J M Bishop 20/3/90

South African Computer Journal
Number 1
January 1990

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 1
Januarie 1990

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

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

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

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083

Assistant Editor: Information Systems
Professor Peter Lay
Department of Accounting
University of Cape Town
Rondebosch 7700

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

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

Editorial

At last the first edition of SACJ is available. I trust that readers will find it worth the waiting. There have been a number of teething problems in getting things together, the many details of which need not be spelt out here. One significant challenge was to cope with the consequences of the resignation of Quintin Gee, QI's highly competent production editor. He assisted in the initial phases of getting this publication together but had to resign for personal reasons. It is fitting to acknowledge here not only his initial advice and assistance in getting this first issue of SACJ off the ground, but also the many hours of work that he spent in previously producing QI.

Quintin's resignation meant that a new *modus operandi* for typesetting and printing had to be established. The exercise was not only time-consuming, but also has significant cost implications. Fortunately, the Unit for Software Engineering (USE) at Pretoria University has generously agreed to sponsor this first edition. On behalf of the South African computing community, I should like to thank them for their generosity. Now that they have made a first issue of SACJ possible, it is hoped to solicit the sponsorship of one of the larger computer companies for future editions.

It might be of interest to take readers on a walk through the new journal to highlight various aspects. To begin with, the cover design follows that of several journals whose titles have the format: *The South African Journal of Subject / Die Suid-Afrikaanse tydskrif vir Vakgebied* (where *Subject* and *Vakgebied* are appropriately instantiated). While colours vary, these journals generally have *Subject* and *Vakgebied* restated on the darker portion of the cover. SACJ's title was chosen in preference to a more descriptive but also more cumbersome title such as *The South African Journal of Computer Science and Information Systems*. The appearance of the words *Computer Science and Information Systems / Rekenaarwetenskap en Inligtingstelsels* on the cover are thus out of step with the original inspiration, but seem appropriate under the circumstances.

The inside cover is of interest for several reasons. Firstly, note that Peter Lay has kindly agreed to lighten my task by acting as an assistant editor. He will deal with matters relating to Information Systems. *Contributions in this area should henceforth please be sent directly to him.* Also note that an editorial board of distinguished persons has been assembled. I should like to once again thank board members for adding status to SACJ by agreeing to serve in this capacity. They will be consulted on matters of editorial policy whenever appropriate. Finally, the subscription costs have been increased to keep pace with production costs. This increase does not affect SAICS members, who will continue to receive the journal as one of the benefits of membership.

The guest editorial by Pieter Kritzinger makes for interesting reading. Several points of concern about computer-related research in South Africa are raised. I trust that the article will focus attention on these problems and stimulate a debate which will lead to eventual solutions. It is hoped to make guest editorials a regular feature of future SACJ issues.

Of the eight research papers offered in the journal, four have been gone through the normal channel of refereeing and revisions. The remainder were submitted to the Vth SA Computer Symposium and are published here by invitation. Each paper submitted to the chairman of the symposium's program committee was sent to three referees. A ranking scheme, reflecting an aggregate measure of referee evaluation, was used as a basis for deciding on papers to be presented. After further editorial evaluation, the authors of four of the five highest ranking papers were invited to submit their papers to SACJ. While it was not possible to contact the fifth author in time for this edition, but it may be possible to publish that paper, together with a selection of others from the symposium, in future SACJ editions.

In the section marked *Communications* various items of news arriving at the editor's desk have been published. It was particularly gratifying to receive book review submissions in response to a prior general appeal. There has also been an enthusiastic response from book publishers, who have sent in a number of books for review. Titles are listed in the *Communications* section. Please contact me if you are willing to review one (or more) of these. Naturally, reviews of other books of interest in your possession will also be welcomed.

The final point to highlight in this walk through the journal is the increase in page charges indicated on the back inside cover. These reflect the increased cost of production. Since research papers in SACJ qualify for state subsidy at academic institutions, the charges should not, in general, present major problems for authors. However, it is worth pointing out that the final format of papers submitted significantly impacts on both the financial and editorial load. Submissions in camera-ready format (or nearly so) result in both a cost savings and a speed up of turn-around time by several orders of magnitude. Since many readers may not be familiar with the printing process, it may be helpful to say something about it in order to substantiate this claim.

The printing process basically involves typesetting, shooting (or photographing), and then reproduction and binding. Apart from limiting the amount of material, the printer's client has very little control over the cost of shooting, reproduction and binding. On the

other hand, anyone equipped with moderate text- or word processing facilities and a laser printer can go a long way (if not all the way) towards typesetting a paper. Even a partially typeset paper helps significantly, as I will explain below.

By typesetting I simply mean knocking the paper into the right shape and producing a laser printout. The printers regard this is a tedious, error-prone task, even if they start off with an ASCII file rather than a hardcopy of the paper. Consequently, they tend to handle large-scale typesetting by subcontracting the task. Moreover, while they may be willing to typeset uncomplicated text, they tend to balk at text containing specialized mathematical and other notation. However, they are quite skilful at cutting and pasting text, and at enlarging or reducing photographed or scanned diagrams. They are even willing to redraw sketches which are not too complicated.

As a result of the above, I have pressed several authors to do their own typesetting. In cases where it was problematic to produce double column format, a single column of appropriate width was requested. While this is a second-best option, it allows for cutting and pasting to be done by the printers. Some sketches have either been directly reduced from the author's original, while others have been redrawn by the printer. By way of exception, I have personally undertaken the typesetting of a few papers using WordPerfect. However, I would like to avoid this as far as possible in future, and consequently appeal to potential authors to make every effort to do their own typesetting.

From SACJ's point of view encouraging authors to do their own typesetting involves a compromise in that there will inevitably be slight variations in the print from one article to the next (as is in fact the case in this issue). If you are pedantically inclined, you might consider this to be a disaster. Personally, I regard it as a rather neat advertisement for the typesetting skills of SACJ contributors.

As an aside, since the handling of TEX files was initially a problem for me, I was pleased to discover that Peter Wood and his colleagues at UCT have mastered the art of producing TEX printout in the format now before you. Future authors who use TEX should consult them on details.

As to the future, it is not possible at the this stage to commit to a fixed number of SACJ issues per year. The number of issues is constrained by finance, submissions of the right quality, and time available to the editorial staff (including our anonymous and unsung heroes - the referees). The ideal is to produce four issues per year, but this may not always be attainable.

In conclusion, if readers have as much fun in reading this first issue of SACJ as I have had in editing it, the hours spent on it will have been well worthwhile. Hopefully SACJ is destined not only to be a permanent feature of the Southern African computing scene, but also to significantly contribute to research in the region.

Derrick Kourie
Editor

This SACJ issue is sponsored by
The Unit for Software Engineering
(USE)
Department of Computer Science
Pretoria University

Guest Editorial

Funding Computer Science Research in South Africa

P S Kritzinger
Department of Computer Science, University of Cape Town, Rondebosch 7700

The word *research* has many connotations and is often abused. In everyday language a person does not simply *search for information in a library*, for example, but rather does *research*, thus pretentiously conferring an aura of intellectual activity on an effort which requires very little original thought.

Here I will interpret the term to mean work which generates results that gain international recognition. This implies that the work is published in good international journals or presented at international conferences. I believe this is the only valid index of the quality of research.

With very few exceptions, the computer industry in South Africa is a consumer of computer technology, rather than a developer. In contrast with, say, the chemical industry, there is therefore no tradition of research in computer science in the South African computer industry and computer science researchers therefore have, as virtually their only source of funding, the Foundation for Research Development (FRD) which has its origins in the CSIR.

The FRD was formed in April 1984 with the development and use of research expertise in the natural and applied sciences and engineering as its mission. This mission is primarily directed at the universities, museums and technicons with the ultimate aim of improving the life of all South Africans.

Although the FRD has several programmes, the two which are of main concern to computer scientists are the Core Programmes and the Special Programmes.

