This is the new access floor at the David Jones Department Store Computer Centre in Sydney.

It uses Series 800 metal panels, with a high pressure laminate surface finish, which support a NCR Criterion 8550 with 4 tape drives, 4 disc packs, 1 printer, 1 card reader and 1 console. Special features include vibration dampeners under the pedestal feet. Cemac Tate's 701 Snaplock stringer system supports the load comfortably.

Once again, Cemac Interiors had single source responsibility for the whole project. In fact, a section of warehouse was transformed to build the computer room.

Cemac Tate’s all-metal access floor system offers four panel sizes, with or without stringers, and the widest range of finishes — high pressure laminate, wood parquet, cork and vinyl asbestos — as well as antistatic carpet.

Cemac also has a patented, adjustable pedestal base which ensures a level floor on a sloping slab. So, for access flooring, or complete office interior layout (floors, systems furniture, ceilings, partitions and task or ambient lighting), call Cemac Interiors.

Brochures and details from Cemac Interiors.

Sydney . . . . . . . . . . 6993122
Melbourne . . . . . . 4198233
Brisbane . . . . . . . . . . 2215699

Licensees:
Adelaide . . . . . . . . . . 453656
Hobart . . . . . . . . . . . . . 295444
Perth . . . . . . . . . . . . . 4447888
“News Briefs from the Computer World” is a regular feature which covers local and overseas developments in the computer industry including new products, interesting techniques, newsworthy projects and other topical events of interest.

8TH WORLD COMPUTER CONGRESS

Members of the Australian Organising Committee for the 8th World Computer Congress to be held in Tokyo and Melbourne this year returned from a visit to Tokyo recently full of praise for the Congress facilities in Japan.

Mr Ashley Goldsworthy, the Chairman of the Australian Congress Organising Committee, together with Mr Geoff Donnelly, the Vice-Chairman of the Australian Committee, and Professor Tony Montgomery, Chairman of the Congress Facilities Sub-Committee, visited Japan for discussions with their Japanese counterparts, and to inspect the venue being used for the Japanese segment of the Congress.

A similar delegation from Japan visited Australia some 12 months ago for much the same purpose.

Mr Goldsworthy said that the venue in Tokyo was a new complex, Sunshine City, which was completed in April of this year. Sunshine City comprises a towering office block of 60 storeys, a new hotel of 38 storeys, completed in April, a culture centre and a wide range of shops, restaurants and other attractions.

The Congress will be held in theatres and conference rooms in this centre. Covering an area of some 60,000 square metres, Sunshine City brings together cultural and recreational facilities, shopping and business establishments, parks and plazas and an international atmosphere, to create a complete living and working environment.

Ashley Goldsworthy said that Sunshine 60 is Japan’s — and Asia’s — largest office building.

THE YEAR OF THE CONGRESS

Australian computer practitioners, in a fashion reminiscent of traditional Chinese practice, are being urged to regard 1980 as the Year of the Congress. The reason being, of course, that in October this year the 8th World Computer Congress will be held in Melbourne. The first week of the Congress will be held in Tokyo from October 6 through 9 and the second week in Melbourne from October 14 through 17. As Ashley Goldsworthy, the Chairman of the Australian Organising Committee pointed out, this is the first time in the history of IFIP (The International Federation for Information Processing) that their triennial Congress has been held in the southern hemisphere.

Australian and Japanese Organising Committees have been planning the Congress since 1974. It was in that year that the General Assembly of IFIP selected Australia and Japan as the site of the 1980 Congress.

It is also the first time in IFIP history that the Congress has been held at two sites. In the past the Congress has always been held at one location. The splitting of the Congress has naturally created many additional problems for the organisers. However, according to Ashley Goldsworthy, arrangements have proceeded very smoothly and plans are well advanced in all facets of the Congress preparations.

In addition to the Congress there will be an exhibition of computer equipment and related services in both Tokyo and Melbourne. The Australian Exhibition has exceeded all expectations. Ashley Goldsworthy said that the original plan had been to have an exhibition covering an area of some 60,000 sq. ft. This was booked out immediately on release and another 30,000 sq. ft. of space...
was offered. This was also immediately snapped up by exhibitors and a further 30,000 sq. ft. was then offered. As Ashley Goldsworthy pointed out even the most optimistic of the organisers were amazed when this was easily taken by exhibitors. He said that the keenness and enthusiasm of suppliers and others to take part in the exhibition had been very encouraging and gratifying for the organisers. It had given them even greater confidence in the ultimate success of the Congress.

Computer conferences inevitably tend to have a certain sameness about them. However, the 8th World Computer Congress promises to be something different from what most Australians in the past will have had an opportunity to participate in.

AUSTRALIAN COMPUTER FOR U.S.
An Australian designed and built general purpose minicomputer was installed in the United States' engineering plant in April.

The machine is the Spectrum-11 D ($9500 Aust.), a dual-sided double density floppy disk computer designed and manufactured by D.D. Webster Electronics Pty. Ltd. of Bayswater, Victoria.

The Spectrum was installed at Dynatron Inc., in Wallingford, Connecticut, a company which manufactures energy conservation systems for both U.S. and world markets.

It will administer the Dynatron bookkeeping using software created by another Australian company, Tri-technic Pty. Ltd., of South Yarra.

The Chief Executives of the respective companies, Mr David Webster and Mr Ray Moloney, personally supervised the installation which they believed was the first Australian computing system to operate in a U.S. business site.

NSF PROVIDES IFIP CONGRESS '80 TRAVEL GRANTS THROUGH AFIPS
The American Federation of Information Processing Societies (AFIPS) is pleased to announce a competitive program of travel grants to assist qualified United States scientists in attending the Eighth World Computer Congress (IFIP '80) supported by a grant to AFIPS from the National Science Foundation. The Congress will be held in two locations: Tokyo, Japan, October 6-9, and Melbourne, Australia, October 14-17.

The grants will provide up to $1000 support to individuals attending either the Tokyo or the Melbourne portion of the Congress (not both). Applicants should supply a letter including the following information: (1) specific Congress site for which support is requested (Melbourne or Tokyo); (2) approximate dates of travel and point of departure; (3) participation in activities of the Congress; (4) expected benefits from attendance at the Congress; (5) a statement indicating willingness to supply a post-Congress report to AFIPS; (6) sources of support for the trip; (7) brief vita. Additionally, applicants are encouraged to have three letters of reference submitted in support of their application.

Applications should be addressed to Nancy LeFebvre at AFIPS, 1815 North Lynn Street, Suite 800, Arlington, Virginia 22209, Attention: IFIP Congress '80 Travel Grants. All materials must be received by May 30, 1980.

Announcement of awards will be made by approximately June 30, 1980. Payment to successful applicants will be made by cheque prior to the Congress. Each grant recipient will be required to supply a letter detailing the benefits derived by attendance at the Congress within 30 days of return.

Applicants for this program must be US citizens, and there will be no discrimination based on race, sex or national origin in the administration of the program. Travel under this program must follow NSF policies and will normally involve transportation on US flag carriers. It is expected that travel grant recipients will wish to take advantage of low cost group travel arranged by Garber Travel, the official US travel agent (800-225-4570).

Additional information of the Congress is available from AFIPS, 1815 North Lynn Street, Suite 800, Arlington, Virginia 22209. AFIPS serves as the official US representative to IFIP.

ICL CAFS GOES TO TOP DIAMOND COMPANY
The world's largest rough diamond marketing organisation, De Beers' associated Central Selling Organisation, has signed the first order for ICL's Content Addressable File Store (CAFS).

CAFS is a new and revolutionary method of selecting and retrieving information at high speeds from computer files. It enables complex searches of large data files to be completed in a few seconds and gives ICL a world lead in computerised information retrieval techniques. It is now being marketed world-wide under the name CAFS 800.

Trials carried out on this new ICL development show that it operates 50 times faster than existing software-based information retrieval systems and can handle up to 10 times as many simultaneous enquiries.

In addition, an information retrieval system based on CAFS 800 will be up to 50 per cent cheaper to set up.

The ICL CAFS 800 subsystem will be installed at De Beers' London headquarters early in 1981. It will work in conjunction with their ICL 2960 computer and will provide senior management with a sophisticated information service.

De Beers markets rough diamonds from most of the world’s major producers to the main diamond manufacturing centres. Computer models are used to help in forecasting trends and planning.

Harry Pennington, Head of De Beers’ Management Information, said: “I am convinced that CAFS 800 will provide a very fast and easy-to-use method of information retrieval.”

CAFS was originated at ICL's Research and Advanced Development Centre in Stevenage. The hardware for the ICL CAFS 800 subsystem is being manufactured at ICL's factories in Letchworth, and the software is produced at ICL's Development Centre in Bracknell.
Editorial
A Short Essay on Some Problems with Teaching of Computing in Australian Government Institutions Today

There is a crisis currently in Computing Education in Australia and ACS members should be aware of it. This crisis stems from two sources. The teacher of computing is starved of funds by his fellow academic "colleagues" and his case for improved funding is not supported by those involved in the commercial computing sphere.

On the first count, academics from the traditional science disciplines, in particular the physical sciences, still view Computer Science as not being a proper discipline at all or, if they are prepared to accept that there is now a Computer Science, then they assert that the subject is so fragmentary and of such small scope that it is a discipline scarcely worthy of serious intellectual endeavour! To be more specific, most academics from non-Computer Science disciplines make three gross generalisations pertaining to the study of computing in its own right. Firstly, that a computer is merely a tool, secondly, that the kind of basic material to be understood is really quite small, and thirdly, that all so-called Computer Science material is quite special and evolving.

There is insufficient space here to deny these beliefs in detail but we submit that all three assertions are incorrect.

Now, one major manifestation of the lack of acceptance of this "new" discipline by fellow academics is impoverishment of funding. Since their inception 10 or more years ago, Computer Science Departments have been starved of funds; funds for teaching staff, support staff (secretarial, programming and engineering) and funds for equipment. The situation can be summarised by a few case examples:

1. At one university, current total funding for the Computer Science Department is four per cent of the funds of the Faculty of Science, but the Computer Science Department teaches 12 per cent of the students in that faculty.

2. At another university, the ratio of students to staff in Engineering and Physics is 6:1, but in the Computer Science Department, 18:1. In the same university the same ratio for the Department of Russian is about 0.6:1.

3. At another university, the aggregate amortised equipment funding level per student for the traditional Physical Science Departments is estimated to be about five times that for Computer Science. This impoverished equipment funding situation has been reached by inequitable funding formulae over many years.

In many Institutes of Technology similar disparities of funds allocation between Computer Science and other Physical Sciences exist.

If we assume that Computing is no less difficult to keep abreast of than the other physical sciences — and there are those who would assert that it is more difficult — then several conclusions, inevitably, have flowed from these pauperised circumstances.

1. Unless Computing staff members work at least three times as hard as their fellows, students of Computing receive about one-third the teaching attention and assistance available to those in other Physical Science disciplines.

2. Staff have little time for self study and even less for research work or industrial consulting, which latter are two major reasons for selecting an academic life in the first place.

3. Because their research output is poor, their promotion prospects are much diminished (since research output is a prime criterion for promotion selection).

4. Academics become disillusioned and after developing marketable skills, leave for a lucrative industrial life.

This latter factor has become so prevalent in the USA that there are 12 academic positions available for each person prepared to take such a post.

In addition, all Australian Computer Science Departments are funded so that they must keep a predominance of junior positions with insufficient posts at the higher levels to make promotion reasonably attainable. Compare this with the situation in some Physical Science Departments where 95 per cent of staff positions are at Professor, Associate Professor, Reader or Senior Lecturer level; there are no lecturers and no tutors in some such Departments!

5. With no time for self study and little up-to-date equipment, students are being trained in outmoded techniques, by tired and untrained staff, in some cases drastically so.

This lack of funding then is the first major facet of the crisis.

Turning now to consider the attitude of the practitioner to the activities of the teacher of Computing. It seems to be generally held that the academic life is one of considerable leisure with at least 13 weeks holiday per year and other perquisites, of a kind unspecified, but quite clearly extremely desirable, including the pot of gold at the end of the rainbow, external studies leave. The fact that university academics, for example, work from 9am to 5pm, are entitled to at most four weeks paid vacation per year and that, whilst students are not present at the university, academic staff members have their only opportunity to indulge in their research work, seems to be
little understood. How many practitioners would like to have their promotion and reputation measured by work which is carried out in (not entirely) full time effort during but nine weeks in the year? How many practitioners would like to be so measured when their funds for attendance at conferences and purchase of books amounts to $140 per staff member per annum, and their time for self study is, at best, at night?

And what of this business "Research"? The very word strikes confusion into many a practitioner's heart, for it is a totally foreign concept to him. Practitioners seem to fall into two classes, (1) those who have to do what must be done and are chained to their desks in progressing programs specified by someone slightly further up the hierarchy, and (2) those who sell equipment or services — the entrepreneurs in our midst!

We find no difficulty in understanding why entrepreneurs totally fail to appreciate the importance or even the meaning of the word "research" — their whole nature is to seek out short-term environmental opportunities and obtain the maximum possible benefit therefrom! This attitude of seizing an immediately available opportunity is totally at variance to the stance predominantly adopted by the researcher, who looks to the long term benefit of his profession and of mankind, by seeking after "fundamental truths" with the hope that he can make some contribution to his nation and society at large and thereby justify his existence, at even a subsistence level.

But what of the desk-chained practitioner, surely he realises that what is today's research work will be incorporated in tomorrow's computing systems? Surely he also realises that the necessity for research in computing here in Australia is no less pressing than it is in the US or Japan? Indeed the level of the "research" carried out in Australia today does little more than ensure a small trickle of trained people who, with good luck, might be competent to comprehend the technology incorporated in the equipment imported from overseas in the years ahead. Surely also the practitioner realises that, if Australia does not develop high technology industries, the unemployment rate will continue to increase because it requires far fewer people to import equipment from overseas than those displaced from their jobs by that equipment? Surely again he knows that high technology industries are dependent upon highly trained work forces spending considerable effort on research and development? Australia has the natural latent brain power and talent to develop a high technology infra-structure but is failing to do so through lack of planning and commitment to that goal.

It is the failure of the practitioner to lobby Government to ensure proper funding of Computing teaching and research that is the second major feature of this crisis. If Australia is to be other than a technological colony, subject to economic domination by the USA, Europe and Japan; if Australia is to be anything more than a hole in the ground from which our resources are dug, which resources must, with a generation or two, be totally exhausted, then we should be developing a major national plan for increasing the funding of research work in Australia and this, presumably, will only be possible when those involved in this sphere recognise the contribution to the society at large which is made by those who are involved in researching into, and teaching Computing in Australia today.

Commitment to Research funding does not mean simply supporting "mission oriented" projects such as those supported by today's Government; it requires financial support for more fundamental Research activities into both software and hardware, such as could be conducted in Governmental teaching institutions, were such support forthcoming.

We hope you are better aware of the problems which exist in the teaching of Computing at Government institutions today, and we hope you will lend your voice to see that our Government earmarks funds to correct the situation, immediately.

A. Y. Montgomery

15/4/80
## PROGRAM OF SPEAKERS FOR IFIP-80

### INVITED PAPERS

#### THEORETICAL - LUCENA

1.1 Neuhold, E.J. Prof.  
Univ. of Stuttgart,  
F.R.G.  
The Vienna Development Method (VDM) and its Application for the Specification of a Relational Data Base System  
Tokyo and Melbourne

1.2 Nivat, M. Prof.  
Univ. of Paris,  
France  
Non-deterministic Programs: an algebraic overview  
Tokyo and Melbourne

1.3 Brzozowski, J.A. Prof.  
Univ. of Waterloo,  
Canada  
Developments in the Theory of Regular Languages  
Tokyo and Melbourne

1.4 Manna, Z., Prof.  
Weizmann Inst.,  
Israel  
Logics of Programs  
Tokyo and Melbourne

#### HARDWARE - GOTO

2.1 Recoque, Alice Mm.  
CII-Honeywell Bull,  
France  
Survey of Main Trends in Computer Hardware Architecture  
Tokyo and Melbourne

2.2 Borgerson, B.R.,  
Manu. Data Systems Inc.,  
U.S.A.  
Beyond CAD to Computer Aided Engineering  
Tokyo and Melbourne

2.3 Terajima, M.  
NTT, Tokyo, Japan  
Recent Progress in Memory Devices and their Prospects  
Tokyo and Melbourne

2.4 Strecker, W.D.,  
Digital Equipt. Corp.,  
U.S.A.  
Design Considerations for the VAX  
Tokyo and Melbourne

#### SOFTWARE - ERSHOV

3.1 Tyugu, E. Prof.  
Inst. of Cybernetics,  
U.S.S.R.  
Towards Practical Synthesis of Programs  
Tokyo and Melbourne

3.2 Cocke, J.,  
I.B.M.,  
U.S.A.  
Measurement of Program Improvement Algorithms  
Tokyo and Melbourne

3.3 Kotov, V.E.  
Academy of Sciences,  
U.S.S.R.  
On Basic Parallel Language  
Tokyo and Melbourne

3.4 Mills, H.D.  
I.B.M.,  
U.S.A.  
Function Semantics for Sequential Programs  
Tokyo and Melbourne

#### DATABASE - TSICHRITZIS

4.1 King, W.F.  
I.B.M.,  
U.S.A.  
Relational Database Systems: where we stand today  
Tokyo and Melbourne

4.2 Bachman, C.W.,  
Honeywell,  
U.S.A.  
The Impact of Structured Data throughout Computer Based Information Systems  
Tokyo and Melbourne

4.3 Bubenko, J.A. Prof.  
Chalmers Univ.,  
Sweden  
Information Modeling in the Context of System Development  
Tokyo and Melbourne

4.4 Delobel, C.,  
IMAG Grenoble,  
France  
An Overview of the Relational Data Theory  
Tokyo and Melbourne

#### NETWORKS - LANCE

5.1 Manning, E. Prof.  
Univ. of Waterloo,  
Canada  
Interprocess Communication in Distributed Systems: one view  
Tokyo and Melbourne
### Science and Industry — Andersin

**5.2 Kirstein, P.T. Prof.**
Univ. of London, U.K.
New Text and Message Services
Tokyo and Melbourne

**5.3 Texier, A.G., P.T.T., Rennes, France**
New Telecommunication Services and Network Architecture: a Tentative Survey
Tokyo

**5.4 Inose, H. Prof.**
Univ. of Tokyo, Japan
Aspects of Data Communication and Computer Networks
Tokyo

### Business and Government — Berthet

**6.1 Brent, R.P. Prof.**
Australian Nat. Univ., Australia
Unrestricted Algorithms for Elementary and Special Functions
Tokyo and Melbourne

**6.2 Hearn, A.C. Prof.**
Univ. of Utah, U.S.A.
The Personal Algebra Machine
Tokyo and Melbourne

**6.3 Freeman, H. Prof.**
Rensselaer Poly. Inst., U.S.A.
Lines, Curves and the Characterisation of Shape
Tokyo and Melbourne

**6.4 Kobonen, T. Prof.**
Helsinki Univ. of Tech., Finland
Automatic Pattern Recognition — A Challenge to Computer Technology
Tokyo and Melbourne

**6.5 Evtushenko, Y.G.**
Academy of Sciences, U.S.S.R.
Integrated Optimisation — Simulation Systems for Industry and Regional Planning
Tokyo and Melbourne

### Social and Economic — Losano

**8.1 Steinmuller, W. Prof.**
Univ. of Regensburg, F.R.G.
Rationalisation and Modelling: Two Complementary Implications of Information Technologies
Tokyo and Melbourne

