South African
Computer
Journal
Number 8
November 1992

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 8
November 1992

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal
An official publication of the Computer Society of South Africa and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Rekenaarverenging van Suid-Afrika en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Subscriptions

	Annual	Single copy
Southern Africa:	R45,00	R15,00
Elsewhere:	$45,00	$15,00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: +27 (12) 420-2504 Fax: +27 (12) 43-6454

Sigma Press

Guest Contribution

The paper below was given as an invited address by Prof Roode at the July 1992 Conference of the South African Computer Lectures' Association. (Editor)

The Ideology, Struggle and Liberation of Information Systems

Dewald Roode
Department of Informatics, University of Pretoria

In 1989, Denning *et al* presented the final report of the Task Force on the Core of Computer Science in an article entitled "Computing as a Discipline" [3]. This was said to present a new intellectual framework for the discipline of computing and proposed a new basis for computing curricula.

In the words of the authors, "an image of a technology-based discipline is projected whose fundamentals are in mathematics and engineering." Algorithms are represented as the most basic objects of concern and programming and hardware design as the primary activities. Although there is wide consensus that computer science encompasses far more than programming, the persistent emphasis on programming "arises from the long-standing belief that programming languages are excellent vehicles for gaining access to the rest of the field" [3].

The new framework sets out to present the intellectual substance of the field in a new way, and uses three paradigms to provide a context for the discipline of computing. These paradigms are *theory*, rooted in mathematics; *abstraction*, rooted in the experimental scientific method and *design*, with its roots in engineering.

Programming, the report recommends, should still be a part of the core curriculum and programming languages should be seen and used as vehicles for gaining access to important aspects of computing.

The following short definition is offered of the discipline of computing [3]:

> The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all of computing is, *"What can be (efficiently) automated?"*

In the same issue of Communications, tucked away towards the end of the journal, an article by Banville and Landry asked the innocent question "Can the Field of MIS be disciplined?" [1]. It is not clear whether the use of the word "discipline" in both articles was purely coincidental – however, the implications were quite clear: computer science was able to talk about "computing as a discipline," and indeed, could present a report which, in a sense, was a culmination of more than twenty years' efforts. Yet, its sister discipline was still asking questions of a very introvertive nature about itself.

It has become quite clear that the fields (leaving aside for the moment the questions of "disciplines") of computer science and information systems (or MIS, informatics, or whatever other name we want to attach to it) have different aims and objectives, different problems that confront it, and, yes, if we want to be truly scientific, different paradigms. To support the latter statement, it is sufficient to contrast the three paradigms of computing with the four paradigms of information systems development described by Hirschheim and Klein [5]. It can be said that a central activity in information systems is the development of information systems, and that therefore, these paradigms have implications for the field of information systems. The four paradigms can be characterized briefly, as follows:

- The analyst as systems expert
- The analyst as facilitator
- The analyst as labour partisan
- The analyst as emancipator or social therapist.

In the same spirit, Lyytinen sees the "systems development process as an instrument in organizational change" [6] and remarks that analysts' principal problems are "in understanding the goals and contents of such change instead of solving technical problems." Already in 1987 Boland [2] observed that: "designing an information system is a moral problem because it puts one party, the designer, in the position of imposing an order on the world of another."

This is clearly a far cry from Denning *et al*'s statement that the fundamental question is "what can be automated?" At the same time, within the context of the field of computing, there is nothing wrong with this question, and it is probably the right question for practitioners of computing to continually ask themselves. But it is a disastrous question for a practitioner of informatics to ask. And it has taken us quite a long time to realise this – that the two disciplines have fundamentally different roles to play. These roles are complementary and supportive, and not destructively opposed.

The liberation of information systems lies in realising this elemental truth: that information systems are man-made objects designed to effect organisational change and that, as such, they can ill be studied using the paradigms of abstraction and engineering mentioned above.

What then is needed? Banville and Landry offer the consolation that we need not concern ourselves too much about the lack of discipline, and that we can indeed even pride ourselves in being a fragmented adhocracy. It is, in fact, even healthy to continue in all sorts of directions. During this process of finding itself, a discipline should be allowed a considerable degree of latitude, and many avenues should be explored. This obviously makes the field of information systems extremely exciting: it is in the process of discovering remarkable truths, discovering that there are in reality people out there using the systems which analysts design and build, and that the most intriguing problems centre around the role of people in all of this: the analyst, the user, their interaction, the impact of sytems on the work lives of workers on all levels, the impact on organizations. These are questions which have mostly been ignored or lightly treated over the years, but which have emerged as *the* problems to be solved. We do not have the tools to solve them – not yet; but a good starting point would certainly be to first understand more about our field and its research tools, for the empirical, positivist approach so often employed will not suffice to solve the above problems.

In the spirit of contributing to the liberation movement of information systems, we have embarked on a study of research on research in Information Systems, and will report on the results more fully in the near future. We define Information Systems as follows [4]:

> Information Systems is an inter-disciplinary field of scholarly inquiry, where information, information systems and the integration thereof with the organisation is studied in order to increase the effectiveness and efficiency of the total system (of technology, people, organisation and society).

In Information Systems then, we see the fundamental question underlying the entire discipline, to be the problem of balancing the need to contribute, through information systems, to the achievement of the mission of the organisation with the moral responsibility to develop and implement socially accepted information systems.

Each of the fields, computer science and information systems, benefits enormously from the activities of the other. Nonetheless, we must recognize the different approaches used by the two disciplines and allow them to complement each other. It should not be our business to convince one another that the universal truth is that which we use in our discipline – whether that be computer science or information systems. Instead, we should seek out the opportunities for synergy, and for complementing each other. If we succeed in doing this at SACLA, then we could indeed do ourselves proud.

References

1. C Banville and M Landry. 'Can the field of MIS be disciplined?'. *Communications of the ACM*, **32**(1):48–60, (1989).
2. R J Boland and R A Hirschheim, eds. *Critical Issues in Information Systems Research*. John Wiley & Sons Ltd., 1987.
3. P J Denning, D E Comer, D Gries, M C Mulder, A Tucker, A J Turner, and P R Young. 'Computing as a discipline'. *Communications of the ACM*, **32**(1):9–23, (1989).
4. N F Du Plooy, L D Introna, and J D Roode. 'Notes on research in information systems'. Unpublished research report, Department of Informatics, University of Pretoria, (1992).
5. R Hirschheim and H K Klein. 'Four paradigms of information systems development'. *Communications of the ACM*, **32**(10):1199–1216, (1989).
6. K Lyytinen. 'New challenges of systems development: A vision of the 90's'. *Data Base*, pp. 1–12, (Fall 1989).

Editor's Notes: To Compete or Collaborate

Human interaction invariably brings with it a blend of competition and collaboration. Competition means that one enjoys the exhilaration of winning while the other endures the shame of loosing. Because of this reward/punishment mechanism, it is a widely assumed that competition enhances performance and efficiency. This dogma pervades not only commerce, sport and politics, but is found in practically all areas of human endeavour, including research.

The competitive spirit in research is found in the well-known saga of Watson and Crick racing to unravel the double helix structure of DNA. Not so well-known, though equally illustrative, is the intensity of Newton's stratagems to oust Leibnitz from receiving any credit for differentiation. Recently there have been reports of scientists who have either tolerated or manufactured fraudulent results in order to win some or other scientific race. The space race, the arms race, the race for an AIDS cure, the scurry for faster smaller hardware, the race for awards, the drive for publications, Nobel prizes: all of this attests to a profoundly competitive international research culture.

But while competition might be the handmaiden of commerce and sport, it is the harlot of research – an unfortunate concomitant of the silly side of human nature. The archetypal researcher not only rises above the incidentals of human accolades; he disdains them. By tradition, the definitive research qualification is a PhD – a Doctor of Philosophy – a lover of thought. Discovery and thought are not only by their very nature rewarding, they are also humbling. When the archetypal researcher moves outside his interior thought-world, it is to share his discoveries. If he is childish, it is not the little boy flexing his biceps and saying: "I'm stronger than you" but the child rushing to

tell everyone: "Wow – look at this!" He is forgetful of self: Pythagoras, oblivious of the invading enemy and his impending death while he researches in the sand; Archimedes shouting "Eureka" without care for his nudity. The competitive spirit is a crass intrusion into this ancient legacy of innocence and selflessness.