FRD Core Programmes foster the optimum development of a scientific and technological knowledge base by supporting individual self-initiated research. These programmes, started only about 4 years ago, have met with considerable acclaim, particularly in regard to the way in which research funding for a particular individual is decided. To qualify for support within a Core Programme, researchers must obtain a certain evaluation status within the FRD and funding is then linked directly and exponentially to the merit of the individual concerned, rather than being linked to the specific project proposed.

In the evaluation process, peer review is strongly emphasised. The researcher himself is expected to nominate referees, whose status and reports play a decisive role in the evaluation. As a result of this evaluation, an applicant is assigned a specific evaluation status category. There are currently 9 categories in all, but the ones of main interest are:

- **A** researchers who are without any doubt accepted by the international community as being amongst the leaders in their field (52);
- **B** researchers not in category **A** but who nevertheless enjoy considerable international recognition as independent researchers of high quality (182);
- **C** proven researchers who have maintained a constant high level of research productivity and whose work is regularly made known internationally, or proven researchers whose current research output is less but who are actively engaged in scholastic activity (433);
- **P** researchers younger than 35 years of age who have already obtained a doctoral degree and who have shown exceptional potential as researchers (10); and
- **Y** young researchers usually under 35 years of age, who are highly likely to achieve C status by the end of their support period (108).

The number of researchers in the various categories as of August 1989 has been indicated in parentheses above. Of these, only 7 persons are computer scientists: 1 in category B; 3 in category C; and 3 in category Y. Only 4 departments of computer science are involved.

The other main programmes of concern to computer persons are the Special Programmes which aim at developing research manpower in priority areas. After identification of an area that merits particular research development, given local expertise, a Special Programme is launched to address the problem in the national interest.

Although a manager of a Special Programme has to be an FRD evaluated researcher, the same need not be true for the other team members. Regular peer evaluation of researchers as well as evaluation of the progress and results of Special Programmes are considered essential. Special Programme awards will be made for the first time towards the end of 1989. It is therefore not yet known whether proposals already submitted for programmes in computer science have been successful.

It is clear that, in the context explained above, there is virtually no computer science research being done in South Africa - a scary thought which has considerable implications for this country! Why is this so? There are several reasons, but I would like to single out two in particular.

Qualified faculty and students is an abiding problem at the heart of computer science departments. Acquisition of new faculty members is an issue intimately linked to the number of graduate students successfully completing PhD degrees. This problem is by no means unique to South Africa. For instance, data gathered in North America indicates that in 1983 there were over 200 vacancies in the 91 departments that have doctoral programmes in computer science. At the same time, only approximately 250 PhD's were granted in North America - a figure that has remained relatively unchanged for the past several years. A large number of those graduates were attracted to industry and industrial research laboratories. Although I do not have solid data at my disposal, I would think that South Africa produces at most one PhD graduate in computer science per year. There are currently 20 departments of computer science at universities is South Africa. It will therefore take us 20 years to locally produce one new faculty member with a PhD in computer science for every university.

Contributing to the above problem is our current academic image. The graduate student usually sees concerned computer science faculty members as rather harried individuals, having large undergraduate classes, much committee and professional work, and labouring under an ill-fitting model (applicable to more established disciplines) for decisions on tenure, salary and promotion. Further, as undergraduates, many prospective graduate students were not engaged in research projects involving computer science faculty, and for that reason were not exposed to graduate students doing research, and rarely developed a camaraderie with any computer science professionals. At last count there were only 5 individuals in South Africa who completed their computer science doctorate at a university outside South Africa where they had the good fortune to work in an environment in which sufficient faculty and funds were available to create an ethos of research. It is difficult to convince students that their interests and goals can be served by a PhD in computer science or by an academic career.

The second problem, which is of greater concern to me since there is no immediate solution to it, has to do with the fact that senior persons who decide the fate and fortune of academic computer science departments are, in general, individuals whose professional careers started well before computing machines came into every day use - that is to say, in the years B.C. (Before Computers). These persons of influence do not always understand what "computers" are, and what their potential influence upon the workplace in particular and society in general are. As far as research (as opposed to teaching) is concerned, most of them understand that a medical school needs special and expensive equipment (not to mention, expensive faculty) and that engineers must have a workshop and special machinery to teach their students and conduct research. They understand that if one needs to build up a defense industry, it will cost billions of rands; but they are not so sure about computer science, even though many other countries have recognised it as of national strategic importance.

I believe that only time and dedication will lead to a solution of these seemingly insurmountable problems and allow computer scientists to take their rightful place in the research community in South Africa.

Bibliography

P J Denning, [1981], Eating our seed corn, *Communications of the A.C.M.*, **24**(6), pp.341 - 343.

J Tartar (Ed.), [1985], The 1984 Snowbird Report: Future issues in computer science, *Communications of the ACM*, **28**(5), pp490 - 493.

D Gries, R Miller, R Ritchie and P Young, [1986], Imbalance between growth and funding in academic computing science: two trends colliding, *Communications of the A.C.M.*, **29**(9), pp.870 - 878.

J E Hopcroft and D B Krafft, [1987], Towards better computer science, *IEEE Spectrum*, pp.58 - 60.

Finding Regular Paths in Acyclic Graphs

P T Wood
Department of Computer Science, University of Cape Town, Rondebosch 7700

Abstract

Increasing the expressive power of relational query languages by providing some form of recursion is currently a topic of much research. For many recursive queries in relational databases, a relation can be represented as labelled directed graph G, while the query can be viewed as finding simple paths in G that satisfy a given regular expression. Based on such a query language, it has been shown recently that, in this context, the general query evaluation problem on cyclic graphs is intractable. In this paper, we present an efficient algorithm for the case in which the graphs are acyclic.
Keywords: *Algorithms, graph databases, finite state automata, query processing, regular expressions.*
Computing Review Categories: *G.2.2, H.2.3.*

Presented at V th S.A. Computer Symposium.

1 Introduction

There has been much interest recently in the problems presented by adding the power of recursion to relational query languages [3]. This research has been prompted, on the one hand, by the fact that certain simple queries, such as finding the transitive closure of a binary relation, are not expressible in traditional query languages [2], and, on the other, by the desire to integrate relational databases and logic programming [4].

While most researchers have been studying the language Datalog as a possible candidate for a recursive query language [13], we have been considering a graph-based language, called $\mathbf{G}^+$ [6,7]. The design of $\mathbf{G}^+$ is based on the observation that many of the recursive queries that arise in practice – and in the literature – amount to graph traversals. In $\mathbf{G}^+$ we view the database as a directed, labelled graph, and pose queries which are graph patterns; the answer to a query is the set of subgraphs of the database that match the given pattern. In our prototype implementation [5], queries are drawn on a workstation screen and the database and query results are also displayed pictorially.

Example 1 Consider a software engineering application in which the call-graph G of the modules comprising a program is to be represented. Assume that each module is denoted in the graph by a pair of nodes: one being the module header, the other the module body. For each module m, there is an edge labelled b from the node denoting the header of m to the node denoting the body of m. There is also an edge labelled c from the node denoting the body of a module m to the header node of each module called by module m.

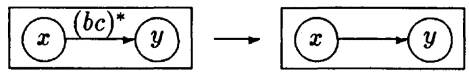

Figure 1: Query to find pairs of modules connected in a call-graph.

Assume that we want to find all pairs (m, n) of module headers such that module m calls module n either directly or indirectly. This query can be expressed by the graph pattern of Figure 1. The left-hand box in the figure contains the pattern graph, while the right-hand box contains the summary graph which specifies how the output is to be presented to the user. The edges of a pattern graph can be labelled with regular expressions; in this case the desired expression is $(bc)^*$. This regular expression is used to match the edge labels along paths in G, thereby satisfying our original request. □

Although queries in $\mathbf{G}^+$ can be a lot more general than exemplified above, this special case is challenging enough from an algorithmic point of view if we want to process queries efficiently. The basic query evaluation problem is: given a regular expression R and a graph G, find all pairs of nodes in G which are connected by a simple path p, where the concatenation of edge labels comprising p is in the language denoted by R.