**8.2 Gottlieb, C.C. Prof.**
Univ. of Toronto, Canada
Computers — A Gift of Fire
Tokyo and Melbourne

### Education — Deussen

**9.1 Nievergelt, J. Prof.**
E.T.H., Zurich, Switzerland
Computer Science Education: An Emerging Consensus on Basic Concepts
Tokyo and Melbourne

**9.2 Gries, D. Prof.**
Cornell Univ., U.S.A.
Educatimg the Programmer: Notation, Proofs of Correctness and the Use of Formalism
Tokyo and Melbourne

### Everyday Life — Negroponte

**10.1 Papert, S. Prof.**
MIT, Massachussets, U.S.A.
Reviewing Childhood — Computers and our Notion of what Children can do
Tokyo and Melbourne

**10.2 Pask, G.**
Systems Res. Ltd., U.K.
The Limits of Togetherness
Tokyo and Melbourne

**10.3 Nelson, Th. H.**
Project Xanadu, U.S.A.
Toward a General-purpose Literary System: Replacing the Printed Word
Tokyo and Melbourne

**10.4 Kay, A.**
Xerox Corpn., U.S.A.
User Interface Design in the Small Talk Personal Computing System
Melbourne
## CLOSING ADDRESS

Zamanek, H. Prof.  
Vienna, Austria  

Dixit Algorizmi  
Melbourne

## SUBMITTED PAPERS

<table>
<thead>
<tr>
<th>Number</th>
<th>Author(s)</th>
<th>Title</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>OYAMAGUCHI etc.</td>
<td>On the equivalence problem for two DPDA's, one of which is real-time</td>
<td>Tokyo</td>
</tr>
<tr>
<td>1.2</td>
<td>Von SOLMS, South Africa</td>
<td>Random context array grammars</td>
<td>Tokyo</td>
</tr>
<tr>
<td>1.3</td>
<td>RAIHA etc. Finland</td>
<td>Balancing syntactic and semantic power in compiler specification</td>
<td>Tokyo</td>
</tr>
<tr>
<td>1.4</td>
<td>YOELI etc. Israel</td>
<td>Control nets for parallel processing</td>
<td>Tokyo</td>
</tr>
<tr>
<td>1.5</td>
<td>COUSINEAU etc. France</td>
<td>Tree semantics: A framework for studying program structure and program logics</td>
<td>Melbourne</td>
</tr>
<tr>
<td>1.6</td>
<td>EHRICH etc. F.R.G.</td>
<td>Proving implementations correct — two alternative approaches</td>
<td>Melbourne</td>
</tr>
<tr>
<td>1.7</td>
<td>NAKAGAWA, Japan</td>
<td>Specifying the underlying control structures of programming languages, etc.</td>
<td>Melbourne</td>
</tr>
<tr>
<td>1.8</td>
<td>FOLKJÆR etc. Denmark</td>
<td>A formal model of a generalized CSP-like language</td>
<td>Melbourne</td>
</tr>
<tr>
<td>1.9</td>
<td>BROY etc. F.R.G.</td>
<td>Semantic Relations in programming languages</td>
<td>Melbourne</td>
</tr>
<tr>
<td>1.10</td>
<td>GERGELY, T. etc. Hungary</td>
<td>Program behaviour specification through explicit time consideration</td>
<td>Melbourne</td>
</tr>
<tr>
<td>2.1</td>
<td>KODAMA etc. Japan</td>
<td>The Japanese word processor JW-10</td>
<td>Tokyo</td>
</tr>
<tr>
<td>2.2</td>
<td>KUBO etc. Japan</td>
<td>A Multi-microprocessor system for threedimensional color graphics</td>
<td>Tokyo</td>
</tr>
<tr>
<td>2.3</td>
<td>GOLDBERG U.S.A.</td>
<td>SIFT: A provable fault-tolerant computer for aircraft flight control</td>
<td>Tokyo</td>
</tr>
<tr>
<td>2.4</td>
<td>KAVIANPOUR etc. Iran</td>
<td>Different diagnostic models for multiprocessor systems</td>
<td>Tokyo</td>
</tr>
<tr>
<td>2.5</td>
<td>LEINWAND etc. Israel</td>
<td>Models of control at register transfer level</td>
<td>Tokyo</td>
</tr>
<tr>
<td>2.6</td>
<td>BABA etc. Japan</td>
<td>MUNAP — A two-level microprogrammed multiprocessor architecture for non-numeric processing</td>
<td>Tokyo</td>
</tr>
<tr>
<td>2.7</td>
<td>SYRE France</td>
<td>The Data Driven LAU MULTIPROCESSOR SYSTEM: results and perspectives</td>
<td>Melbourne</td>
</tr>
<tr>
<td>2.8</td>
<td>SHIBAYAMA etc. Japan</td>
<td>Performance evaluation and improvement of a dynamically microprogrammable computer, etc.</td>
<td>Melbourne</td>
</tr>
<tr>
<td>2.9</td>
<td>MALLER U.K.</td>
<td>Information retrieval using the content addressable file store</td>
<td>Melbourne</td>
</tr>
<tr>
<td>2.10</td>
<td>DEB Canada</td>
<td>A 'Regular' non-periodic skewing scheme for optimal conflict-free memory access</td>
<td>Melbourne</td>
</tr>
<tr>
<td>3.1</td>
<td>CATTO etc. U.K.</td>
<td>Nondeterministic data flowgraphs</td>
<td>Tokyo</td>
</tr>
<tr>
<td>3.2</td>
<td>HIKITA etc. Japan</td>
<td>Parallel sieve methods for generating prime numbers</td>
<td>Tokyo</td>
</tr>
<tr>
<td>3.3</td>
<td>NAGATA etc. Japan</td>
<td>An interactive supporting system for functional recursive programming</td>
<td>Tokyo</td>
</tr>
<tr>
<td>3.4</td>
<td>CHUSHO Japan</td>
<td>A good program = a structured program + optimization commands</td>
<td>Tokyo</td>
</tr>
<tr>
<td>3.5</td>
<td>BOURRET etc. France</td>
<td>A deterministic model of operating systems</td>
<td>Tokyo</td>
</tr>
<tr>
<td>3.6</td>
<td>JAMMEL etc. F.R.G.</td>
<td>Impacts of message orientation</td>
<td>Tokyo</td>
</tr>
<tr>
<td>3.7</td>
<td>FUTATSUGI etc. Japan</td>
<td>Specification writing as construction of hierarchically structured clusters of operators</td>
<td>Tokyo</td>
</tr>
</tbody>
</table>

3.8 MEYER
France
A basis for the constructive approach to programming
Tokyo
3.9 NAKAJIMA etc.
Japan
The Iota Programming System — A Support system for Hierarchical and modular programming
Tokyo
3.10 XU etc.
China
On the design, implementation and use of the systems programming language XCY
Tokyo
3.11 MARTIN
Netherlands
A distributed implementation method for parallel programming
Tokyo
3.12 GAUDE etc.
France
Distributed processing as a key to reliable and evolving software for real time applications
Melbourne
3.13 BOEHM
U.S.A.
An experiment in small-scale application software engineering
Melbourne
3.14 FOISSEAU etc.
France
Program development with or without coding
Melbourne
3.15 VAQUERO etc.
Spain
Placement of records on linear storage devices
Melbourne
3.16 KEEDY
Australia
Paging and small segments: A memory management model
Melbourne
3.17 GERHART etc.
U.S.A.
An overview of AFFIRM: A specification and verification system
Melbourne
3.18 KANOUI etc.
France
Implementing a very high level language on a very low cost computer
Melbourne
3.19 KAPLAN etc.
U.S.A.
Adding type declaration to INTERSLIP
Melbourne
3.20 ADAMS etc.
U.S.A.
FORTRAN for the 1980s
Melbourne
4.1 TANAKA etc.
Japan
Pipeline searching and sorting modules as components of a data flow database computer
Tokyo
4.2 UEMURA etc.
Japan
The design and implementation of a magnetic-bubble database machine
Tokyo
4.3 CHUNG etc.
U.S.A.
Magnetic bubble memory structures for efficient sorting and searching
Tokyo
4.4 MALY etc.
U.S.A.
H-trees
Tokyo
4.5 TKAIZAWA etc.
Japan
Query translation in distributed databases
Tokyo
4.6 DADAM etc.
F.R.G.
Recovery in distributed databases based on non-synchronized local checkpoints
Tokyo
4.7 CARLSON etc.
U.S.A.
Integrating dialog management and database management
Tokyo
4.8 DE JONG etc.
U.S.A.
The System for Business Automation (SBA): A unified application development system
Tokyo
4.9 ADIBA etc.
France
An overview of the POLYPHEME distributed database management system
Melbourne
4.10 MUNZ
F.R.G.
Transaction management in the distributed data system VDN
Melbourne
4.11 ASTRAHAN etc.
U.S.A.
Performance of the system R access path selection mechanism
Melbourne
4.12 SCHEK
F.R.G.
On index intervals and their optimization
Melbourne
4.13 CERI etc.
Italy
Integrated specification of static and dynamic requirements of data base applications
Melbourne
4.14 DAVENPORT
U.K.
Data administration: the need for a new function
Melbourne
5.1 BOZZETTI
Italy
A general overview of the Olivetti Network File System
Tokyo
5.2 KLEINROCK etc.
U.S.A.
Interfering queuing processes in packet switched broadcast communication
Tokyo
5.3 LE REST etc.
France
NTI: International gateway to TRANSPAC
Tokyo
5.4 JOHNSON
U.K.
Identifying the main applications of data communications in Western Europe
Tokyo
5.5 MORINO etc.
Japan
A technique for describing High-level Data Link Control (HDLC) Procedures
Tokyo

| 5.6 | ONOZATO etc. | IFIP-80 Dynamic characteristics of a satellite communication system employing the slotted ALOHA scheme | Tokyo |
| 5.7 | FOLEY | Optimum design of two-computer networks | Melbourne |
| 5.8 | HAESSIG etc. | Partitioning and allocating computational objects in distributed computing systems | Melbourne |
| 5.9 | AZEMA etc. | The design and validation by Petri nets of a mechanism for the invocation of remote servers | Melbourne |
| 5.10 | MAHABALA etc. | Pseudo-ternary coding to incorporate parity checking without a parity bit | Melbourne |
| 6.1 | ACIL'DIEV | The method of normal solutions in linear programming | Tokyo |
| 6.2 | MARCHUK etc. | Parallel computations in grid methods for solving mathematical physics problems | Tokyo |
| 6.3 | DAVID etc. | CAD in small and medium sized industries | Tokyo |
| 6.4 | ROLSTADAS etc. | Prototyping factory management systems | Tokyo |
| 6.5 | KATO etc. | A hand-writing input system for Japanese sentences | Tokyo |
| 6.6 | BERTHOD etc. | Using context in the global recognition of a set of objects: an optimization approach | Tokyo |
| 6.7 | NISHIDA etc. | A knowledge-based literature guide system — a new approach to document retrieval | Tokyo |
| 6.8 | DINCBAS | A knowledge-based expert system for automatic analysis and synthesis in CAD | Tokyo |
| 6.9 | ALBRECHT | A survey of recent results of composite integration methods for ordinary differential equations, etc. | Tokyo |
| 6.10 | GEAR | Automatic multirate methods for ordinary differential equations | Melbourne |
| 6.11 | AXELSSON etc. | A preconditioned conjugate gradient method for finite element equations, which is stable, etc. | Melbourne |
| 6.12 | GAMBOLATI etc. | An improved iterative scheme for refining the solution of ill-conditioned systems | Melbourne |
| 6.13 | ZHURIN etc. | Recent advances in the numerical solution of the Boltzmann equation | Melbourne |
| 6.14 | LAWSON etc. | High-order near best uniform approximations to the solution of heat conduction problems | Melbourne |
| 6.15 | DE MORI etc. | Structural knowledge for the recognition of syllables in continuous speech | Melbourne |
| 6.16 | HERNAN etc. | Image analysis of experimental data in turbulence | Melbourne |
| 7.1 | STEVENS | Some cautionary aphorisms for user-oriented computer management | Tokyo |
| 7.2 | GABAY etc. | Computers in economic policy making: challenging advances in computer systems and econometric models | Tokyo |
| 7.3 | GERGELY, C. etc. | The financial information system in Hungary, based on a three level equipment hierarchy | Tokyo |
| 7.4 | YOKOYAMA etc. | Prescription order service in Kanto Teishin Hospital information system | Tokyo |
| 7.5 | TYSAN | South Australian land information system | Melbourne |
| 7.6 | BRUSSAARD etc. | Information and organization policies in public administration | Melbourne |
| 7.7 | NEPMIASTCHY | MODULLECO: Software for macroeconomic modelling | Melbourne |
| 7.8 | BARROS etc. | Decision support systems for economic policy analysis in a market oriented environment | Melbourne |
| 7.9 | MATTHEWS | Trends in user-oriented software within the U.K. Central Electricity Generating Board | Melbourne |
7.10 STANLEY
Australia
A design approach to the audit of computer information systems
Melbourne

8.1 MURPHY
U.K.
The development of the British, French and German native computer industries, 1960-1978
Tokyo

8.2 GENUYS
France
The impact of computerization on the relationship between public administration, etc.
Tokyo

8.3 SMYTH etc.
Canada
The encouragement of effective computer use in developing countries
Tokyo

8.4 VRECION
Czechoslovakia
Some basic applications of computing in the law-making process
Tokyo

8.5 BLACKMORE
Australia
A computer training scheme for the disabled
Melbourne

8.6 BLUM etc.
U.S.A.
Computer technology and medical costs
Melbourne

8.7 LINCOLN
U.K.
Information systems constraints — a strategic review
Melbourne

8.8 COHEN-HADRIA
France
Automation, organizational choices and social effects: the case of the cement industry
Melbourne

8.9 GILCHRIST
U.S.A.
Computers and Employment: the U.S. experience
Melbourne

9.1 MACLEOD etc.
Australia
A multisensory electronic system for developing handwriting skills with blind trainees
Melbourne

9.2 SCHWEIKHARDT
F.R.G.
A computer based education system for the blind
Tokyo

9.3 PAGES etc.
France
A general structure for building personalized information: An alternate approach, etc.
Tokyo

9.4 KUBBA
Iraq
The impact of computers on Arabic writing, character processing and teaching
Tokyo

9.5 THOMAS etc.
Canada
Computers and education: A remote learning application
Melbourne

9.6 HAEFNER
F.R.G.
The concept of an integrated system for information access and telecommunication (ISIT) etc.
Melbourne

9.7 KLOPPROGGE
F.R.G.
Teaching systems analysis and planning in informatics: a practical course
Melbourne

9.8 ISOMOTO etc.
Japan
A computer aided advising system for FORTRAN program debugging: ADVISOR
Tokyo

10.1 SCHMANDT
U.S.A.
Soft Typography
Melbourne

10.2 ISHII
Japan
Feasibility experiments on versatile pattern information processing systems
Tokyo

10.3 HEROT
U.S.A.
A spatial graphical man-machine interface
Melbourne

10.4 HUNTER
U.S.A.
Teleconference in virtual space
Melbourne

10.5 TURKLE
U.S.A.
Personal computation and personal meaning
Melbourne

10.6 POLLARD etc.
Australia
Using a home microcomputer to teach a dyslexic child to read
Melbourne

THEORETICAL — LUCENA

1.1 Igarashi, S. Prof.
Univ. of Tsukuba, Japan
Specification, Verification and Synthesis of Programs
Tokyo

1.2 Darlington, J.
Univ. of London, U.K.
Program Transformation — Techniques and Applications
Melbourne

HARDWARE — GOTO

2.1 Aiso, H. Prof.
Keio Univ., Japan
Parallelism in Computer Architectures
Tokyo

PANEL MEETINGS

53
2.2 Ishida, H. Prof.  
Univ. of Tokyo, Japan  
Micro Processor Application  
Melbourne

2.3 Piloty, R. Prof.  
T.H. Darmstadt, FRG  
An Overview of CONLAN: a formal construction method for hardware description languages  
Tokyo and Melbourne

SOFTWARE – GOTO
3.1 Carlson, W.  
DARPA, Arlington, USA  
The ADA Language: Design Objectives and Early Experience  
Tokyo and Melbourne

3.2 Dewar, R.B.K. Prof.  
Univ. of New York, USA  
Very High Level Languages and Specification Languages  
Tokyo

3.3 Zamenek, H. Prof.  
Vienna, Austria  
Qualified Programmers for the Eighties — How to Get Them  
Melbourne

3.4 Sandewall, E. Prof.  
Linjoping Univ., Sweden  
Programming Environment  
Melbourne

DATA BASE – TSICHRITZIS
4.1 Sibley, E.H. Prof.  
Univ. of Maryland, USA  
The Very Large Database: its Theory, Practice and Problems  
Tokyo and Melbourne

4.2 Lowenthal, E.I.  
MRI Systems Corp., USA  
Product Management Requirements for Commercial DBMS Suppliers  
Tokyo

4.3 Yao, S.B. Prof.  
Purdue Univ., USA  
Decision Support Systems for Database Design  
Melbourne

4.4 Hsiao, D.K. Prof.  
Univ. of Ohio, USA  
Data Base Computer Architecture  
Melbourne

NETWORKS – LANCE
5.1 Landweber, L. Prof.  
Univ. of California, USA  
The Future of Networks as a Research Tool  
Tokyo

5.2 Uhlig, R.P.  
Bell Nthn. Res. Ltd., Canada  
International Computer Message Services  
Melbourne

5.3 Davies, D.W.  
Nat. Physical Lab., U.K.  
Security with Special Reference to Computer Networks  
Melbourne

5.4 Hughes, P.  
Logica, U.K.  
Data Communications in Western Europe  
Melbourne

SCIENCE AND INDUSTRY – ANDERSIN
6.1 Niemi, A.J. Prof.  
Tech. Res. Centre, Finland  
Industrial Automation and Robotics  
Tokyo

6.2 Lassez, J.L.  
Univ. of Melbourne, Australia  
Voice Analysis and Synthesis  
Melbourne

6.3 Gear, C.W. Prof.  
Univ. of Illinois, U.S.A.  
Standardization of Floating-Point Arithmetic: Benefits and Implications  
Melbourne

BUSINESS AND GOVERNMENT – BERTHET
7.1 Wallin, S.E.  
Swedish Telecomm. Admin., Sweden  
The Computerisation of Telecommunications Administrations  
Tokyo and Melbourne

7.2 Checroun, A.D. Prof.  
Univ. of Paris, France  
Generalisation of Computerisation of Information Systems: Models and Simulation Methods  
Tokyo and Melbourne

7.3 Moriguti, S. Prof.  
Univ. of Tokyo, Japan  
Government Data Systems  
Tokyo

7.4 Chevion, D.  
Min. of finance, Jerusalem, Israel  
The proceedings of a symposium held in June to be available to the
reviewer before the end of October.

addition, one can comment favourably on the cooperation evident
is of the usual high standard expected from the publishers. In
author's manuscripts as is now common for conference proceedings,
face; Integration; Applied Algebra; Languages and Designs.

The standard of the contributions is, in general, very high.
These areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra systems or the non-specialist seeking an overview of the subject. In fact, their highly theoretical
nature makes most of the papers inaccessible to the reader without a
relevant research degree.

