ verifies that $4 \cong 5$ and $\sigma_F(G)$ is changed to $\{(1)\,(2)\,(3)\,(4\,5)\}$. $P3$ verifies that 3 is similar to 4 and 5 and $\sigma_F(G)$ becomes $\{(1)\,(2)\,(3\,4\,5)\}$. $P4$, however, adds no new information since it has already been verified that 5 is similar to 4 and 3 is similar to 5. The same holds true for $P5$ and $P6$. $P7$ changes $\sigma_F(G)$ to $\{(1)\,(2\,3\,4\,5)\}$ and $P9$ finally produces the automorphism

	permutation	$\sigma_F(A)$
$P1$	12345	( )
$P2$	12354	(45)
$P3$	12435	(345)
$P4$	124$\underline{5}$3	*no change*
$P5$	12$\underline{5}$34	*no change*
$P6$	12543	*no change*
$P7$	13245	(2345)
$P8^*$	1$\underline{*}$***	*no change*
$P9$	21345	(12345)
$P10^*$	$\underline{*}$****	*no change*

FIG. 10. Permutation trace for K_5.

partition $\{(1\,2\,3\,4\,5)\}$. $P8^*$ and $P10^*$, representing the remainder of the permutations, add no new information. Observe that no new information is added when the current and all previous vertex mappings in the partial permutation X are between vertices that have already been determined similar by some previous permutation. For example, in $P5$ 1 and 1 are similar, 2 and 2 are similar, and 5 and 3 are similar. The current permutation must be fully explored when a vertex is initially mapped to itself. $P2$ illustrates this point; even though 1 is similar to 1 the partial permutation must still be fully explored. In addition, if there is any vertex mapping on a previous level between vertices that have not been determined similar, then the search must fully explore the current partial permutation.

The consistency check (3) can be modified to reflect the selections of the previous example by adding a Boolean *cut* vector C. C is initialized to $C[i]=1$, $1 \leq i \leq p$ and the consistency check becomes:

$$\begin{aligned} &(R(X[k],k) \neq \mathrm{DIFF}) \text{ and } (\bar{C}[k] \text{ or } R(X[k],k) \neq \mathrm{SAME}) \\ &\qquad\qquad\qquad \text{and } (X[i],X[j]) \in E(G) \\ &\Leftrightarrow (i,j) \in E(G), \quad 1 \leq i,j \leq p, \end{aligned} \tag{4a}$$

$$C[k+1] = C[k] \quad \text{and} \quad (R(X[k],k) = \mathrm{SAME}). \tag{4b}$$

In addition, whenever a new automorphism is generated the cut vector can be reinitialized to $C[i]=1$, $1 \leq i \leq p$. The consistency checks (4a) and (4b) force the tree search to fully explore any permutations in which $R(X[i],i)=$ UNKNOWN, where i is less than or equal to the current level. At level k, $C[k]=1$ indicates that the current branch will produce no changes in R if $R(X[k],k)=\mathrm{SAME}$. This is a direct consequence of the transitive closure performed on R. The addition of the cut vector makes the backtracking tree search extremely efficient for graphs containing many automorphisms.

The tree search can be further improved by employing the *forward checking* algorithm [9]. The forward checking tree search performs a partial lookahead consistency check of future vertex mappings against past and present vertex mappings. A possible mapping table is maintained at each level of the tree search. This table contains, for each unmapped vertex, all vertex mappings that are consistent with past mappings contained in the partial permutation X. Additions to X are selected from the possible mapping table entry for the current vertex. Consistency checks are only done when a new possible mapping table is constructed for the next level. All vertices not yet added to X must have at least one entry in the possible mapping table. Otherwise the current partial permutation X fails and the tree search backtracks to the previous level, where a new selection is made from the possible mapping table of that level. The forward checking tries to make a failure occur early in the tree search by determining that there is no possible mapping for some vertex on a deeper level of the tree search. The number of early failures can be increased by selecting vertices with the smallest number of entries in the possible mapping table first, increasing the probability of failure because of the reduced number of possible future mappings. We refer to this type of selection as *optimal ordering*. The forward checking tree search does incur the computational cost of passing and manipulating the possible mapping table by level. This cost is only noticeable for small graphs ($p < 8$), and for larger graphs the computations saved by early failures of partial permutations far outweighs the computations required for table maintenance. Experiments indicate that forward checking combined with the transitive closure of R can be more efficient than either method.

