BIT 32 (1992), 197–214.

CANONICAL REPRESENTATIONS OF PARTIAL 2- AND 3-TREES

STEFAN ARNBORG* and ANDRZEJ PROSKUROWSKI**

Department of Numerical Analysis and Computing Science Royal Institute of Technology 100 44 Stockholm, Sweden

Department of Computer and Information Science, University of Oregon Eugene, OR 97403, USA

Abstract.

We give algorithms constructing canonical representations of partial 2-trees (series parallel graphs) and partial 3-trees. The algorithms can be implemented in log-linear space, or in linear time using quadratic space.

CR categories: G.2.2.

1. Introduction.

A canonical representation of a family of graphs assigns to each member of the family a label that is independent of any arbitrary vertex numbering: two graphs have the same canonical representation if and only if they are isomorphic. Thus, the graph isomorphism problem can be solved efficiently using canonical representations if such representations can be efficiently computed and compared. Other uses of canonical representations are to investigate the structure of the automorphism group of a graph and to generate random graphs with some distribution over isomorphism classes.

Most graph representations are not canonical since vertices are arbritrarily numbered. But if we consider all possible vertex permutations, compute the corresponding representations, and select the lexicographically smallest, then we get a canonical representation. The set of vertex permutations yielding the lexicographicaly smallest representation is a coset of the automorphism group for the graph, regarded as a subgroup of the symmetric group on the vertex set.

A straightforward application of the above procedure has cost, exponential in the

* Supported in part by a grant from the Swedish Natural Science Research Council.
** Research supported in part by the Office of Naval Research Contract N00014-86-K-0419.
Received January 1991. Revised December 1991.

graph size, since there are exponentially many vertex permutations to minimize over. But in some cases it is possible to constrain the set of explicitly considered permutations in such a way that the whole procedure can be performed in polynomial time. We need only consider a set guaranteed to contain at least one representative of each coset of the automorphism group of the given graph. In this paper we show how these ideas yield an algorithm which produces a canonical representation for partial 3-trees in log-linear time, and thus also solves the isomorphism problem for partial 3-trees in log-linear time. Previously, the graph isomorphism problems for the graphs of bounded valence (Luks [18]), genus (Filotti and Mayer [12], Miller [19], [20]), and tree-width (Bodlaender [7]) (none of which is a subfamily of the other) have been shown solvable in polynomial time. Linear time algorithms for isomorphism of planar graphs (and thus also for partial 2-trees, which are planar) are already known (Fontet [13]; Hopcroft and Wong [15]; Colbourn and Booth [10]).

For a fixed value of the integer parameter k, partial k-trees are exactly subgraphs of chordal graphs with the maximum clique size $k + 1$. Thus, partial 1-trees are the acyclic graphs (forests), and partial 2-trees are the series-parallel graphs with no K_4 minors or homeomorphs).

Partial k-trees have been in the focus of attention in recent years because of their interesting algorithmic properties. For a large number of inherently difficult (on general graphs) discrete optimization problems, partial k-trees admit a linear time solution algorithm when the value of k is fixed and the partial k-tree is given with its k-tree embedding. Somewhat discouraging is the fact that, for a general value of k, we do not know how to construct a k-tree embedding of such a graph in less than $O(n^{k+2})$ time. The only more efficient and practical recognition (and embedding) algorithms known are for $k \leq 3$. A quadratic time recognition algorithm for any given k exists as a consequence of Robertson and Seymour's [23] results, but it uses a list of minimal forbidden minors which it is not known how to find and which can be of astronomical size.

The class of partial k-trees is also identical with the class of graphs of tree-width k (Scheffer [26], Wimer [30]). In the next section, we will give an iterative definition of partial k-trees that is the basis of our approach to solve problems for this class of graphs. Bodlaender proposed an algorithm for deciding isomorphism of partial k-trees [7]. His method is based on the brute force k-tree embedding method of Arnborg, Corneil and Proskurowski [3], where all k-vertex separators of the given partial k-tree are tested for suitability as separators in a k-tree embedding. This algorithm requires solving bipartite matching problems and takes $O(n^{k+4.5})$ time. We follow a different approach for $k = 2, 3$. For these values of k there exist small complete sets of safe reduction rules that determine k-tree embeddings of a given partial k-tree. With help of these reduction rules we produce a canonical string representing the graph in log linear time, thus lowering the computation time from polynomial (actually, $O(n^{7.5})$) to log-linear or linear time for isomorphism of partial 3-trees.

The procedure we use is based on a canonical reduction sequence obtained from the safe reduction rules reported in Arnborg and Proskurowski [5]. For any given graph, the set of vertices reducible according to a given rule is fixed by the automorphism group of the graph[1]. Each reduction involves a separator of the graph with one, two or three vertices. Whether two reduced vertices are automorphic depends on symmetries between the corresponding separators. Our method keeps a record of symmetries of the reduced parts of the graph through a sequence of labels and orientations attached to the separators used in the reduction process. Two reduced subgraphs cut off by such separators are isomorphic (the isomorphism mapping one separator to the other) if and only if the labels of the two separators are equal, and their orientations indicate the correspondence between the separator vertex sets.

A reducible vertex represents a k-leaf in an embedding k-tre. Thus, adjacent vertices cannot be reduced in parallel. To deal with this, we refine the reduction rules to deal with overlapping reduction instances. These refined reduction rules allow us to construct a *parse tree*, where each node is associated with a reduction instance and two nodes are adjacent if the reduction instance corresponding to one node creates a (hyper-) edge involved in the reduction corresponding to the other. This tree is used to implement efficiently the algorithms computing canonical representations.