When trying to find an efficient solution for this problem to incorporate in our implementation of $\mathbf{G}^+$, we were somewhat surprised to discover that the query of Example 1 is in fact NP-complete on general cyclic graphs [12]. It turns out that, because we would expect a call-graph as described

to be bipartite, the query of Example 1 can be evaluated on valid call-graphs in polynomial time [12]. In this paper, however, we will assume that the graphs being queried are acyclic (e.g. call-graphs corresponding to nonrecursive programs). We show that, in this case, an efficient query evaluation algorithm is possible.

Section 2 covers the necessary background material, such as the definition of $\mathbf{G}^+$ as well as some material from formal language and automata theory. The query evaluation algorithm is described and proved correct in Section 3, which also includes examples of the best- and worst-case behaviour of the algorithm. Conclusions and topics for future research are presented in Section 4.

2 Background

We begin in Section 2.1 by defining the graph structures as well as the class of queries over these structures in which we are interested. Additional definitions from formal language theory, which are required in order to describe the query evaluation algorithm, are presented in Section 2.2.

2.1 The language $\mathbf{G}^+$

Definition 1 A *database graph* (*db-graph*, for short) $G = (N, E, \psi, \Sigma, \lambda)$ is a directed, labelled graph, where N is a set of *nodes*, E is a set of *edges*, and ψ is an *incidence function* mapping E to $N \times N$. Multiple edges between a pair of nodes are permitted in db-graphs. The labels of G are drawn from the finite set of symbols Σ, called the *alphabet*, and λ is an *edge labelling function* mapping E to Σ. □

Definition 2 Let Σ be a finite alphabet disjoint from $\{\epsilon, \emptyset, (,)\}$. A *regular expression* R over Σ and the language $L(R)$ *denoted* by R are defined in the usual way. Let $G = (N, E, \psi, \Sigma, \lambda)$ be a db-graph and $p = (v_1, e_1, \ldots, e_{n-1}, v_n)$, where $v_i \in N$, $1 \leq i \leq n$, and $e_j \in E$, $1 \leq j \leq n-1$, be a path (not necessarily a simple path) in G. We call the string $\lambda(e_1) \cdots \lambda(e_{n-1})$ the *path label* of p, denoted by $\lambda(p) \in \Sigma^*$. Let R be a regular expression over Σ. We say that the path p *satisfies* R if $\lambda(p) \in L(R)$. The *query* Q_R on db-graph G is defined as

$$Q_R(G) = \{(x, y) \mid \text{there is a simple path from } x \text{ to } y\ (x \neq y) \text{ in } G \text{ which satisfies } R\}$$

If $(x, y) \in Q_R(G)$, then (x, y) *satisfies* Q_R. □

2.2 Automata theory

Definition 3 A *nondeterministic finite automaton* (NDFA) M is a 5-tuple $(S, \Sigma, \delta, s_0, F)$, where

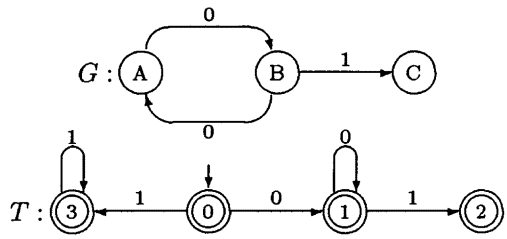

Figure 2: A db-graph G and transition graph T of an NDFA accepting $L(R)$, where $R = 0^* + 1^* + 0^*1$.

S is a finite set of *states*, Σ is the *input alphabet*, δ is the *state transition function*, $s_0 \in S$ is the *initial state*, and $F \subseteq S$ is the set of *final states*. The concepts of the *extended* transition function δ^*, the language $L(M)$ *accepted* by M, and the *transition graph* associated with M are defined in the standard way, as is the notion of a *deterministic finite automaton* (DFA) [8]. □

Definition 4 Given an NDFA $M = (S, \Sigma, \delta, s_0, F)$, for each pair of states $s, t \in S$, the *language from s to t*, denoted by L_{st}, is the set of strings that takes M from state s to state t. In particular, for a state $s \in S$, the *suffix language* of s, denoted by L_{sF} (or $[s]$, for short), is the set of strings that takes M from s to some final state. Clearly, $[s_0] = L(M)$. □

Given a regular expression R over Σ, an ϵ-free NDFA $M = (S, \Sigma, \delta, s_0, F)$ which accepts $L(R)$ can be constructed in polynomial time [1]. From now on, we will assume that all NDFAs are ϵ-free.

Let R_1 and R_2 be regular expressions. In the subsequent analysis, it will be useful to refer to an NDFA which accepts the language $L(R_1 \cap R_2)$. The construction of such an NDFA is defined as follows.

Definition 5 Let $M_1 = (S_1, \Sigma, \delta_1, p_0, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, q_0, F_2)$ be NDFAs. The NDFA for $M_1 \cap M_2$ is $I = (S_1 \times S_2, \Sigma, \delta, (p_0, q_0), F_1 \times F_2)$, where, for $a \in \Sigma$, $(p_2, q_2) \in \delta((p_1, q_1), a)$ if and only if $p_2 \in \delta_1(p_1, a)$ and $q_2 \in \delta_2(q_1, a)$. We call the transition graph of I the *intersection graph* of M_1 and M_2. □

Example 2 Let R be the regular expression $0^* + 1^* + 0^*1$. A db-graph G and the transition graph T of an automaton M accepting $L(R)$ are shown in Figure 2. State 0 is the initial state of M, while all states are final (denoted by a double circle). (We do not show (reject) states which are not on some path from the inital state to a final state.) The suffix language of state 1 is $[1] = 0^* + 0^*1$, while $[2] = \epsilon$. The intersection graph I of G and T is depicted in Figure 3. Isolated nodes have

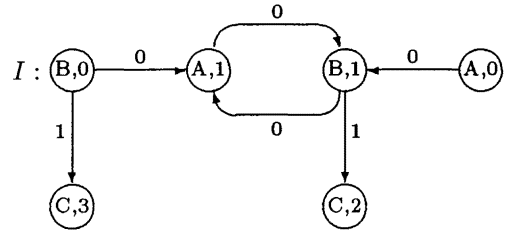

Figure 3: The intersection graph of G and T from Figure 2.

been omitted from Figure 3, so that I is shown with only 6 rather than 12 nodes. □

We saw in the previous section that, in general, it is very unlikely that we will find an algorithm for evaluating Q_R on an arbitrary graph G which will always run in time polynomial in the size of G. However, if G is acyclic, Q_R can be evaluated in polynomial time for any regular expression R. The reason is that every path in an acyclic graph is simple.

Lemma 1 *Given query Q_R, db-graph G, and nodes x and y in G, deciding whether $(x, y) \in Q_R(G)$ can be done in polynomial time.*

Proof: We can view the db-graph G as an NDFA with initial state x and final state y. Construct the intersection graph I of G and $M = (S, \Sigma, \delta, s_0, F)$, an NDFA accepting $L(R)$. There is a path p from x to y satisfying R if and only if there is a path in I from (x, s_0) to (y, s_f), for some $s_f \in F$. Since G is acyclic, p is simple. All this can be done in polynomial time [9]. □

3 The Query Evaluation Algorithm

A naive method for evaluating a query Q_R on an acyclic db-graph G is to traverse every path satisfying R in G exactly once. The penalty for this is that such an algorithm takes exponential time when G has an exponential number of paths. Although, in general we know we cannot expect an algorithm to perform much better, for acyclic graphs we have already shown in Lemma 1 that polynomial time evaluation can be guaranteed. In fact, query evaluation on such graphs can be performed quite efficiently, as we demonstrate in this section.