The preceding remarks are not, of course, intended as a con-
demnation of the book, merely as a warning to potential readers. After all, who among users of operating systems and compilers, for
example, reads proceedings of conferences on the latest theories in
these areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
better use of space, and other contributions to more effective use of
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra computation. The main weapons used are an
equally wide variety of mathematical techniques, usually deep and
abstract; the readers who will gain most from this book will probably therefore be pure mathematicians, and these people have
the best chance of continuing the development of the subject.
It is inappropriate to attempt to review individual papers
here, even if it were possible for one person to try. The range of
topics discussed is best indicated by the section headings in the
table of contents: Simplification; Applications in Physics; Matrices
and Equations; Algebraic Fields; Differential Equations; Polynomial
Algorithms; Systems; Algorithm Analysis; Symbolic-Numeric Inter-
face; Integration; Applied Algebra; Languages and Designs.

Production of this book, photographed directly from
the casual user of symbolic algebra systems or the non-specialist
nature makes most of the papers inaccessible to the reader without a
relevant research degree.

The preceding remarks are not, of course, intended as a con-
demnation of the book, merely as a warning to potential readers. After all, who among users of operating systems and compilers, for
example, reads proceedings of conferences on the latest theories in
these areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
better use of space, and other contributions to more effective use of
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra computation. The main weapons used are an
equally wide variety of mathematical techniques, usually deep and
abstract; the readers who will gain most from this book will probably therefore be pure mathematicians, and these people have
the best chance of continuing the development of the subject.
It is inappropriate to attempt to review individual papers
here, even if it were possible for one person to try. The range of
topics discussed is best indicated by the section headings in the
table of contents: Simplification; Applications in Physics; Matrices
and Equations; Algebraic Fields; Differential Equations; Polynomial
Algorithms; Systems; Algorithm Analysis; Symbolic-Numeric Inter-
face; Integration; Applied Algebra; Languages and Designs.

Production of this book, photographed directly from
the casual user of symbolic algebra systems or the non-specialist
nature makes most of the papers inaccessible to the reader without a
relevant research degree.

The preceding remarks are not, of course, intended as a con-
demnation of the book, merely as a warning to potential readers. After all, who among users of operating systems and compilers, for
example, reads proceedings of conferences on the latest theories in
these areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
better use of space, and other contributions to more effective use of
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra computation. The main weapons used are an
equally wide variety of mathematical techniques, usually deep and
abstract; the readers who will gain most from this book will probably therefore be pure mathematicians, and these people have
the best chance of continuing the development of the subject.
It is inappropriate to attempt to review individual papers
here, even if it were possible for one person to try. The range of
topics discussed is best indicated by the section headings in the
table of contents: Simplification; Applications in Physics; Matrices
and Equations; Algebraic Fields; Differential Equations; Polynomial
Algorithms; Systems; Algorithm Analysis; Symbolic-Numeric Inter-
face; Integration; Applied Algebra; Languages and Designs.

Production of this book, photographed directly from
the casual user of symbolic algebra systems or the non-specialist
nature makes most of the papers inaccessible to the reader without a
relevant research degree.

The preceding remarks are not, of course, intended as a con-
demnation of the book, merely as a warning to potential readers. After all, who among users of operating systems and compilers, for
example, reads proceedings of conferences on the latest theories in
these areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
better use of space, and other contributions to more effective use of
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra computation. The main weapons used are an
equally wide variety of mathematical techniques, usually deep and
abstract; the readers who will gain most from this book will probably therefore be pure mathematicians, and these people have
the best chance of continuing the development of the subject.
It is inappropriate to attempt to review individual papers
here, even if it were possible for one person to try. The range of
topics discussed is best indicated by the section headings in the
table of contents: Simplification; Applications in Physics; Matrices
and Equations; Algebraic Fields; Differential Equations; Polynomial
Algorithms; Systems; Algorithm Analysis; Symbolic-Numeric Inter-
face; Integration; Applied Algebra; Languages and Designs.

Production of this book, photographed directly from
the casual user of symbolic algebra systems or the non-specialist
nature makes most of the papers inaccessible to the reader without a
relevant research degree.

The preceding remarks are not, of course, intended as a con-
demnation of the book, merely as a warning to potential readers. After all, who among users of operating systems and compilers, for
example, reads proceedings of conferences on the latest theories in
these areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
better use of space, and other contributions to more effective use of
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra computation. The main weapons used are an
equally wide variety of mathematical techniques, usually deep and
abstract; the readers who will gain most from this book will probably therefore be pure mathematicians, and these people have
the best chance of continuing the development of the subject.
It is inappropriate to attempt to review individual papers
here, even if it were possible for one person to try. The range of
topics discussed is best indicated by the section headings in the
table of contents: Simplification; Applications in Physics; Matrices
and Equations; Algebraic Fields; Differential Equations; Polynomial
Algorithms; Systems; Algorithm Analysis; Symbolic-Numeric Inter-
face; Integration; Applied Algebra; Languages and Designs.

Production of this book, photographed directly from
the casual user of symbolic algebra systems or the non-specialist
nature makes most of the papers inaccessible to the reader without a
relevant research degree.

The preceding remarks are not, of course, intended as a con-
demnation of the book, merely as a warning to potential readers. After all, who among users of operating systems and compilers, for
example, reads proceedings of conferences on the latest theories in
these areas? The general level of facilities offered by symbolic
algebra systems has changed little in the past decade, although close
inspection of the internals will reveal more sophisticated algorithms,
better use of space, and other contributions to more effective use of
resources than were present in earlier systems. The main thrust of
research is in fact towards obtaining greater insight into the wide
diversity of problems collected together under the heading of
symbolic algebra computation. The main weapons used are an
equally wide variety of mathematical techniques, usually deep and
abstract; the readers who will gain most from this book will probably therefore be pure mathematicians, and these people have
the best chance of continuing the development of the subject.
It is inappropriate to attempt to review individual papers
here, even if it were possible for one person to try. The range of
topics discussed is best indicated by the section headings in the
table of contents: Simplification; Applications in Physics; Matrices
and Equations; Algebraic Fields; Differential Equations; Polynomial
Algorithms; Systems; Algorithm Analysis; Symbolic-Numeric Inter-
face; Integration; Applied Algebra; Languages and Designs.
An Introduction to Knowledge-Based Expert Systems

J.R. Quinlan*  

This tutorial paper discusses the concepts underlying knowledge-based systems, and shows how they differ from more conventional (algorithm-based) systems. Three principal dimensions are identified: the representation of knowledge, architectures for deploying knowledge, and techniques for acquiring knowledge. Key ideas are illustrated from a selection of current expert systems covering a wide range of applications.

Keywords: rule-based, frames, productions, inference.
CR categories: 1.3, 3.6.

INTRODUCTION

Fifteen years ago, computers were expensive, esoteric devices used primarily for numerical computation. Even within this subdomain, applications tended to be either large 'numbercrunching' jobs such as calculations motivated by research in the physical sciences, or 'data processing' and similar jobs in the more forward-thinking sections of industry. Today computers are cheap, every-day tools whose principal uses have little to do with the manipulation of numeric information. (Anyone surprised that they could be described as 'cheap' should read the article entitled "It's like buying a Rolls-Royce for a fiver" [Michie 1979].)

The real explosion in the application of computers to problems of practical importance has had a startling consequence. Increasingly, the commonplace uses have been or are being exhausted. In a recent talk, Feigenbaum put it succinctly when he said that much of the world's work lacks a mathematical (i.e., analytically tractable) core, and such tractable areas as do exist have already been exploited to near their limit. More and more attention is shifting to areas for which analytic methods are not known, but where human beings are able to achieve results. Not surprisingly, systems for such areas have turned out to be unlike more familiar programs for analytically tractable tasks.

How can we characterise these more familiar programs? For one thing they tend to reflect the orderly domain for which they were written; they are deterministic and possess no redundancy. For any given input there is a single computational path that is always followed, and there is a single mechanism capable of producing the correct output for that input. Human beings, however, are often equipped with an armoury of overlapping techniques for handling a problem, and forgetting one of them does not prevent a solution being found by other means. Again, a conventional program usually exhibits a sharp distinction between code and data — between the recipes for how to manipulate structures and the structures themselves. Since only the latter are accessible to the program, it thus lacks any ability to reason about or explain the techniques and mechanisms that it employs. Finally, complex programs almost invariably are opaque and difficult to modify. The algorithms on which they are based are intermixed with details of representation and so forth. They are useless as a codification of expertise or as a vehicle for communicating it: imagine trying to learn how to play chess from poring over a conventional chess-playing program!

In contrast with programs like this (which will be called algorithm-based) are the more recent knowledge-based systems. Here the whole focus is on knowledge in the ordinary meaning of the term: facts about the task domain, and heuristics or rules of thumb that guide the use of knowledge to solve problems in the domain. Most current knowledge-based systems are divided, not into code and data, but into a corpus of knowledge and a comparatively simple mechanism for applying the knowledge in an opportunistic way to solve problems. The power of the system does not come principally from this knowledge application mechanism (the inference engine) but from the richness, pertinence and redundancy of the knowledge itself.

Scientists involved in building these knowledge-based systems have placed considerable importance on expressing the knowledge in a form that is usable by the inference engine, but also comprehensible to human beings. These systems can then be understood by 'digesting' the knowledge rather than by tracing the intricate computational paths arising from possible interactions between knowledge elements. In this way it has proved possible to construct complex systems that are still relatively transparent. When the knowledge elements are also fairly independent (as is usually the case), incremental modification and improvement of the system is easy. Finally, if the knowledge is redundant, the absence of one fact or mechanism does not necessarily prevent the system arriving at a result by another route.

The applicability of ideas like these is not limited to the 'difficult' areas for which no algorithms exist; even more familiar processes can benefit from them. For example, it has been shown that a common prime-number testing algorithm can be improved by the addition of the redundant information that all prime numbers larger than 5, when divided by 30, have a remainder which is one of 1, 7, 11, 13, 17, 19, 23 or 29 (Michie 1977). However...
this paper is specially concerned with expert systems, where the task domain is regarded by humans as complex, requiring special training, and where a high level of performance is demanded. Almost by definition, effective algorithms do not exist for this sort of domain. It is becoming apparent that knowledge-based systems are the natural way (some would argue the only way) to build expert systems.

The process of building expert systems by way of assembling knowledge was christened 'epistemological engineering' by McCarthy but Michie's amendment 'knowledge engineering' is the term in most common use. Interestingly enough, it is these two who have begun the difficult task of developing a theory of knowledge per se; see for example [McCarthy 1979], [Michie 1977]. I see this knowledge engineering process as one requiring three design decisions:

(i) How is the knowledge to be represented, so that it is both usable by the system and comprehensible to human beings?

(ii) What architecture should the system have so that the knowledge can be brought to bear on problems in this domain?

(iii) How can the knowledge be acquired and tested for internal consistency and completeness?

There are no pat answers to these questions, which provoke some quite sharp disagreements among practising knowledge engineers. The following sections examine these issues in turn, and exhibit the answers to them that have been embodied in a sample of current systems.

ENCODING KNOWLEDGE

Of the many ways in which knowledge has been represented in expert systems, three in particular have achieved widespread acceptance: production systems, first-order logic statements, and frame systems. This section contains a thumbnail sketch of each: much more complete tutorial expositions can be found in (Davis and King, 1977), (van Emden 1977) and (Winston 1977) respectively.

Production systems were first proposed in the 1940s but their use today stems from the work of Newell and Simon (Newell and Simon 1972). The basic idea is very simple: we have a data base and a collection of production rules each of the form

\[ \text{situation} \rightarrow \text{action} \]

The situation is anything that can be determined to be part of the molecule, and the problem to a goal clause of the form

\[ R \leftarrow A_1 A_2 \ldots A_x \]

Situation \( \rightarrow \) action rules are widely used to represent knowledge for expert systems work, particularly in the United States. An example is the pioneering DENDRAL project (Felgenbaum, Buchanan and Lederberg 1971). This system was not originally formulated as a rule based system; the change was forced on its designers by the failure of conventional programming methodology as the system grew in complexity. (Incidentally, DENDRAL was one of the first expert systems and after more than a decade is still arguably the best.) The problem domain is the task of inferring plausible molecular structures of an organic chemical from its atomic composition and mass spectrogram. The latter is obtained by bombarding a sample of the compound with high-energy particles, causing pieces of the molecule to break away, and recording the masses and relative abundance of the fragments. DENDRAL is made up of three principal modules. The first derives constraints on the molecular structure by examining the spectrogram, using rules like

**situation:** there is a pronounced peak at each of 43, 71 and 86, and an additional peak at 58

**action:** record the constraint that part of the molecule is an N-PROPYL-KETONE structure

The second component is not rule based, but a clever algorithm that derives all possible structures consistent with these derived constraints and with further general chemical knowledge that some theoretically possible structures never occur in stable chemicals. The final module tests these hypothesised structures one at a time. The structure becomes the data base, and the rules that operate upon it are situations which are molecular substructures and actions marking probable bonds that will fragment. An example:

\[ H \]
\[ \text{---} N - C - C - C - \Rightarrow - N - C * - C - \]
\[ H \]

where '---' represents a broken bond. After all rules have had their say, all possible broken bonds have been found and the mass spectrogram that would arise from this structure can be predicted. DENDRAL matches the predicted spectrogram against the actual one to judge the plausibility of the proposed candidate structure.

The general concept of a rule-based system does not constrain the form that situations and actions can take. The situation is anything that can be determined to be satisfied or not by a particular state of the data base, and the action can be anything that can change it in some way. The second representational scheme gives up some of this flexibility in exchange for the well-understood semantics of first-order logic. Here information about the domain and the problem is represented by logic formulae and augmented using rules of inference. This formulation has been further restricted in PROLOG, a widely known approach in this direction. All knowledge is reduced to a collection of Horn clauses of the form

\[ R \leftarrow A_1 A_2 \ldots A_x \]

and the problem to a goal clause of the form

\[ \neg B_1 \land B_2 \ldots \land B_i \land \ldots \land B_y \]

where each R, A and B is an atomic formula. The semantics of a Horn clause can be thought of as being either declarative

"if A_1, A_2, \ldots, A_x all hold then so does R"

or procedural

"if you want R then do A_1, then A_2, \ldots, then A_x"
thus combining both the idea of declaring eternal verities
with the sort of 'do it this way' instruction sequences found
in more conventional programming languages. The speci-
fied inference rule used to deduce new information is
that, given the goal clause and Horn clause above, if there
is a substitution $S$ of terms for variables that will make
$Bi$ and $R$ identical, then we can derive the new goal clause
\[
< - (B1 \land B2 \land \ldots \land Bi-1 \land A1 \land \ldots \\
\land Ax \land Bi+1 \land \ldots \land By) \land S
\]
In production systems, computation stops when a solu-
tion is found or no rule has a situation satisfied by the
current data base; a PROLOG program halts when the
empty goal clause is generated or no new goal clause can
be formed.

Bundy's Mecho (Bundy et al. 1979) is an example of
an expert system written in PROLOG. This program's area
of expertise is described in the above paper as
"... mechanics problems which deal with idealised
objects such as smooth planes, light inextensible
strings, frictionless pulleys, etc... To date, our pro-
gram has tackled problems in the area of: pulley
problems, statics problems, motion on smooth
complex paths and motion under constant accel-
eration."

This is only half the story, however, because the prob-
lems are stated in English, so Mecho also tackles the
task of natural language understanding in this domain.
The following example is given of the sort of problem as
presented to the system:
"Two particles of mass $b$ and $c$ are connected by a
light string passing over a smooth pulley. Find the
acceleration of the particle of mass $b". This
is translated into a collection of Horn clauses
that assert facts about this particular problem, and com-
bined with more clauses defining the genus of pulley
problems (such as "if the pulley has no mass the tensions
in the left and right hand parts of the string are the same").
This final set of clauses, called object clauses, is manipu-
lated by a second set of meta-level clauses to extract
equations for solution. The meta-level clauses are also
first-order, but the predicates used in the object clauses
become terms! An example quoted in the above paper is
\[
\text{rewrite (accel (P,A,Div,Time),} \\
\text{relaccel (P,earth A,Div,Time),} \\
\text{strategy (d binf)}) < -
\]
which asserts that "any accel(eration) predicate can be
rewritten to a rel(ative) accel(eration) predicate with
the earth as the other point of reference, and that the standard
inference strategy is then applicable".

A number of difficulties confront any system such
as Mecho which deals with detailed real-world informa-
tion. Everything in the domain is related to just about
everything else, and a solution to the whole problem
cannot usually be found by considering parts of it in
isolation. (For example, the pulley problem cannot be
solved by just thinking about the particle of mass $b$.) Also,
a great deal of necessary information is not specified
directly in the problem, but is 'taken for granted': that
the pulley itself has no mass, that its diameter is irrele-
vant, that it is fixed relative to the earth and so on. In
Mecho these difficulties are tackled by mechanisms such as
type predicates:
\[
\text{type-constraint (light, physobj)}
\]
which asserts that in order to be light (massless) an entity
must be a physical object; and 'cueing' which adds to the
list of clauses produced from a problem statement a set of
default assertions about everything not specifically
mentioned.

Another clean and powerful technique for over-
coming these difficulties was proposed by Minsky in
1974. The kernel of the idea is the representation of
things and events by a collection of frames. Each frame
corresponds to one entity and contains a number of label-
led slots for things pertinent to that entity. Slots in turn
may be blank, or be specified by terminals referring to
other frames, so the collection of frames is linked together
into a network. As an illustration, consider the entity "a
cricket match". A frame for this might contain slots for
the names of the teams, a description of the players, the
dates played, the result of the match and so on. Some
slots may be blank, such as the result in the case of a future
match. Some slots may have default specifications (e.g.,
the players are men) by way of appropriate links, and
these default links would only be broken when they were
explicitly contradicted. The slots delimit the information
that is relevant to the entity; there would be no question
of trying to find the weight of a cricket match because no
such slot would exist. Frames can be used to represent
vastly different things: facts ("The West Indies won the
second Test against Australia in 1979"), objects ("the
second Test . . . ") and concepts or prototype objects
("Test") are just a few. The interconnections between
frames allows information to be "inherited" by one frame
from another, for example from the frame describing a
prototype to one describing an object covered by that
prototype.

The NUDGE system (Goldstein and Roberts 1979)
is an expert system for scheduling events that is based on
the frame representation. The authors state
"Traditionally, scheduling programs apply simple
but powerful decision analysis techniques to find-
ing the optimal schedule under a well defined set
of constraints . . . .
For well-defined, formal situations the traditional
power-based approach is appropriate. But for the
problem of defining these formal situations when
given only informal specifications, a knowledge-
based approach is necessary."

NUDGE takes very incomplete requests for events
to be scheduled and transforms them into more com-
plete requests by filling in information that we all take
for granted. Conflicts that arise in the fleshed-out requests
are then satisfied by an independent scheduling program.
An example simplified from the above paper should illus-
trate the ideas. Part of the frame for the concept 'PA
meeting' is
\[
\text{PA-MEETING} \\
\text{AKO} \text{ $\textit{VALUE}$ $\textit{MEETING}$} \\
\text{WHY} \text{ $\textit{VALUE}$ $\textit{PA-PROJECT}$} \\
\text{WHERE} \text{ $\textit{DEFAULT}$ $\textit{AI-PLAYROOM}$} \\
\text{WHEN} \text{ $\textit{DEFAULT}$ ((ON FRIDAY) [FOR 1 HOUR])} \\
\text{WHO} \text{ $\textit{DEFAULT}$ IRA [ROLE: MANAGER]} \\
\text{Bruce [ROLE: FRL]} \\
\text{CANDY [ROLE: SEMANTICS]} \\
\text{MITCH [ROLE: SYNTAX]} \\
\text{The Australian Computer Journal, Vol. 12, No. 2, May 1980}
\]
When the data in the database is noisy and there is a powerful model of what to expect in the data, backward chaining is more powerful (Nii and Feigenbaum 1978).