Our paper is organized as follows. After defining the necessary terminology in Section 2 we introduce the method in Section 3 by applying it to partial 2-trees. This special case is much simpler. Then the additional reduction instances necessary for partial 3-trees are derived in Section 4. The algorithm is presented and analyzed in Section 5.

2. Definitions and terminology.

We will use standard graph theory terminology, as found, for instance, in Bondy and Murty [8]. We will also make use of concepts from the realm of hypergraphs, but will introduce them first in Section 4. Some elementary and completely standard group theory is also used; see, for example, Rotman [25]. We will now define some basic concepts.

A *walk* is a sequence of vertices such that every two consecutive vertices are adjacent. If all the vertices are different, we have a *path*. A walk forms a *cycle* if only its first and last vertices are identical. A set of k vertices, every two of which are adjacent, is called a k-clique. The graph on k vertices whose vertex set is a k-clique will be denoted K_k. A (minimal) subset of vertices of a graph such that their removal disconnects the graph is a (minimal) *separator*. A *k-tree* is a connected graph with no K_{k+2} subgraph such that every minimal separator is a k-clique. Equivalently (Rose

[1] This means that the automorphism group of the graph permutes these vertices among themselves.

[24]), the complete graph on k vertices (K_k) is a k-tree, and any k-tree with $n > k$ vertices can be constructed from a k-tree with $n - 1$ vertices by adding a new vertex adjacent to all vertices of a k-clique of that graph. In this new graph, the added vertex is a *k-leaf.* A *partial k-tree* is any subgraph of a k-tree.

While partial k-trees are undirected, simple graphs (without multiple edges or self-loops), in the course of our presentation we will allow both undirected *edges* and directed *arcs* (ordered pairs of vertices), as well as parallel edges and arcs. Those mixed graphs will be intermediate results of applying *graph rewriting rules*, consisting of replacing a subgraph isomorphic to the lefthand side of such a rule by the righthand side subgraph. In our case, the latter has always fewer vertices than the former and thus a set of such rules defines possible *reduction sequences.* Given a class of graphs, a rewriting rule such that its application preserves membership in both the class and its complement is called a *safe* rule. A set of reduction rules such that any non-trivial graph in the class contains as a subgraph the lefthand side of some rule is called a *complete* set of rules.

Complete sets of safe reduction rules for partial k-trees are known for $k < 3$ (Arnborg and Proskurowski [5]). Intuitively, they correspond to *pruning* of k-leaves in an embedding k-tree, safe in the sense of the existence of such a k-tree. For partial 1-trees (forests), the set of reduction rules consists of the removal of pendant vertices (of degree 1) and of isolated vertices. For partial 2-trees (series-parallel graphs) we have additional *series* and *parallel* rules, in which a path of degree 2 vertices is replaced by an edge, and multiple edges are replaced by one edge, respectively. For partial 3-trees, the additional rules deal with three cases of degree 3 vertex reductions: the *triangle*, the *buddy*, and the *cube* rules (*cf.* Figure 1, where only subgraph vertices matching the white vertices of the lefthand side can have additional adjacencies than those shown). The cube-like configuration with the 'hub' (vertex x in Figure 1) of degree greater than 3 can be safely reduced, as well.

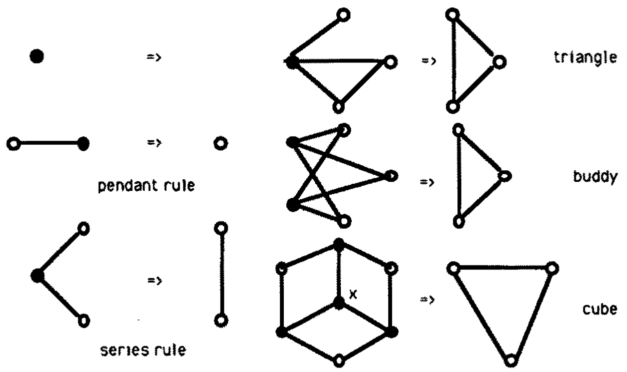

Figure 1: Reduction rules for forests, series-parallel graphs and partial 3-trees.

In a k-tree, vertices may be partially ordered with respect to the last $(k+1)$-complete subgraph ('the root') [21]. This partial order might be inherited by a partial k-tree, once we decide an embedding. Non-adjacent vertices reduced according to applicable reduction rules are not related in partial orders corresponding to embeddings having those vertices as k-leaves. Such reductions can be performed simultaneously (or, emulating simultaneity, consecutively in any order), leaving the necessary information as labels on other affected elements of the graph (for instance, on the pendant vertex in the pendant rule or on the added edge in the series rule).

Unfortunately, some instances of the reduction rules may deal with adjacent proposed k-leaves; obviously, for a k-tree embedding, only one of two such vertices is a k-leaf. This situation has to be dealt with separately. We note that there is a simple solution if the conflicting rules are different, *e.g.*, a vertex reducible according to the triangle rule is adjacent to a vertex reducible according to the buddy rule. We simply order the rules and say that a higher priority rule takes precedence. The remaining case of conflict is where adjacent vertices are reducible according to the same rule. To break the ties in this case, we consider a refined list of rules derived from the complete set of safe rules presented in [5].

3. Canonical representation of partial 2-trees.

We start our exposition by presenting the algorithm for partial 2-trees and then we generalize it to partial 3-trees.

Our algorithm is based on a construction of (vertex and edge) labels that record the sequence of reduction of the original graph G leading to a set of labeled isolated vertices. From these, we construct the final label that is a canonical representation of G and, as such, would also allow a unique (and efficient) reconstruction of G.