Experimental data suggests the following order (from best to worst) of tree search efficiency based on the number of operations required to find the solution for graphs with $p \geq 8$:

forward check + R + closure	40
forward check + R	9
backtrack + R + closure	2
backtrack + R	1

The last column indicates a rough estimate of the average factor of improvement over the basic backtracking tree search with no transitive closure on R. The results were averaged over a wide range of graph sizes and symmetries, with results on the smaller graphs sometimes deviating greatly from the mean. For example, the backtrack tree search with closure on R performed better than the forward checking tree search without closure on R for $p < 8$, at which point the number of branches required by the backtracking tree search began to outweigh the table manipulations of the forward checking tree search.

5.1. Squeeze tree search preceded by vertex partitioning

Instead of initializing $R(u, v) = \text{UNKNOWN}$ for $u \neq v$, one of the vertex partitioning schemes of the previous section can be used to construct R. Assume that the vertex partitioning produces a class label vector CLASS, where CLASS$[i]$ is the class label for vertex i. R can then be initialized as follows:

(1) $R(u, u) = \text{SAME}$,

(2) $\text{CLASS}[u] \neq \text{CLASS}[v] \Rightarrow R(u, v) = \text{DIFF}$,

(3) $R(u, v) = \text{UNKNOWN}$, otherwise.

When $|\sigma_C(G)| = p$ the automorphism partition is trivial and no tree search is necessary. This is the case for most random graphs when the 0-subdivision vertex partition is used to compute $\sigma_C(G)$. In graphs where $\sigma_C(G) = \pi_A(G)$ the tree search is reduced to a verification of the automorphism partition, and most of the operations occur during the initial vertex partitioning. This is the case for most graphs when the 1-subdivision vertex partition is used to generate $\sigma_C(G)$, and it is always the case for graphs in our catalog when the 2-superline vertex partition is used.

6. Discussion of Results

In this section we outline the average complexity of the squeeze tree search when combined with some of the graph transform partitions of Section 4. Before discussing the complexity of the vertex partitioning algorithms and tree searches we define the basic units of computation. We consider each data reference to be one *operation*. Data references fall into two major categories: reading the value of a data item and writing the value of a data item. A data item may be an element of a vector, matrix, or some form of linked list. For example, a check to see if two vertices are adjacent requires one operation, and a request for the current class label of a vertex requires one operation. All operations are weighted equally, so our analysis of complexity does not take into account any speedups attainable through special data structures that may take advantage of particular computer hardware configurations.

In cases where the initial vertex partitioning produces the automorphism partition the squeeze tree search is trivial and most of the operations are incurred by the initial vertex partitioning. When a weak vertex partitioning technique is used, the initial partition is usually coarser than the automorphism partition and the squeeze tree search typically exhibits exponential behaviour. This suggests that the choice of the initial partitioning algorithm should be based on the expected structure of the input graphs so as not to waste operations on a powerful partitioning when a simpler one would do just as well, being aware that if the initial partitioning is too weak the tree search will most likely require exponential operations.

On the average, computation of $\pi_D(G)$ requires $O(|V(G)|^2)$ operations. Therefore, when the initial vertex partitioning involves an invariant graph transform H, the computation of $\pi_D(H(G))$ requires $O(|V(H(G))|^2)$ operations on the average.

For transitive graphs the squeeze tree search functions the same regardless of the initial partitioning is used. With no initial vertex partitioning the squeeze tree search requires $O(p^4)$ operations on the average to verify the transitivity of these graphs.