By its nature, collaboration thrives in a climate of easy social intercourse. It may initially feel uncomfortable for researchers, who are inclined to be socially inept and are wont to bury themselves in work away from society. However, once the plunge to collaborate is taken there is ample evidence that it leads to successful research. In maximizing the use of available talent, it brings about a synergy in which two heads are better than one. All participants enjoy its rewards and no individual has to endure the full weight of its failures. In fact, the notion of collaboration is now so commonplace that significant research seems impossible without it. The tendency, however, is to encourage research collaboration within an organisation, but to emphasize competition in relation to outside organisations.

During a forum discussion at the July South African Computer Lecturers' Association (SACLA) conference, an appeal was made for greater collaboration between universities. Not surprisingly, the information technology disciplines at local universities have always had both a competitive and a collaborative relationship. The competitiveness usually takes the form of friendly rivalry, while the very existence of SACLA bears testimony to a rather unique collaborative relationship. In latter years the competitiveness seems to have intensified, while electronic mail and other developments have improved the prospects for collaboration. At issue, then, is whether there is an imbalance between these dual forces. The appeal at the SACLA forum implied that there is, and I would strongly agree. It is my view (my prejudice, if you will) that competition between universities is a self-indulgent and wasteful dissipation of energy.

Those who are inclined to compete should seriously examine what is to be gained. It is unconvincing to argue that winning makes a significant impact on the way in which students select universities: in the main, this is a matter of geography and language preference. To some extent, the same might be said about staff, although research reputation perhaps plays a more important role here. Neither are research funding agencies (e.g. the FRD) influenced by whether X is "better" in some or other sense than Y. On the contrary, it has wisely been decided to fund on the basis of criteria that are believed to be objective, without any reference whatsoever to the performance of competitors. True enough, funds are limited, but it is precisely for this reason that it is wasteful to divide the little there is between divergent research efforts.

It seems to me that there is a wealth of research talent out there, but that each researcher selects an area of interest almost as a matter of whim. There is an urgent need for well-coordinated collaboration on focussed research areas that have been carefully selected as directly relevant to the country. It is especially incumbent on those who finance, manage and lead research to identify such areas and to encourage collaboration in every possible way.

I look forward to the manifestation of such collaboration in SACJ publications authored by researchers from different university departments. To date there have been none of consequence. If we fail to collaborate, we are in danger of becoming little Don Quixotes who spend our lives attacking windmills and defending castles of xenophobia and irrelevance.

A New Algorithm for Finding an Upper Bound of the Genus of a Graph

D I Carson and O R Oellermann
Department of Computer Science, University of Natal (Durban), King George V Avenue, Durban 4001

Abstract

In this paper we discuss the problem of finding an upper bound on the genus of a graph. This problem has applications to circuit layouts. An electronic circuit may be modelled by a graph. By punching holes into the circuit board, one may be able to lay out the circuit so that no two wires cross. The smallest number of holes that are required for a given graph (that models such a circuit) is called the genus of the graph. The number of holes in the surface equals the genus of the surface. Thus, finding an algorithm which approximates the genus of the graph which models the circuit is important. We present a new algorithm for finding an upper bound on the genus of a graph, which uses a combinatorial data structure called the PQ-tree data structure. Four additional PQ-tree templates are used to extend the basic combinatorial reduction process of the PQ-trees to consider a genus approximation algorithm.
Keywords: *PQ-trees, genus, circuit layouts*
Computing Review Categories: *G.2.2, E.1*

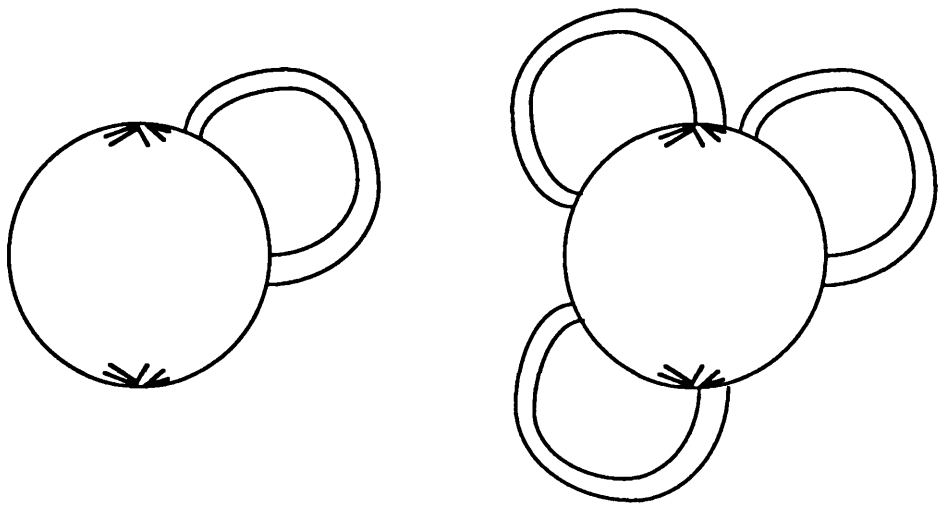
Received May 1991, Accepted July 1992

1 Introduction

The genus of the sphere is 0. Any surface obtained from the sphere by inserting $n \geq 0$ handles on the sphere is called a *surface of genus n*. For example, Figure 1(a) shows a surface of genus 1, while Figure 1(b) shows a surface of genus 3.

A graph is *2-connected* if the removal of fewer than two vertices does not disconnect the graph. A *block* of a graph is a maximal 2-connected subgraph. The *genus* $\gamma(G)$ of a 2-connected graph G is the minimum among the genera of the surfaces on which G may be embedded (i.e. drawn so that no two of its edges cross). The genus of a 2-connected graph is always defined (see [2]). In this paper we develop an efficient algorithm for finding an upper bound on the genus of a graph.

We follow the notation of [2]. If a graph G has $|E(G)| = q$, and $|V(G)| = p$, then we say G is a (p, q) graph. The genus of a graph equals the sum of the genera of its blocks (Battle, Harary, Kodama and Youngs [1]). Therefore, it suffices to develop algorithms for finding the genus of a 2-connected graph. Determining the genus of an arbitrary 2-connected graph efficiently is believed to be a difficult problem. In 1979 Filotti, Miller and Reif [6] developed, for a fixed k, an $O(p^{O(\gamma(G))})$ algorithm for determining if $\gamma(G) \leq k$. In 1989 Thomassen [10] showed that the following decision problem is NP-complete : For a given graph G and integer $k \geq 0$, is $\gamma(G) \leq k$? Thus it is reasonable to



(a) (b)

Figure 1 . Genera of Surfaces.

consider heuristics that approximate the genus of a 2-connected graph. In this paper we present an algorithm for finding an upper bound on the genus of a 2-connected graph.

2 The PQ-tree Data Structure

The basis for the algorithm relies on the PQ-tree data structure. This data structure was developed by Booth and Lueker [3], and allows for the efficient implementation of combinatorial problems, in particular those relating to graph theory. The most important result using PQ-trees is a linear time (in the number of vertices) modification by Booth and Lueker [3] of an algorithm for testing whether a given 2-connected graph is planar [9]. For further applications of the PQ-tree data structure see [4]. We present here an introduction to PQ-trees. This introduction to PQ-trees is of necessity short and for further reference please refer to [3] or [4].

The first operation we perform is an st-numbering of the vertices of G, the details of which can be found in [9] and [4]. It is known that every 2-connected graph has an st-numbering [9]. Let H be a subgraph of a graph G such that the highest st-number of a vertex of H is k, and $|V(H)| = k$. The st-numbering provides an ordering of the vertices of a G, such that, if G is a plane graph, then **all edges from vertices of V(H) to vertices of V(G) - V(H) lie in one face of H** [9].

Let $\mathcal{U} = \{a_1, a_2, ..., a_m\}$ be a universal set. Then the class of *PQ-trees* over the set $\mathcal{U}$ is defined to be the set of all rooted trees whose leaves are elements of $\mathcal{U}$ and whose internal vertices are either *P-nodes* or *Q-nodes* where a P-node is a node whose children may be permuted freely amongst themselves and a Q-node is a node whose children remain in a fixed order, although the order in which they appear in any planar realisation of the PQ-tree may be reversed. The properties of the P-nodes and Q-nodes are shown in the way we draw them in the PQ-tree. For a P-node, we use a circle which represents freedom of order, whereas a Q-node is represented by a rectangle which portrays the fact that the order amongst the children is fixed. As for the drawing of normal trees, we draw the children of a node below the node in question (thus the root is at the top of the diagram). When it is not important whether a node is a P-node, Q-node or a leaf, then we shall represent it by a triangle.