Initially, every node label is (0) and every edge label is (0,0). The second component in an edge label is the orientation information, recorded as "an arrow" or its absence, depending on the set of end vertex permutations that yield the minimum label value. When the identities of vertices are neglected in a subsequent reduction, the arrow is replaced by a 0, 1, or -1, depending on the direction (with or against the arrow) in which the edge is scanned in that reduction.

The algorithm will reduce a given connected graph to the single labeled vertex in phases performed in the following manner. Every phase begins with finding the first applicable reduction rule (closest to the beginning of the list of reductions, to be presented shortly). All instances of this rule in the entire graph are then found and the corresponding reductions are performed, resulting in a modified (reduced) graph, and some new labels. Since each reduction decreases the size of the graph, in $O(n)$ phases the graph will be reduced to a single vertex.

3.1. *Reduction rules.*

In this subsection we describe the rewriting rules (tailored to the needs of unique labeling) that reduce a given connected partial 2-tree to a single labeled vertex. The rules are given in their scanning order, together with the associated new label(s).

0 multiple vertices and edges. The rules below may create several edges with the same end-points (series rules) or several labels for a vertex (pendant and chain with identical end-vertices). The latter is referred to in the pendant rule 1 as a label being merged into the label of a vertex.

0.1 vertex label merge. The merging of multiple vertex and self-loop labels into a vertex label means that we keep a set of labels with multiplicities to describe the merged labels. Before the vertex itself is reduced, we must construct a proper label for it. This is done simply by lexicographically sorting the labels, and keeping duplicates in the sorted list l. The label will be $(0.1; l)$.

0.2 parallel rule. This rule is applicable when m edges have the same end-points s_1 and s_2 ($s_1 \neq s_2$, arbitrarily ordered). Let the labels of these edges be l_i and let d_i be 1 if edge i has an arrow from s_1 to s_2, -1 if an arrow is directed from s_2 to s_1, and 0 if it is not present. The parallel edges are replaced by one new edge between s_1 and s_2. This edge has a label $(0.2; l)$, where l is the lexicographically smallest of two sorted lists corresponding to edge orientations from s_1 to s_2 and from s_2 to s_1. The new edge will have arrow $(s_1 \to s_2)$ if the first list is smaller, $(s_2 \to s_1)$ if the second is smaller, and none if they are equal. The first list has the members $(l_i, d_i)_{i=0}^{m-1}$, the second $(l_i, -d_i)_{i=0}^{m-1}$.

1 pendant rule. This rule applies to vertices of degree 1 (pendant vertices) and to edges with both endpoints the same (self-loops).

1.1 dipole. If there is a pair of adjacent pendant vertices, they must form a connected component of the graph consisting of two vertices labeled l_1 and l_2 both incident to an edge labeled l', directed from the first to the second vertex ($d = 1$) or not at all ($d = 0$). This graph is reduced to a single vertex with label $(1.1; l_1, (l', d), l_2)$ or $(1.1; l_2, (l', -d), l_1)$, whichever is lexicographically smallest.

1.2 pendant vertices. Assume the set of pendant (degree 1) vertices is independent. One vertex and one edge is removed by a pendant vertex removal. The label l' together with the orientation d of the edge form a pair (l', d) with d defined as 1 if the arrow of the edge is directed towards the pendant vertex, -1 if it is directed the other way, and 0 if it is not present. Combined with the label of the vertex, l, they form the label $(1.2;(l',d),l)$, which is merged subsequently into the label of the separating vertex according to rule 0.1.

1.3 self-loops. The label l' of a self-loop is merged into the label of its end-vertex according to rule 0.1.

2 series rule. This rule applies to vertices of degree 2. A set of such vertices inducing a connected graph will induce a path or a cycle in the graph, as detailed below:

2.1 chain rule. A maximal set of degree 2 vertices inducing a connected graph, and ending in two vertices, s_1 and s_2, of degree higher than 2 consists of a path $(v_1,\ldots,v_m)$ with v_1 adjacent to s_1 and v_m adjacent to s_2. Let l_i be the label of $v_i, i = 1,\ldots,m$ and l'_i of the edge $(v_i, v_{i+1}), i = 0,\ldots,m$, with $v_0 = s_1$ and $v_{m+1} = s_2$. Let d_i be 1, -1 or 0, depending on whether the arrow of (v_i, v_{i+1}) is directed from v_i to v_{i+1}, from v_{i+1} to v_i, or not present. The vertices $v_i, i = 1,\ldots,m$ and their incident edges are replaced by an edge connecting s_1 and s_2. The label of this edge is the lexicographically smallest of two labels, $(2.1;(l'_0,d_0),l_1,(l'_1,d_1),\ldots,l_m,(l'_m,d_m))$ and $(2.1;(l'_m,-d_m),l_m,(l'_{m-1}, -d_{m-1}),\ldots,l_1,(l'_0,-d_0))$. The arrow will point from s_1 to s_2 if the first alternative is smaller, from s_2 to s_1 if the second is smaller, and is absent if they are equal or if $s_1 = s_2$. (In the latter case the edge is a self-loop).

2.2 ring rule. A connected two-regular graph is a cycle consisting of vertices $v_0,\ldots,v_{m-1}$, with vertex v_i adjacent to vertex v_{i+1} for $i = 0,\ldots,m-2$ and with v_0 adjacent to v_{m-1}. It is reduced into a single vertex labeled with the lexicographically smallest of $2m$ labels $(2.2;(l'_{i+jk}, kd_{i+jk}), l_{i+jk})_{j=0}^{m-1}$ where $i = 0,\ldots,m-1$, $k = 1, -1$, and indices are computed modulo m.