Given a query Q_R and an acyclic db-graph G, the evaluation algorithm implicitly traverses the intersection graph of G and an NDFA M accepting $L(R)$. In fact, the algorithm traverses G and marks the nodes of G with sets of states from M. Since G is acyclic, its nodes can be topologically ordered so that if there is an edge from x to y in G, then $x < y$. By accumulating partial results while traversing the nodes of G in descending topological order, the algorithm ensures that, in effect, each node in the intersection graph is visited at most once during the entire execution of the algorithm. This evaluation algorithm, shown in Figure 4, is based on an algorithm for computing the transitive closure of an acyclic graph given in [11].

The algorithm starts by constructing an NDFA M accepting $L(R) - \{\epsilon\}$ (since $(x, y) \in Q_R(G)$ only if $x \neq y$) and by setting up adjacency lists for G. In Line 3, the nodes of G are sorted into topological order and the nodes on each adjacency list are sorted into ascending order (with respect to the topological ordering). Previous markings (initialized in Line 5) are used to avoid revisiting nodes. The fact that previous markings are never removed from a node means that each node in the intersection graph I of G and M can be visited at most once during the execution of the entire algorithm. Consequently, after visiting a node (v, s), we must already have found all the nodes reachable from (v, s). The sets $R[v, s]$ for each node (v, s) (initialized in Line 6) serve this purpose.

Some of the efficiency of Algorithm T is derived through the use of bit vectors, which allow set membership to be determined in constant time. For each node v in G, $BR[v]$ (initialized in Line 7) provides a bit vector representation of $R[v, s]$ between Lines 15 and 21 of procedure SEARCH-T (Figure 5). $BR[v]$ is non-empty only when some (v, s) is on the stack of SEARCH-T and since G is acyclic, no (v, t) $(s \neq t)$ can be on the stack at the same time. Lines 13, 14 and 20 of SEARCH-T ensure that $BR[v] = R[v, s]$ between Lines 15 and 21, while Line 22 restores $BR[v]$ to $\emptyset$ before SEARCH-T(v, s) exits.

By processing the nodes of G in descending order (Line 8), we ensure that if searching from (v, s) we encounter a node (w, t) for which $t \in PM[w]$ (that is, (w, t) has already been visited), then all nodes reachable from (w, t) are in $R[w, t]$ and can be added to $R[v, s]$ (Lines 19 and 20). The efficiency of the algorithm derives from the fact that nodes adjacent to the current node are visited in ascending order. In this way, the longest path between two nodes is always traversed first and significant work is done only for this path, not for any "shortcuts". This is made precise in the complexity analysis given below.

Our next task is to show that Algorithm T is correct. We begin by proving that the set $R[v, s]$ contains all the nodes reachable from (v, s).

Lemma 2 *Let I be the intersection graph of a db-graph $G = (N, E, \psi, \Sigma, \lambda)$ and the transition graph of the NDFA $M = (S, \Sigma, \delta, s_0, F)$ constructed in Line 1 of Algorithm T. After termination of Algorithm T, node (w, t) in I is in $R[v, s]$*

Algorithm T: Evaluation of a query on an acyclic db-graph.

INPUT:
Query Q_R and db-graph $G = (N, E, \psi, \Sigma, \lambda)$, where $|N| = n$.

OUTPUT:
$Q_R(G)$, the value of Q_R on G.

METHOD:
1. Construct an NDFA $M = (S, \Sigma, \delta, s_0, F)$, where $|S| = q$, accepting $L(R) - \{\epsilon\}$.
2. Set up an adjacency structure representing G (e.g. adjacency lists with edge labels included).
3. Topologically order the nodes of G, and sort the nodes in the adjacency lists into ascending order.
4. $Q_R(G) \leftarrow \emptyset$
5. **for** each node $v \in N$ **do** $PM[v] \leftarrow \emptyset$
6. **for** each node $v \in N$ and state $s \in S$ **do** $R[v, s] \leftarrow \emptyset$
7. **for** each node $v \in N$ **do** $BR[v] \leftarrow \emptyset$ /* $BR[v]$ is a bit vector of size qn */
8. **for** each node $v \in N$ **do** /* in descending order */
   begin
9. SEARCH-T(v, s_0)
10. $V \leftarrow \emptyset$ /* V is a bit vector of size n */
11. **for** each $(w, t) \in R[v, s_0]$ **do**
12. **if** $t \in F$ **and** $w \notin V$ **then** add (v, w) to $Q_R(G)$ and w to V
    end

Figure 4: Evaluation of a query on an acyclic db-graph.

procedure SEARCH-T (v, s)
begin
13. $R[v, s] \leftarrow \{(v, s)\}$
14. $BR[v] \leftarrow \{(v, s)\}$
15. **for** each edge (v, w) labelled a in G **do** /* in ascending order on w */
16. **for** each $t \in \delta(s, a)$ **do**
    begin
17. **if** $t \notin PM[w]$ **then** SEARCH-T (w, t)
18. **if** $(w, t) \notin BR[v]$ **then**
19. **for** each $(x, q) \in R[w, t]$ **do**
20. **if** $(x, q) \notin BR[v]$ **then** add (x, q) to $R[v, s]$ and $BR[v]$
    end
21. $PM[v] \leftarrow PM[v] \cup \{s\}$
22. **for** each $(w, t) \in R[v, s]$ **do** remove (w, t) from $BR[v]$
end SEARCH-T

Figure 5: SEARCH procedure for Algorithm T (Figure 4).

if and only if there is a path in I from some (u, s_0) to (w, t) which passes through (v, s).

Proof: (Only if) It is not hard to show that Algorithm T effectively traverses paths in I. Since $R[v, s]$ is initialized to $\emptyset$ in Line 6, (w, t) is in $R[v, s]$ only if SEARCH-T(v, s) was called. Hence, there is a path in I from some (u, s_0) to (v, s). We prove by induction on the position of v in the topological ordering that if $(w, t) \in R[v, s]$, then there is a path from (v, s) to (w, t) in I. Assume there are n nodes in G. If v is in position n, then (v, s) has no successors in I and so (w, t) must have been added to $R[v, s]$ in Line 13. Hence $(w, t) = (v, s)$, and there is a trivial path of length zero from (v, s) to (w, t). Assume the result holds for all nodes in positions greater than k, $k < n$, and that v is in position k. Now (w, t) can be added to $R[v, s]$ only in Line 13 or Line 20. If it is added in Line 13, the result follows as before. If (w, t) is added to $R[v, s]$ in Line 20, then $(w, t) \in R[x, q]$ for some successor (x, q) of (v, s) in I. Since x is in a position greater than k in the topological ordering, we know by the inductive hypothesis that there is a path from (x, q) to (w, t). Hence there is a path from (v, s) to (w, t) in I.

(If) Clearly, SEARCH-T(u, s_0) is called in Line 9 at some iteration of the loop starting at Line 8. A simple induction shows that since there is a path from (u, s_0) to (v, s), SEARCH-T(v, s) is called. Let p be a path of maximal length from (v, s) to (w, t) in I. The proof proceeds by induction on the length of p. If p is of length zero, the result follows from Line 13 in SEARCH-T. Assume that the hypothesis is true for paths of length less that n, $n \geq 1$, and that p is of length n. Let (x, q) be the successor of (v, s) on p. By the inductive hypothesis, $(w, t) \in R[x, q]$. We must show that $(w, t) \in R[v, s]$. The definition of I ensures that Lines 15 and 16 execute for the edge from (v, s) to (x, q) during SEARCH-T(v, s). There are two cases to consider, corresponding to whether or not q is in $PM[x]$ at Line 17.