On random graphs the sequeeze tree search using $\pi_D(G)$ as an initial partition requires $O(p^2)$ operations on the average to determine the automorphism partition. This is because $\pi_D(G)$ is almost always the automorphism partition for random graphs, reducing the tree search to a verification that $\pi_D(G)$ is the automorphism partition. In addition, the automorphism partition is almost always trivial (each vertex in its own class) for random graphs, so in most cases the verification tree search is trivial. However, for non-transitive strongly regular and balanced incomplete block design graphs this tree search requires exponential operations on the average. This is because $\pi_D(G)$ fails to fully partition the vertices of these graphs, leaving most of the work for the tree search.

When $(S^1)^{-1}(\pi_D(S^1(G)))$ is used as the initial vertex partition, the squeeze tree search requires $O((p+q)^2)$ operations on the average for strongly regular and random graphs in our catalog. This tree search requires exponential operations for non-transitive balanced incomplete block designs and any other non-transitive graphs for which the 1-subdivision partition is strictly coarser than the automorphism partition. Since q is bounded above by $\frac{1}{2}p(p-1)$, this tree search has an average upper bound of $O(p^4)$ operations for graphs in which the 1-subdivision partition is the automorphism partition.

$(\mathrm{SL}^2)^{-1}(\pi_D(\mathrm{SL}^2(G)))$ is the most powerful vertex partitioning method and is also the most costly. The number of vertices in the 2-superline graph is

$$r = p + \tfrac{1}{2}\Big(\sum_V d(v)^2\Big).$$

The squeeze tree search using the 2-superline vertex partition then requires $O(r^2)$ operations on the average. The upper bound on r is $O(p^3)$ (using a complete graph), so this search has an average upper bound of $O(p^6)$ operations for all graphs in our catalog. Of course, if there are any non-transitive graphs for which the 2-superline graph transform does not produce the automorphism partition, the squeeze tree search will exhibit exponential behaviour, but, as mentioned earlier, no graphs of this type have been found.

The average complexity of $O(p^2)$ for random graphs required by the forward checking squeeze tree search with $\pi_D(G)$ as the initial partition compares favorably with the results of Corneil and Gotlieb ($O(p^2)$) and Ullmann ($O(p^3)$). On transitive graphs the squeeze tree search average complexity of $O(p^4)$ (assuming no initial vertex partitioning) also agrees with the $O(p^4)$ reported by Corneil and Gotlieb. For strongly regular graphs the average complexity of the squeeze tree search with $(S^2)^{-1}(\pi_D(S^2(G)))$ as the initial partition is $O((p+q)^2)$, with an average upper bound of $O(p^4)$, whereas Corneil and Gotlieb report an absolute upper bound of $O(p^7)$. Ullmann did not report any bounds on the complexity of his algorithm in the case of strongly regular graphs. Corneil and

Gotlieb were not aware of the balanced incomplete block design graphs at the time of their publication [5]. These graphs turned out to be a counterexample to a main conjecture of their vertex partitioning algorithm, so their tree search would most certainly exhibit exponential behaviour for these graphs. On the other hand, we report an average upper bound of $O(p^6)$ for balanced incomplete block designs in our catalog when the 2-superline graph is used to generate an initial partition for the squeeze tree search.

Tables 1 and 2 list the number of operations required for various combinations of tree search techniques, vertex partitionings, and graphs. An asterisk marks graphs in which the initial partition is strictly coarser than the automorphism partition for tree searches with some initial vertex partitioning.