MYCIN is a backward chaining rule-based system that gives advice to physicians on the diagnosis of bacterial infections and appropriate drug therapy (Shortliffe 1976). The data base in this case is the physician himself providing information in response to questions generated by the program. In this application the model-driven system architecture is important because the resulting goal structure enables the requests for information to follow a 'sensible' pattern by pursuing a coherent line of inquiry. MYCIN has achieved an impressive level of performance, reputedly considerably better than an intern. A panel of experts in this field agreed with the program's diagnosis and therapy in 90 per cent of the cases examined (Feigenbaum 1979). And yet the major contributions of MYCIN seem to lie along other dimensions: it deals with 'facts' having different degrees of certainty, it communicates with users in comprehensible English, it can explain and answer questions on its conclusions (a feature that is most important to the degree of acceptance of those same conclusions), and it is able to modify and add to its store of knowledge by interactions with experts.

As the volume of knowledge in a rule-based system grows, two new problems emerge. From the human designer's point of view it becomes more difficult to understand unless it is organized in some way. On the computational side the task of checking each rule to see if its situation is satisfied in the database (or alternatively if its action part is relevant to a goal) consumes more time on every cycle. Why doesn't this sort of problem affect human beings in the same way? One obvious suggestion is that we partition our knowledge into labelled compartments, and we do not 'activate' our knowledge of integral calculus, for instance, when we are trying to solve a crossword puzzle. Furthermore, we attempt to focus on one section of the problem at a time, using as much local context as possible: the crossword puzzle is a good example of this also.

The blackboard model, first developed in the context of a speech understanding system (Mostow and Hayes-Roth 1978), is an architecture that tries to carry these ideas over to rule-based systems. The anthropomorphic model is a collection of independent experts co-operating to solve a problem, who communicate by writing on a blackboard visible to all. The blackboard contains the problem data and a hierarchy of hypotheses relevant to the problem. At any one time a single hypothesis or bit of data becomes the centre of attention. Those experts to whom this is meaningful each examine it, and may suggest changes to the blackboard — the addition of new hypotheses or modifications to existing hypotheses. The focus then shifts to another area of the board. From the point of view of system architecture, each expert or knowledge source is a packet of situation-action rules which may have its own internal database. Experts, rather than individual rules, are activated by events, which correspond to certain types of changes to the blackboard.

The CRYSTALIS system (Englemore and Terry 1979), described by its authors as being in its "early adolescence", is an ambitious project to recover three-dimensional protein structures from electron density data produced by X-ray crystallography. Its architecture closely follows the blackboard model: its rules are packaged into
knowledge sources each of which is invoked by an event. A knowledge source is like a mini expert system which contains all CRYSLALIS's expertise on some facet of crystallography such as what strategy to apply next, inferences to be drawn from properties of the data, or expectations generated by some sort of hypothesis. The action part of a rule in a knowledge source can trigger events as it changes some part of the data base, thus invoking other knowledge sources focussed on the changed part. The data base itself is complex. At the lowest level of abstraction there are the raw electron density profiles, on which are built three levels of more abstract data descriptions: local maxima, the protein skeleton or backbone, and the independent segments or building blocks of the skeleton. Above this data hierarchy are three levels of model hypotheses: the presence of atoms, common groups of atoms, and large identifiable portions of the protein. The hierarchical organization allows the system as a whole to obtain the effect of forward and backward chaining simultaneously. For example, a data-driven knowledge source can hypothesise from part of the data the existence of (say) a group of atoms, while a model-driven knowledge source can then search other parts or levels of the data for more evidence to support or contradict this hypothesis.

It is significant that both architectures discussed in this section have proved useful for more than one system, and have subsequently been embodied into packages. E-MYCIN (van Melle 1979) is the MYCIN system stripped of all its knowledge about bacterial infections: the potential user need only add knowledge in the same format but concerning some other task domain and he has built a new expert system! AGE (Nii and Aiello 1979) is a system that knows about blackboard models, and helps the user to design, code and test one for his own application. The development of tools such as these allows the system-builder to concentrate on getting the knowledge right without having to worry about the tedious environmental details.

**KNOWLEDGE ACQUISITION**

Rather surprisingly, only a quite small amount of knowledge is necessary to drive an expert system at approximately human levels of performance. I wrote a program that used Bridge bidding and leads to deduce information about which hand held which high cards (Quinlan 1979c). The program was based on a book devoted to this topic, and about one-third of the book reduced to only 30-odd rules! SACON (Bennett and Englemore 1979), a consultative system that helps engineers use a structural analysis package, contains 170 rules. I know of only one current expert system that exceeds the 300-rule barrier.

One would think that it would therefore be possible to build an expert system very quickly — say between lunch and afternoon tea — but experience indicates that this is not so. The Bridge program above took me two months. The SACON project required four months of the experts' time, let alone that of the knowledge engineers. DENDRAL can be measured in man-years, perhaps man-decades, of work. The process of extracting knowledge from experts, making sure that all cases are covered, hunting out internal contradictions, etc., is one of trial and error, and can be laborious.

It is heartening therefore to see the development of programs that can obtain this knowledge more or less under their own steam. The process used is induction: by looking at examples from a task domain these systems formulate small collections of knowledge sufficient to explain what is going on. We invoke this process ourselves each time we hypothesise some property shared by members of one set of things that differentiates them from everything else. At its most general level, an inductive inference system is capable of discovering regularities that can explain observed phenomena in some domain. While most existing expert systems are based on knowledge obtained as above from a human expert, ability to discover regularities gives a new possible prescription for acquiring knowledge: the expert, instead of trying to specify the knowledge directly, guides an inductive inference system in its search for regularities in collections of examples drawn from his domain of expertise. The remainder of this section describes a few modern inductive inference systems, indicates the mechanisms they employ, and mentions one or two notable successes of each. No attempt is made at completeness: for a more comprehensive treatment the reader is referred to the survey papers by (Mitchell 1979) and (Diettrich and Michalski 1979).

Probably the most successful application of inductive inference techniques to the problem of knowledge acquisition is the Meta-DENDRAL program (Buchanan and Mitchell 1978). Meta-DENDRAL's original task was to discover the rules for the third component of DENDRAL, or in other words to develop a theory of how molecules fragment in the mass spectrometer. For each run the data consisted of a large number of known molecules together with their observed mass/abundance readings. The program first identified all possible places that the molecules could have broken in order to explain the observed fragments: only those breaks consistent with a 'weak' theory of what bonds can break were considered. Next a coarse search was made for possible rules to explain these breaks in terms of the local context of each broken bond. Finally the set of rules so obtained was refined, so that rules predicting breaks that did not occur were made more specific, and other rules were made more general. The rules resulting from this three-phase process not only accurately reflected expert knowledge, but also included previously unknown rules. When the same approach was taken for nuclear magnetic resonance spectroscopy, once more new and useful rules were found. So successful was this work that the rules found have been published as Chemistry.

Meta-DENDRAL is an example of a special-purpose inductive system: it can only be used for the particular induction tasks for which it was designed. (It does however embody powerful kernel ideas, such as using a weak model to guide the discovery of a strong one, that have wide applicability.) General induction systems, on the other hand, are able to attempt problems of discovering regularities in domains where no semantic information is available to guide the inference. This class of systems is differentiated primarily by constraints on the form that discovered knowledge can take, and to a lesser extent by the way that search for this knowledge is carried out.

Michalski's INDUCE package (Michalski 1978) takes the description of a number of objects of different classes and constructs one or more generalized descriptions of each class. The descriptions of both objects and classes are couched in VL21, a language based on first-order logic
but allowing typed variables. To find generalized descriptions of a class, the descriptions of all objects of that class are placed in one set (the positive instances) and all other objects into another set (the negative instances). A single description is then chosen from the positive set, and some of its atomic descriptors are built into more and more complex descriptions until a number are found that are sufficiently specific so that no object matching them can be in the negative instances. The best of them is selected as one of the descriptions of the class, the positive instances are reduced by removing all those that match this description, and the process repeated until all the original positive instances are covered by one or more of the generalizations. This system has also been applied to the problem of building expert systems, specifically for constructing a rule to test for a soybean disease. From the given descriptions of some hundreds of plants it was able to construct a generalized description of diseased plants that was more accurate diagnostically than a human expert working from the same data.

Vere's THOTH-P (Vere 1978) is an example of a system that, while it is less generally applicable than INDUCE, can still tackle a range of tasks. Its data is a set of pairs of objects viewed as before-and-after snapshots, and it attempts to find the smallest number of relational productions that explain the changes. A relational production specifies that in a given context some stated properties are invalidated and new ones created, where the context and properties are again expressed in a language derived from logic. The method used to find these relational productions is an exhaustive search for maximal common generalizations which expands exponentially with the number of pairs of objects, and so cannot be applied to more than a small amount of data. (This exponential time problem is shared by other 'complete' systems such as SPROUTER [Hayes-Roth and McDermott 1977].) Examples of its applications are finding the smallest number of primitive actions sufficient to explain a sequence of changes in the microcosm known as the 'blocks world', and discovering rules to change a restricted class of sentences from active to passive voice.

ID3 (Quinlan 1979a, b) is another general-purpose system, designed specifically for sifting knowledge from large numbers of objects. As with INDUCE, each object can belong to one of two classes, but must be described in a way much more constrained than that used by INDUCE. We presuppose a fixed collection of attributes or properties, each having a small number of possible values. Every object is then represented by the particular values it takes for each attribute. ID3 starts by selecting a workable subset (called a window) of these objects, and produces a rule in the form of a decision tree that accurately accounts for the class membership of the objects in the window. It then tests this rule on the other objects and forms a new window from the old one and the exceptions to the rule. The process repeats until a rule is discovered that is accurate for the entire collection of objects. Experiments with ID3 have been largely confined to finding pattern-based rules for the chess endgame king-rook versus king-knight. Rules for deciding whether the knight's side is lost 2- and 3-ply have been discovered from collections of up to two million objects, and these rules have been up to 5 times faster than search-style programs for the same problem. Results are reported in detail in (Quinlan 1980).

CONCLUSION

These examples outlined here give a taste of the exciting work that has been done on the production of knowledge-based systems. Many existing systems can perform at the level of a human expert, while others are not far behind. Altogether these systems must number in the hundreds, and, when one reflects that almost all of them have been built over the last seven or eight years, it seems obvious that expert systems are going to have a significant impact on computing in this decade.

REFERENCES


QUINLAN, J.R. (1979a), Induction over large data bases. Report

This book is the proceedings of an IFIP TC 2.5 Working Conference held in Baden, Austria, in December 1978.

The papers mainly deal with two areas, one with the general aspects of performance evaluation of numerical software that deals with the performance evaluation of numerical software in three specific areas — linear algebra, ordinary differential equations, and optimization. In addition, there are three papers dealing with reliability and warranty of numerical software and also the report of a panel discussion on use of mathematical software outside the mathematical software community.

Three sessions at the conference were devoted to general aspects of performance evaluation. The nine papers presented during these sessions illustrate how diverse questions of performance evaluation of numerical software can be. For example, papers by T.E. Hull and T.J. Dekker look at correctness proofs for numerical software while W.S. Brown discusses the difficulties of modeling a floating point number system, reviews a model presented by him in an earlier paper, and then shows how the model can be used in the development and analysis of a simple algorithm. J.N. Lyness discusses the fundamental question of what performance assessment of numerical software should mean and how it should be measured in those situations when no finite algorithm can be guaranteed to solve all the problems to the specified accuracy. Lyness points out the weaknesses of battery type testing and proposes use of performance profiles. The use of a statistical statement from these tests is much more informative than results of battery tests but are also more expensive. L.D. Fosdick discusses ways of detecting errors in programs. This is a topic of interest to everyone in computing but, as Fosdick points out, the published empirical knowledge of errors, error types and error rates is very limited. Fosdick also discusses use of data flow analysis to detect errors in programs.

Jordan discusses the impact of new machine architectures on linear algebra software. He attempts to compare the performance of several of the parallel machines available in solving linear equations using direct methods. For each purpose, the same software can not be tested on all machines. Special software is required for each machine to achieve the performance for which they were designed.

Other papers dealing with the general aspects of performance evaluation are those by J.R. Rice, J.A. Nelder, and B. Ford, C.S. Hodgson and D.K. Sayers. Rice discusses the experimental approach of testing mathematical software and suggests standardization of problem populations of about 50 to 100 members so that concise description of the computational experiment and brief, objective conclusions based on the results may be published in a journal while the raw data is made available. Interested persons Nelder discusses experiment design and its role in software evaluation. Ford, et al. discuss multi-machine evaluation of software and the problems associated with such evaluations.

Three papers dealing with performance evaluations of linear algebra software include I. Molchanov's paper which presents a survey of error analysis for the problem of solving systems of linear equations and for the eigenvalue problem, and presents some new augmentation results. Another paper by B. Smith who discusses the use of automated theorem provers to evaluate linear algebra software. Smith illustrates his approach by an example of Cholesky decomposition. The third paper, by J.K. Reid and I.S. Duff, discusses performance evaluation of codes for sparse matrix problems. Reid and Duff discuss the importance of testing sparse matrix software on a realistic battery of test problems because most real life sparse matrices have special features. They also discuss their collection of test problems which includes matrices of more than 1000 rows.

The three papers dealing with performance evaluation in ordinary differential equations show the wide range of activity in this area. The paper by H.J. Stetter discusses the possibilities of modeling the behaviour of the software or a specific component of some software. He illustrates this approach by two examples and suggests that both testing and modeling software together will help us to understand more fully the functioning and numerical software and to design better codes. The paper by P.J. Van der Houwen and J.G. Verwer discusses solving parabolic partial differential equations in two dimensions by approximation with ordinary differential equations. The authors discuss the difficulties involved in comparing various algorithmic possibilities introduced in the discussion, the splitting of the equations, and the handling of boundary conditions. W.H. Enright discusses testing codes for stiff and non-stiff ordinary differential equations using a testing package which includes a set of test problems.

The papers dealing with performance evaluation in optimization and non-linear equations are by P.E. Gill and W. Murray, K. Schittkowski, R.B. Schnabel, J.J. More, and F.A. Lootsma. Gill and Murray discuss the important aspects of performance, approaches used in testing and testing methodology. The authors conclude that many aspects of performance evaluation of optimization software are far from satisfactory. Schittkowski discusses a major project to compare optimization software, and presents results of testing six codes. Each code is subjected to 240 test runs on randomly generated test problems. Schnabel discusses modular development of software for uncontaminated optimization. Several advantages of this approach are discussed. More discusses standards for optimization software. He discusses the need for robust, reliable, transportable, usable, and valid software. He presents example of testing optimization software for robustness, reliability, performance. Lootsma discusses selection of non-linear programming codes in a particular environment by assigning priorities to relevant performance criteria.

The session dealing with reliability and warranty of numerical software includes papers by B. Niblett, E.L. Battiste, and C. Tanner. Niblett discusses various ways of protecting property rights in numerical software and Battiste discusses difficulties of providing reasonable warranties about the performance of software. Tanner discusses legal remedies for misrepresentation of software.

Also included in the book is the report of a panel discussion on the use of mathematical software outside the mathematical software community. A.M. Erisman lists some of the reasons why mathematical software has had limited use in application areas and calls for stronger communication ties with application communities.

To conclude, this book brings together the views of many leading research workers in the field of software evaluation and performance evaluation. Some of the issues discussed are common to all software — like program correctness and testing of software — while others are unique to mathematical software because it is dealing with problem spaces of very high dimension whose overall nature is not well understood. In addition, large scale performance evaluation tends to be very expensive and that is why one of the issues discussed in several papers is the collection of test batteries of realistic and expensive-to-solve problems.

It is an important book for those involved in designing and evaluating mathematical software.

G.K. GUPTA,
Monash University and
Asian Institute of Technology, Bangkok

Virtual Memory

J.L. Keedy*

Before the invention of virtual memory, a clear distinction was drawn between main memory (which was the memory level reserved for executing programs, i.e., the computational memory) and auxiliary memory (which was the set of magnetic media, e.g., discs and drums, used for the long-term storage of programs and data, i.e., the long-term memory). Since the relatively high cost of main memory led to only limited amounts being available, serious restrictions had to be imposed on the size of programs executing in the computational memory. To overcome this restriction, programs (including the operating system) were decomposed into several phases which could be overlayed into the same area of main memory under program control. Not only was it inconvenient for programmers to organise their programs in this way, it also led to many programming errors and above all to much duplication both of effort and of mechanisms. For example, in at least one commercially available operating system distinct overlaying mechanisms are provided for (i) the operating system, (ii) timesharing system programs, (iii) timesharing user programs, (iv) batch programs, (v) transaction processing applications, and (vi) remote job entry.

Virtual memory was originally invented mainly as an automatic solution to the problems of program overlaying, by extending the scope of computational memory to include a part of the auxiliary memory, i.e., by pretending that some of the disc/drum space was main memory. The processor accepts a single range of addresses for all parts of the computational memory, and examines a map set up by the operating system to determine whether a location addressed is currently in main memory or auxiliary memory. In the latter case, it cannot complete the current instruction, but instead generates a virtual memory fault interrupt so that the operating system can transfer the appropriate location into main memory and modify the map to indicate this. Once the location is in main memory the faulting process is permitted to continue. This unified addressing scheme relieves the programmer of all concerns about the actual size of main memory, and presents him with a view of the computational memory as if it were a 'one-level store'.

The address translation maps can be organised to reflect either fixed-length units (paging) or variable-length units (segments). In paging schemes the program has a single contiguous set of addresses, with the least significant p bits representing the offset from the start of a page (of size 2^p) and the higher order bits representing a page number. This page number is used as an index into the map, and the appropriate entry includes a presence bit to indicate whether the page is currently in main memory. One of the main advantages of paging schemes is that the allocation of space in both the main memory and the computational part of the auxiliary memory is relatively straightforward, since all pages have the same size.

The alternative to paging is the segmented virtual memory, which provides a map entry for each 'logical' segment of a program (e.g., each procedure or array). Compilers generate two dimensional addresses, consisting of a segment number (used as a map index) and an offset within segment. Since such segments have different lengths the map entry holds, in addition to a presence bit and a main memory start address, an indication of the segment's length. It is usual also to provide information about how a segment may be accessed (e.g., read only for constants, read/write for most data segments, execute or read/execute for code procedures). Segmentation facilitates hardware bounds checking — by comparing the segment offset with the segment length — and provides protection against errors such as attempts to execute data as code or to modify code or constants.

The main disadvantage of segmentation is that the operating system's task of allocating space in main memory and auxiliary memory is complicated by the need to handle units of different lengths. If, as is usual, the maps are themselves held in main memory, an additional memory access is necessary to translate a virtual address into a main memory address. Many systems provide special cache memory hardware to avoid the overhead of this additional memory cycle for most address translations.

The first virtual memory system, Atlas (Fotheringham, 1961) was a paging scheme. The first segmentation scheme was implemented on Burroughs B5000 computers (Burroughs Corporation, 1961), albeit in a slightly different way to that described above. The Multics system (Bensoussan, et al, 1972) and the ICL2900 Series (Keedy, 1977) have combined segmentation and paging. In fact the Multics system has taken the further step of providing a unified virtual memory which includes both computational and long-term memory, thus dispensing with the need for a separate technique for organising files.

The principles of virtual memory are more fully described by Denning (1970).

REFERENCES:

*Department of Computer Science, Monash University, Clayton, Vic, 3168.

A Fast and Stable List Sorting Algorithm

B. Cheek*

"QUICKSORT" has wide applicability as a fast exchange sorting technique. The algorithm is not however "stable" with respect to equal keys.