Case 1: If $q \notin PM[x]$ at Line 17, then SEARCH-T(x, q) cannot have been called previously (because I is acyclic and previous markings are never removed). We want to show that, after SEARCH-T(x, q) is called, $(x, q) \notin BR[v]$ (so that Lines 19 and 20 will be executed). Since G is acyclic, no call of SEARCH-T that occurs before SEARCH-T(v, s) exits can involve v. Hence, $BR[v]$ can be altered only in SEARCH-T(v, s). It follows that, between Lines 15 and 21, $R[v, s] = BR[v]$. Before SEARCH-T(x, q) is called, $R[x, q] = \emptyset$ and (x, q) cannot appear in any $R[y, r]$ (by Lines 19 and 20); in particular, $(x, q) \notin R[v, s]$. Hence, $(x, q) \notin BR[v]$. This is still true after SEARCH-T(x, q) exits, since $BR[v]$ can be changed only in SEARCH-T(v, s). We conclude that Lines 19 and 20 are executed, and, since $(w, t) \in R[x, q]$, (w, t) is added to $R[v, s]$.

Case 2: If $q \in PM[x]$ at Line 17 and $(x, q) \notin BR[v]$, the result follows from Lines 19 and 20. If $q \in PM[x]$ and $(x, q) \in BR[v]$, then $(x, q) \in R[v, s]$. Node (x, q) can have been added to $R[v, s]$ only in Line 20. Since there is no path of length at least two from (v, s) to (x, q) (because p is of maximal length), (x, q) can only have been added to $R[v, s]$ when $R[x, q]$ was added to $R[v, s]$. Hence, $(w, t) \in R[v, s]$. □

Theorem 1 *Algorithm T is correct; that is, given query Q_R and acyclic db-graph G, Algorithm T adds (x, y) to $Q_R(G)$ if and only if (x, y) satisfies Q_R.*

Proof: Let $M = (S, \Sigma, \delta, s_0, F)$ be the NDFA accepting $L(R)$ constructed in Line 1 of Algorithm T and I be the intersection graph of G and M. If the algorithm adds (x, y) to $Q_R(G)$, then $(y, t) \in R[x, s_0]$ for some $t \in F$ (by Line 12). By Lemma 2, there is a (simple) path from (x, s_0) to (y, t) in I. Because $t \in F$, there is a simple path from x to y in G satisfying R; hence, (x, y) satisfies Q_R.

If (x, y) satisfies Q_R, then there is a simple path from x to y in G that satisfies R; hence, there is a simple path in I from (x, s_0) to (y, t), for some $t \in F$. Certainly SEARCH-T(x, s_0) is called at some stage during the execution of Line 9 of Algorithm T. By Lemma 2, $(y, t) \in R[x, s_0]$ after SEARCH-T(x, s_0) has executed. Since $t \in F$, Line 12 adds (x, y) to $Q_R(G)$. □

In order to analyze the complexity of Algorithm T, we introduce the concept of the reduction of an intersection graph I. The reduction of I can have an order of magnitude fewer edges than I. Assume that I is the intersection graph of db-graph G and NDFA M, where G has n nodes and M has q states. We show that Algorithm T does $O(qn)$ work for each edge in the reduction of I, while it does only $O(q)$ work for each edge of I not in its reduction.

Definition 6 Let $G = (N, E, \psi, \Sigma, \lambda)$ be a db-graph and $M = (S, \Sigma, \delta, s_0, F)$ be an NDFA accepting $L(R)$. Assume that we ignore the edge labels of the intersection graph I of G and M (although there may still be multiple edges between a pair of nodes in I). The *reduction* of I, written I_{red}, has the same set of nodes as I, and has an edge from (v, s) to (w, t) if and only if there is an edge from (v, s) to (w, t) in I and there is no path of length at least two from (v, s) to (w, t) in I; that is, there are no transitive edges in I_{red}. In

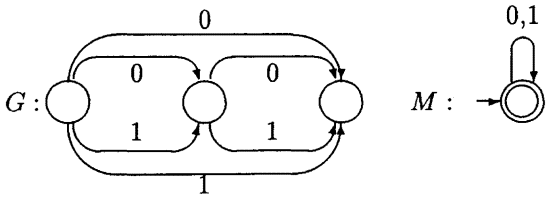

Figure 6: A db-graph G and an automaton M.

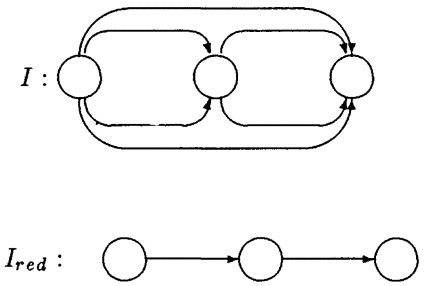

Figure 7: The intersection graph I and its reduction I_{red} for G and M of Figure 6.

addition, there is at most one edge between any pair of nodes in I_{red}. The set of edges in I_{red} is denoted $E_{I_{red}}$. □

Example 3 Consider the db-graph G and automaton M shown in Figure 6. The intersection graph I for G and M (ignoring edge labels) and its reduction, I_{red}, are given in Figure 7. Generalizing this example, assume that I has n nodes, such that each node v is connected by two edges to every node greater than v. Then I has $n(n-1)$ edges, while I_{red} has only $n-1$ edges. □

Theorem 2 *Let G be an acyclic db-graph comprising n nodes and e edges, and let Q_R be a query where R is a regular expression of length q. In addition, let I be the intersection graph of G and M, where M is the NDFA accepting $L(R) - \{\epsilon\}$ constructed in Line 1 of Algorithm T. Let i denote the number of edges in I_{red} and i^* be the number of edges in the transitive closure of I_{red}. Given Q_R and G as input, Algorithm T runs in $O(qn^2 + q^3e + qni + i^*)$ time.*

Proof: Line 1 of Algorithm T takes $O(q^2)$ time to construct the NDFA M which has $O(q)$ states (in fact, $|S| \leq 2|R|$ [1]). Setting up the adjacency lists for G takes $O(n+e)$ time, as does the topological sort and the sorting of nodes in the adjacency lists if a bucket sort is used. Line 4 takes $O(1)$ time, while Line 5 takes $O(n)$ time, and Line 6 takes $O(qn)$ time. Because $BR[v]$ is a bit vector of size $O(qn)$, Line 7 takes $O(qn^2)$ time.

Next, we consider the time taken in a single call of SEARCH-T. Lines 13, 14 and 21 can all be done in constant time ($PM[v]$ and $\{s\}$ are disjoint). Ignoring the recursive call to SEARCH-T, Line 17 takes $O(q)$ time. The set membership tests in Lines 18 and 20 can be done in $O(1)$ time, since $BR[v]$ is a bit vector. Thus the total time taken in Lines 19 and 20 is $O(qn)$. We show next that if Lines 19 and 20 are executed for an edge e, then $e \in E_{I_{red}}$.

Assume that SEARCH-T(v,s) has just been called and that $((v,s),(w,t))$ is not in $E_{I_{red}}$. Then there is a node (x,q) in I such that $((v,s),(x,q))$ is in $E_{I_{red}}$ and there is a path from (x,q) to (w,t) in I. Hence, the position of x in the topological ordering of the nodes of G is less than that of w, so x is considered before w in Line 15. By Lemma 2, $(w,t) \in R[x,q]$ and hence $t \in PM[w]$. So SEARCH-T is not called at Line 17. Since (x,q) is on a path of maximal length from (v,s) to (w,t), $R[x,q]$ must have been added to $R[v,s]$ and $BR[v]$ (by Lemma 2). Hence, $(w,t) \in BR[v]$ at Line 18, so Lines 19 and 20 are not executed for the edge $((v,s),(w,t))$.

It is not hard to see that, over all calls of SEARCH-T, each edge in I is considered at most once. From the preceding paragraphs, we conclude that $O(q)$ time is spent on each edge in $E_I - E_{I_{red}}$ and $O(qn)$ time on each edge in $E_{I_{red}}$. Since there can be $O(q^2e)$ edges in $E_I - E_{I_{red}}$, these two components take $O(q^3e + qni)$ time. Turning to Line 22, we note that, by Lemma 2, (w,t) is added to $R[v,s]$ only if (w,t) is reachable from (v,s) in I. Hence, over all calls of SEARCH-T, Line 22 takes $O(i^*)$ time.