Figure 2 shows an example of a PQ-tree. From now on we shall consider only *proper* PQ-trees; that is PQ-trees in which each element a_i appears exactly once as a leaf, every P-node has at least two children and every Q-node has at least three children. Consider a PQ-tree T. We define the *frontier* of T to be the leaves of T as read from left to right in the embedding. An *equivalence transformation* on a PQ-tree node either :

(i) Arbitrarily permutes the children of a P-node; or

(ii) Reverses the children of a Q-node.

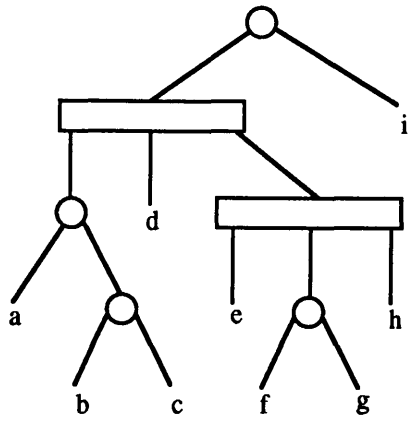
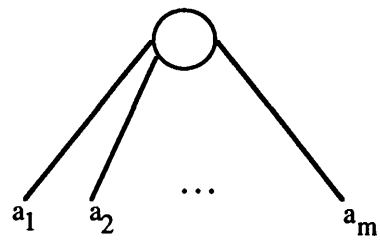


Figure 2 . An example PQ-tree.

We say that two PQ-trees T_1 and T_2 are *equivalent* if and only if the T_1 can be transformed via a series of zero or more equivalence transformations to T_2. If two PQ-trees are equivalent, then we write $T_1 \equiv T_2$. We note that this is an equivalence relation. For example, for the PQ-tree T in Figure 2, there are 64 trees in the equivalence class of T. Every two PQ-trees in the same equivalence class have different frontiers.

It is important to note that a PQ-tree T of a universal set $\mathcal{U}$ is a description of the "allowable" permutations of $\mathcal{U}$. The frontier of every PQ-tree $T' \equiv T$ represents a valid permutation. Given a universal set $\mathcal{U}$, there is a spectrum of possible PQ-trees. At the ends of the scale, we have the *null tree* and the *universal tree*. The null tree is the empty PQ-tree, which is a PQ-tree without any nodes or leaves. The null tree represents the most restricted PQ-tree - one in which we do not know anything about the allowable structure of the universal set $\mathcal{U}$. The set of valid permutations of the null tree is empty. The universal tree is the most unrestricted PQ-tree - a single P-node with children a_i for all $a_i \in \mathcal{U}$. Thus the universal tree represents every possible permutation on the universal set $\mathcal{U}$. These two trees are shown in Figure 3.



Null tree Universal Tree

Figure 3. The Null tree and Universal tree.

Turning to Q-nodes, there are always two *endmost* children. These are the children which are always at the ends of the sequence of children of the Q-node (irrespective of reversal). The rest of the children are

interior. Furthermore, we say that an *immediate sibling* of a Q-node's child is a child of the Q-node which appears adjacent to that node in every PQ-tree T' such that T' ≅ T. Conceptually, the immediate siblings of a node are the neighbours of that node in $\hat{T}$, the embedding of T. Thus, every interior node has exactly two immediate siblings and every endmost child has exactly one immediate sibling. Figure 4 illustrates these terms.

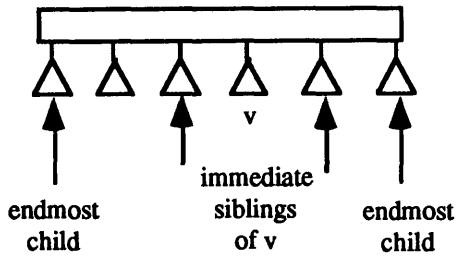


Figure 4. Children of a Q-node.

We now discuss the general application of PQ-trees. Let $\mathcal{S}$ be a class of subsets of a universal set $\mathcal{U}$ (note that the elements of $\mathcal{S}$ need not be disjoint). Consider the following problem $\mathcal{P}$: Determine all permutations π on $\mathcal{U}$ so that for every S ∈ $\mathcal{S}$, the elements of S appear consecutively in π. Algorithm 1 gives a formal description of a method which solves $\mathcal{P}$.

Algorithm 1: Reduction ($\mathcal{U}$, $\mathcal{S}$)
{ find all permutations π of $\mathcal{U}$ such that, for every set S ∈ $\mathcal{S}$, all elements of S appear consecutively in π }

```
Π = {π | π is a possible permutation of U }
For every S ∈ S do
    Π = Π ∩ { π | all objects of S are
consecutive within π}
        [ the reduction phase ]
end
```

We now describe how the PQ-tree data structure can be used to implement this reduction efficiently. We begin with a universal tree T whose leaves are the elements of $\mathcal{U}$. Note that at this stage the trees equivalent to T represent all permutations of the elements of $\mathcal{U}$. The inner loop of Algorithm 1 is actually a modification procedure which adjusts T to reflect the new constraint, namely that for each S ∈ $\mathcal{S}$ the elements of S appear consecutively in the frontier of T. Effectively we are *reducing* the number of trees equivalent to T (i.e. the set Π). We say that T is *S-reduced* if, for every T' ≅ T, the elements of S appear consecutively in Frontier(T'). We only need a single operation on T, denoted by Reduce (T,S), which modifies T so that it becomes S-reduced. Informally, the process of performing such a reduction consists of scanning the PQ-tree node by node, starting from the leaves, and then looking for *patterns* in the structure of the subtree at each node, and structurally modifying the subtree rooted at the node with a *replacement*. We call each pair of pattern and replacement at a node a *Template*. A formal description of Algorithm 2 is now given.

Algorithm 2: Reduce (T, S)
{ Constrain the PQ-tree T so that S appears consecutively in Frontier(T) }

```
QUEUE = empty
Finished = false
    { Queue contains nodes which can be
matched and then replaced }
for each leaf ∈ U do
    Add_to_Queue (leaf)
while not Finished do
        { not all elements of S consecutive }
    Current = Head of Queue
    Try to match Templates to Current
    if a Template matches
        then
            Replace subtree rooted at Current
            with replacement pattern
        else{ error - no match }
            T = Null Tree
            Halt
    if S ⊆ {Leaf| Leaf is a leaf of the
     subtree of T rooted at Current }
        then Finished = true
            { Arranged all leaves of S }
            { consecutively in Frontier(T) }
        else
            if parent of Current_Node has all
                its children queued
            then Add parent of Current to
                                    Queue
end
```

Notice that the changes are local, we only modify the node and its children. Also, the pattern which is matched depends only on the node and its children. We match the children of a node before we match the node itself.

There are three possible states a child of a node can be in. Consider any child X. If none of the descendants of X which are leaves are in S, we say that X is *empty*. If all of the descendants of X which are leaves are in S, we say X is *full*, and lastly if some of the descendants of X that are leaves are in S, but some are not, then we say X is *partial*. A node is said to be *pertinent* if some or all of its children are either full or partial with respect to S. The *pertinent subtree* of T with respect to S, denoted Pertinent(T,S), is the unique subtree, rooted at a vertex X, of T of minimum height whose frontier entirely contains S. The root of the pertinent subtree is denoted by Root(T,S). For example, in Figure 5 we show the

pertinent subtree of the PQ-tree given in Figure 2 when S = {a, c, e, f}. Intuitively we should only have to perform equivalence transformations on Pertinent(T,S).

After matching each template and performing the appropriate replacement, there is no information loss. In other words, the PQ-tree Reduce(T,S) represents exactly every possible valid permutation of the frontier of T which has all elements of S appearing consecutively (see [3] and [4] for further details).

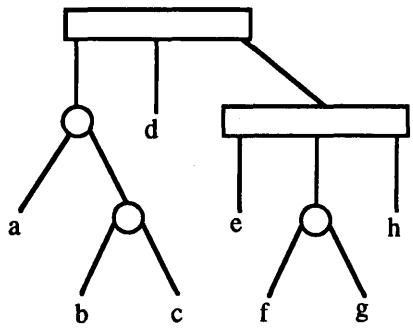


Figure 5. Pertinent(T,S) when S = {a, c, e, f}.

The following convention is adopted. Full nodes are shaded and partial nodes have only the right-hand side of the node shaded. Empty nodes are left unshaded. Similarly, whenever a child's node type does not matter, we shall represent it by a shaded or unshaded triangle. Furthermore, we always arrange the children of a partial node X in T so the pertinent descendants which are leaves of the maximal subtree rooted at X appear consecutively in Frontier(T). For brevity we omit the template descriptions here (see [3] and [4] for further details).