The algorithm "LISTSORT" described in this paper is stable but still retains most of the advantages of the "QUICKSORT" method.

Keywords and Phrases: Sorting, Quicksort, Linked Linear List, Stable Sorting.

CR Categories: 3.7.

INTRODUCTION

"QUICKSORT" is a widely used sorting technique which although relatively simple to understand and implement, still has an average running time proportional to n log n.

"QUICKSORT" sorts a file of records R[1], R[2],..., R[n] with keys K[1], K[2],..., K[n] by "partitioning" the records as follows. A record is selected from the file (normally the first record). The remaining records are arranged on either side of this record such that the keys of all the records to the left are less than or equal to the key of the selected record while all those records to the right have keys greater than the selected record. After "partitioning" the whole file can be sorted simply by independently sorting the left and right subfiles just created. This is done recursively using the same partitioning technique. The process stops when all subfiles contain less than two records.

The algorithm is usually implemented as shown in Program 1 which sorts the integer keys stored in the array K[1],..., K[r] into ascending order. Note that K[r+1] must exist and be larger than any of the keys K[1],..., K[r].

Program 1 (language Pascal)

begin
j:=-1;
end
until k[j] <= b;
if i < j then
begin
  temp:=k[i];
  k[i]:=k[j];
  k[j]:=temp;
end;
util j < i;
temp:=k[l];
k[i]:=k[j];
k[j]:=temp;
qusort (i+1);
qusort (i,n);
end;
end;

The program selects the first element from those to be sorted. The keys are then scanned from left to right searching for a key greater than the first. Next it scans from right to left until it finds a key smaller than the first. The two keys found are interchanged and the scanning continues until the left and right pointers cross. The first element (K[l] and the last element of the left sublist (K[j]) are interchanged to finish the partition. The program then calls itself recursively to sort the left and right partitions just formed.

Table 1 shows how the keys 4 3 5 2 4 8 3 would be sorted by Program 1. All exchanges in the first partitioning stage are shown, and the result of each subsequent partitioning stage is listed.

As can be seen from Table 1 "QUICKSORT" as implemented is not a "stable" algorithm (i.e., the original ordering of equal keys is not retained after sorting - see key 4). This instability occurs because the algorithm selects and exchanges records from both ends of the file when partitioning.

In many applications, for example when sorting using multiple keys, the sorting algorithm employed must be "stable".

The following algorithm sorts a singly linked list. It does this by continually dividing the list into two sublists such that all records in one sublist have key values less than the first key in the original list while all keys greater than or

* Computing Centre and Mathematics Department, University of Newcastle, NSW. Manuscript received 25th September, 1979.
LISTSORT

TABLE 1 Sorting Using "QUICKSORT" Algorithm

Note 1. The subscripts on equal keys indicate their original ordering, they are only documentary and have nothing to do with the sorting process.
Note 2. Underlining is used to indicate the keys currently being partitioned.

L[i] — a link field pointing to the next record in the list;
D[i] — the date portion of the record.

LISTSORT (LIST)

L1: If LIST is empty return
L2: Remove first record from LIST and call its key B
Step through the remainder of LIST partitioning it into two sublists, UPPER and LOWER such that
K[i] < B implies R[i] is placed in LOWER
K[i] >= B implies R[i] is placed in UPPER
L3: Perform LISTSORT (LOWER)
L4: Add first element which was removed at step L2 to end of SORTED LIST (create SORTED LIST if it does not exist).
L5: Perform LISTSORT (UPPER)
L6: Return

Note. The final list is called SORTED LIST.

Table 2 shows how LISTSORT works on the list 4 3 5 2 4 8 4 3. The state of each of the sublists after partitioning (L2:) can be seen by examining the columns. Indentation indicates the level of recursion.

The algorithm shown is compact and easy to understand, however several minor changes make the method far more suitable for use in practice. The major problem in Algorithm 1 is the stack handling which the recursive calls necessitate. In Algorithm 2 the recursion is removed.

Algorithm 2

Instead of using recursion or an explicit stack as in Algorithm 1 this algorithm stores the unsorted sublists in a new list (UNSORTED LIST). Of course we must know where each of these sublists ends. The easiest way to do this is to add a flag to the link field of each record to indicate "end of sublist". A scheme such as the following can be used

TABLE 2 Sorting Using Algorithm 2

LISTSORT

63 indicates that next record in sublist is record no. 63.
-63 indicates that this record is the last in the sublist, and
that the next sublist begins at record no. 63.
0 indicates that this record is the last record both in
this sublist and in UNSORTED LIST.

LISTSORT (LIST)
L1: If LIST is empty go to L5.
L2: Set a variable B equal to the key of the first record in
LIST. Place this record in UPPER.
L3: Step through remainder of LIST partitioning it into
two sublists UPPER and LOWER such that
K[i] < B implies R[i] is placed in LOWER
K[i] >= B implies R[i] is placed in UPPER
L4: Place the UPPER sublist on the front of UNSORTED
LIST (create UNSORTED LIST if it does not exist).
Rename the LOWER sublist LIST (the previous LIST
is no longer needed)
Go to L1
L5: If UNSORTED LIST is empty the sorting is comple­
ted, the sorted records are in SORTED LIST, rename
this LIST and return.
L6: Remove the first record from UNSORTED LIST (this
record was used as a pivot in a previous partitioning
and is in its final position), place it at the back of
SORTED LIST.
Remove the first sublist from UNSORTED LIST, this
sublist now becomes LIST.
Go to L1

Algorithm 2 produces many singleton lists. It would
be possible to modify the algorithm so that this does not
happen, however the overhead involved results in longer
sorting times.

Table 3 documents how the revised algorithm works
on the sample keys. The lists are shown before step L1 is
executed each time. Note. "*" indicates the end of a sub­
list.

Implementation of Algorithm 2

The following Pascal procedure carries out sorting as
described in Algorithm 2. Note. At step L4 LIST is placed
on the front of UNSORTED LIST before each partitioning
stage. If this is not done another link pointer to store the
start of UNSORTED LIST is necessary.

Program sort (input, output):
begin
{This procedure uses the LISTSORT algor­
ithm to sort a linked linear list which has
previously been stored in "sortrec".
"sortrec.key" contains a pointer to the
next record
End of list is indicated by "0" in the link
field.
End of sublist is shown by negating the link.
"sortrec[0]" is used only to point the first ele­
ment in the list or when partitioning to the
first element of LOWER, it does not contain
a record to sort.

<table>
<thead>
<tr>
<th>List</th>
<th>Unsorted List</th>
<th>Sorted List</th>
</tr>
</thead>
<tbody>
<tr>
<td>41 31 5 2 42 8 43 32</td>
<td>41 5 42 8 43</td>
<td>2</td>
</tr>
<tr>
<td>31 2 32</td>
<td>empty</td>
<td>2 31 5 42 8 43</td>
</tr>
<tr>
<td>2</td>
<td>31 2 32 41 5 42 8 43</td>
<td>2</td>
</tr>
<tr>
<td>empty</td>
<td>31 2 32 41 5 42 8 43</td>
<td>2</td>
</tr>
<tr>
<td>31</td>
<td>41 5 42 8 43</td>
<td>2 31</td>
</tr>
<tr>
<td>empty</td>
<td>31 2 41 5 42 8 43</td>
<td>2 31</td>
</tr>
<tr>
<td>41 5 42 8 43</td>
<td>empty</td>
<td>2 31 32</td>
</tr>
<tr>
<td>empty</td>
<td>41 5 42 8 43</td>
<td>2 31 32</td>
</tr>
<tr>
<td>5 42 8 43</td>
<td>empty</td>
<td>2 31 32 41</td>
</tr>
<tr>
<td>42 43</td>
<td>5 8</td>
<td>2 31 32 41</td>
</tr>
<tr>
<td>empty</td>
<td>43 5 8</td>
<td>2 31 32 41</td>
</tr>
<tr>
<td>5 8</td>
<td>empty</td>
<td>2 31 32 43</td>
</tr>
<tr>
<td>empty</td>
<td>8</td>
<td>2 31 32 41 42 43</td>
</tr>
</tbody>
</table>
empty
| empty    | empty        | 2 31 32 41 42 43 |

"start" is the list head for the SORTED
LIST
"uperstrt" is the list head for UPPER
partition.}

var firstime : boolean;
listend, liststrt, current, b, lower, upper, start,
uperstrt: integer;
begin
firstime:=true;
listend:=0;
current:=sortrec[0].link;
while current Oo do
begin
if current >0 then
begin
b:=sortrec[current].key;
lower:=0;
uperstrt:=current;
upper:=current;
current:=sortrec[current].link;
while current >0 do
begin
if sortrec[current].key < b then
begin
sortrec[lower].link:=current;
lower:=current
end
else
begin
sortrec[upper].link:=current;
upper:=current
end;
current:=sortrec[current].link;
end;
end;
{step L5}
{step L1}
{step L2}
{step L3}
end;
end;
end;}
LISTSORT

srec[upper].link:=current;
src[lower].link:=superstrtr;
current:=src[0].link
end
else
begin {step L6}
if firstime then
begin {sets up pointer to smallest key}
start:=src[0].link;
firstime:=false
end;
current:=current;
srecpistend .link:=current;
listend:=current;
current:=src[current].link
end;
end;
srec[0].link:=start {finished set up pointer to smallest key}
end;

Average Running Time of Algorithm 2

The speed of Algorithm 2 is not significantly affected by the presence of equal keys (Sedgewick 1977), therefore to simplify the analysis we will assume that all keys in the list are distinct.

The time taken to sort a list of size "n" using Algorithm 2 can be found once the following quantities are known:

A(n) The number of times step L2 is executed;
B(n) The number of times step L5 is executed;
C(n) The number of comparisons performed during partitioning at step L3.

If we define the time taken to execute step i as Ti then the total running time of Algorithm 2 can be expressed as:

\[ T(n) = A(n) \times (T1+T2+T4) + B(n) \times (T1+T5) + B(n-1) \times (T6) + C(n) \times \text{Time taken to partition one element at L3} \]

All the following relations which will be derived rely on the observation that if the keys in the list being partitioned are in random order, then this randomness will be retained in both sublists produced by the partitioning process (L3).

Step L3 makes by far the largest contribution to the running time and therefore it is appropriate to analyse the associated quantity C(n) first.

The total number of key comparisons made at step L3 is equal to the number of comparisons made during the first partitioning \((n-1)\); plus the comparisons made in all later stages, i.e.,

\[ C(n) = (n-1) + \frac{1}{n} \sum_{k=1}^{n} (C(k-1) + C(n-k)) \]

or

\[ C(n) = (n-1) + \frac{1}{n} \sum_{k=1}^{n} C(k-1) \]

Solving this recurrence relation and using \( C(0) = 1 \) we find

\[ C(n)/n+1 = 2/n+1 + 2/n + 2/n+1 + \ldots + 2/3 \]

\[ - (2/n(n+1) + 2/(n-1)n + \ldots + 2/6) \]

\[ = 0 (n \ln(n)) \]

Similarly the recurrence relations and initial conditions for A(n) and B(n) are:

\[ A(n) = 1 + 2/n \sum_{k=1}^{n} A(k-1); A(1) = 1 \]
\[ B(n) = 2/n \sum_{k=1}^{n} B(k-1); B(2) = 3 \]

Therefore we find:

\[ A(n) = n \]
\[ B(n) = n + 1 \]

On local PDP 11/45 the times (in seconds) to execute each of the steps are:

L1 . . 6.85e-5
L2 . . 13.4e-5
L4 . . 10.7e-5
L5 . . 5.1e-5
L6 . . 15.3e-5

Finally, the time taken to partition one element is dependent on the length of the keys, for the simplest key type (integers) the local PDP 11 takes 23.8e-5 seconds to partition one record.

Therefore for simple keys on a PDP 11/45 computer, average running time is:

\[ T(n) = A(n) \times 30.95e-5 \]
\[ + B(n) \times 11.95e-5 \]
\[ + (B(n)-1) \times 15.3e-5 \]
\[ + C(n) \times 23.8e-5 \]

So the analysis confirms that the partitioning stage (L3) accounts for approximately 70% of running time for 50 keys, rising steadily until step L3 uses almost 85% of total running time when 5000 keys are being sorted. (These figures are for simple integer keys on a PDP 11/45.)

The above omits two factors, neither of which have any significant effect on running time. First, minor housekeeping both at the start and finish of the algorithm is not included. Second, after each element is partitioned at step L3 a check for end of list is made. For simplicity the final successful check has been ignored in these calculations.

Adding an Insertion Sort

"QUICKSORT" is very efficient for "large" files, however, it is relatively inefficient for "small" files. Of course, "QUICKSORT" must generate many "small" files to sort a single "large" file. Hoare (Hoare, 1962) suggested that these "small" subfiles should be sorted using an algorithm more suitable for small numbers of records. "QUICKSORT" algorithms usually make use of a simple insertion sort for these "small" files. Not surprisingly this approach is also useful for LISTSORT.

The introduction of the insertion sort to "LISTSORT" does not affect its "stability". The "LISTSORT" strategy ensures that equal records will be in their original relative order when the subfiles are passed to the insertion sort and this ordering will, of course, be maintained during the "stable" insertion sort.

Algorithm 3 uses a List Insertion Sort for "small" sublists (those of length < "M").

Algorithm 3

Obviously to implement List Insertion Sorting for "small" subfiles we must know the length of the subfiles. The best way to do this is to count the length of each subfile as it is being produced (when partitioning). The choice of value for "M" is not critical but for most
implementations ‘M’ should be between 6 and 15.
Rather than store the exact length of each sublist in UNSORTED LIST it is split into two lists SHORT UNSORTED and LONG UNSORTED. Sublists smaller than “M” are placed on SHORT UNSORTED while other sublists go to LONG UNSORTED.

LISTSORT (LIST)
L1: Set a variable B equal to the key of the first record in LIST
Step through LIST partitioning it into two sublists UPPER and LOWER such that
\[ K[i] < B \implies R[i] \text{ is placed in LOWER} \]
\[ K[i] \geq B \implies R[i] \text{ is placed in UPPER} \]
L2: If length of UPPER < M
then
place UPPER on front of SHORT UNSORTED
else
place UPPER on front of LONG UNSORTED
Rename the LOWER sublist LIST
If length of LIST >= M then go to L1
L3: Perform “List Insertion Sort” on LIST
Add LIST to the back of SORTED LIST
L4: If both SHORT UNSORTED and LONG UNSORTED are empty the sorting is completed, the sorted records are in SORTED LIST. Rename this LIST and return. If the key of the first record of SHORT UNSORTED is less than the first key of LONG UNSORTED
then
Remove the first record from SHORT UNSORTED, place it at the back of SORTED LIST
Remove the first sublist from SHORT UNSORTED, this now becomes LIST
Go to L3
else
Remove the first record from LONG UNSORTED, place it at the back of SORTED LIST
Remove the first sublist from LONG UNSORTED, this now becomes LIST
Go to L1

Choice of Bound (“B”) – Worst Case Behaviour
Once again, like “QUICKSORT”, LISTSORT exhibits its worst behaviour for lists which show a marked degree of pre-order. Consider sorting the keys \( K[1] \ldots K[n] \) which are already in order. \( K[1] \) is the partitioning element and LOWER will be empty after the first partitioning, UPPER consists of \( K[2] \ldots K[n] \). Thus the algorithm produces sublists of length \( n,(n-1),(n-2), \ldots \) and its total running time will be proportional to \( n^2 \) squared. Lists in reverse order produce almost the same behaviour.
QUICKSORT algorithms often use the median of the first, last and middle elements in the subfile being sorted as the partitioning element. If this is done the algorithm works well for files showing a marked degree of pre-order (although, of course, it still exhibits \( O(n^2) \) worst case behaviour). Unfortunately as we are processing lists we do not have random access to the records and so cannot locate either the middle or last record without sequentially reading the complete list (a comparatively slow process).
For “LISTSORT” a method which both gives protection against “worst-case” behaviour and which minimises the extra time spent scanning the list is desirable. One way to do this is to use the median of the keys of the second
and two other records approximately one-third of the way through the list as the pivot value (“B’”). This compromise results in roughly equivalent sorting times for ordered or randomly distributed keys. Other choices may well be more appropriate for other applications.
This scheme is implemented in the following algorithm.

Algorithm 4
LISTSORT (LIST)
L1: Set a variable B equal to the median of the keys of the second and two records one-third of the way through LIST
Step through LIST partitioning it into two sublists UPPER and LOWER such that
\[ K[i] < B \implies R[i] \text{ is placed in LOWER} \]
\[ K[i] \geq B \implies R[i] \text{ is placed in UPPER} \]
If length of LOWER = 0 then remove the first element with key = B from UPPER and place it on LOWER
L2: If length of UPPER < M
then
place UPPER on front of SHORT UNSORTED
else
place UPPER on front of LONG UNSORTED
Rename the LOWER sublist LIST
If length of LIST >= M then go to L1
L3: Perform “List Insertion Sort” on LIST
Add LIST to the back of SORTED LIST
L4: If both SHORT UNSORTED and LONG UNSORTED are empty the sorting is completed, the sorted records are in SORTED LIST, rename this LIST and return
If the key of the first record of SHORT UNSORTED is less than the first key of LONG UNSORTED
then
Remove the first sublist from SHORT UNSORTED, this now becomes LIST
Go to L3
else
Remove the first sublist from LONG UNSORTED, this now becomes LIST
Go to L1

Note. The record whose key is used as the partitioning element will not necessarily be in its final position after partitioning is complete and so must take part in subsequent partitioning operations. (If it were placed in its final position the sort would become “unstable”.)
A stack is once again necessary to store the length of each sublist stored in LONG UNSORTED. This information which will be needed when calculating B later can be most efficiently calculated when the sublist is being formed. Although this stack can in theory become as large as \( n/m+1 \), in practice it normally does not grow to a size greater than \( \log n \).

Running Times of Some Common Sorting Algorithms
These following figures give some indication of the relative speeds of the four algorithms mentioned. Of course it is important to realise that the time taken to sort a particular set of records is not dependent simply on the sorting algorithm used; many other factors (e.g. type of computer, length of record and distribution of key values) contribute to the final run time.
The times shown are for programs run on a PDP.
One of the conclusions obtained is that a cross interaction among research workers in each field would lead to more fruitful results. Philosophies, objectives and analytical techniques between them have similarities and differences. WILLSKY, A.S. (1979): Digital Signal Processing and Control and Estimation Theory: Points of Tangency, Areas of Intersection, and Parallel Directions, MIT Press, Cambridge, Mass. 253pp. $A33.75.

CONCLUSION

"LISTSORT" and "QUICKSORT" are similar in many respects, however there are several significant differences which affect their relative suitabilities for certain applications.

1. If the sorting algorithm must be stable then "LISTSORT" is more suitable.
2. "QUICKSORT" is a true minimum storage sort, whereas "LISTSORT" must have an extra link field associated with each record. Therefore if available memory is limited "QUICKSORT" may be preferable.
3. "QUICKSORT" physically reorders the records in the computer’s memory and so processes which require direct access to the records (e.g. binary searching) are possible immediately after sorting. "LISTSORT" produces a Linked Linear List which must undergo further processing before direct access to the records is possible.
4. Obviously if the records to be sorted are stored in a Linked Linear List then "LISTSORT" is particularly appropriate.
5. The "median of three" modification is implemented more elegantly in "QUICKSORT" algorithms than has been done for "LISTSORT".
6. "QUICKSORT" partitions by exchanging records as necessary, an operation which takes longer than the pointer updating employed by "LISTSORT". (It must be noted however that while "QUICKSORT" exchanges approximately 1/3 of the records in a partitioning stage "LISTSORT" must update all link fields.) For "small" records this exchanging does not significantly affect the running time of "QUICKSORT", however for larger records the time spent exchanging records becomes significant. As a result the speed of "QUICKSORT" is dependent on record size. The running time of "LISTSORT" (which only updates link pointers) is unaffected by record size. (This problem is normally solved in "QUICKSORT" programs by the introduction of "tags" to records and subsequently exchanging only the tags. When this is done however the limitations documented for "LISTSORT" in Points 2 and 3 apply equally to "QUICKSORT".)