Returning to the main program, Line 10 takes $O(n)$ time. Line 12 can be done in constant time so that Lines 11 and 12 together require $O(qn)$ time. Over all iterations of the loop at Line 8, Lines 10 to 12 take $O(qn^2)$ time. We conclude that, assuming there is at least one edge in G, Algorithm T takes $O(qn^2 + q^3e + qni + i^*)$ time. □

There can be $O(q^2e)$ edges in I_{red}, so the complexity of the algorithm remains $O(q^3ne)$ in the worst case, although, even if there are $O(n^2)$ edges in G, it can be as good as $O(q^2n^2)$. Below we give examples of both situations while demonstrating the execution of Algorithm T.

Example 4 It turns out that examples of both extremes of the complexity behaviour exhibited by Algorithm T can be constructed from *undirected* graphs of the form $K_{m,m}$, that is, complete bipartite graphs with the same number of nodes in each partition. Let $I = (V_1, V_2, E)$ be an intersection graph of this type. Assume that I has $n = 2m$ nodes and e edges. Let E_{red} denote the set of edges in the reduction of I and e_{red}

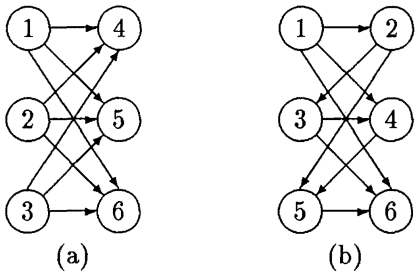

Figure 8: Graphs demonstrating (a) worst-case, and (b) best-case behaviour of Algorithm T.

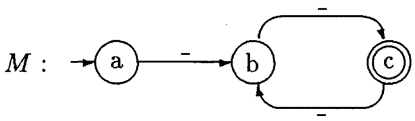

Figure 9: An NDFA M accepting $L(R)$, where $R = (\_\_)^*$.

denote the number of such edges. For (worst) case (a), where $e_{red} = e = \Theta(n^2)$, I is defined by $V_1 = \{1, \ldots, m\}$, $V_2 = \{m+1, \ldots, n\}$ and $E = \{(i,j) | i \leq m < j\}$. The graph for $m = 3$ is shown in Figure 8(a). For (best) case (b), where $e = \Theta(n^2)$ while $e_{red} = \Theta(n)$, I is defined by $V_1 = \{1, 3, \ldots, n-1\}$, $V_2 = \{2, 4, \ldots, n\}$ and $E = \{(i, i+2j+1) | 1 \leq i \leq n-1, 0 \leq j \leq (n-i-1)/2\}$. Then $E_{red} = \{(i, i+1) | 1 \leq i \leq n-1\}$. The graph for $m = 3$ is shown in Figure 8(b).

Assume that we are given the graph of Figure 8(b) as a db-graph G along with a query Q_R, asking for nodes connected by paths of even length, that is, $R = (\_\_)^*$ (where the underscore denotes any symbol in Σ). An NDFA M accepting $L(R) - \{\epsilon\}$ is shown in Figure 9, while the intersection graph I of G and M is depicted in Figure 10. The significance of the solid and dotted edges in Figure 10 will be explained below.

We will describe how Algorithm T evaluates Q_R on G in terms of its (implicit) traversal of the intersection graph I of Figure 10. The algorithm begins searching from node $(6, a)$, that is, SEARCH-T$(6, a)$ is called first in Line 9. Node $(6, a)$ is added to $R[6, a]$ and $BR[6]$ in Lines 13

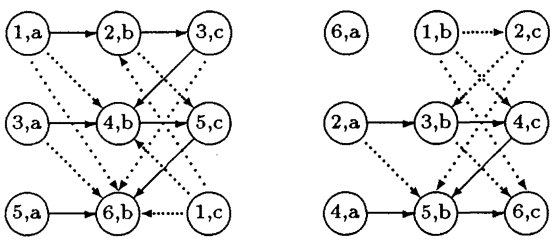

Figure 10: The intersection graph of the db-graph of Figure 8(b) and M of Figure 9.

and 14, respectively, and since no edges leave $(6, a)$ in I, the loop at Line 15 does not execute. So a is added to $PM[6]$ (Line 21) and $(6, a)$ is removed from $BR[6]$ (Line 22), thereby restoring it to $\emptyset$.

Next, SEARCH-T$(5, a)$ is called in Line 9. Since there is an edge from $(5, a)$ to $(6, b)$ in I and $b \notin PM[6]$, SEARCH-T$(6, b)$ is called in Line 17. Again, no edges leave $(6, b)$ in I, so b is added to $PM[6]$ and control returns to Line 18. Node $(6, b) \notin BR[5]$ (since $BR[5] = \{(5, a)\}$), so all of $R[6, b]$ (namely $\{(6, b)\}$) is added to $R[5, a]$ which now becomes $\{(5, a), (6, b)\}$. No other edges leave $(5, a)$; hence, a is added to $PM[5]$ and the procedure exits.

Searching from $(4, a)$ leaves $R[6, c] = \{(6, c)\}$, $R[5, b] = \{(5, b), (6, c)\}$, and $R[4, a] = \{(4, a), (5, b), (6, c)\}$. Because c is a final state, the edge $(4, 6)$ is added to $Q_R(G)$ in Line 12. Now consider searching from $(3, a)$. Two edges leave $(3, a)$ in I, but Line 15 ensures that the edge to $(4, b)$ is traversed first. SEARCH-T$(4, b)$ in turn calls SEARCH-T$(5, c)$ during which it is discovered at Line 17 that $b \in PM[6]$. Consequently, SEARCH-T$(6, b)$ is not called. However, $(6, b) \notin BR[5]$ (since $BR[5] = \{(5, c)\}$) so all of $R[6, b]$, namely $\{(6, b)\}$, is added to $R[5, c]$ (Lines 19 and 20). On backtracking to $(4, b)$, the algorithm finds that $(5, c) \notin BR[4]$ at Line 18, so $R[5, c] = \{(5, c), (6, b)\}$ is added to $R[4, b]$. The algorithm then backtracks to $(3, a)$, where $R[4, b]$ is added to $R[3, a]$ in Lines 19 and 20. Now, on considering the edge from $(3, a)$ to $(6, b)$, it is discovered that not only is $b \in PM[6]$ but $(6, b)$ is already in $BR[3]$; hence, Lines 19 and 20 are not executed. Since $(5, c) \in R[3, a]$, edge $(3, 5)$ is added to $Q_R(G)$.

The traversals from nodes $(2, a)$ and $(1, a)$ proceed similarly, adding $(2, 4)$, $(2, 6)$, $(1, 3)$ and $(1, 5)$ to $Q_R(G)$. The solid edges in Figure 10 represent those edges for which Lines 19 and 20 of Algorithm T are executed, while the dotted edges denote edges for which Lines 19 and 20 are not executed (and which are possibly not traversed at all, such as the edges from $(1, b)$, $(1, c)$ and $(2, c)$). From the above complexity analysis, we conclude that Algorithm T does at most $O(q)$ work for each of the dotted edges and at most $O(qn)$ work for each of the solid edges. It can be seen that the solid edges in Figure 10 are actually a proper subset of E_{red}. □

It would be interesting to test empirically whether the theoretical benefits of Algorithm T presented above actually give rise to performance gains in practice. This, of course, also depends on the extent to which graphs encountered in applications exceed their reductions in size.

If we modify Algorithm T to construct an intersection graph explicitly, then its applicability

is no longer restricted to acyclic db-graphs alone. The algorithm would then be appropriate whenever the *intersection* graph of a db-graph G and the transition graph T of an NDFA was acyclic. A sufficient condition for this is that either G or T is acyclic. Hence, the modified Algorithm T would also apply to cyclic db-graphs when the given regular expression contained no closure operator.