As can be seen, the method chosen for determining the automorphism partition primarily depends on the nature of the input graphs. The forward

TABLE 1. Sample operation counts for backtracking tree searches

p	$p+q$	r	c	A	B	C	D
6	14	29	4	1522	1364	4612	15497
8	24	72	1	86081	1923	13920	88743
8	24	72	1	7137	2644	14018	90385
8	24	72	3	5187	2749	13611	92268
12	24	108	1	296689	43689	63622	232483
12	24	108	5	89271	49850	28485	208023
12	24	108	12	60481	60744	24723	204009
16	48	144	1	3646081	550841	855978	1153687
16	48	144	5	1797351	537001	49290	362318
16	48	144	12	1697195	1398582	43151	359994
16	48	171	16	—	1240331	35795	487295
22	66	198	5	13672267	5020188	155731	722149
22	66	198	9	12386153	6034289	79136	651237
22	66	198	10	—	10527923	73428	658618
25	85	325	4	—	—	645924*	1744759
25	85	325	5	—	—	828495*	1736071
32	96	288	12	—	106270886	175245	1377505
32	96	288	14	—	165683189	161882	1381731
32	96	351	32	—	—	121039	1972706
44	132	396	17	—	—	333938	2597635
44	132	396	18	—	—	309978	2555029
44	132	396	21	—	—	277021	2543451

r: number of vertices in 2-superline graph,
c: number of cells in automorphism partition,
A: backtrack + R,
B: backtrack + R + closure,
C: backtrack + R + closure + 1-subdivision partition,
D: backtrack + R + closure + 2-superline partition,
*: initial partition > automorphism partition.

TABLE 2. Sample operation counts for forward checking tree searches

p	$p+q$	r	c	E	F	G	H
6	14	29	4	1555	1639	4724	15609
8	24	72	1	68015	3983	15980	90803
8	24	72	1	8383	3148	14399	90766
8	24	72	3	6603	4959	14143	92800
12	36	108	1	138527	12083	37374	206235
12	36	108	5	37748	29338	28876	208414
12	36	108	12	16759	17022	24723	204009
16	48	144	1	816639	32851	77355	375064
16	48	144	5	55666	41085	52398	365426
16	48	144	12	33274	33199	43958	360801
16	48	171	16	74074	74553	35795	487295
22	66	198	5	505395	211246	103732	670150
22	66	198	9	132722	107820	81957	654058
22	66	198	10	435543	253263	76160	661318
25	85	325	4	1433602	356857	254561*	1756848
25	85	325	5	1443789	395286	348013*	1736071
32	96	288	12	1307110	894106	183313	1385573
32	96	288	14	1449795	1100293	163734	1380419
32	96	351	32	1486136	1488119	121039	1972706
44	132	396	18	3989174	2886972	320756	2565807
44	132	396	21	5940968	4178936	288168	2554638
44	132	396	17	5464674	3531624	346505	2610202

r: number of vertices in 2-superline graph,
c: number of cells in automorphism partition,
E: forward check + R,
F: forward check + R + closure,
G: forward check + R + closure + 1-subdivision partition,
H: forward check + R + closure + 2-superline partition,
*: initial partition > automorphism partition.

checking tree search with closure on R and no vertex partitioning is the most efficient for small graphs ($p < 8$), but as the graph size increases the benefits of vertex partitioning present themselves. In fact, when the graph size is large enough ($p \geqslant 40$), even the most complex vertex partitioning is worth the effort because of the exponential nature of the tree search when the initial partition is strictly coarser than the automorphism partition.

Fig. 11 is a log-log plot of the number of operations versus the number of vertices in the 2-superline graph for the forward checking squeeze tree search with an initial partition generated by the 2-superline graph transform.

We note that the operation counts predominantly represent the computation of the 2-superline graph distance partition. This plot indicates polynomial complexity for all graphs tested.

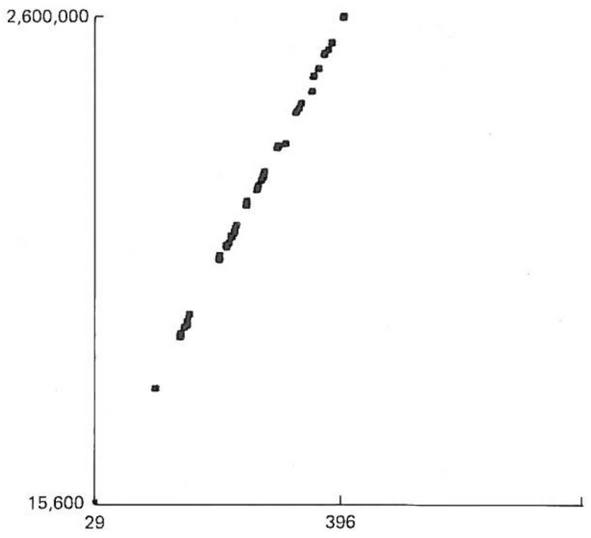


FIG. 11. log-log plot for the 2-superline squeeze tree search.