7. Another apparent advantage of QUICKSORT over LISTSORT (Algorithm 4) is that in the former, the stack space is held to log n, even in the worst case. However if the stack size in Algorithm 4 exceeds log n this indicates sub-optimal behaviour (complexity approaching O(n^2)), in which case a change to a more suitable method can be made to complete the sort. Obviously then, the choice of whether to use "QUICKSORT" or "LISTSORT" for a particular application will depend on considerations such as those set out above.

ACKNOWLEDGEMENT

I would like to thank Professor A.J. Guttmann, both for introducing me to the subject of sorting and for his assistance in the preparation of this document.

REFERENCES


Book Review


The content of this book is well explained by its long title. Appropriately, the book explores several specific areas of current research in the field of digital signal processing and modern control theory. By in-depth study of key results in each field, the author managed to show the similarities and differences in the philosophies, objectives and analytical techniques between them. One of the conclusions obtained is that a cross interaction and collaboration among research workers in each field would lead to more fruitful results.

There are six chapters in this book. Each of the first five chapters examines a specific area: Chapter 1 discusses the various approaches of stability analysis — viz., Lyapunov functions, passivity and frequency domain criteria. Chapter 2 considers the problems of parameter identification, linear prediction, least squares, and Kalman filtering. Chapter 3 compares state-space realization techniques and the realization of digital filters. Chapter 4 addresses the more complicated multi-dimensional problems such as those found in image processing. Chapter 5 examines issues in nonlinear system techniques. Chapter 6 consists of a series of questions raised by the author on the possibilities of cross culture between digital signal processing and modern control theory.

The book is fairly readable; its presentation is concise and relies heavily on the reader’s familiarity with the subject. Research

KEN KWONG
Capricornia Institute of Advanced Education
TWONET: A New Program for the Computation of a Two-Neighbour Network

W.T. Williams*

A network based on the first two nearest neighbours produces "network-groups" comparable with classificatory groups, but with an overt internal network structure. The procedure has proved powerful in a number of biological applications, but the program originally used had several shortcomings. An improved program, TWONET, faster and with a more informative output, is now available on the TAXON library of the CSIRO Cyber 76 computer in Canberra.

Keywords: classification, ordination, networks, minimum spanning tree.
CR categories: 3.12, 5.19, 5.32

In a recent investigation into methods of analysis of genetic resources data Williams, Burt, Pengelly and Robinson (1980) described a simple program which, given an inter-element distance matrix, computes a network based on the first two nearest neighbours. This provides an informative compromise between a classification and a minimum spanning tree, in that it produces 'network-groups' of elements linked internally by 'strong' bonds, the groups themselves being linked externally by 'weak' bonds. This procedure has three advantages over a conventional classification: (i) the internal structure of the groups is evident; (ii) the number of groups is an inherent property of the data, not an arbitrary estimate of the user, and (iii) the existence in a prior classification of a 'non-conformist group', i.e., a group of elements associated simply because they are rather unlike everything else, is immediately obvious. The method has proved useful in further studies in genetic resources data (Robinson, Burt and Williams 1980; Burt, Pengelly and Williams 1980) and a study of inter-relationships between the families of monocotyledons (Clifford and Williams 1980); papers are also in preparation on its application in north-eastern Australia to the ecology of rain-forests (L.J. Webb and J.G. Tracey) and mangroves (J.S. Bunt).

The original program was cumbersome, used an excessive amount of storage-space, and was slow-running. Moreover, the output was relatively uninformative in that it did not specify the number or constitution of the network-groups, which could only be ascertained by the laborious task of drawing out the entire network. In view of the evident usefulness of the method, it was therefore decided to write a more efficient program.

The purpose of this communication is simply to announce the existence and availability of a new program TWONET. Given the inter-element distance matrix, it now provides output in three parts: (i) A list which, for each element, enumerates those elements of serial number higher than its own (if any exist) to which it is to be directly joined. All element numbers will appear at least once somewhere in this list. (ii) A summary of the network-groups by their constituent elements. The latter are listed in each group in numerical order, the within-group configuration being obtained from the list previously printed. (iii) A summary of weak links between groups, specifying the individual elements concerned, as an inter-group upper triangle. Program TWONET is available on the TAXON library of the Cyber 76 computer in the CSIRO Division of Computing Research, Canberra.

REFERENCES


*Australian Institute of Marine Science, Private Bag No. 3, Townsville, Qld., 4810. Manuscript received 29th January 1980.
Communications. Papers submitted for publication should be sent to The Editor, The Australian Computer Journal, Department of Computer Science, Monash University, Clayton, Victoria, 3168, Australia. All papers submitted will be subject to consideration by an Editorial Committee in consultation with referees authoritative in their particular fields. Authors will be notified as soon as possible whether the paper has been accepted for publication, of the date of the journal when it will probably appear, and of any modifications suggested by the referees.

Submissions should include the name and address of the person to whom the proof is to be sent, the name and address of the author’s organisation or laboratory, and the author’s title and full name. Honours and degrees may be included for editorial reference but these will not be published. Contributors who reside outside Australia should, wherever practicable, nominate a representative in Australia to whom their proofs may be sent for correction.

Eligibility of Papers. Papers submitted must be in English and they may be in the categories of review articles, exploratory articles, research articles or papers on advances in application. Papers submitted should be certified by the authors as original and that they have not been previously published or are not being considered for publication in any other publication. If accepted for publication in The Australian Computer Journal a paper may not be published elsewhere in the same form, in English or any other language, without the consent of the Editor. Submission of a paper will be held to imply that it has been cleared for publication from military or commercial security standpoints, and no responsibility will be accepted by the Editor or the Australian Computer Society for any consequences, arising from the failure by an author to obtain such a clearance.

Form of Papers. A high standard of preparation is expected of papers submitted for publication and those not deemed satisfactory will be returned to the authors for revision.

Papers should be in double-spaced typing on one side only of sheets of A4 size with 3 cm wide margins. A progressive count of each hundred words of the paper should be shown in the left-hand margin. Normally papers in excess of 4000 words will not be accepted where each Table or Figure should be counted as about 100 words. Three copies of the paper should be submitted. The title of the paper should be as short as possible and should be set out clearly at the head of the first sheet. Each succeeding sheet should show a shortened version of the title suitable for a running title on each page of the published work. The sheets of the paper should be consecutively numbered in Arabic numbers and firmly stapled together in the top left-hand corner. Each paper must be accompanied by a summary of about 100 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate Key Word list and a list of relevant Computing Review categories (see last page). The first sheet of the paper should show the exact form of the author’s name required for printing under the heading (last name, and given name or initials only) and the name and address of his organisation suitable for inclusion as a footnote.

Sub-headings. Sub-headings and paragraphs of papers should be clearly indicated. The hierarchy and required indentation of sub-headings and the sub-structures of paragraphs must be presented unambiguously.

Figures. Diagrams and flow-charts must be well drawn in Indian ink and clearly lettered on stiff white paper or Bristol board. Directional arrows must be clearly indicated. The diagram should be approximately twice the size of the finished block. The size limits for finished blocks are as indicated for tables. Each diagram should be on a separate sheet, packed flat and have the author’s name and figure number on the back. Where photographs are submitted, glossy prints are required, clips should not be used and heavy pressure should be avoided when writing on the back. The numbering, heading and positioning of figures should be shown as for tables.

Programs which are short and which relate to important algorithms will be published. All programs will be typeset and sent to the authors for careful checking.

Mathematical Formulae. Mathematical formulae must be clearly written; avoid symbols or arrangements that are difficult to set up. They should be consistent with the printing width available.

Appendices. If papers require the presentation of extensive mathematical treatments or supporting data (where quotation of references will not suffice) such detail should be included in appendices at the end of the paper. Appendices should be clearly identified as such, be consecutively numbered in Arabic if the paper has more than one, and have self-explanatory headings. Subject matter should be prepared in accordance with the rules for papers.

References. References should be identified in the text thus: Curtis and Osborne (1966); (Cohen and Nguyen-Dinh, 1966). Where the paper cited has more than two authors, the names of all the authors should be given when
Notes on Submission of Papers

Reference is first made, e.g. (Fair, Flowerdew, Munro and Rowley, 1966). Subsequent citations of the same paper should appear as (Flair et al., 1966). Where more than one paper by the same authors has appeared in the one year the reference should be shown as follows: Lance and Williams (1966a); Lance and Williams (1966b); Lance and Williams (1966a, b); (Lance and Williams, 1966a, 1967).

At the end of the paper, references should be given in alphabetical order according to the names of the first authors of the publications quoted, names and prefixes being entered under the prefix. The references should include the authors’ initials, year of publication, title of paper, the name of the journal, volume and first page number. References to books and monographs should include year of publication, the title and edition, first page number of the section referenced, place of publication and the name of the publisher. Examples:


It is the responsibility of contributors to check the accuracy of references in their papers before submission. The official abbreviations to be used are those of Mathematical Reviews and in particular this Journal is referred to as Austral. Comput. J.

Proofs. The authors are responsible for seeing that their type-scripts are in the correct form for publication. Proofs are sent to the authors or their nominated representatives to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alternatives may have to be disallowed or the cost charged against the author. The symbols used to indicate any corrections should be those laid down in the "Style Manual" published by the Commonwealth Government Printer, Canberra, pp. 76-79.

Reprints. Fifty reprints will be supplied free of cost. A price list and order form will be provided by the printer, with the proofs, for notification of any additional reprints required.

Correspondence. All correspondence submitted for publication should be addressed to the Editor. Letters must be typed, well presented, dated and clearly indicate the reference to which discussion is directed. They must be signed to show the correspondent’s name and address in typescript. Tables and mathematical formulae included should comply with the rules for papers.

Copyright. Authors are requested to assign the copyrights for their paper to the Australian Computer Society Incorporated which in turn may grant the right to reproduce the article to a third party: an agreement form is sent to the authors of accepted papers for this purpose.

Invitation to Contributors to Submit Short Communications to

"The Australian Computer Journal"

Introduction

There are a number of classes of documents relating to Computer Science research and Data Processing Practice being currently produced in Australia, the existence of which is not commonly known either by fellow practitioners in Australia or overseas. This invitation solicits Short Communications by which information about work of interest or novel ideas can be promulgated.

Classes of Documents

1) Communications regarding new ideas.
2) Internal reports discussing Data Processing applications implemented, of a nature thought by the Editorial Panel to be novel or of interest to A.C.J. readers.
3) Internal Research reports of Tertiary Educational organizations and other bodies involved in Computer Science research (e.g., CSIRO).
4) Post Graduate Research Theses accepted for degrees. Referees of classes 2 to 4 above will be asked to comment more on the style and presentation of the communication and the interest of the reported work than on the correctness in detail of the work itself. It is assumed that work in these classes would have already, in effect, been thoroughly “refereed” by their own or other organisations.

The referees of class 1 will require that both the stylistic content be satisfactory and the technical content be original, without, at the same time, necessarily being of interest to a significant proportion of ACJ readers.

All documents summarised in short communications should be publicly available.

Form of Communication

The form of the communication should be similar to the normal Journal articles with

1) Title, Author and Affiliation
2) Abstract including Key Words and Computing Review Categories
3) Objectives, Scope, Main Development, and Conclusions of work
4) References in ACJ format
5) Source of supply of related documents and price.

Communications should be of one printed page length, about 500 words of main text.

Purpose of Communications

The purpose of the short communication is

a) that ACJ readers should be informed of work of interest,
b) that information about Australian research work be abstracted and published overseas,
c) that a medium for quick reporting of new ideas in Computer Science be available in Australia.

Notes for the Guidance of Referees

GENERAL

The Referees are selected by the Editor on a personal basis. If the person selected considers he is unable (because of lack of competence or for any other reason) to undertake the task he should not pass the paper to a colleague but send it to the Editor by return post. If the Referee does return a paper for this reason then the Editor would welcome suggestions of an alternative Referee.

Referee

The Referee's comments should not be included in a covering letter to the Editor but should be typed on the report form provided, so that the Editor can send the comments to the author without disclosing the Referee's identity and without having to retype the comments. The paper should be returned to the Editor with the Referee's comments.

Referees are requested to make every effort to deal with papers as expeditiously as possible, and a target date of one month would seem reasonable for all concerned (Referee, Author, Editor and Publisher). The Editor makes every effort to avoid overworking particular individuals.

TYPES OF PAPERS

Full Papers

The Referee should feel free to comment on the style of English, the spelling and punctuation, the style of presentation of Diagrams, Tables, Appendices and any other stylistic matter which is found unsatisfactory.

A Referee may find it convenient to classify a paper before examining it in detail, as the considerations which make publication desirable vary with different types of papers.

Having done so, he may then ask himself specific questions and report accordingly. A number of categories into one of which a paper might fall, and the respective questions that might be asked, are as follows:

(a) Papers which make significant contributions to computing knowledge:
   Does its originality or fundamental nature make it a significant contribution to computing knowledge?

(b) Papers which collect existing knowledge and summarize it in a form convenient for reference:
   Does it provide a broad and critical survey or does it merely repeat information readily available in a textbook or other accessible publication?

(c) Papers which describe works of magnitude, or which are unusual or otherwise notable:
   Is the work of such importance that a permanent record should be made?
   Will other computing professionals in this field be anxious to have details of it?
   Are there novel or unusual features in the work?
   Will the prestige of Australian computing professionals be enhanced by circulation overseas?
   Will it provide a useful historical record of methods in use at the time?
   Does it define a methodology which could find broader application?
   Does the paper describe an application of particular current or lasting interest?

(d) Papers which discuss the investigation of problems of interest or of unusual magnitude:
   Does it provide valuable data for others in the same field?
   Is it of general interest?
   Is it of special interest to a limited number only, but nevertheless of great value to them?

(e) Papers which deal with professional or ethical matters in such a way as to promote higher professional standards:
   Is it constructive and likely to be effective?

REFEREES' REPORTS

A Referee is at liberty, and is invited, to submit as comprehensive a report as he desires. He should state whether the author has conformed to "ACJ Notes on Submission of Papers" and, if not, in what respects. The Referee will greatly assist if, in his report, he recommends that a paper should be:

(a) published in full;
(b) published after being reduced in length;
(c) published after suitable modification or amendment;
(d) published with additional information;
(e) not published.

Computing Review Categories:

1. GENERAL TOPICS AND EDUCATION
   1.0 GENERAL
   1.1 TEXTS, HANDBOOKS
   1.2 HISTORY, BIOGRAPHIES
   1.3 INTRODUCTORY AND SURVEY ARTICLES
   1.4 GLOSSARIES
   1.5 EDUCATION
      1.50 General
      1.51 High School Courses and Programs
      1.52 University Courses and Programs
      1.53 Certification Degrees; Diplomas
      1.59 Miscellaneous
   1.9 MISCELLANEOUS

2. COMPUTING MILIEU
   2.0 GENERAL
   2.1 PHILOSOPHICAL AND SOCIAL IMPLICATIONS
      2.10 General
      2.11 Economic and Sociological Effects
      2.12 The Public and Computers
      2.19 Miscellaneous
   2.2 PROFESSIONAL ASPECTS
   2.3 LEGISLATION; REGULATIONS
   2.4 ADMINISTRATION OF COMPUTING CENTRES
      2.40 General
      2.41 Administrative Policies
      2.42 Personnel Training
      2.43 Operating Procedures
      2.44 Equipment Evaluation
      2.45 Surveys of Computing Centres
      2.49 Miscellaneous
   2.9 MISCELLANEOUS
Notes for the Guidance of Referees

3. APPLICATIONS
   3.1 NATURAL SCIENCES
      3.10 General
      3.11 Astronomy; Space
      3.12 Biology
      3.13 Chemistry
      3.14 Earth Sciences
      3.15 Mathematics; Number Theory
      3.16 Meteorology
      3.17 Physics; Nuclear Sciences
      3.19 Miscellaneous
   3.2 ENGINEERING
      3.20 General
      3.21 Aeronautical; Space
      3.22 Chemical
      3.23 Civil
      3.24 Electrical; Electronic
      3.25 Engineering Science
      3.26 Mechanical
      3.29 Miscellaneous
   3.3 SOCIAL AND BEHAVIORAL SCIENCES
      3.30 General
      3.31 Economics
      3.32 Education; Welfare
      3.33 Law
      3.34 Medicine; Health
      3.35 Political Science
      3.36 Psychology; Anthropology
      3.37 Sociology
      3.39 Miscellaneous
   3.4 HUMANITIES
      3.40 General
      3.41 Art
      3.42 Language Translation and Linguistics
      3.43 Literature
      3.44 Music
      3.49 Miscellaneous
   3.5 MANAGEMENT DATA PROCESSING
      3.50 General
      3.51 Education; Research
      3.52 Financial
      3.53 Government
      3.54 Manufacturing; Distribution
      3.55 Marketing; Merchandising
      3.56 Military
      3.57 Transportation; Communication
      3.59 Miscellaneous
   3.6 ARTIFICIAL INTELLIGENCE
      3.60 General
      3.61 Induction and Hypothesis-Formation
      3.62 Learning and Adaptive Systems
      3.63 Pattern Recognition
      3.64 Problem-Solving
      3.65 Simulation of Natural Systems
      3.66 Theory of Heuristic Methods
      3.69 Miscellaneous
   3.7 INFORMATION RETRIEVAL
      3.70 General
      3.71 Content Analysis
      3.72 Evaluation of Systems
      3.73 File Maintenance
      3.74 Searching
      3.75 Vocabulary
      3.79 Miscellaneous
   3.8 REAL-TIME SYSTEMS
      3.80 General
      3.81 Communications
      3.82 Industrial Process Control
      3.83 Telemetry; Missiles; Space
      3.89 Miscellaneous
   3.9 MISCELLANEOUS

4. SOFTWARE
   4.0 GENERAL
      4.1 PROCESSORS
         4.10 General
         4.11 Assemblers
         4.12 Compilers and Generators
         4.13 Interpreters
         4.19 Miscellaneous
      4.2 PROGRAMMING LANGUAGES
         4.20 General
         4.21 Machine-Oriented Languages
         4.22 Procedure- and Problem-Oriented Languages
         4.29 Miscellaneous
   4.3 SUPERVISORY SYSTEMS
      4.30 General
      4.31 Basic Monitors
      4.32 Multiprogramming; Multiprocessing
      4.33 Data Base
      4.34 Data Structures
      4.35 Operating Systems
      4.39 Miscellaneous
   4.4 UTILITY PROGRAMS
      4.40 General
      4.41 Input/Output
      4.42 Debugging
      4.43 Program Maintenance
      4.49 Miscellaneous
   4.5 PATENTS, SOFTWARE
   4.6 SOFTWARE EVALUATION, TESTS, AND MEASUREMENTS
   4.9 MISCELLANEOUS

5. MATHEMATICS OF COMPUTATION
   5.0 GENERAL
      5.1 NUMERICAL ANALYSIS
         5.10 General
         5.11 Error Analysis; Computer Arithmetic
         5.12 Function Evaluation
         5.13 Interpolation; Functional Approximation
         5.14 Linear Algebra
         5.15 Nonlinear and Functional Equations
         5.16 Numerical Integration and Differentiation
         5.17 Ordinary and Partial Differential Equations
         5.18 Integral Equations
         5.19 Miscellaneous
   5.2 METATHEORY
      5.20 General
      5.21 Logic; Formal Systems (includes: theorem proving; excludes: switching, Boolean algebras, ternary logic, etc.)
      5.22 Automata: finite-state; cellular; stochastic; sequential machines
      5.23 Formal Languages: nondeterministic
processors; grammars; parsing and translation; abstract families of languages
5.24 Analysis of Programs: schemata; semantics; correctness
5.25 Computational Complexity: machine-based; machine-independent; efficiency of algorithms
5.26 Turing Machines; Abstract Processors
5.27 Computability Theory: unsolvability; recursive functions
5.29 Miscellaneous
5.3 COMBINATORIAL AND DISCRETE MATHEMATICS
5.30 General
5.31 Sorting
5.32 Graph Theory
5.39 Miscellaneous
5.4 MATHEMATICAL PROGRAMMING
5.40 General
5.41 Linear and Nonlinear Programming
5.42 Dynamic Programming
5.49 Miscellaneous
5.5 MATHEMATICAL STATISTICS; PROBABILITY
5.6 INFORMATION THEORY
5.7 SYMBOLIC ALGEBRAIC COMPUTATION
5.9 MISCELLANEOUS
6. HARDWARE
6.0 GENERAL
6.1 LOGICAL DESIGN; SWITCHING THEORY
6.2 COMPUTER SYSTEMS
6.3 COMPONENTS AND CIRCUITS
6.30 General
6.31 Circuit Elements
6.32 Arithmetic Units
6.33 Control Units
6.34 Storage Units
6.35 Input/Output Equipment
6.36 Auxiliary Equipment
6.39 Miscellaneous
6.4 PATENTS; HARDWARE
6.9 MISCELLANEOUS
7. ANALOG COMPUTERS
7.0 GENERAL
7.1 APPLICATIONS
7.2 DESIGN; CONSTRUCTION
7.3 HYBRID SYSTEMS
7.4 PROGRAMMING; TECHNIQUES
7.9 MISCELLANEOUS
8. FUNCTIONS
8.0 GENERAL
8.1 SIMULATION AND MODELING
8.2 GRAPHICS
8.3 OPERATIONS RESEARCH/DECISION TABLES
8.9 MISCELLANEOUS

Book Reviews


This book is really for those knowledgeable in Artificial Intelligence. However, for some time I have been strongly suggesting that Australian DP practitioners make themselves aware of possible future developments in computing by reading the latest literature. Those with a desire to obtain an accurate insight into the current ability to represent and exploit knowledge held in a Data Base — which ability will eventually give rise to improved question answering capabilities for Data Base Ad-hoc Retrieval languages — will obtain singular benefit from careful study of the series of articles in this book.