4 Conclusions

We have described a limited subset of a recursive query language, called $\mathbf{G^+}$, which is described more fully elsewhere [6,7]. In particular, we presented an efficient query evaluation algorithm for this subset of the language when applied to acyclic database graphs. Algorithms for more general query evaluation, which are not as efficient as the algorithm described here (and indeed are sometimes compelled to run in exponential time), are considered in [7,12]. Application areas for $\mathbf{G^+}$ include knowledge-based systems, in particular semantic or conceptual networks, CAD/CAM systems, and Hypertext systems.

An implementation of $\mathbf{G^+}$, in which graphs are assumed to fit in main memory, is described in [5]. Future research involves the investigation of how these algorithms can be adapted to run efficiently when graphs are kept on secondary storage. Transitive closure algorithms for this case are investigated in [10], although they do not appear to have incorporated the ideas of [11] as we have done, and therefore do not derive the benefit of improved efficiency for certain acyclic graphs. Our query evaluation algorithm is concerned only with finding all pairs of nodes connected by a path satisfying a given regular expression. A single-source query is one in which one endpoint of the path is fixed. Another topic for research is to investigate whether the algorithm can be adapted to process single-source queries efficiently as well.

References

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

[2] A.V. Aho and J.D. Ullman, [1979], Universality of Data Retrieval Languages, *Proceedings of 6th ACM Symposium on Principles of Programming Languages*, ACM Press, New York, 110–120.

[3] F. Bancilhon and R. Ramakrishnan, [1986], An Amateur's Introduction to Recursive Query Processing Strategies, *Proceedings of ACM SIGMOD Conference on Management of Data*, ACM Press, New York, 16–52.

[4] M. Brodie and M. Jarke, [1984], On Integrating Logic Programming and Databases, *Proceedings of 1st International Workshop on Expert Database Systems*, Benjamin-Cummings, Menlo Park, CA, 40–62.

[5] M. Consens, [1988], Query Processing in a Graph-Based Language, M.Sc. Thesis, University of Toronto.

[6] I.F. Cruz, A.O. Mendelzon, and P.T. Wood, [1987], A Graphical Query Language Supporting Recursion, *Proceedings of ACM SIGMOD Conference on Management of Data*, ACM Press, New York, 323–330.

[7] I.F. Cruz, A.O. Mendelzon, and P.T. Wood, [1988], $\mathbf{G^+}$: Recursive Queries Without Recursion, *Proceedings of 2nd International Conference on Expert Database Systems*, Benjamin-Cummings, Menlo Park, CA, 355–368.

[8] J.E. Hopcroft and J.D. Ullman, [1979], *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA.

[9] H.B. Hunt, D.J. Rosenkrantz, and T.G. Szymanski, [1976], On the Equivalence, Containment, and Covering Problems for the Regular and Context-free Languages, *J. Comput. Syst. Sci.*, **12**, 222–268.

[10] Y.E. Ioannidis and R. Ramakrishnan, [1988], Efficient Transitive Closure Algorithms, *Proceedings of 14th International Conference on Very Large Data Bases*, Morgan Kaufmann, Palo Alto, CA, 382–394.

[11] K. Mehlhorn, [1984], *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, EATCS Monographs in Theoretical Computer Science, Springer-Verlag, Berlin.

[12] A.O. Mendelzon and P.T. Wood, [1989], Finding Regular Simple Paths in Graph Databases, *Proceedings of 15th International Conference on Very Large Data Bases*, Morgan Kaufmann, Palo Alto, CA, 185–193.

[13] J.D. Ullman, [1985], Implementation of Logical Query Languages for Databases, *ACM Trans. on Database Syst.*, **10**, 289–321.

Computers and the Law

Submitted by Antony Cooper
CSIR

The SA Law Commission has established a commission on "The Legal Protection of Information".

The commission is still in its preliminary stages and the assigned researcher, Mr Herman Smuts, is still preparing the working paper. He does not know when it will be finished, but once the working paper has been prepared, they will invite comments for about two years, before preparing the final report. I have contacted Mr Smuts, and he would be most grateful to receive input at this stage, especially regarding the terms of reference of the commission. His address is:

C/o SA Law Commission
Private Bag X668
PRETORIA
0001

In addition, there is an ad-hoc committee at the Registrar of Copyright investigating numerous copyright issues, including those relating to software and data. Mr Smuts' commission will be liaising with the ad-hoc committee.

I feel that SAICS has an obligation to submit evidence to the commission, and I would appreciate it if you would circulate the members of the Council of SAICS, and perhaps the general membership as well, to solicit ideas concerning SAICS's input.

I shall prepare something for the commission, either in my personal capacity, or in my professional capacity here at CSIR. I would be willing to assist in the preparation of any evidence SAICS might submit.

4th National MSc/Phd Computer Science Conference

Report by Danie Behr
University of Pretoria

This conference was held from 7th to 10th September 1989 at the Cathedral Peak Hotel in the Drakensberg. The conference was attended by 61 postgraduate students from 11 South African universities. Most were engaged in MSc studies, although 5 Phd students also attended. These numbers are encouraging for the South African computer science community. This type of conference is rather unique in that it affords students the opportunity of sharing their research, and getting to know other researchers in the country. The number of Afrikaans and English speaking students attending the conference were roughly equal. Presentations were made in the language preferred by the student. Invitations were sent to all universities with computer science departments. The conference was organized by the students themselves.

Some of the more popular research topics that were presented included expert systems, data communications, computer security, graphics, software engineering, user interfaces and data bases. The main sponsor for this year's conference was the Division for Microelectronic Systems and Communication Technology of the CSIR. The conference was opened with an interesting talk on the myths and motivations of post graduate studies by Prof DG Kourie, acting head of the Computer Science Department at Pretoria University.

The next conference will be presented by the University of Port Elizabeth. People requiring further information about the next conference should contact Andre Calitz, Charmaine du Plessis or Jean Greyling of the Department of Computer Science at UPE.

A list of authors and papers presented at the symposium follows:

S Crosby, University of Stellenbosch
Performance Analysis of Wide Area Computer Communication Networks

A B Joubert, PU for CHE Vaal Triangle Campus
Image Processing Libraries

A Calitz, University of Port Elizabeth
An Expert System Toolbox to assist in the classification of objects

L von Backström, University of Pretoria
Integrated Network Management

R Foss, Rhodes University
The Rhodes Computer Music Network

A McGee, University of Natal
On Fixpoints and Nondeterminism in the Sigma-Lambda Calculus

P G Mulder, Randse Afrikaanse Universiteit
A Formal Language and Automota approach to Data Communications

A Tew, Randse Afrikaanse Universiteit
Drie dimensionele grafiek grammatikas

T C Parker-Nance, University of Port Elizabeth
Human-Computer Interaction: What Determines Computer Acceptance