It is important to note that the order of the initial vertex labeling has little effect on the characteristics of the forward checking tree search because of continuous optimal vertex ordering. However, the initial vertex order does have an effect on the backtracking tree search. For example, consider the automorphism partitions of two isomorphic copies of the same graph

$$(1\ 2)\,(3)\,(4)\,(5)\,(6)\,(7)\,(8)\,,$$
$$(1\ 8)\,(2)\,(3)\,(4)\,(5)\,(6)\,(7)\,.$$

Using natural selection order, a backtracking tree search would immediately find the permutation mapping 1 to 2 in the first graph, whereas the mappings 1 to {2, 3, 4, 5, 6, 7} would be tried in the second graph before the mapping 1 to 8 would succeed.

We draw the following conclusions based on experimental data.

(1) Use the backtracking squeeze tree search if the graphs to be tested are small ($p < 10$).

(2) Use the forward checking squeeze tree search if the graphs are large ($p \geq 10$).

(3) If the graphs are very small ($p < 8$), then no initial partitioning is necessary.

(4) If the graphs are random with few vertices having the same degree, then the 0-subdivision distance partition will produce the finest partition most of the time.

(5) If some or all of the graphs are regular, then the 1-subdivision distance partition will produce the finest partition most of the time.

(6) If some of the graphs are derived from BIBD's and the number of

vertices is greater than 40, then the 2-superline distance partition should be used; in all known cases the tree search should merely be a verification of the automorphism partition.

Finally, we briefly comment on a few variations of the tree search that did not perform as well as the ones mentioned above. When R is passed and modified by level, the $O(p^2)$ operations required to manipulate R on each level far outnumber the operations that are saved over searches that maintain R at the top level only. The same is true for tree searches which refine the initial vertex partition $\pi_C(G)$ level by level, even when the refinements are restricted to the top few levels of the tree search. The data indicates that a majority of the vertex partitioning computations should take place before the tree search commences.

7. Conclusion

A method employing invariant graph transforms has been developed to induce vertex partitions in graphs. The method is stronger than any other deterministic (non-tree search) vertex partitioning algorithm presented in the literature, and is in fact capable of producing the automorphism partition in a large collection of graphs. No graph has been found in which the method failed to produce the automorphism partition. The squeeze tree search technique has been introduced. This technique greatly reduces the number of automorphisms needed to determine the automorphism partition of a graph. The combination of the vertex partitioning method and the squeeze tree search technique provides a means of efficiently generating and verifying graph automorphism partitions in polynomial time, even when the graphs are strongly regular or balanced incomplete block designs.

REFERENCES

1. Bhat, K.V.S., Refined vertex codes and vertex partitioning methodology for graph isomorphism testing, *IEEE Trans. Systems Man Cybernet.* **10**(10) (1980) 610–615.
2. Bitner, J. and Reingold, E., Backtrack programming techniques, *Comm. ACM* **18**(11) (1975) 651–656.
3. Bolles, R.C., Robust feature matching through maximal cliques, *Proc. SPIE Tech. Symp. Imaging and Assembly*, Washington, DC, 1979.
4. Chen, J.K. and Huang, T.S., A subgraph isomorphism algorithm using resolution, *Pattern Recognition* **13**(5) (1981) 371–379.
5. Corneil, D.G. and Gotlieb, C.C., An efficient algorithm for graph isomorphism, *J. ACM* **17**(1) (1970) 51–56.
6. Gaschnig, J., A general backtrack algorithm that eliminates most redundant tests, *Proc. International Conf. Artificial Intelligence*, Cambridge, MA (1977) 457.
7. Gati, G., Further annotated bibliography on the isomorphism disease, *J. Graph Theory* **3** (1979) 95–109.
8. Ghahraman, D.E., Wong, A.K.C. and Au, T., Graph optimal monomorphism algorithms, *IEEE Trans. Systems Man Cybernet.* **10**(4) (1980) 181–188.