Since the above rules refine the complete set of confluent reduction rules recognizing partial 2-trees ([29,5]), any such graph will be assigned a label by an algorithm implementing the labeling process. The uniqueness of such a label is due to the fact that each intermediate label depends only on the unique labeling phase during which the particular reduction is performed. This follows from the independent nature of application of reduction rules. The phase is defined by the first reduction rule on the list that is applicable, and the order of applying any two reductions in the same phase does not influence the values of resulting labels.

3.2. *Example.*

We will illustrate the process of creating a canonical label by reduction of the partial 2-tree whose adjacency lists are:

vertex	neighbors
1	13
2	13,14
3	14
4	15
5	6,15
6	5,7
7	6,15
8	12
9	10,12
10	9,11
11	10,12
12	8,9,11,13,15
13	1,2,12,14,15
14	2,3,13,15
15	4,5,7,12,13,14

The canonical reduction sequence produced is the following, where groups of parallel reductions are separated by horizontal lines:

rule	reduced	edges removed	label	labeled	arrow
1.2	1	(1,13)	l_1	13	
	3	(3,14)	l_1	14	
	4	(4,15)	l_1	15	
	8	(8,12)	l_1	12	
0.1			l_2	13	
			l_2	14	
			l_2	15	
			l_2	12	
2.1	2	(13,2), (2,14)	l_3	(13,14)	
	5,6,7	(15,5), (5,6), (6,7), (7,15)	l_4	(15,15)	
	9, 10, 11	(12,9), (9,10), (10,11), (11,12)	l_4	(12,12)	
0.1			l_5	12	
			l_5	15	
0.2		(13,14)	l_6	(13,14)	
2.1	12	(13,12), (12,15)	l_7	(13,15)	
	14	(13,14), (14,15)	l_8	(15,13)	→
0.2		(13,15)	l_9	(13,15)	→
1.1	13, 15	(13,15)	l_{10}		

The label abbreviations used above are as follows:

name	value
l_1	(1.2; (0, 0), (0))
l_2	(0.1; (0), l_1)
l_3	(2.1; (0, 0), (0), (0, 0))
l_4	(2.1; (0, 0), (0), (0, 0), (0), (0, 0), (0), (0, 0))
l_5	(0.1; l_2, l_4)
l_6	(0.2; (0, 0), (l_3, 0))
l_7	(2.1; (0, 0), l_5, (0, 0))
l_8	(2.1; (0, 0), l_2, (l_6, 0))
l_9	(0.2; (0, 0), (l_8, -1), (l_7, 0))
l_{10}	(1.1; l_2, (l_9, 1), l_5)

and we can get the explicit label from l_{10} by expanding the above abbreviations.

4. Canonical representation of partial 3-trees.

The idea behind the algorithm for partial 3-trees is similar to that of the algorithm constructing a canonical representation of partial 2-trees. The reduction of vertices is performed in consecutive stages, where each stage consists either of independent reduction instances or of groups of dependent reduction instances of the same kind. The situation is more complicated for partial 3-trees for two reasons. For one, the reduction information (recorded in labels) often concerns three vertices, and thus the resulting label must be associated with triples of vertices. We will present this as the labeling process for hyperedges of order 3. The second reason for a more complicated algorithm is the large number of ways in which reduction rules of the same kind can involve adjacent vertices. For instance, more than one vertex of a triangle can have degree 3. Below, we elaborate on these differences leading to an algorithm constructing a canonical label for a given connected partial 3-tree.

4.1. *Labeling of hyperedges.*

Each of the three reduction rules involving a degree three vertex causes the elimination of that vertex and edges incident with it and replaces them by edges between the neighbors of the vertex. (We call this reduction of a single vertex v adjacent to the separator vertices s_1, s_2, s_3 a *basic degree* 3 *reduction.*) We will separate this *topological* action from the *labeling* action of creating a labeled hyperedge corresponding to the three neighbors of the eliminated vertex.

Let a *sufficient set of permutations* denote permutations of vertices of a subgraph that are able to represent all symmetries (automorphism group) of the subgraph. (This can be achieved by a smaller than full symmetric group set of permutations when permutations exchanging non-symmetric vertices are omitted.)

Let 0 be the label of hyperedges (vertices, edges, or triangles) in the original graph.

A label of a hyperedge is determined with respect to a permutation of its vertices and is given an *orientation* represented by a subset of permutations of these vertices. These permutations are symmetric with respect to the label (they constitute the projection, into the symmetric group on the vertices of the hyperedge, of a coset of the automorphism group for the graph reduced into the hyperedge). When a hyperedge is removed in a reduction operation involving one of its vertices, its label will form a component of the label of the created hyperedge, together with an indication of how the orientation of the removed hyperedge corresponds to the orientation of the created hyperedge. Consider an orientation D of a removed hyperedge and a permutation σ of the union of vertices in the removed hyperedges. D is coded with respect to σ in the following way. Recall that D is a set of permutations of a subset of the vertices appearing in σ, so each vertex occurring in D can be replaced by its index in σ. The resulting set of integer lists is then sorted lexicographically.

As an example, consider reduction of a degree 3 vertex v with neighbors a, b and c, This reduction removes at most one 1-hyperedge, three 2-hyperedges, and three 3-hyperedges. Now, for instance, the permutation $\sigma = (a, b, c, v)$ will cause the orientation $D = \{(a, b, v), (a, v, b)\}$ of a 3-hyperedge $\{a, b, v\}$ to be encoded as $((1,2,4),(1,4,2))$.