Let us consider an example of the template matching process as applied to a graph planarity testing algorithm. Consider the (non-planar) st-numbered graph K_5 given in Figure 6.

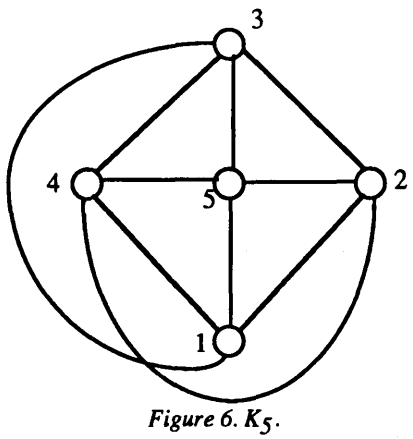


Figure 6. K_5.

Let H be a subgraph of a graph G such that the highest st-number of a vertex of H is k, and |V(H)| = k. Recall that the st-numbering provides an ordering of the vertices of a graph G, such that, if G is planar, then in an embedding of G in the plane all edges from vertices of V(H) to vertices of V(G) - V(H) lie in one face of H. We start with the Universal Tree T representing all edges directed out of vertex 1 and with H being the single vertex labelled 1. In general when we add the vertex with st-number k ($2 \leq k < p$) to H, we attempt to modify T to construct a new tree T' so that all leaves representing edges directed into vertex k are arranged consecutively in the frontier of every tree $T'' \cong T'$. If such a T' exists, then in H we identify in a vertex labelled k, all the edges directed into vertex k. Further, in T' we replace Pertinent (T, S) by a P-node whose children are leaves representing all the edges directed out of vertex k.

Suppose we are reducing for a vertex k. Since the PQ-tree represents all valid permutations of the edges from H to V(G) - V(H), if we are unable to find an equivalent PQ-tree T which has all edges directed into vertex k arranged consecutively in Frontier(T), then this implies that the edges directed into vertex k cannot be arranged consecutively in any embedding of V(H) ∪ {k}. Thus we may conclude that G is non-planar.

The initial PQ-tree T is shown in Figure 7. Note how T represents all the edges incident with vertex 1, yet does not restrict them into any particular combination in a planar realisation of G.

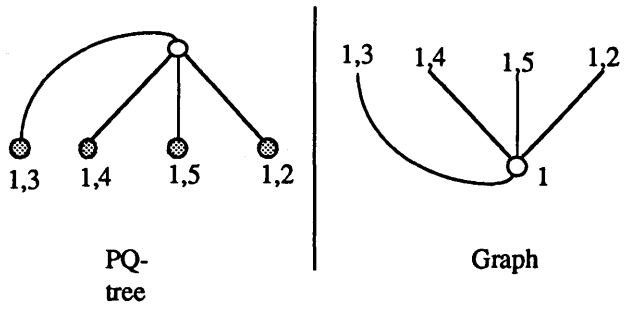


Figure 7. Initial PQ-tree T.

Now, we gather all leaves representing edges directed into vertex v_2. In this case there is only one such leaf. Then we replace the leaf (v_1,v_2) with a P-node having as children leaves representing the edges directed out of vertex v_2, namely the leaves (v_2,v_3), (v_2,v_4) and (v_2,v_5). See Figure 8.

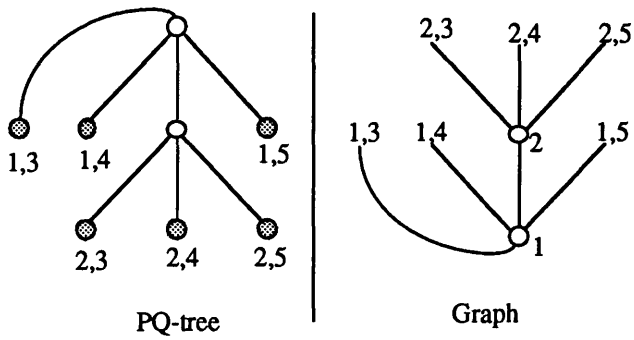
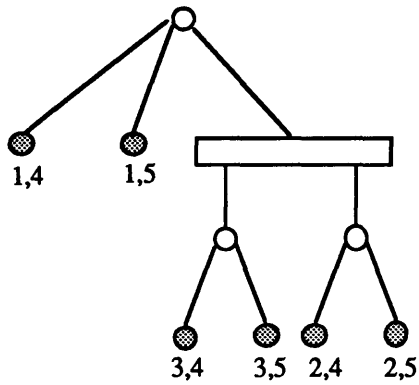


Figure 8. After reduction is complete for vertex v_2.

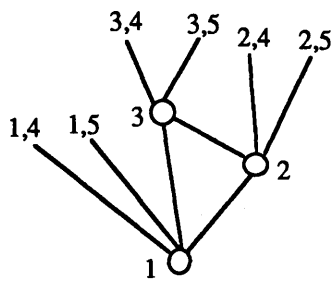
Now we consider the set $S = \{(v_1,v_3), (v_2,v_3)\}$. We match a number of templates to complete the reduction

for vertex v_3 (see [4] for further details). The resulting PQ-tree is shown in Figure 9. Again observe how the PQ-tree represents exactly every possible valid permutation. Observe that the subtree rooted at the Q-node in Figure 9 represents the 2-connected subgraph induced by the vertices 1, 2 and 3.

Take the next set, namely $S = \{(v_1,v_4), (v_2,v_4), (v_3,v_4)\}$. After matching a number of templates the reduction process fails and we conclude that K_5 is non-planar.



PQ-tree



Graph

Figure 9. After reduction is complete for vertex v_3.

To see that the reduction process must fail is easy. Consider Figure 10 which shows the PQ-tree T and what the corresponding sub-graph would have looked like had we applied as many valid templates as possible to T. We observe that, in order to group (v_1,v_4) with the other pertinent edges, we would have to cross other edges.

The algorithm we present in this paper exploits the flexibilty of the PQ-tree data structure. When it is not possible to add the next vertex to H without allowing edges to cross, then we may conclude that the graph is non-planar, and consider our options. It is at this stage that we propose to insert handles into the corresponding embedding of H, and to allow the algorithm to continue.

3 The Basis for the Algorithm

Suppose G is an arbitrary non-planar 2-connected graph. Using the PQ-tree data structure we obtain, at each stage of the reduction algorithm, a description of the partial embedding of G. Figure 11 shows an st-numbered non-planar graph G. That G is non-planar is easy to see since a subgraph of G isomorphic to the non-planar Kuratowski graph $K_{3,3}$ is induced by the vertex set {1, 3, 4, 5, 6, 7}, with {1, 4, 6} and {3, 5, 7} being the two partite sets.

G:

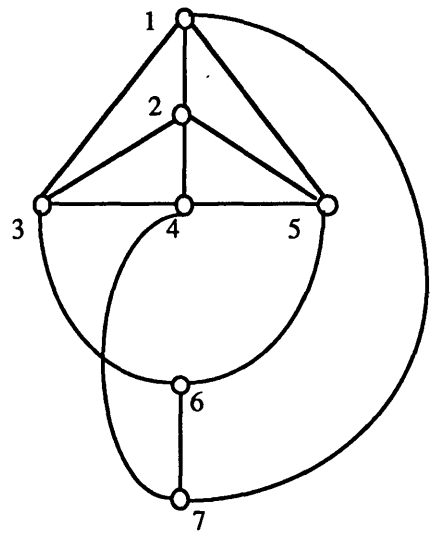
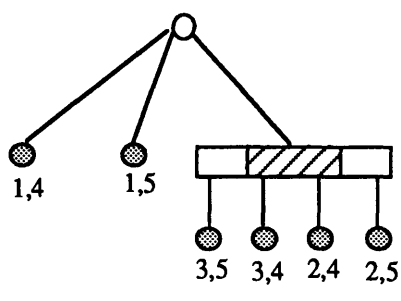
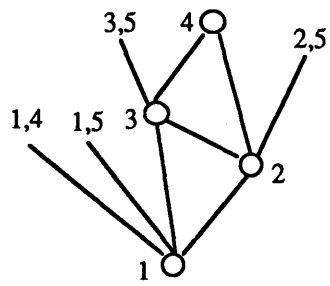


Figure 11. An st-numbered non-planar graph G.

Now, during the reduction algorithm, the reduction



PQ-tree



Graph

Figure 10. PQ-tree after the reduction fails.

for vertex 6 fails, because both the pertinent leaves are children of a Q-node X, and they have an empty child (the leaf representing the edge from vertex 4 to vertex 7) between them, in the order as they appear as children of the Q-node. However, if we insert a handle with ends in regions R_1 and R_2, as shown in Figure 12(a), then we obtain an embedding of G on the torus, as shown in Figure 12(b). The ideas of embedding the graph G of Figure 11 on the torus, as shown in Figure 12(b), illustrate the key ideas to our algorithm.