The contents consist of 14 offerings divided into three major parts:

I Overview and General Systems
   I Theoretically Oriented Efforts, and
   III Areas of Application


The other parts I will leave until I have fully digested the articles in the overview part.

Lay people in this area, such as myself, will not find the reading easy. But eventually systems of storing and systematically exploiting large tracts of human knowledge will be commonplace, and desirable so if mankind is to use his past and rapidly expanding current knowledge as a base from which to mount larger scale intellectual endeavours. Readers of this book will have foreknowledge of how such knowledge systems might work.

A.Y. MONTGOMERY
Monash University


"Information Privacy" is a journal published every two months by IPC Science and Technology Press Limited in the United Kingdom. The first issue appeared in September, 1978.

"Information Privacy" is a welcome addition to the list of professional journals. The journal publishes papers on the technical, legal and social aspects of computer-based information processing systems. It should be of interest to all managers, users, and computer professionals who deal with such systems.

The Journal has an Editorial Advisory Board which comprises people interested in relevant fields from some eleven countries. The Journal includes technical papers as well as reports on developments throughout the world in the area of information privacy. The issues published to date have contained a wide range of editorial content from some of the leading writers in the field. The growth of information systems, particularly those that are computer-based, has given rise to a whole new host of problems in the privacy area. The invasion of individual privacy has become a major topic for debate and discussion throughout the world and hence a publication devoted specifically to this and related issues is particularly appropriate. It is likely that the privacy, security and integrity of information systems will continue to increase in both importance and sensitivity. It becomes increasingly important, therefore, for interested computer professionals to be able to keep abreast of relevant developments. "Information Privacy" certainly helps in this regard.

I believe that there is little doubt that the problem of invasion of privacy and related issues is destined to become a major facet in the design and control of information systems. Computer professionals, therefore, would be well advised to add "Information Privacy" to their list of required reading.

ASHLEY W. GOLDSWORTHY
State Govt. Insurance Office Building Society
This book is about the design and construction of AI programs for an important class of rule based systems. For a decade or more computer science, particularly AI, has tried to design machines which use large knowledge bases to achieve relatively intelligent and sensible behaviour. To AI workers 'knowledge' has a technical meaning. It means the representations of objects, situations, relationships, operations etc. at levels of abstraction where they are essentially independent of the specific ways they become involved in achieving particular goals. The search for such 'neutral' descriptions has gravitated from the atomic, fact-based systems such as those which use the order predicate calculus and its theorem provers, to systems based on highly modular procedures and sophisticated control structures. The latter provide extremely flexible representations for knowledge whilst preserving a good measure of the modularity and declarative nature of use-independent descriptions. Usually, the procedures in these systems are derived from two-part rules, and interact through side effects on a data base, the state of which in turn governs procedure invocation. Two-part rule structures support a number of useful meta associations; for example the two sides of a production rule (hence the term production systems PSs), an occurrence and the preconditions for its application, the antecedent and consequent of a logical implication, the 'before' and 'after' of permissible state transitions, and so on. The different uses of rule structures can lead to different meta-level control structures though most are based on the familiar state space search strategies of forward and backward chaining.

Rules which work in a wide range of contexts will inevitably contain some pollution caused by the systems. Occasioned by rule deployment are often thought of as inferences. Appropriately then, the term 'Pattern Directed Inference Systems', PDISs, was coined for the workshop held in June 1977 to exchange experience with such systems. The book is a selected papers from the proceedings, together with useful Editorial brackets providing substantial summary/abstract as a prologue and an interesting though not quite so substantial 'unifying' taxonomy of PDISs as an epilogue. The book will disappoint readers who expect more than workshop proceedings. The contributions contain many speculative remarks, in-house descriptions of work in progress, and experiential accounts of systems, the technical details of which being available only from local reports. Such frustrations are to be expected from workshop proceedings. Fortunately, embedded in the 600 pages are some hard results and some firm advice to intending system constructors.

The importance of the area is firstly in its potential for founding a computational semantics of cognition, and secondly in the effectiveness of its techniques for constructing knowledge based systems. (Currently these techniques are the dominant ones.) The papers in this book exemplify this. The book is divided into book sections; Architecture and Design, Deductive Inference, Learning, Cognitive Modelling, Natural Language Understanding, Multilevel Systems and Complexity. The papers don't cluster very strongly into these groupings. Readers with ideas of the control of the computer system from a modular/operativity/specifiCity spectrum will impose quite different organisations. For review purposes, interests in the latter will be assumed. With one or two exceptions the papers on architecture, design or complexity are about production systems. Production rules represent state antecedent-consequent pairs. The operation cycle consists of comparing (matching) either the antecedent or the consequent with the data base to produce a conflict set of 'executable rules' followed by a selection phase in which a rule from the conflict set is chosen for execution (to the exclusion of all other rules) according to some problem independent strategy. The simplicity of the rule structure and control cycle gives PSs their wide applicability. At the same time, the simplicity limits their effectiveness though unconstrained growth in conflict set recovery and rule selection times according to the size of the data base and the number of rules. McDermott, Newell and Morely analyse this problem and give results of their attempts at holding the essentially N* S growth down to a 'practical' level (a system shouldn't spend too much time carrying out computation which we can see is avoidable). Their approach exploits co-occurrence of the matching conditions in the rules and the data base elements (assuming at worst, changes will be incremental) to reduce costly recomputation of matching conditions when the environment doesn't change. In a companion paper, Kendal and Wekey show how decisions and operational generalisation across 'before and after' pairs, drawn from "Blocks for a dose of problem knowledge! Just how inefficient a PS will be depends on the nature of the problem domain to such an extent that a theory of PSs must be closely allied to classification of problem domains. This central problem is recognised by Rosenschein who has attacked it with computational constructors and operational semantics. The plan is to match particular problem domain characteristics to a class of PSs (a PS is a <RULES,CONTROL> pair) thereby avoiding the inefficiencies caused by otherwise ad hoc decisions.

The remarkably small degree of attention given to the foundations of pure PSs is presumably a consequence of a widespread belief that pure PSs are less desirable than practical in real problem domains. A clear treatement of the contrast between the practitioners views of PSs and the intellectually appealing purist stand is given by Lenat and Harris who informally argue the inevitability of corruption systems when applied to real problems. The story is a familiar one in AI. The attractive qualities of simplicity, generality, and clarity of candidate rule schema often become intolerable constraints on our ability to express problem domain structures and, as if that's not enough, they often promote crippling inefficiencies reminding us again that generality is not necessarily power. A good example of such difficulties in applied PSs is contained in the Mostow and F. Hayes-Roth report on HERSAY II. The authors relate the problems caused by the uniformity and minimal structure of PSs in a problem area which area, a-priori, a quite suitable case for treatment. They found that by using a grammar (which was mechanically converted to production rules) and a weighting function PSs could be used to limit the size of the search space, they could not engineer their way to acceptable performance without resorting to non PS technology by which additional problem semantics were introduced. The combinatory complexity and treatment. The book provides a complete data in their paper highlights again the way good heuristics (relatable predictors) are often integrations of structurally distant parts of formal models of a problem domain, and therefore probabilistic treatment. The book puts forward a proposal to overcome the difficulties through system construction based on partial matches. The proposal is not developed.

Several papers advocate overcoming the rigidity of PSs by extending the machinery. An obvious target is the control provisions for problem knowledge, that is the dependence of one state transition on another. Independent provision for control usually manifests itself as all sorts of temporary debris in the data base. Zisman proposes extension by the superposition of petrie nets on PSs. The success or otherwise of petrie nets in this role will no doubt repeat the success or otherwise of petrie nets as process coordinators in the larger computer science context, where they are one of a number of contenders: the problems and the promises are the same. Zucker's paper on PSs with feedback is another direct assault on control problems, this time addressing a class of problems for which the control theoretic feedback paradigm is appropriate. Though his paper does not exhaust sufficient work accurately to the reader of the viability (even naturalness) of viewing filtering problems as control problems. The data base is likened to the plant to be driven into a final state by rule invocations determined by the current state of PSs (hence feedback). The paper stops short of making the model for system design.

As pure PSs are extended, their guarantee of simplicity and uniformity of representation are compromised. Above all what needs to be preserved for AI purposes is the facility for constructing highly modular data driven systems. Analysis of data driven systems in their own right leads Reiger to formulate a Spontaneous Computation model in which computation proceeds from triggers in the everchanging data, departing quite dramatically from PSs through a provision for 'partially' triggered computations via a splitting mechanism. The ideas in his essay are cognate with the motivations for partial matching given elsewhere. Indirectly Zisman's and Reiger's papers relate the study of PSs to the study of co-ordinated processes, the former identifies with abstracted control and the latter with message passing. The field is crying out for a contribution which addresses that relationship directly; the profit is guaranteed!

The task of 'learning' (sometimes called inductive inference or 'adaptation') is to synthesise rules for a wide class of problems from the combination of a relatively narrow class and some ideas about the environment. PSs seem too rigid to form target systems for this task largely because of the independence between acquisition and use order of 'causal' relationships inferred by the systems. A formal, abstract model of inductive learning is given in this paper. Were they in an operational setting we refer to them as 'Blocks
The search for global organisation principles for knowledge and associated inferencing. A large number of the papers in this year's proceedings are discussions of database management systems, and while the book does provide a useful set of examples, it does not provide a comprehensive guide to the world of database management.

The book is primarily addressed to mathematicians rather than computer specialists. There are no computer programs and the numerical examples are generally designed to illustrate important aspects of the theory. Some hints are given on efficient computer implementation of some of the algorithms discussed and information is given on the CPU time required for some of the numerical examples, but the main concern of the book is with the mathematical foundations of the algorithms and much of the exposition uses the "definition, theorem, proof" style.

Apart from a few cases where two papers have a common author, the papers are quite independent and some idea of the major aspects of the theory. Some hints are given on efficient computer implementation of some of the algorithms discussed and information is given on the CPU time required for some of the numerical examples, but the main concern of the book is with the mathematical foundations of the algorithms and much of the exposition uses the "definition, theorem, proof" style.

Exception: In a few cases where two papers have a common author, the papers are quite independent and some idea of the major aspects of the theory. Some hints are given on efficient computer implementation of some of the algorithms discussed and information is given on the CPU time required for some of the numerical examples, but the main concern of the book is with the mathematical foundations of the algorithms and much of the exposition uses the "definition, theorem, proof" style.

A.L. ANDREW, La Trobe University

97
The first part of the article on computer chess is an historical survey describing the progress made by chess programs from Class C level to International Master level; the main emphasis of the survey being on the period following the first world championship in St. Louis in 1974. Some important games are documented in full and are accompanied by interesting comments and statistics. The tree-searching strategies used by the programs are described in the latter part of the article. The author predicts that chess programs will be playing at the Grand Master level by 1988.

M.H. Halstead's article 'Advances in Software Science' begins by discussing the basic metrics used by software science and then reviews relations which endeavour to explain the natural laws governing language and associated mental activity. These relations are tested by experimentation and are characterized by their simplicity and lack of arbitrary constants.

In 'Current Trends in Computer-Assisted Instruction', the author begins by discussing computer aided instruction (CAI) in elementary and secondary education. Certain projects and courses offered in the US are described. There follows an interesting appraisal of the current research in those areas most significant for CAI. These areas include natural language processing, the uses of audio, informal mathematical proofs and modelling of the student.

The final article analyzes the various factors which have affected the productivity of software in recent years. The author classifies these factors into four categories: those that depend on hardware availability, those that are related to priorities in the allocation of effort and other resources, those that depend on economic factors, and those that are due to technology transfers from foreign sources. Most of the discussion is limited to non-military general computing.

Dr. R. Sacks-Davis, Dept. of Computer Science, Monash University

This new monograph on computer security deserves more than casual consideration at its price of $25.20, as it carries the reputation of the Association for Computing Machinery Inc. as an authoritative review of computer security research progress.

The approach taken by the authors in providing extensive references at the end of each chapter, with an abstract of the key contribution of the cited reference included with each bibliography, is somewhat unusual, and may well not appeal to support the authors' objectives. However, the monograph, in focusing on current research references and directions, will no doubt appeal more to technical managers, computer planning officers and systems personnel rather than the growing army of concerned users, including the general public, of computer equipment.

The book is divided into eight chapters, including an excellent introduction. The introduction, for example, stimulates the reader's interest in a number of areas, particularly the goals of software and database security. The second chapter, which is concerned with privacy and the concise summary of privacy protection measures studied introduced in the USA and of current security issues arising from privacy requirements, including the implications of electronic funds transfer technology.

On the question of operational security, there is explicit recognition that "much of the literature . . . has been primarily intended to motivate and introduce the concept of security to managers and other individuals that had little or no background or experience in this field". Chapter 3 is consequently one of the most valuable treatments of the problems of evaluating security systems, due to its practical orientation and concessions to the interests of senior managers and auditors. Despite a creditable analysis of the organisational impact of security operations and of the cost-effectiveness of specific measures, the growing nexus of security-auditability-systems design did not emerge as strongly as management was noted, however, where policy/operational/ economic decisions are involved.

Personally, I was very pleased to see the "conventional wisdom" reinforced by the present concept of risk analysis and risk management was noted, however, where policy/operational/ economic decisions are involved.

The five articles in the 1979 edition of Advances in Computers are:

(i) 'Image Processing and Recognition' by A. Rosenfeld.
(ii) 'Recent Progress in Computer Chess' by M.M. Newborn.
(iii) 'Advances in Software Science' by M.H. Halstead.
(iv) 'Current Trends in Computer-Assisted Instruction' by P. Suppes.
(v) 'Software in the Soviet Union: Progress and Problems' by S.E. Goodman.

The articles are well written and provide an introduction to the respective topics. The article by A. Rosenfeld surveys techniques used in image processing for the (a) digitization, (b) coding and approximation, (c) enhancement, restoration and reconstruction, (d) segmentation, (e) representation and (f) interpretation of images. The survey avoids mathematics in describing these techniques although a knowledge of two dimensional Fourier transforms is assumed in parts of the text. The style is lucid and the discussion is well illustrated by many excellent photographs and diagrams.
security features are to be implemented!

In summary, the monograph offers an expert roundup of technical trends in security research, drawing heavily on the more recent literature (1973-78), and will be of value to those requiring a systematic exposure to recent developments, such as cryptographic transformations and the use of microprocessors/mini-computers as security aids.

DR. B.J. GARNER, Professor of Computing, Deakin University


The general theme of this Journal is similar to that of the ACJ itself. The articles are contributed by Australian practitioners and represent a balance between somewhat esoteric subjects and case studies of practical applications.

The issue reviewed consisted of 10 contributions which I have outlined below. The authors are currently CSA employees assigned, in most cases, as team leaders on a variety of projects many of which are the subjects of their papers.

1. A Development Methodology for Diagnostic Software (G.J. Monaghan)
   A discussion of the role of diagnostic software in hardware fault detection and localisation. Attention is drawn to the necessity of testing the diagnostic software itself prior to its release to users.

2. An Efficient Coding System for Long Source Sequences (C.B. Jones)
   An explanation of a source coding scheme which permits a source sequence of practically unlimited length to be coded as a single codeword using arithmetic of only limited precision. The application of the system to data compression and cryptography is explained and encoding and decoding algorithms are provided.

3. The Implementation of Control and Security Provisions for the GENIUS System (S.C. Johnson)
   A discussion of security aspects applied to a system developed for the Education Department of Victoria on Burroughs 7700s.

4. Database Design for a Large Personnel/Payroll System (M. Doraby and R. Sherratt)

5. The York Motors Parts System (S.J. Cook)

6. Woolnet, Electronic Auction System (D.N. Coleman)

7. Development of the RAAF LRMPA Flight Simulator (A. Page)
   These four papers are essentially case studies in the design and implementation of computer-based systems the functions of which are obvious from their titles.

8. The Computer Move (M. Copeman and N. Palmer)
   The experience of a team involved in the relocation of a revenue-earning time-sharing system, discussed from both a technical and managerial viewpoint.

   An examination of the methodology of testing modifications to existing software with emphasis on the cost-effectiveness of the testing approach taken.

   A discussion of the role of both software and hardware monitors in the deviation of a performance model of a computer system, together with case studies on a Burroughs 6700 and Univac 1100 AMPNET Mass Storage.

   This Journal should be of wide appeal as a treatment of both theoretical and practical issues on current computing.

   P.J. JULIFF, Prahran C.A.E.


This book consists of a preface, acknowledgements, two introductions, three Parts (I to III), the conclusion, 39 pages of notes and references, and an index. The first Part includes two Phases (I and II) which are classed as a chapter. Each Phase is divided into two sections (I and II). The second Part includes an introduction, four chapters (3 to 6) and a conclusion. Two of these chapters (4 and 5) are each divided into two sections (I and II). The third Part includes an introduction, three chapters (7 to 9) and a conclusion. The conclusion consists of chapter 10.

Confused? Yes, so was I. But there is more.