We can now describe the reduction of a set of vertices $R = \{v_i\}_{i=1}^{k}$ separated from the rest of the graph by vertices $S = \{s_i\}_{i=1}^{l}$:

Produce a sufficient set P of permutations of $R \cup S$. For each permutation p in P, produce a label as follows: consider the set of (*label, orientation*) pairs that contain some vertices of R. Replace each orientation with its encoding *wrt* p and sort the pairs to get a label *wrt* p. The subset of permutations that yield the lexicographically smallest label l constitute the orientation of the new edge. The minimum value l and the rule number r according to which the reduction is made are used to build the label $(r; l)$ of the new hyperedge.

Observe that the previous label construction for partial 2-trees was only slightly different and can be easily changed to the present one. As an example, in the chain rule, each d_i would be changed from 1 to $((i, i+1))$, from -1 to $((i+1, i))$ and from 0 to $((i, i+1), (i+1, i))$. An arrow on an edge between vertices a and b would be represented as an orientation $((a, b))$ or $((b, a))$ and its absence would be represented by orientation $((a, b), (b, a))$.

4.2. *Reduction rules for partial* 3-*trees*.

Except for applications of the parallel rules, the sequence of reductions made is decided solely on basis of adjacency information – we never investigate which type of hyperedge (2- or 3-hyperedge) makes two vertices adjacent. So in this section we can consider the graph represented by its clique representation when looking for vertex sets to reduce. After this set has been decided, we must use another data structure to find the set of hyperedges containing some reduced vertex, as detailed in Section 5.

The data structure for hyperedges is also used to decide when parallel rules (rule group 3) are to be invoked. The correctness of our algorithm rests on the fact that for every partial 3-tree there is a unique way to select a number of non-overlapping reduction rules and to describe the reduced parts of the graph uniquely in the labels created. This in turn follows from a careful reading of this section: for every rule reducing a partial 3-tree, have we identified all ways in which reduction instances of this rule can overlap? The numbering of the rules is intended to simplify the understanding of the case analysis.

3 parallel triangles. This rule is applicable when two or more 3-hyperedges have the same set of vertices. Let the vertices be $S = (s_1, s_2, s_3)$, and the labels of the hyperedges l_i, $i = 1, \ldots, m$. The orientations d_i, $i = 1, \ldots, m$ are sets of permutations of S. Let $d_i^{(p)}$ be the coding of d_i *wrt* a permutation p of S. For each permutation p of S, form the set $\{(l_i, d_i^{(p)})\}_{i=1}^{m}$, sort it lexicographically and extract the lexicographically smallest list (over p) as well as those p giving a minimum, and call the smallest list l and the set of permutations P. The new hyperedge with vertices s_1, s_2, s_3 has label $(3; l)$ and orientation P.

4 isolated reduction instances.

4.1 triangle. A vertex that is triangle-reducible and not adjacent to another triangle-reducible vertex can be reduced directly. This reduction of a single vertex adjacent to three separator vertices will be called the *basic degree* 3 *reduction.* The sufficient set of permutations consists of every permutation of the s_i, each followed by v.

4.2 buddy. Vertices in a buddy configuration not adjacent to another buddy configuration can be reduced simultaneously with the basic degree 3 reduction in all occurrences.

4.3 cube. The three reducible vertices in a cube configuration not adjacent to another cube are reduced with the basic degree 3 reduction.

5 conflicting triangles.

5.1 diamonds. We can consider separately the cases when two adjacent triangle-reducible degree 3 vertices v_1, v_2 have two neighbors s_1, s_2 in common.

5.1.1 K$_4$. More than one occurrence of the diamond rule with adjacent reducible vertices is possible only if a connected component of the graph is the 4-clique K_4. The sufficient set consists of all 24 permutations of the vertices $v_i, i = 1, 2, 3, 4$.

5.1.2 K_4^-. If the vertices v_1 and v_2 are not adjacent, they induce, together with their common neighbors s_1, s_2, the four-clique without an edge, K_4^-. A sufficient set of permutations to consider is each of the two permutations of $\{s_1, s_2\}$ followed by each permutation of $\{v_1, v_2\}$.

5.2 Subgraph H. The remaining configurations of adjacent degree 3 vertices reducible according to the triangle rule and with at most one common neighbor (triangle-reducible for short) are discussed below. For a given partial 3-tree G, let us consider a maximal connected subgraph H consisting of triangle-reducible vertices subject to conflicting reductions according to the *triangle* rule. We will investigate the structure of those subgraphs and will make unique choices of *independently* reducible vertex sets in G. We consider the following cases of reductions in G depending on the degrees of vertices in H:

5.2.1 degree 1 only. Two adjacent vertices of degree 1 in H, v_1 and v_2, represent one of the two subgraphs in G shown in Figure 2. In the case of a four-vertex separator, other reduction rules must be applicable in the rest of the graph ('beyond the four separating vertices'), or else G cannot be a partial 3-tree. In the case of a 3-vertex separator, a separate reduction rule allows us to reduce v_1 and v_2, creating a hyperedge containing the separator vertices, with a unique label describing the reduced subgraph of the original graph (none of the separator vertices is triangle-reducible).

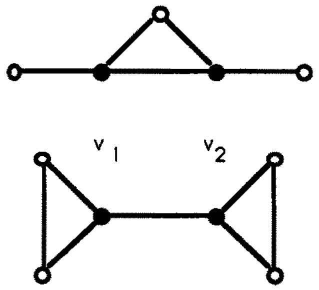

Figure 2: Subgraphs of H with two adjacent degree 1 vertices.

5.2.2 degree 1 and 2 or 3. In this case we have a vertex of degree 1. If there are more, they must be nonadjacent. Nonadjacent vertices of degree 1 in H can be subjects to the basic degree 3 reductions in G, since these do not conflict with each other.