We now generalise the above idea to any non-planar graph. Observe that all we did in the above example was, given a template matching failure, to insert a handle from the one consecutive sequence of pertinent edges to a final region on whose boundary vertex 6 lies. This technique of inserting a handle during the reduction process is the main idea behind our algorithm.

4 The Algorithm

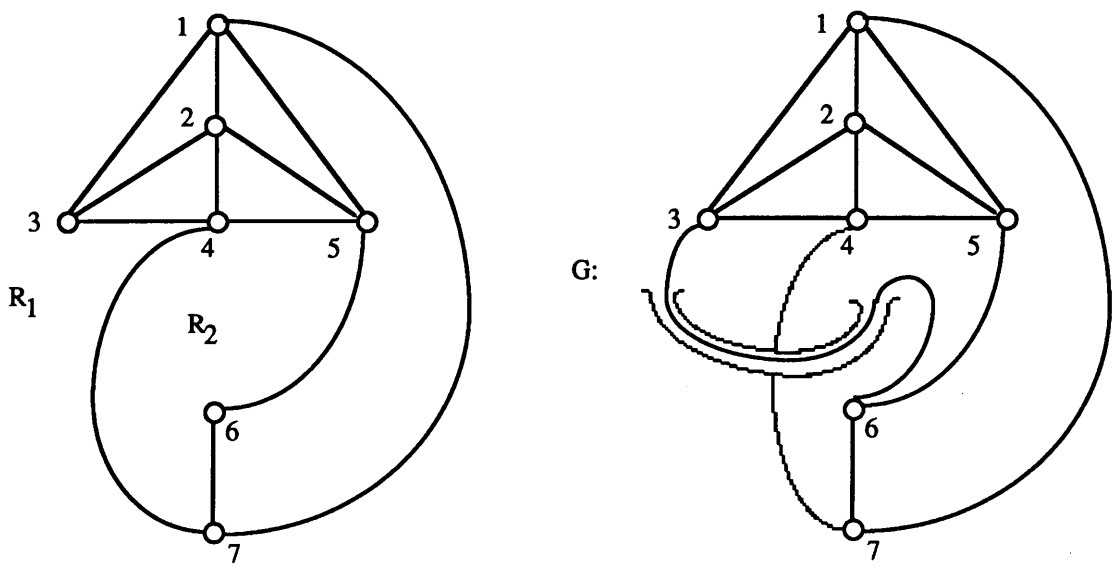
Suppose we are reducing with respect to some vertex having st-number i, and suppose that, during the reduction, a node X in our PQ-tree T does not match one of the valid reduction templates. Our goal is to delete maximal subsequences of full leaves from Frontier(X) to allow the reduction process to proceed. At the end of the reduction we only have one maximal subsequence of full leaves in Frontier(T). Given a maximal subsequence of full leaves to delete, we identify the edges represented by the full leaves in a *pseudo-vertex*. The single remaining subsequence of full leaves in Frontier(T) after the reduction is complete is identified in a vertex called the *base*. We place a handle from each pseudo-vertex to the base. Finally we undo the edges identified at each pseudo-vertex and place the corresponding identified edges along that handle at the pseudo-vertex so that they are now identified at the base vertex.

For example, when reducing with respect to vertex 6 in Figure 12 none of the templates match and thus a pseudo-vertex representing the edge between vertex 3 and vertex 6 is placed next to vertex 3 and joined to this vertex by an edge. This new edge is not represented in the PQ-tree. The reduction can now proceed. After the base for vertex 6 is identified we replace the pseudo-vertex adjacent with vertex 3 by a handle to the base and we replace the edge from vertex 3 to the pseudo-vertex with an edge between 3 and 6 that proceeds along the handle.

Let T' be the PQ-tree equivalent to the maximal subtree rooted at a Q-node X such that in Frontier(X) there are as few maximal subsequences of full leaves as possible. Suppose that there are k maximal subsequences of full leaves in Frontier(T'). We define $Pert_i(X)$ to be the sequence of pertinent children of X whose full leaf descendants are precisely the i-th maximal sequence of full leaves in Frontier(T') as we proceed from left to right in Frontier(T').

We define an operation Reduce($Pert_i(X)$) to simplify $Pert_i(X)$ in the following manner. If there are no partial children in $Pert_i(X)$, then do nothing. Otherwise, for each partial child Y, merge the children of Y into the sibling lists of X so that the full or partial immediate sibling of Y in $Pert_i(X)$ is now an immediate sibling of the full endmost child of Y. Thus, the frontier of $Pert_i(X)$ still contains the same number of full leaves. Figure 13 illustrates the Reduce($Pert_i(X)$) operation.

The basic templates introduced by Booth and Lueker were extended in [8] to include the case when the template matching process fails; a complete description was given using four additional templates describing when the template matching process fails. These



(a) (b)

Figure 12 . An embedding of a graph G on the torus after inserting a handle on the sphere.

templates were used in [8] to discover subdivisions of $K_{3,3}$ and K_5 in the input graph. We follow the description of [4] and describe the four templates given in [8].

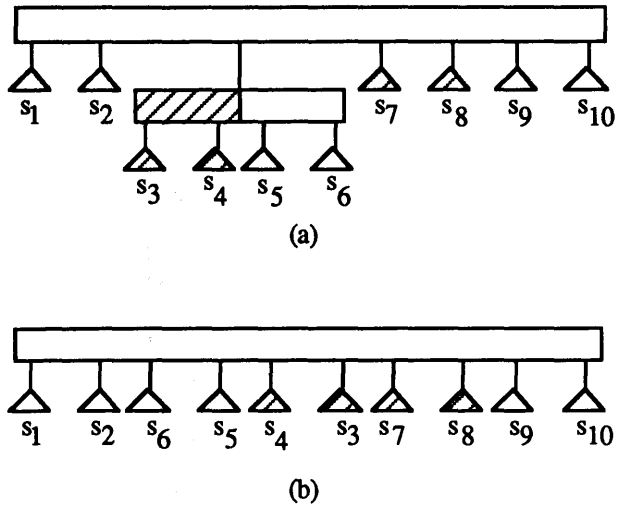


Figure 13. The Reduce(Pert$_i$(X)) operation; (a) before Reduce(Pert$_i$(X)); (b) after Reduce(Pert$_i$(X)).

Let X be a Q-node of T. Now, if in every PQ-tree T' equivalent to the maximal PQ-subtree rooted at X, the number of maximal sequences of consecutive pertinent leaves in Frontier(T') is at least 2, then Template N_1 is matched. One of the situations that satisfies Template N_1 is shown in Figure 14.

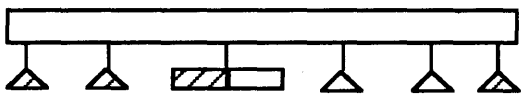


Figure 14 . Template N_1.

Let X be a Q-node of T, where X is not the pertinent root. We say that X matches Template N_2 if X does not match Template N_1, and every PQ-tree T', equivalent to the maximal subtree rooted at X, in which the pertinent leaves of Frontier(T') appear consecutively, both the leftmost and rightmost elements of Frontier(T') are empty. One of the situations that satisfies Template N_2 is shown in Figure 15.

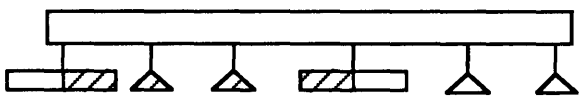


Figure 15 . Template N2.

The templates which cater for non-planar cases when the node is a P-node are much easier. For Template N_3 to match we must have a P-node which is the pertinent root, with three or more partial children. Template N_3 is shown in Figure 16. For clarity we omit the rest of the partial children (if any), and the full and empty children (zero or more may be present) from Figure 16.

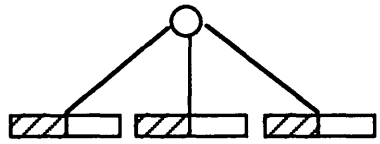


Figure 16. Template N_3.

Template N_4 is similar to Template N_3, except that, in this case, the P-node in question may not be the pertinent root, and it must have at least two partial children. Figure 17 illustrates this template, where, again, we have omitted the rest of the partial children, and the full and empty children from the figure.

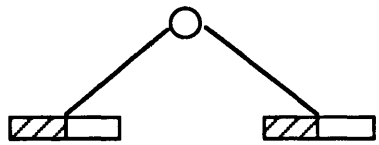


Figure 17 . Template N_4.