The first introduction, which introduces the revised edition, in fact contains two new Phases and a conclusion. These appear complete with headings but are not mentioned in the contents page. The two new Phases (III and IV) are to be read after Phases I and II in Part I. This particular conclusion, one of eight such headings on the book, is to be read after Part III, apparently in place of the conclusion.

The book contains 561 notes and references which are printed at the back and accessed by numbers in the body of the text. Unfortunately the numbering sequence restarts no less than 15 times, making it necessary to use a bookmark in the notes and references section in order to be able to use it at all.

In short, the presentation and layout of this book is dreadful.

Dreyfus reviews 20 years of research from 1957 to 1977 in four Phases to which he gives the titles “Cognitive Simulation”, “Semantic Information Processing”, “Manipulating Micro-Worlds” and “Facing the Problem of Knowledge Representation”. He analyses the differences between predictions and subsequent reality, and much of what he says is, of course, critical; his belief that Artificial Intelligence research is ill-founded caused him to write the book originally. Even accepting this, one does begin to doubt his judgement upon finding that, for example, all 63 of his references to the writings of Minsky are critical to the point of being scathing. I hold no brief for Minsky, but I doubt that such extreme damnation is realistic.

Dreyfus examines the assumptions implicit in AI work and illustrates why he believes these assumptions to be wrong. He sets up an alternative set of assumptions and draws conclusions from these regarding how far AI can ever go. Whilst he takes a strong position, he writes with considerable perception and I found this part of the book most interesting.

Taken overall, however, I think readers would be well advised to wait for a better organised and less partisan book on the subject.

J. WHITTLE, Whittle Programming
<table>
<thead>
<tr>
<th>DATE</th>
<th>EVENT</th>
<th>LOCALE</th>
<th>SOURCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td><strong>June</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-6</td>
<td>Symposium on Machine Processing of Remotely Sensed Data</td>
<td>W. Lafayette, Ind., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>3-5</td>
<td>Networks 80</td>
<td>London, UK</td>
<td>BCS</td>
</tr>
<tr>
<td>3-6</td>
<td>4th International IFAC Conference on Instrumentation and Automation in the Paper, Rubber, Plastics, and Polymerization Industries</td>
<td>Ghent, Belgium</td>
<td>IFAC</td>
</tr>
<tr>
<td>5-7</td>
<td>SIAM Spring 1980 Meeting</td>
<td>Alexandria, VA., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>15-20</td>
<td>Sixth Workshop on Graphtheoretic Concepts in Computer Science</td>
<td>Bad Honnef, Bonn, Germany</td>
<td>ACM</td>
</tr>
<tr>
<td>16-18</td>
<td>1980 International Conference on Communications</td>
<td>Seattle, Wash., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>16-18</td>
<td>IFAC/IFIP Symposium on Automation for Safety in Shipping and Offshore Operations</td>
<td>Trondheim, Norway</td>
<td>IFAC, IFIP, SINTEF</td>
</tr>
<tr>
<td>16-19</td>
<td>First Annual Conference of the National Computer Graphics Association</td>
<td>Arlington, Va., USA</td>
<td>NCGA</td>
</tr>
<tr>
<td>16-20</td>
<td>European Conference on Databases in the Humanities and Social Sciences</td>
<td>Madrid, Spain</td>
<td></td>
</tr>
<tr>
<td>18-24</td>
<td>Annual Meeting of the Association for Computational Linguistics</td>
<td>Philadelphia, Pann., USA</td>
<td>ACL</td>
</tr>
<tr>
<td>23-25</td>
<td>Seventeenth Design Automation Conference</td>
<td>Minneapolis, Minn., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>23-26</td>
<td>Workshop on Abstraction, Databases and Conceptual Modeling</td>
<td>Pingree Park, Col., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>23-27</td>
<td>Symposium on Research and Development in Information Retrieval</td>
<td>Cambridge, England</td>
<td>ACM</td>
</tr>
<tr>
<td>23-27</td>
<td>World Conference on Transborder Data Flow Policies</td>
<td>Rome, Italy</td>
<td>IBI World Conf.</td>
</tr>
<tr>
<td>24-26</td>
<td>2nd IFAC Symposium, 'Large Scale Systems — Theory and Applications'</td>
<td>Paris, France</td>
<td>IFAC</td>
</tr>
<tr>
<td>24-26</td>
<td>APL 80 International Congress</td>
<td>Noordwijkerhout, Neth.</td>
<td>DSI</td>
</tr>
<tr>
<td>25-27</td>
<td>Simulation 80</td>
<td>Interlaken, Switzerland</td>
<td>IASTED</td>
</tr>
<tr>
<td><strong>JULY</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-4</td>
<td>International Conference on Databases</td>
<td>Aberdeen, Scotland</td>
<td>BCS</td>
</tr>
<tr>
<td>7-11</td>
<td>Tutorial Conference on Practical Optimization</td>
<td>Stanford, USA</td>
<td></td>
</tr>
<tr>
<td>8-11</td>
<td>5th Conference on Automated Deduction</td>
<td>Les Arcs, Savoire, France</td>
<td>IAPR</td>
</tr>
<tr>
<td>13-25</td>
<td>ISP 1980 Congress</td>
<td>Hamburg, FRG</td>
<td></td>
</tr>
<tr>
<td>14-16</td>
<td>Nonlinear Programming Symposium 4</td>
<td>Madison, Wis., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>14-18</td>
<td>SIGGRAPH 80, Seventh Annual Conference on Computer Graphics and Interactive Techniques</td>
<td>Seattle, Wash., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>14-18</td>
<td>SIGGRAPH 80, Seventh Annual Conference on Computer Graphics and Interactive Techniques</td>
<td>Seattle, USA</td>
<td>ACM SIGGRAPH</td>
</tr>
<tr>
<td>14-18</td>
<td>Seventh International Colloquium on Automata, Languages and Programming</td>
<td>Noordwijkerhout, Neth.</td>
<td>EATCS</td>
</tr>
<tr>
<td><strong>AUGUST</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17-21</td>
<td>1980 Urban and Regional Information Systems Association Conference</td>
<td>Toronto, Canada</td>
<td>URISA</td>
</tr>
<tr>
<td>18-22</td>
<td>4th Symposium on Computational Statistics</td>
<td>Edinburgh, Scotland</td>
<td>IFIP</td>
</tr>
<tr>
<td>24-27</td>
<td>1980 LISP Conference</td>
<td>Stanford, Calif., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>25-29</td>
<td>5th Australian Statistical Conference</td>
<td>Sydney, Australia</td>
<td>ACS</td>
</tr>
</tbody>
</table>

### Forthcoming Events

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
<th>Location</th>
<th>Organizer</th>
</tr>
</thead>
<tbody>
<tr>
<td>26-29</td>
<td>1980 International Conference on Parallel Processing</td>
<td>Pellston, Mich., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>27-28</td>
<td>Second Workshop on Picture Data Description and Management</td>
<td>Asilomar, Calif., USA</td>
<td>IEE-CS</td>
</tr>
<tr>
<td>27-29</td>
<td>SSA Conference 1980 and 4th Biennial Conference</td>
<td>Brisbane, Australia</td>
<td>SSA, ACS</td>
</tr>
<tr>
<td><strong>SEPTEMBER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-5</td>
<td>Eurographics 80</td>
<td>Geneva, Switzerland</td>
<td>IFIP</td>
</tr>
<tr>
<td>8-12</td>
<td>Integrated Systems 80</td>
<td>Washington DC, USA</td>
<td>ACM</td>
</tr>
<tr>
<td>16-19</td>
<td>EUSIPCO-80 First European Signal Processing Conference</td>
<td>Lausanne, Switzerland</td>
<td>IFIP</td>
</tr>
<tr>
<td>17-19</td>
<td>Joint ACM SIGSMALL/SIGPC Symposium on Small Systems</td>
<td>Palo Alto, Calif., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>25-28</td>
<td>Euro IFIP 79</td>
<td>London, UK</td>
<td>IFIP</td>
</tr>
<tr>
<td>29-Oct 3</td>
<td>MEDINFO 80</td>
<td>Tokyo, Japan</td>
<td>IFIP</td>
</tr>
<tr>
<td>29-Oct 4</td>
<td>COLING 80, 1980 World Meeting on Natural Language and Computer</td>
<td>Tokyo, Japan</td>
<td>IFIP</td>
</tr>
<tr>
<td><strong>OCTOBER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-3</td>
<td>FTCS-10, 10th International Symposium on Fault-Tolerant Computing</td>
<td>Kyoto, Japan</td>
<td>IEEEECS</td>
</tr>
<tr>
<td>1-3</td>
<td>6th International Conference on Very Large Data Bases</td>
<td>Montreal, Canada</td>
<td>ACM</td>
</tr>
<tr>
<td>5-12</td>
<td>Annual Meeting of the American Society for Information Science</td>
<td>Anaheim, Calif., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>6-9</td>
<td>8th World Computer Congress</td>
<td>Tokyo, Japan</td>
<td>IFIP</td>
</tr>
<tr>
<td>7-10</td>
<td>IKD 1980 International Conference of Informatics</td>
<td>Berlin, FRG</td>
<td>IFIP</td>
</tr>
<tr>
<td>8-10</td>
<td>1980 International Conference on Cybernetics and Society</td>
<td>Cambridge, Mass., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>13</td>
<td>Computing in Developing Nations</td>
<td>Melbourne, Australia</td>
<td>ACS</td>
</tr>
<tr>
<td>14-17</td>
<td>8th World Computer Congress</td>
<td>Melbourne, Australia</td>
<td>IFIP</td>
</tr>
<tr>
<td>20-22</td>
<td>CPEUG 80, 16th Meeting of the Computer Performance Evaluation Users Group</td>
<td>Orlando, Fla., USA</td>
<td>ACM</td>
</tr>
<tr>
<td>21-24</td>
<td>SEARCC 80, South East Asian Computer Congress</td>
<td>Jakarta, Indonesia</td>
<td>ACS, SEARCC</td>
</tr>
<tr>
<td>20-23</td>
<td>International Conference on Computer Aided Design</td>
<td>Melbourne, Australia</td>
<td>ACADS</td>
</tr>
<tr>
<td>27-29</td>
<td>ACM 1980 Annual Conference — “Previewing the Computer Age”</td>
<td>Nashville, USA</td>
<td>ACM</td>
</tr>
<tr>
<td>27-30</td>
<td>5th International Conference on Computer Communication:</td>
<td>Atlanta, USA</td>
<td>ICCC</td>
</tr>
<tr>
<td><strong>NOVEMBER</strong></td>
<td>Increasing Benefits for Society</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6-8</td>
<td>6th International Conference ‘System Dynamics and the Analysis of Change’</td>
<td>Paris, France</td>
<td>AFCET</td>
</tr>
<tr>
<td>12-14</td>
<td>Pacific 80, Conference on Distributed Processing</td>
<td>San Francisco, USA</td>
<td>ACM</td>
</tr>
<tr>
<td>30-Dec. 2</td>
<td>13th Annual Workshop in Microprogramming</td>
<td>Colorado Springs, USA</td>
<td>ACM</td>
</tr>
<tr>
<td><strong>DECEMBER</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-4</td>
<td>Fifth International Joint Conference on Pattern Recognition</td>
<td>Miami Beach, USA</td>
<td>ACM</td>
</tr>
<tr>
<td>9-11</td>
<td>SIGPLAN Symposium on the ADA Programming Language</td>
<td>Boston, Mass., USA</td>
<td>ACM</td>
</tr>
<tr>
<td><strong>1981 JANUARY</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-14</td>
<td>Pacific Telecommunications Conference</td>
<td>Honolulu, Hawaii</td>
<td>HAS</td>
</tr>
<tr>
<td>26-28</td>
<td>Third Hungarian Computer Science Conference</td>
<td>Budapest, Hungary</td>
<td>HAS</td>
</tr>
<tr>
<td><strong>FEBRUARY</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24-26</td>
<td>ACM Annual Computer Science Conference</td>
<td>St Louis, USA</td>
<td>ACM</td>
</tr>
<tr>
<td><strong>MARCH</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-12</td>
<td>5th International Conference on Software Engineering</td>
<td>San Diego, Calif., USA</td>
<td>ACM</td>
</tr>
<tr>
<td><strong>JULY</strong></td>
<td>3rd World Conference on Computers in Education (WCCE 81)</td>
<td>Lausanne, Switzerland</td>
<td>IFIP</td>
</tr>
</tbody>
</table>

For further information on these events please contact the Chief Executive Officer, PO Box 640, Crows Nest, NSW, 2065.

Computer Science Theses

COMPUTER SCIENCE D.Sc., Ph.D. AND MASTERS THESES ACCEPTED AT AUSTRALIAN UNIVERSITIES 1977-79

The following consolidates D.Sc., Ph.D. and Masters thesis lists for the period 1977-79 which have been provided by computer science and associated departments of Australian universities.

The Australian National University

Deakin University

Flinders University of South Australia

University of Melbourne

Monash University

University of Queensland

Book Review


The best comment on this book can be found in the preface. It says: This book is directed primarily towards those who are confronted with practical processing problems and understanding at the "grass roots" level. The basic theoretical background covered by the book contains random processes, series and transformations, spectral analysis, correlation analysis, and filtering concept. However, the authors only introduce the reader to some concept required for the understanding of signal analysis. There is little difference from any other books as there are some misprints. For example, in Fig. 6.2, p. 217 the equation

\[
\delta t = \lim_{\Delta t \to 0} f(t) \\
H_{\rightarrow \infty}
\]

should be in the form of

\[
\delta(t) = \lim_{\Delta t \to 0} f(t) \\
H_{\rightarrow \infty}
\]

In general, this is a very good reference book for people who are interested in computer and system science. H.R. HWA, Basser Department of Computer Science, University of Sydney.
With a Tandem NonStop System, expansion is easy.
Simply add processors as your processing needs grow.

No one else can do it. In every other system, you must replace the processor when it's outgrown. Not with Tandem. This is the only multiple processor on-line system in the world which lets you simply add processors, one or more at a time, when your needs have outgrown your original installation. And you never lose a cent on software either. All your original programs work without modification on the expanded system. Work is automatically spread among all the processors and all terminals access any processor or data base record, completely under the control of the non-stop operating system. From start to finish, you maintain the lowest cost-per-transaction in the industry. And you don't ever have to invest more hardware dollars than you need at the time to cover a possible future need. That's incredible in the world of computers.

And of course, the truly beautiful part of the Tandem NonStop System is that it is NonStop. It's the only system on the market today which can come through a failure intact. Without loss or duplication of any transaction in process at the time of a failure anywhere in the system. And Tandem offers an extraordinary measure of protection for the data base as well.

The Tandem NonStop System is on-line in major banks, retailers, distribution, manufacturing, medical, governmental and library service organizations performing credit verification, deposit and withdrawal, electronic fund transfers, communications, order entry, inventory and process controls, and emergency vehicle dispatch to name just a few. The era of non-stop computing is finally here, and Tandem Computers has an exclusive, successful jump on the field. For further information, or a demonstration of our unique capabilities, contact us today.

TANDEM

TANDEM COMPUTERS
DISTRIBUTED IN AUSTRALIA BY:

MANAGEMENT INFORMATION SYSTEMS PTY. LTD.

S. G. I.O. Building
Cnr TURBOT & ALBERT Sts.
BRISBANE. QLD. 4000
Telephone 229 38 30
The Australian Computer Journal is an official publication of the Australian Computer Society Incorporated.

Subscription and orders should be sent to PO Box 640 Crows Nest, 2065.

Office Bearers: President: G.E. Wastie; Vice-Presidents: R.C.A. Paterson, A.W. Goldsworthy; Immediate past president: A.R. Benson; National treasurer: C.S.V. Pratt; Chief executive officer: R.W. Rutledge, P.O. Box 640, Crows Nest, N.S.W., 2065, telephone (02) 929 2219.

Editorial Committee: Editor: A.Y. MONTGOMERY, Department of Computer Science, Monash University, Clayton, Victoria, 3168. Associate Editors: J.M. BENNETT, T. PEARCEY, P.C. POOLE.

SUBSCRIPTIONS: The annual subscription is $15.00. All subscriptions to the Journal are payable in advance and should be sent (in Australian currency) to the Australian Computer Society Inc., PO Box 640, Crows Nest, NSW, 2065.

PRICE TO NON-MEMBERS: There are now 4 issues per annum. The price of individual copies of back issues still available is $2.00. Some are already out of print. Issues for the current year are available at $5.00 per copy. All of these may be obtained from the National Secretariat, PO Box 640, Crows Nest, NSW, 2065. No trade discounts are given, and agents should recover their own handling charges. Special rates apply to members of other Computer Societies and applications should be made to the Society concerned.

MEMBERS: The current issue of the Journal is supplied to personal members and to Corresponding Institutions. A member joining partway through a calendar year is entitled to receive one copy of each issue of the Journal published earlier in that calendar year. Back numbers are supplied to members while supplies last, for a charge of $2.00 per copy. To ensure receipt of all issues, members should advise the Branch Honorary Secretary concerned, or the National Secretariat, promptly of any change of address.

REPRINTS: 50 copies of reprints will be provided to authors. Additional reprints can be obtained, according to the scale of charges supplied by the publishers with proofs. Reprints of individual papers may be purchased for 50 cents each from the Printers (Publicity Press).

PAPERS: Papers should be submitted to the Editor, authors should consult the notes published in Volume 9, pp. 45-49 (or request a copy from the National Secretariat).

MEMBERSHIP: Membership of the Society is via a Branch. Branches are autonomous in local matters, and may charge different membership subscriptions. Information may be obtained from the following Branch Honorary Secretaries. Canberra: PO Box 446, Canberra City, ACT, 2601. NSW: Science House, 35-43 Clarence St, Sydney, NSW, 2000. Qld: Box 1484, GPO, Brisbane, Qld, 4001. SA: Box 2423, GPO, Adelaide, SA, 5001. WA: Box F320, GPO Perth, WA, 6001. Vic: PO Box 98, East Melbourne, Vic, 3002. Tas: PO Box 216, Sandy Bay, Tas, 7005.

Copyright © 1979, Australian Computer Society Inc.

Published by: Associated Business Publications, 28 Chippendale Street, Chippendale, NSW, 2008. Tel: 699-5601, 699-1154. All advertising enquiries should be referred to the above address.


THE Australian Computer Journal

Price: $5.00 per copy to non-members $2.00 per copy to members

VOLUME 12, NUMBER 2, MAY 1980

CONTENTS


48-55 Program of Speakers for IFIP-80

56-62 An Introduction to Knowledge-Based Expert Systems — J.R. QUINLAN

63 Virtual Memory — J.L. KEEDY

64-69 A Fast and Stable List Sorting Algorithm — B. CHEEK

70 TWONET: A New Program for the Computation of a Two-Neighbour Network — W.T. WILLIAMS

SPECIAL FEATURES

55, 62, 69, 75-79, 82 Book Reviews

71-72 Notes on Submission of Papers

73-75 Notes for the Guidance of Referees

80-82 Forthcoming National and International Events

82 Computer Science Theses

This Journal is Abstracted OR Reviewed by the following services:

Publisher Service

ACM Bibliography and Subject Index of Current Computing Literature.

ACM Computing Reviews.

AMS Mathematical Reviews.

CSA Computer and Information Systems Abstracts.

Data Processing Digest.

ENGINEERING INDEX INC. Engineering Index.

INSPEC Computer and Control Abstracts.

INSPEC Electrical and Electronic Abstracts.

SPRINGER-VERLAG Zentralblatt fur Mathematik und ihre Grenzgebiete.