5.2.3 degree 2 only. The vertices of degree 2 form a cycle in H. Let us call an edge t if it is a side of a triangle of G and f otherwise. Notice that end vertices of adjacent t edges have one common neighbor in G. Consider those vertices incident to both a t and an f edge; call this set A. Depending on the relation between A and H, we have three subcases (Figure 3).

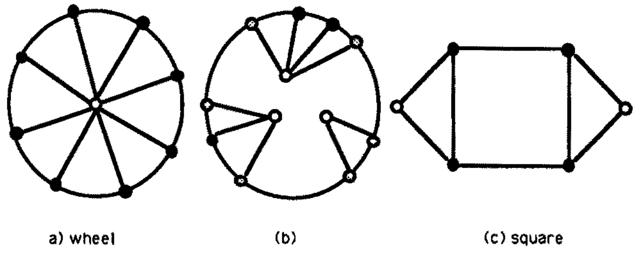

Figure 3: Cycles in H (a) a wheel, (b) a general case, (c) the square.

5.2.3.1 wheel. The set A is empty: all edges of H are t edges. This is the wheel configuration, reduced to a single vertex label ("the hub's") according to a separate rule.

5.2.3.2. collection of paths. A does not contain all vertices of H and its vertices partition the set of the remaining vertices of H into connected components. Those can be dealt with in a manner similar to that in rules 4.1, 5.2.1, and 5.2.2.

5.2.3.3 square. If H consists of alternating t and f edges (all the vertices are in A), then we have to consider subcases depending on the number of edges in H (it is trivially greater than 3). When H has 4 edges (only two triangles are involved), the triangles' "third vertices" form a separator of G. The corresponding configuration (square, Figure 3(c)), is the left hand side of a separate reduction rule. If there are more than two triangles, then there must be another instance of a reduction rule "beyond the separating vertices" (i.e., in the subgraph of G induced by vertices of G other than in H), or else G is not a partial 3-tree.

5.2.4 degrees 2 and 3. If H consists of both vertices of degree 2 and vertices of degree 3, then the vertices of degree 2 form in H paths that end in degree 3 vertices. When such a path has exactly two degree 2 vertices, these can be reduced according to rule 5.2.1. Otherwise, there are unique and nonadjacent vertices of degree 2 in H and the corresponding vertices in G can be reduced with the basic degree 3 reduction, similarly to the situation in 5.2.2.

5.2.5 degree 3 only (prism). Since a vertex in H is of degree 3 in the original partial 3-tree G, a cubic H is identical with G. To analyse this case, we consider the multigraph $\bar{H}$ obtained from H in the following manner: The set of vertices of $\bar{H}$ is the set of edges of H *not* in the triangles. A vertex of $\bar{H}$ is incident with an edge of $\bar{H}$ if the corresponding triangle contains a vertex of H incident with that edge in H. We will show that $\bar{H}$ is a series-parallel graph, which has important consequences in determining unique triangle reductions in H (or, equivalently, G). A multigraph is *series-parallel* if and only if it does not contain a subgraph homeomorphic to K_4. However, if $\bar{H}$ contains such a subgraph, then H contains as a minor the 4-regular Duffin graph (see Figure 4(b)), and is thus not a partial 3-tree. Since all vertices of $\bar{H}$ have degree 3, there must be instances of independent parallel edge reductions in $\bar{H}$. Unless H has six vertices and nine edges ("the prism", see Figure 4(a)) reduced to a single vertex by a separate rule, an instance of the parallel edge reduction in $\bar{H}$ corresponds to the subgraph of the 5.2.3.3 *square* rule in H. Thus, the only new configuration for this case is where H, and thus also G, is the prism, Figure 4(a).

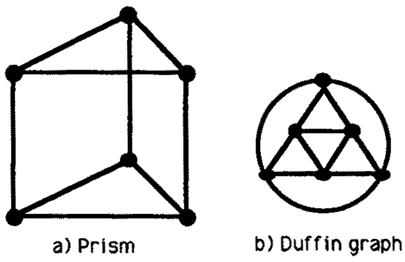

Figure 4: 3-regular subgraphs (a) the prism, (b) a minimal forbidden minor.

6 conflicting buddies. The 3-leaves v_1, v_2 in an instance of the buddy reduction can be adjacent to 3-leaves u_1, u_2 in another instance of the buddy reduction only if u_1, u_2 are commonly adjacent to a third vertex w. This third vertex may be identical with or different from the third common neighbor of v_1, v_2, leading to two configurations that can be incorporated as the left-hand-sides of new reduction rules. The former is a case of the 5.2.3.1 *wheel* rule discussed above, the latter, *cat's cradle*, is shown in Figure 5.

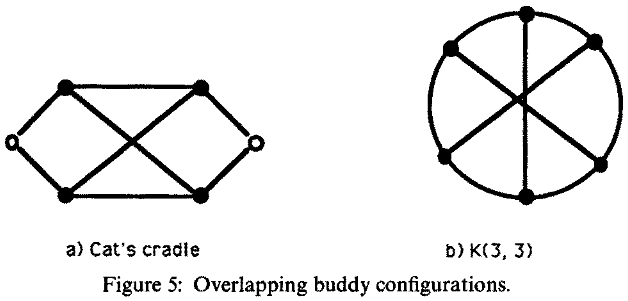

Figure 5: Overlapping buddy configurations.

K$_{3,3}$. Two cat's cradles can overlap only when the graph has a connected component which is $K_{3,3}$.

6.2 cat's cradle. This is the other case where $w \neq \Gamma(\{v_1, v_2\})$. The separator vertices s_1, s_2 and the cycle (v_1, v_2, v_3, v_4) have edges (s_1, v_1), (s_1, v_3), (s_2, v_2) and (s_2, v_4) between them.