We show in [4] that these templates are sufficient to describe all the cases for when the normal PQ-tree reduction process fails. Using the Templates N_1, N_2, N_3 and N_4 we obtain the following algorithm, Algorithm Place_Handles_Non_Planar_Graph(X) which deals with the non-planar situation. In Algorithm 2 when the reduction process fails at a node Current instead of assigning T to be the null tree and halting, we call Algorithm 3, match Current to a valid template and then allow the reduction to proceed.

Algorithm 3: Place_Handles_Non_Planar_Graph(X)
{ The template matching in PQ-tree T has failed
at a node X, we place handles to
allow the reduction to proceed }

If Template_N$_i$(X) $\{i \in \{1,2,3,4\}\}$
then Place_Handle_N$_i$(X)
end

We denote by $\gamma_g(G)$, the number of handles which we place to embed G. Before we begin the reduction for vertex 2, we assume that we are to embed G on the sphere, i.e. $\gamma_g(G) = 0$.

We consider each of the cases in turn. Note that the algorithms may place a number of handles. Suppose we are reducing for vertex i. At first we shall only place the pseudo-vertices. The base can only be determined at the end of the reduction for vertex i. Suppose we have determined that the full leaves from a sequence Pert$_j$(X) of pertinent children of X must be deleted. When we place a pseudo-vertex for some of the sequences Pert$_j$(X), we shall mark the pertinent leaves which are in Pert$_j$(X) as *void*. This marking process removes all references to the sequence of pertinent leaves from the Q-node X. This will then allow the reduction algorithm to continue the template matching process.

For the rest of this section, in the diagrams used to illustrate the different cases we encounter, we will use black circles to denote the ends of the handles which we insert, and hence the pseudo-vertex which we place.

Template N_1 is matched for a Q-node X, when we have at least two maximal subsequences of pertinent children in any Frontier(T'), where T' is equivalent to the maximal subtree of T rooted at X. The first operation we perform is Reduce($Pert_i(X)$) for every $Pert_i(X)$. Thus, our maximal subsequences of pertinent children are now only full children. We have two subcases, depending on whether X is the pertinent root or not.

Case 1: Suppose that X is the pertinent root. Then, we select some subsequence $Pert_i(X)$. At the end of the reduction we will identify the edges represented by the pertinent leaves in the frontier of $Pert_i(X)$ in the base. For all other subsequences $Pert_j(X)$, we identify the pertinent leaves to a pseudo-vertex, and mark these leaves void.

Case 2: Suppose that X is not the pertinent root. In this case, if a full child is endmost, we select a sequence $Pert_i(X)$ containing that child. For all other sequences $Pert_j(X)$, we identify the pertinent leaves to a pseudo-vertex, and mark these full leaves void. We do not identify the sequence $Pert_i(X)$ to a pseudo-vertex, because the reduction is now able to proceed without the placement of a further handle. In this way we attempt to minimise the number of handles placed.

Algorithm 4 gives the full algorithm.

Algorithm 4: Place_Handle_N_1(X)
{ Place Handle(s) to enable reduction to continue after reduction failed and X matched Template N_1 }

```
{ Q-node, two or more isolated full
  sequences of children see Figure 18(a) }

Let the maximal sequences of pertinent
children be
   Pert1(X), Pert2(X), ..., Pertk(X);
   where k ≥ 2
For all sequences Perti(X) do
   Reduce(Perti(X))
If X is not the pertinent root
   then
      If any full child Y of X is endmost
         then
            let Perti(X) be the sequence to
            which that full child belongs
         else
            Perti(X) = ∅
   else          { X is the pertinent root }
      Select any sequence Perti(X)

      { Perti(X) now contains a sequence of
        full children from which we
        do not want to place a handle }
for all sequences Pertj(X) ≠ Perti(X) do
   Identify Pertj(X) in a pseudo-vertex
   γg(G) = γg(G) + 1
   Mark Pertj(X) void  { delete Pertj(X) }
      { see Figure 18(b) }
{ Now the reduction may proceed }
end
```

In Figure 18, we show a possible situation for some 2-connected non-planar graph G when we are reducing for vertex 17. The reduction fails because the edges going to vertex 17 cannot be identified without crossing either the edge to vertex 18 or the edge to vertex 19. (Recall that the vertices of higher st-number must remain on the exterior.)

We may now consider the next algorithm, Algorithm Place_Handle_N_2(X), which caters for the case when the reduction process fails, Template N_1 does not match, and

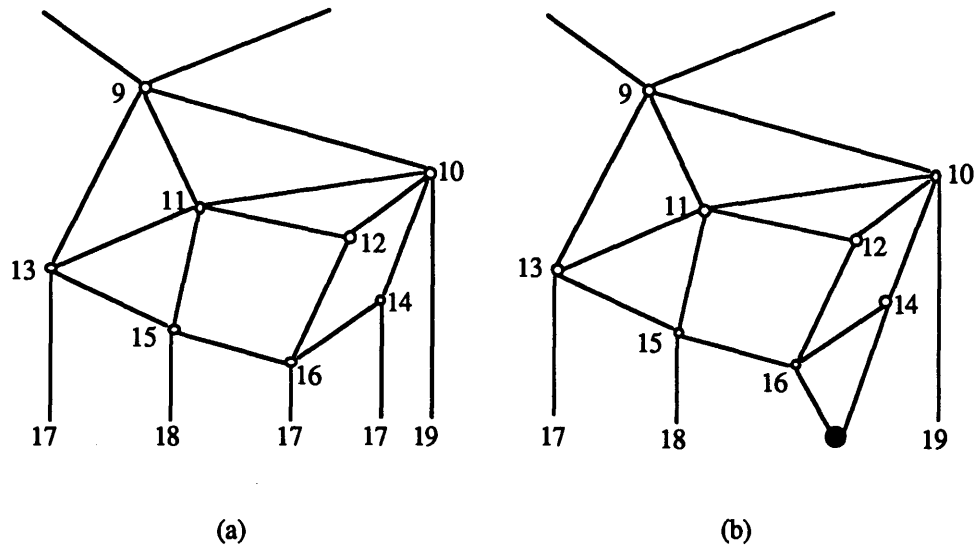


Figure 18. Insertion of Handles when Template N_1 is matched
(a) Before reduction for vertex 17; (b) after placement of pseudo-vertex.

Template N_2 is matched. Since we have all pertinent children appearing in one maximal sequence $Pert_i(X)$, it is sufficient to place a single handle. Again, we perform Reduce($Pert_i(X)$). Then, we identify all the pertinent leaves to a single pseudo-vertex. Algorithm 5 gives the full algorithm.

Algorithm 5: Place_Handle_N_2(X)
{ Place Handle(s) to enable reduction to continue after reduction failed, Template N_1 does not match, and X matched Template N_2}

{ Q-node, not pertinent root, no full child endmost - see Figure 19 }
Let the maximal sequence of pertinent children be $Pert_i(X)$
Reduce($Pert_i(X)$)
Identify $Pert_i(X)$ in a pseudo-vertex
$\gamma_g(G) = \gamma_g(G) + 1$
Mark $Pert_i(X)$ as void
{ see Figure 19(b) }
{ Now the reduction may proceed }
end

Figure 19 shows Template N_2. Once again, the psuedo-vertex is represented by a black circle, and we are reducing for vertex 17.

The rest of the algorithms deal with the Templates N_3 and N_4, which apply when X is a P-node. Consider Algorithm Place_Handle_N_3. Suppose a P-node X satisfies Template N_3, so X must be the pertinent root. Further, suppose that there are k (k > 2) partial children of X. Let P_ℓ and P_m be any two partial children of X. We place the full children of X between P_ℓ and P_m. We then merge P_ℓ and P_m to create a new Q-node Y with both endmost children empty, and all full children appearing in one consecutive subsequence $Pert_i(Y)$ of the children of Y. We then pair off the other partial children, except for possibly one, and merge each pair, thereby reducing the number of subsequences of full children to $\left\lceil \frac{k}{2} \right\rceil$. For every partial child W and corresponding maximal subsequence $Pert_j(W) \neq Pert_i(Y)$, we identify the pertinent leaves of Frontier(W) in a pseudo-vertex, and mark all full children void. At the end of the reduction, $Pert_i(Y)$ is identified as our base.

Algorithm 6 gives Algorithm Place_Handle_N_3.