9. Haralick R.M. and Elliott, G.L., Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence* (1980) 263–313.
10. Haralick, R. and Shapiro, L., The consistent labeling problem I, *IEEE Trans. Pattern Anal. Machine Intelligence* **1**(2) (1979) 173–184.
11. Haralick R. and Shapiro, L., The consistent labeling problem II, *IEEE Trans. Pattern Anal. Machine Intelligence* **2**(3) (1980) 193–203.
12. Karp, R.M., Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher (Eds.), *Complexity of Computer Computations* (Plenum, New York, 1972) 85–103.
13. Karp, R.M., On the computational complexity of combinatorial problems, *Networks* **5** (1975) 45–68.
14. Kuhn, W.W., Graph isomorphism using vertex adjacency matrices, *Proc. 25th Summer Meeting Canadian Mathematical Congress*, Lakehead University, Port Arthur, Ont. (1971) 471–476.
15. Levi, G., Graph isomorphism: a heuristic edge-partitioning-oriented algorithm, *Computing* **12** (1974) 291–313.
16. Mathon, R., Sample graphs for isomorphism testing, *Proc. Ninth Conf. Combinatorics, Graph Theory, and Computing* (1978) 499–517.
17. Nagle, J.F., On ordering and identifying undirected linear graphs, *J. Math. Phys.* **7** (1968) 1588–1592.
18. Pease, M.C., The undirected binary *n*-cube microprocessor array, *IEEE Trans. Comput.* **6** (1977).
19. Read R.C. and Corneil, D.G., The graph isomorphism disease, *Graph Theory* **1** (1977) 339–363.
20. Schmidt, D.C. and Druffel, L.E., A fast back-tracking algorithm to test directed graphs for isomorphism using distance matrices *J. ACM* **23**(3) (1976) 433–445.
21. Shah, Y.J., Davida, G.I. and McCarthy, M.K., Optimum features and graph isomorphism, *IEEE Trans. Systems Man Cybernet.* **4**(5) (1974) 313–319.
22. Stockton, F., Linearization and standardization of graphs, Shell Development Co. Tech. Progress Rept. No. 2–68, Project No. 34430, 1968.
23. Sussenguth, Jr., E.H., A graph-theoretic algorithm for matching chemical structures, *J. Chem. Doc.* **5** (1965) 36–43.
24. Turner, J., Generalized matrix functions and the graph isomorphism problem, *SIAM J. Appl. Math.* **16** (1968) 520–526.
25. Turner, J., Key-word indexed bibliography of graph theory, in: F. Hararay (Ed.), *Proof Techniques in Graph Theory* (Academic Press, New York, 1969).
26. Ullmann, J.R., *Pattern Recognition Techniques* (Crane Russak, New York, 1973) 198.
27. Ullmann, J.R., An algorithm for subgraph isomorphism, *J. ACM* **23**(1) (1976) 31–42.
28. Unger, S.H., GIT—a heuristic program for testing pairs of directed line graphs for isomorphism, *Comm. ACM* **7**(1) (1974) 26–34.
29. Waltz, D., Generating semantic descriptions from drawings of scenes with shadows, in: P. Winston (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).
30. Weisfeiler, B., On construction and identification of graphs, *Lecture notes in Mathematics* **558** (Springer, Berlin, 1976).
31. Barrow, H.G. and Popplestone, R.J., Relational descriptions in picture processing, in: B. Meltzer and D. Michie (Eds.), *Machine Intelligence* **6** (Edinburgh University Press, Edinburgh, 1971).

Received August 1982; revised version received October 1982