7 conflicting cubes. There are two basic cases of possible conflicts between the reduced vertices in two different instances of the cube reduction.

7.1 cube. In one case, the purported 3-leaf vertices v_1, v_2, v_3 in one instance of the cube reduction are adjacent to 3-leaf vertices of another instance. This occurs only in the 8-vertex, 12-edge three-dimensional cube graph. This is because the vertices u_1, u_2, and u_3 (*cf.* Figure 6) have then degree 3 and are adjacent to a common neighbor (the hub of that instance), also of degree 3.

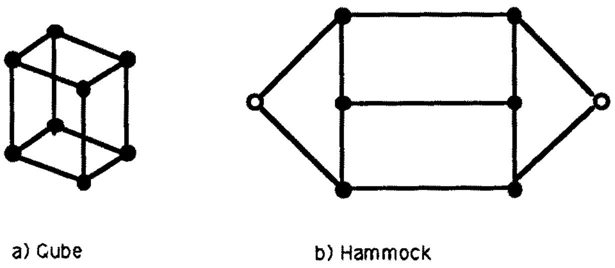

Figure 6: Overlapping cube configurations.

7.2 hammock. In the second case, the hub x of one instance is a 3-leaf of the other instance. This implies that one of the 3-leaves of the first instance, say v_3, is the hub of the second instance and its other neighbors u_1, u_2 are the remaining 3-leaves of the second instance. The vertices u_1, u_2 must have another common neighbor y. If this vertex is identical with the remaining vertex u_3 of the original cube, u_1, u_2 are triangle-reducible. If y is different from u_3, this leads to a new reduction rule with the left-hand-side configurations given in Figure 6(b).

5. Fast canonical labeling and isomorphism test.

We will now describe some implementation details for the simultaneous reduction and computation of hyperedge labels described in Section 4. Let n be the number of vertices of the graph (this reflects the *size* of the input since any partial 3-tree has fewer than $3n$ edges). Moreover, each reduction removes at least one vertex and adds a constant number of adjacencies and labels. Thus, the total number of adjacencies and hyperedges in the graph during reduction is $O(n)$. Hence, also the number of applications of parallel rules (which remove hyperedges but not vertices) is $O(n)$. It can thus be expected that an $O(n \log n)$ (*log-linear*) time realization is possible.

Actually, using the trick of constant time "initialization" of entries in a table (see, for instance, [1]) we can achieve a linear time (albeit quadratic space) algorithm. The anonymous referee convinced us that this algorithm is too complex to present easily. Accordingly, we changed our emphasis to log-linear time, especially since we could not avoid using quadratic space anyway. The log-linear version discussed below is probably better in most repects. Additional "tricks" required by the linear time algorithm are presented in [4].

The following tasks present certain difficulties:

(i) Sorting $O(n)$ labels, each of length $O(n)$, when producing labels for applications of parallel rules.

(ii) Finding the lexicographically smallest label for m shifts of m labels, each of size k, where mk is $O(n)$, when deciding a label for the ring and wheel reductions (2.2 and 5.2.3.1).

(iii) Finding instances of applicable reductions.

Actually, there are already methods available to overcome these obstacles. First, the *abbreviations* introduced in the example of section 3.2 can be formalized into a *canonical numbering* of labels (or objects labeled), as is done for a tree isomorphism algorithm due to Edmonds and described in, *e.g.*, Aho, Hopcroft and Ullman [1] and Colbourn and Booth [10]. In this way, all labels will have size $O(\log n)$. Two identical branches of the graph will get the same label if they are permuted by the automorphism group of the original graph, since they will correspond to reductions made in parallel. With this canonical numbering, any of the corresponding algorithms by Syslo [28], Shiloah [27], or Booth [9] also solves problem (ii) above.

The total number of times vertex adjacencies change during the reduction process is proportional to the graph size. Thus, maintaining "ready lists" for vertices that reach small enough degree to be considered in the reduction rules resolves problem (iii) above. Below, we elaborate on the *Ready lists* and the *Ring* algorithms.

5.1. *Ready lists.*

The reduction instances are either bounded size, connected graphs cut off by a separator (rules 1.1, 1.2, 1.3, 4.1, 4.3, 5.1.1, 5.1.2, 5.2.3.3, 5.2.5, 6.1, 6.2, 7.1, 7.2), variable size configurations involving only vertices of degree not greater than 3 (2.1, 2.2, 5.2.3.1), parallel rules (0.1, 0.2, 3) or the buddy configuration (4.2). The membership of a vertex in such a configuration may vary during the reduction process. However, membership is altered only when one of the adjacencies of reducible vertices is altered. Since each reduction involves at most adjacencies of three remaining vertices, the total number of times a vertex changes its neighborhood or is part of a created hyperedge (by being in the separator associated with a reduction) is $O(n)$, summed over all vertices. Each time this happens we investigate which is the highest ranking rule by which the vertex is reducible, or if it is a vertex of a hyperedge to which a parallel rule applies. The vertex is then linked into a list for that reduction rule.

5.2. *Ring algorithm.*

The problem of determining the label for the ring reduction (2.2) is equivalent to the problem of finding a lexicographically minimum linearization of a cyclic string. A simple solution to this problem involves recording the change of two pairs of positions in the cyclic string. These positions delimit two identical substrings, minimum so far: the *candidate* substring precedes the other, *challenger*, substring (both start with the overall smallest element). One pair of positions indicates the start of the two substrings, and another pair of positions delimits their explored prefixes. The algorithm calls for a comparison of elements possibly extending the two substrings. If the elements are equal but lexicographically smallest, it is easy to see that the challenger substring would never prove lexicographically smaller than the candidate substring. This causes advancement of the position indicating the beginning of the challenger substring to the last scanned position. The same happens when the element comparison indicates explicitly that the candidate substring is lexicographically smaller than the challenger. Otherwise, the challenger substring becomes the current candidate and the position indicating the beginning of the second string is advanced beyond the last scanned.