Algorithm 6: Place_Handle_N_3(X)
{ Place Handle(s) to enable reduction to complete after reduction failed and X matched Template N_3}

{ P-node, pertinent root, three or more partial children - see Figure 20 }
Let the partial children be $P_1, P_2, ..., P_k$; where k > 2
Denote the full and empty endmost children of any P_i by
Full(P_i) and Empty(P_i)
Let $\mathcal{P}$ be a partition of the partial children into $\left\lceil \frac{k}{2} \right\rceil$ sets, so that each set, except for possibly one set (if k is odd), has two elements
Select one of these sets with two partial children P_ℓ and P_m
Remove any full children of X and place them
as children of a new P-node Y
Join P_ℓ and P_m together as follows
If Y has children
then
Add Y as an immediate sibling of Full(P_ℓ) and Full(P_m)

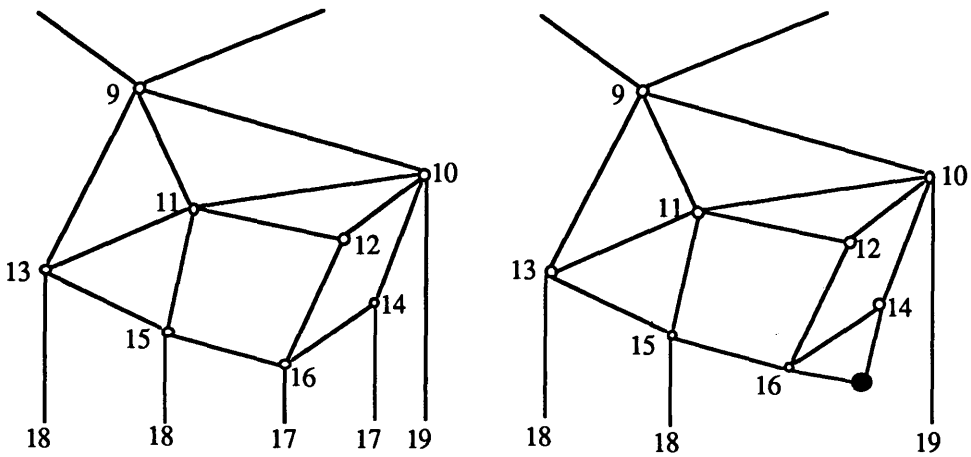
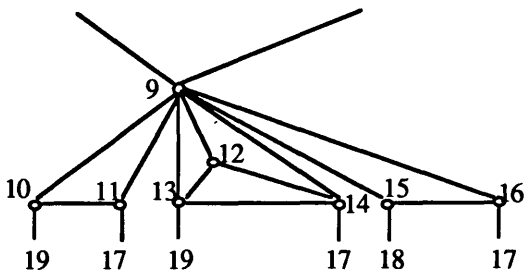


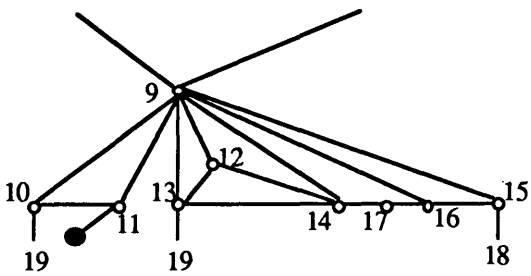
Figure 19 . Insertion of Handles when Template N_2 is matched
(a) Before reduction for vertex 17; (b) after placement of pseudo-vertex.

Let new endmost child of P_ℓ be
Empty(P_m)
else
Let Full(P_ℓ) and Full(P_m) be
immediate siblings
Let new endmost child of P_ℓ be
Empty(P_m)
Denote full sequence of children of P_ℓ by
Main_Group
For every element of $\mathcal{P}$ that contains a pair
P_i and P_j ($i, j \neq \ell, m$) do
Let Full(P_i) and Full(P_j) be immediate
siblings
Let new endmost child of P_i be
Empty(P_j)
Delete P_j
Identify full children of P_i in a pseudo-
vertex
$\gamma_g(G) = \gamma_g(G) + 1$
Mark full children of P_i as void
if k is odd
then
Identify P_k in a pseudo-vertex
$\gamma_g(G) = \gamma_g(G) + 1$
Mark full children of P_k as void
{ Now the reduction may proceed }
end

Figure 20 shows a subgraph of a non-planar graph G, when we are reducing for vertex 17.



(a)



(b)

Figure 20 . Illustration of Handles when Template N_3 is matched
(a) Before reduction for vertex 17; (b) after placement of pseudo-vertex.

When Template N_4 is matched, the situation is almost the same as for Template N_3. Suppose a P-node X matches Template N_4, so X is not the pertinent root. Further, suppose that there are k (k > 1) partial children of X. Let P_ℓ and P_m be any two partial children of X. The full children of X must be placed between P_ℓ and P_m. We then merge P_ℓ and P_m to create a new Q-node Y with both endmost children empty, and all full children appearing in one consecutive subsequence $Pert_i(Y)$ of the children of Y. We then pair off all, except for possibly one, of the remaining partial children and merge those partial children that were paired off, thereby reducing the number of subsequences of full children to $\left\lceil \frac{k}{2} \right\rceil$. For every partial child W which does not have an endmost full child, and corresponding maximal subsequence $Pert_j(W)$, we identify the pertinent edges of Frontier(W) in a pseudo-vertex, and mark all full children void. Note that we avoid placing an extra handle if k is odd. If this situation occurs, then there is some sequence $Pert_i(X)$ which has a full endmost child. We do not identify the sequence $Pert_i(X)$ to a pseudo-vertex, because the reduction is now able to proceed without the placement of a further handle. Again, we attempt to minimise the number of handles placed.

Algorithm 7 details the algorithm Place_Handle_N4.

Algorithm 7: Place_Handle_N4(X)
{ Place Handle(s) to enable reduction to continue after reduction failed and X matched Template N_4 }

{ P-node, not pertinent root, two or more
partial children - see Figure 21 }
Let the partial children be $P_1, P_2, ..., P_k$;
where $k \geq 2$
Denote the full and empty endmost children
of any P_i by
Full(P_i) and Empty(P_i)
Let $\mathcal{P}$ be a partition of the partial children
into $\left\lceil \frac{k}{2} \right\rceil$ sets, so that each set, except for
possibly one set (if k is odd), has two
elements
For every element of $\mathcal{P}$ that contains a pair
P_i and P_j ($i, j \neq \ell, m$) do
Let Full(P_i) and Full(P_j) be immediate
siblings
Let new endmost child of P_i be
Empty(P_j)
Delete P_j
Identify full leaves from P_i in a pseudo
-vertex
$\gamma_g(G) = \gamma_g(G) + 1$
Mark full children of P_i as void
{ Now the reduction may proceed }
end

Figure 21 illustrates the situation when Template N_4 is matched.

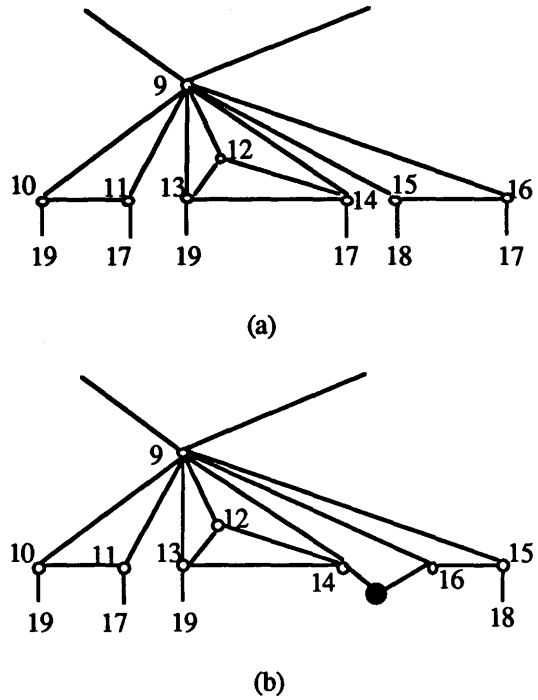


Figure 21. Illustration of Handles when Template N_4 is matched
(a) Before reduction for vertex 17; (b) after placement of pseudo-vertex.

All that remains to be discussed is the actual placing of the second end of the handles which we insert, and to prove the algorithm correctness. The placing of the handles inserted during the reduction for a vertex i is done once the reduction process for vertex i has been completed. The following lemma will aid us in our discussion.