E Coetzee, PU vir CHO Vaaldriehoekkampus
Opsporing van rande in syferbeelde dmv verskerping en drempelbepaling
D A Sewry, Rhodes University
Visual Programming
A Cooper, University of Pretoria
Improvements to the National Exchange Standard
E S Badier, University of Port Elizabeth
A Computer Assisted Diagnostic System (CADS)
C du Plessis, Universiteit van Port Elizabeth
Persoonsidentifikasie dmv naampassing in 'n genealogiese databasis
J Greeff, University of Stellenbosch
The Entity-Relationship Model and its Implementation
D A de Waal, PU vir CHO
Flat Concurrent Prolog (FCP) en Flat Guarded Horn Clauses (FGHC): 'n Vergelyking
E Naude, UNISA
Interne metodes in Liniêre Programmering
A Deacon, University of Stellenbosch
Global consistency in non-locking DDBMS
A Wilks, Rhodes University
The Synchronisation and Remote Configuration of the Resources in a Computer Music Network
J Greyling, University of Port Elizabeth
The design of a User Interface with special reference to an Interactive Molecular Modelling Program
L Drevin, PU vir CHO
Rekenaarsekuriteit: Verskillende vlakke van kontrole
Dieter C Barnard, University of Stellenbosch
The design and implementation of a modest, interactive proof checker
R A Schmidt, University of Cape Town
Knowledge Representation Systems and the Algebra of Relations
J Hartman, Randse Afrikaanse Universiteit
Die Gebruik van Objek-georiënteerde Programmering in die Moderne Sneltrein Omgewing
S Lawrie, Rhodes University
The Design and Implementation of a System for the Interactive Control of a MIDI-based Studio
E Mulder, Rand Afrikaans University
A Formalisation of Object-Oriented Principles
C J Tolmie, UOVS
Die Ontwikkeling van 'n Ekspertrekenaarstelsel vir die beoordeling van die resultate van die Technicon H1-Bloedselanaliseerder
R Breedt, University of Pretoria
Realism with Ray Tracing
J van Jaarsveld, University of Pretoria
Developing Medical Expert Systems: A knowledge acquisition perspective
W Appel, University of Pretoria
TCP/IP Implementation on Ethernet
E Goedeke, University of Natal
Eggspert's Control Structure
M Harmse, University of Stellenbosch
Modelling of I/O Subsystems
H L Viktor, University of Stellenbosch
A Quantitative Model for Comparing Recovery Techniques in a Distributed Database
M Olivier, Randse Afrikaanse Universiteit
Rekenaarvirusse in Suid-Afrika

Book Reviews

An Introduction to Functional Programming Through Lambda Calculus
by Greg Michaelson, Addison-Wesley, 1988.
Reviewer: Dr. E P Wentworth, Rhodes University

Recently we have seen a number of excellent *second generation* texts on Functional Programming. Michaelson's text assumes some previous programming experience with imperative languages, and presents the functional approach as an alternative paradigm. He begins with a very accessible exposition of the Lambda Calculus, and carefully develops this foundation to encompass the important aspects and paradigms of functional programming. The programming notation is language-independent, although the last chapters are devoted to a brief look at two specific languages, Standard ML and Lisp. The examples and exercises are mainly utility in nature, e.g. "insert a sublist after the first occurrence of another sublist in a list", and can generally be solved in a couple of lines. Answers to the exercises are provided in an appendix.

The approach is slanted towards developing a solid base for understanding functional languages and computing. In this respect the book achieves a good balance between the theoretical underpinnings and their practical application. On the practical side, however, I found the lack of more substantial examples and exercises disappointing. Most programming texts tackle a set of 'standard' problems which are well-understood in the academic community and provide an informal benchmark for comparisons. Since the book is targeted for those already versed in imperative languages and standard algorithms, one might expect the examples to clearly demonstrate the elegance and power of the *problem-oriented* functional approach in these areas. Having laid an excellent foundation I was left with the feeling that the book failed to capitalize and deliver the cherry on the top.

The book is highly recommended as one of the new breed of Computer Science books which gives substantial attention to the fundamentals of the subject without becoming bogged down in over-rigorous formality.

Artificial Intelligence and the Design of Expert Systems
by George F Luger & William A Stubblefield, The Benjamin/Cummings Publishing Co., 1989.
Artificial Intelligence: A Knowledge-based Approach
by Morris W Firebaugh, PWS-Kent Publishing Co., 1989.
Reviewer: Prof G D Oosthuizen, University of Pretoria

One of the primary goals of an Honours course is to introduce students to a field in such a way that they arrive at enough insight into relevant issues to enable them to conduct further research on their own. To this end a text book which is used ought to reflect the current view of the field. Because of the rapid expansion of the field of Artificial Intelligence (AI), we have now finally outgrown the era dominated by the books by Winston and Charniak and McDermott. In the past five to ten years much new work has been done, and new insights have been gained. Introducing AI, therefore, requires a marked shift from the previous emphasis on a few historical systems embodying a number of famous methods, to a more generic approach - an approach which highlights those fundamental representation and search models that span all the different application areas and strategies of problem solving. Of course, since AI still does not have a well developed theory, references to seminal systems continues to fulfil an important role.

Both of the above books are good text books, characterised by a balanced coverage of Prolog and Lisp. They also reflect and consolidate much of the work of the past few years done in areas such as knowledge representation, machine learning, the work done under the heading of Expert Systems and even the recent work on neural networks. But the most important feature that they share is the accurate and up to date overall picture of the subject provided; the broad framework for the understanding of AI that is created without neglecting work of historical importance. There are still references to these works, but they are placed in perspective in relation to new developments.

The book of Luger & Stubblefield (L&S) is more language oriented than Firebaugh's book. A characteristic of L&S is that AI approaches to representation are related to the Object Oriented approach. Whereas L&S includes chapters on advanced AI programming techniques in Prolog and Lisp, it does not address pattern recognition, computer vision and robotics. (Firebaugh has chapters on each of these themes.) These omissions are understandable, since AI has diversified so much recently that it is difficult to cover all applications in one book.

If I had to select one of the books, it would be L&S. Although L&S gives poor coverage of Machine Learning, the book's overall presentation is very good. In particular, the chapters are well-organised, and the overall approach to AI - starting with the core aspects of *representation* and *search*, followed by chapters on AI languages - is coherent. The authors also make very good use of graphical representations and illustrations to convey ideas.

Books Received

The following books have been sent to SACJ. Anyone willing to review a book should contact the editor. The book will be sent to him for review, and may be kept provided that a review is received.

- D Bustard, J Elder & J Welsh, [1988], *Concurrent Program Structures*, Prentice-Hall Inc., Englewood Cliffs.
- R Cafolla & A D Kauffman, [1988], *Turbo Prolog Step by Step*, Merrill Publishing Company, Columbus, Ohio.
- S Hekmatpour, [1988], *Introduction to LISP and Symbol Manipulation*, Prentice-Hall Inc., Englewood Cliffs.
- K L Clark & F G McCabe, [1984], *micro-PROLOG: Programming in Logic*, Prentice-Hall Inc., Englewood Cliffs.
- D Crookes, [1988], *Introduction to Programming in Prolog*, Prentice-Hall Inc., Englewood Cliffs.
- M J C Gordon, [1988], *Programming Language Theory and its Implementation*, Prentice-Hall Inc., Englewood Cliffs.
- J G Hughes, [1988], *Database Technology : A software engineering approach*, Prentice-Hall Inc., Englewood Cliffs.
- R Milner, [1989], *Communication and Concurrency*, Prentice-Hall Inc., Englewood Cliffs.
- T J Myers, [1988], *Equations, Models and Programs*, Prentice-Hall, Inc., Englewood Cliffs.
- N C Rowe, [1988], *Artificial Intelligence through Prolog*, Prentice-Hall Inc., Englewood Cliffs.
- D A Protopapas, [1988], *Microcomputer Hardware Design*, Prentice-Hall Inc., Englewood Cliffs.
- H Eisner, [1988], *Computer-aided Systems Engineering*, Prentice-Hall Inc., Englewood Cliffs.
- S H Unger, [1989], *The essence of logic circuits*, Prentice-Hall Inc., Englewood Cliffs.
- R J Young, [1989], *Practical Prolog*, Van Nostrand Reinhold, New York.

NOTES FOR CONTRIBUTORS

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a 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 double-space typing on one side only of A4 paper, and provide wide margins.
- 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 on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin. Distinguish clearly between such cases as:
  - upper and lower case letters;
  - the letter O and zero;
  - the letter I and the number one; and
  - the letter K and kappa.
- 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. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may provided in one of the following three formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies.

Charges

A charge per final page, scaled to reflect scanning, typesetting and reproduction costs, will be levied on papers accepted for publication. The costs per final page are as follows:

Typed format:	R80-00
ASCII format:	R60-00
Camera-ready format :	R20-00

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

Proofs

Proofs of accepted papers 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 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.

South African Computer Journal

Number 1, January 1990
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 1, Januarie 1990
ISSN 1015-7999

Contents