6. Conclusions.

The presented method to construct a canonical label for partial 2- and 3-tree is based entirely on the complete systems of confluent vertex-reduction rules that recognize graphs from these classes. Lagergren [17] shows that such systems do not exist for partial k-trees with $k > 3$. Thus, this method is not readily extensible. However, there exist more general graph reduction systems that decide membership in classes of partial k-trees. Namely, Arnborg *et al.* show in [4] that this is the case for any subclass of partial k-trees (fixed k) definable by the *Monadic Second Order Logic* (*MSOL*) (*cf.*, for instance, Courcelle [11]). Specifically, this gives the existence of a graph reduction system for partial k-trees (any fixed k). Such a graph rewriting system can be implemented as a linear time (although space intensive) algorithm. However, it remains an open question how the rewriting system can be modified to permit a unique labeling in polynomial time.

REFERENCES

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Design and Analysis of Computer Algorithms*, Addison-Wesley (1972).
2. S. Arnborg, *Efficient algorithms for combinatorial problems on graphs with bounded decomposability – a survey*, BIT 25(1985), 2–33.

3. S. Arnborg, D. G. Corneil and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Alg. and Discr. Methods 8(1987), 277–287.
4. S. Arnborg, B. Courcelle, A. Proskurowski and D. Seese, *An algebraic theory of graph reduction*, submitted.
5. S. Arnborg and A. Proskurowski, *Characterization and recognition of partial* 3-*trees*, SIAM J. Alg. and Discr. Methods 7(1986), 305–314.
6. S. Arnborg and A. Proskurowski, *Linear time algorithms for NP-hard problems on graphs embedded in k-trees*, Discr. Appl. Math. 23(1989), 11–24.
7. H. L. Bodlaender, *Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees*, Proceedings of SWAT'88, Springer-Verlag LNCS 318(1988), 227–232.
8. J. A. Bondy and U. S. R. Murty, *Graph Theory with Application*, North Holland (1976).
9. K. S. Booth, *Finding a lexicographic least shift of a string*, Information Processing Letters 10(1980), 240–242.
10. C. J. Colbourn and K. S. Booth, *Linear time automorphism algorithms for trees, interval graphs, and planar graphs*, SIAM J. Computing 10(1981), 203–225.
11. B. Courcelle, *The monadic second order logic of graphs I: Recognizable sets of finite graphs*, Information and Computation 85(1990), 12–75.
12. I. S. Filotti and J. N. Mayer, *A polynomial-time algorithm for determining the isomorphism of graphs of bounded genus*, Proc. 12th ACM Symp. on Theory of Computing (1980), 236–243.
13. M. Fontet, *A linear algorithm for testing isomorphism of planar graphs*, Proc. 3rd Int. Conf. Automata, Languages, Programming, Springer-Verlag LNCS (1976), 1411–423.
14. M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman and Company, San Francisco (1979).
15. J. E. Hopcroft and J. K. Wong, *A linear time algorithm for isomorphism of planar graphs*, Proc. 6th ACM Symp. Theory of Computer Science (1974), 172–184.
16. J. Lagergren, *Efficient parallel algorithms for tree-decomposition and related problems*. Proceedings of IEEE FoCS 1990.
17. J. Lagergren, *The non-existence of reduction rules giving an embedding in a k-tree*, to appear in *Annals of Discrete Mathematics.*
18. E. M. Luks, *Isomorphism of graphs of bounded valence can be tested in polynomial time*, JCSS 25(1982), 42–65.
19. G. L. Miller, *Isomorphism testing for graphs with bounded genus*, Proc. 12th ACM Symp. on Theory of Computing (1980), 225–235.
20. G. L. Miller, *Isomorphism testing and canonical forms for k-contractible graphs*, Proc. Foundations of Computation Theory, Springer-Verlag LNCS 158 (1983), 310–327.
21. A. Proskurowski, *Recursive graphs, recursive labelings and shortest paths*, SIAM J. Computing 10(1981), 391–397.
22. A. Proskurowski, *Separating subgraphs in k-trees: cables and caterpillars*, Discrete Mathematics 49(1984), 275–285.
23. N. Robertson and P. D. Seymour, *Graph minors V, excluding a planar graph*, J. Combinatorial Theory, Ser. B, 41(1986), 92–114.
24. D. J. Rose, *On simple characterization of k-trees*, Discrete Mathematics 7(1970), 317–322.
25. J. J. Rotman, *The Theory of Groups (2nd ed.)*, Allyn and Bacon (1973).
26. P. Scheffler, *Linear time algorithms for NP-complete problems restricted to partial k-trees*, Akad. Wiss. DDR Report R-MATH-03/87 (1987).
27. Y. Shiloah, *A fast equivalence-checking algorithm for circular lists*, Information Processing Letters 8(1979), 236–238.
28. M. M. Syslo, *Linear time algorithm for coding outerplanar graphs*, in Beiträge zum Graphentheorie, Proceedings of Oberhof Conference 1977, H. Sachs (Ed.) (1978), 259–269.
29. A. Wald and C. J. Colbourn, *Steiner trees, partial 2-trees, and minimum IFI networks*, Networks 13(1983), 159–167.
30. T. V. Wimer, *Linear algorithms on k-terminal graphs*, PhD. Thesis, Clemson University (1988).