Lemma 1: Not every sequence of pertinent children during a reduction for a vertex i is marked void.
Proof: First of all, note that, for a node X, Templates N_2 and N_4 do not allow X to be the pertinent root. If Template N_1 is matched, and X is the pertinent root, then Algorithm 4 explicitly selects a sequence $Pert_i(X)$ not to be marked void. If Template N_3 is matched, then we join two partial children, and the pertinent leaves from those two partial children, as well as the full children of X are not marked void. ❑

Using Lemma 1 we may identify the edges represented by full leaves not marked void in the base. Now, we place a handle from each pseudo-vertex created during the reduction for vertex i to the base, and place the corresponding identified edges along that handle.

Repeated application of the PQ-tree reduction algorithm as described in Algorithm 2, together with the vertex addition for a graph has complexity O(q) (see [3] and [4] for more details). We call this algorithm the Pruned Reduction Algorithm. The complexity of Algorithm 3 can now be derived.

Observe that Algorithm 3 only deals with pertinent nodes. Also, each pertinent node dealt with by Algorithm 3 is considered during exactly one invocation of Algorithm 3, since it is removed from the PQ-tree thereafter. From [3] and [4] there are O(q) pertinent nodes in total during the entire reduction process. Observe that the complexity of Algorithms 4, 5, 6 and 7 is O(Reduce($Pert_i(X)$) + |pertinent nodes|). Hence the only detail that needs to be elaborated on is the complexity of Reduce($Pert_i(X)$) operation for all maximal subsequences, $Pert_i(X)$, of pertinent children of a Q-node X. To successfully perform Reduce($Pert_i(X)$), we traverse the children of X from left to right, looking for maximal sequences of pertinent children. It is shown in [4] that Reduce($Pert_i(X)$) has complexity O(|Children of X|). We may scan these children O(p) times during a reduction pass. From [7] it is shown that O(|Children(X)|) = O(p) as well. So, during a reduction pass for a vertex, Reduce($Pert_i(X)$) has complexity $O(p^2)$. Therefore, overall, the complexity of Reduce($Pert_i(X)$) is $O(p^3)$, because there are p reduction passes. We have the following result on Algorithm 3.

Theorem 1: The Pruned Reduction Algorithm, Algorithm 2, together with Algorithm 3, embeds a 2-connected graph G on a surface of genus $\gamma_g(G)$, and has $O(p^3)$ time complexity.

5 Conclusion

Because there is no efficient algorithm for finding the genus of a graph, in general the results of the algorithm are not easy to compare against the actual genus of the graph. An embedding of a 2-connected graph G on a surface of genus n is a 2-cell embedding if a closed curve in a region of this embedding can be continuously deformed (on the surface) to a single point. The maximum genus $\gamma_m(G)$ of a 2-connected graph G is the maximum genus of a surface on which G can be 2-cell embedded. The maximum genus of a graph is defined (see [2]). There are two well-known types of graphs for which the genus $\gamma(G)$ and maximum genus $\gamma_m(G)$ is known. The graphs are the complete graph on p vertices, K_p, and the complete bipartite graph with partite sets m and n, namely $K_{m,n}$.
Table 1 shows the results that were obtained on the complete graph K_p, on p vertices. In Table 2 the results obtained for bipartite graphs $K_{3,n}$ are given.

Table 1.

p	$\gamma(K_p)$	$\gamma_g(K_p)$	$\gamma_m(K_p)$
5	1	1	3
6	1	2	5
7	1	4	7
8	2	6	10
9	2	9	14

Table 2.

n	$\gamma(K_{3,n})$	$\gamma_g(K_{3,n})$	$\gamma_m(K_{3,n})$
3	1	1	2
4	1	2	3
5	1	3	4
6	1	4	5
7	2	5	6

Lastly, we observe that the Pruned Reduction Algorithm together with Algorithm 3 does not necessarily produce a 2-cell embedding of G. Let H be the plane subgraph of G obtained by deleting all the edges of G that were ever marked void. Then, it may happen that we place, for two different reductions i and j, two handles from the same two regions with respect to the embedding of H. Consider Figure 22 with two edges on two handles, namely the edges $e_1 = \{11, 17\}$ and $e_2 = \{13, 18\}$.

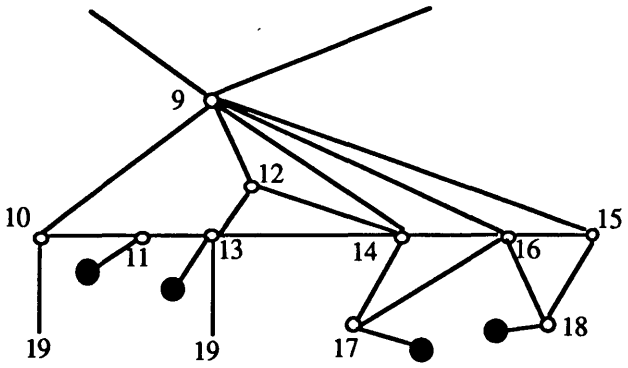


Figure 22 . An example of an embedding of G that is not a 2-cell.

It is easy to see that the embedding of G in Figure 22 is not a 2-cell, if we place a closed curve C along both handles which then cannot be shrunk to a single point.

A refinement of the algorithm could be the addition of a checking pass of the algorithm, where one can check for redundant handles. For example, in Figure 22 only one handle is required, since the handles connect the same "regions". Similarly "cycles" of handles are unnecessary. It remains an open problem to determine if Algorithm 3 may be extended to produce 2-cell embeddings of a non-planar graph G on some surface. Once a 2-cell embedding of G is guaranteed, then it is an easy extension of the basic embedding algorithm by [5] (which also uses PQ-trees) to provide an embedding of G on the surface of genus $\gamma_g(G)$. Hence it provides a description of the corresponding circuit on a surface of reasonably low genus.

References

1. J Battle, F Harary, Y Kodama and J W T Youngs. 'Additivity of the genus of a graph'. *Bull. Amer. Math. Soc.*, **68**:565-568, (1962).
2. M Behzad, G Chartrand and L Lesniak-Foster. *Graphs and Digraphs*, Wadsworth International, California, 1979.
3. K S Booth and G S Lueker. 'Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-Tree Algorithms'. *J. Comput. Syst. Sci.* **13**:335-379, (1976).
4. D I Carson. Planarity Testing and Embedding Algorithms. M.Sc. dissertation, University of Natal, Natal, 1990.
5. N Chiba, T Nishizeki, S Abe and T Ozawa. 'A Linear Algorithm for Embedding Planar Graphs using PQ-trees'. *J. Comput. Syst. Sci.* **30**:54-76, (1985).
6. I S Filotti, G L Miller and J Reif. 'On determining the genus of a graph in $O(V^{O(g)})$ steps'. In *Proc. of the 11th Ann. Symp. on the Theory of Computing*, pp. 27-37, Atlanta Georgia (1979).
7. R Jayakumar, K Thulasiraman and M N S Swamy. '$O(n^2)$ Algorithms for Graph Planarization'. *IEEE trans. Comput.-Aided Design* **8**(3):257-267, (1989).
8. A C Karabeg. *PQ-tree data structure and some Graph Embedding Problems*. Ph.D. thesis, University of California at San Diego, (1988).
9. A Lempel, S Even and I Cederbaum. 'An Algorithm for Planarity Testing of Graphs'. In *Theory of Graphs, Int. Symp.*, pp. 215-232, Rome (1966), Gordon and Breach, New York (1967).
10. C Thomassen. 'The graph genus problem is NP-complete'. *Journal of Algorithms* **10**(4):568-576, (1989).

Acknowledgments:

We wish to thank the Foundation for Research Development for financial support during the completion of this project. We are also indebted to the referees for their useful comments.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript

Manuscripts for *review* should be prepared according to the following guidelines.

- Use wide margins and $1\frac{1}{2}$ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1, 2, 3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) LaTeX file(s), either on a diskette, or via e-mail/ftp – a LaTeX style file is available from the production editor;
2. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

   Further instructions on how to reduce page charges can be obtained from the production editor.
3. In camera-ready format – a detailed page specification is available from the production editor;
4. In a typed form, suitable for scanning.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for LaTeX or camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers in categories 2 and 4 above will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References

1. E Ashcroft and Z Manna. 'The translation of 'goto' programs to 'while' programs'. In *Proceedings of IFIP Congress 71*, pp. 250–255, Amsterdam, (1972). North-Holland.
2. C Bohm and G Jacopini. 'Flow diagrams, turing machines and languages with only two formation rules'. *Communications of the ACM*, 9:366–371, (1966).
3. S Ginsburg. *Mathematical theory of context free languages*. McGraw Hill, New York, 1966.

South African Computer Journal

Number 8, November 1992
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 8, November 1992
ISSN 1015-7999

Contents

Sigma Press