

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

AD-A193 325

DTIC FILE COPY

2

AFOSR-TR- 88-0366

Technical Report No. ODL-88.2

**DATABASE DESIGN AND MANAGEMENT IN
ENGINEERING OPTIMIZATION**

by

JASBIR S. ARORA

OPTIMAL DESIGN LABORATORY

**College of Engineering
The University of Iowa
Iowa City, IA 52242-1593**

Prepared for

**AIR FORCE OFFICE OF SCIENTIFIC RESEARCH
Air Force Systems Command, USAF
Under Grant No. AFOSR 82-0322**

February 1988

**DTIC
ELECTE
APR 01 1988
S H D**

DISTRIBUTION STATEMENT #

**Approved for public release
Distribution Unlimited**

88 3 31 008

A193 325

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ODL-88.2		5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TK- 88-0306	
6a. NAME OF PERFORMING ORGANIZATION Optimal Design Laboratory	6b. OFFICE SYMBOL (If applicable) ODL	7a. NAME OF MONITORING ORGANIZATION AFOSR/NA	
6c. ADDRESS (City, State and ZIP Code) College of Engineering The University of Iowa Iowa City, IA 52242		7b. ADDRESS (City, State and ZIP Code) Bolling AFB D.C. 20332-6448	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Office of Scientific Research	8b. OFFICE SYMBOL (If applicable) NA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Grant No. AFOSR 82-0322	
8c. ADDRESS (City, State and ZIP Code) Directorate of Aerospace Sciences Bolling AFB, D.C. 20332-6448		10. SOURCE OF FUNDING NOS.	
		PROGRAM ELEMENT NO. 61102F	PROJECT NO. 2302
		TASK NO. B1	WORK UNIT NO.
11. TITLE (Include Security Classification) Database Design and Management in Engineering Optimization			
12. PERSONAL AUTHOR(S) J.S. Arora			
13a. TYPE OF REPORT Final Technical	13b. TIME COVERED FROM 10-82 TO 2-88	14. DATE OF REPORT (Yr., Mo., Day) 1988-2-22	15. PAGE COUNT 151
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	Database Management, Design, Evaluation, Engineering, Optimization, Interactive	
	SUB. GR.		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) In this report, the research completed under the project in the area of database management in engineering design and optimization is described. Database management concepts used in business applications were studied and concepts suitable for engineering applications were developed. Data structures that need to be managed were identified. Database design methodologies were studied and a suitable methodology for engineering designs and optimization applications was developed. Several prototype database management systems (DBMS) were designed, developed and evaluated. Several prototype application programs utilizing a database management were developed to evaluate performance of DBMS. Based on these implementations and studies, the usual relational data model was generalized to handle engineering data types. Specifications for an integrated DBMS capable of handling relations, vectors and matrices (of different types) were developed. A system based on the specifications, called MIDAS/GR was implemented and evaluated. MIDAS/GR stands for Management of Information for Design and Analysis of Systems/Generalized Relational Model. Research was conducted to			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input checked="" type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Anthony K. Amos		22b. TELEPHONE NUMBER (Include Area Code) 202-767-4937	22c. OFFICE SYMBOL NA

19. determine appropriate algorithms for implementation of various steps in the database management system. During the research project, several reports were submitted, and papers presented at conferences and published in journals. Several Ph.D. and M.S. Theses were completed. These are listed in the report. All the journal articles published under the project are given in Appendices.


CONFIDENTIAL

CONFIDENTIAL

TABLE OF CONTENTS

1.	INTRODUCTION.....	1
2.	THESES COMPLETED AND REPORTS SUBMITTED.....	3
2.1	Ph.D. Dissertations.....	3
2.2	M.S. Theses.....	3
2.3	Other Reports Submitted.....	3
3.	JOURNAL ARTICLES PUBLISHED.....	4
3.1	A Survey of Database Management in Engineering.....	4
3.2	Database Management Concepts in Computer-aided design optimization.....	4
3.3	MIDAS: Management of Information for Design and Analysis of Systems.....	4
3.4	Data Base Design Methodology for Structural Analysis and Design Optimization.....	4
3.5	SMART: Scientific Database Management and Engineering Analysis Routines and Tools.....	5
3.6	Evaluation of the MIDAS DBMS in an Equation-Solving Environment.....	5
3.7	Uses of Artificial Intelligence in Design Optimization.....	5
3.8	Implementation of an Efficient Run-time Support System for Engineering Design Environment.....	6
3.9	Design and Implementation Issues in an Integrated Database Management System for Engineering Design Environment.....	6
3.10	Role of Data Base Management in Design Optimization Systems.....	6
3.11	An Integrated Database Management System for Engineering Applications Based on an Extended Relational Model.....	6
4.	CONFERENCE PRESENTATIONS.....	8
4.1	Database Management Concepts in Engineering Design Optimization.....	8
4.2	MIDAS: Management of Information for Design and Analysis of Systems.....	8
4.3	A Methodology to Design Database for Finite Element Analysis and Structural Design Optimization Applications.....	8
4.4	Use of AI in Design Optimization.....	8
4.5	Development of an Artificially Intelligent Nonlinear Optimization Expert System.....	9
4.6	Role of Database Management in Design Optimization Systems.....	9
4.7	A Nonlinear Optimization Expert System.....	10
4.8	Interactive Design Optimization of Framed Structures.....	10
4.9	An Integrated Database Management System for Engineering Applications Based on an Extended Relational Model.....	10
4.10	A Structural Optimization Program Using a Database Management System.....	11
	APPENDIX 1. A Survey of Database Management in Engineering.....	12

APPENDIX 2.	Database Management Concepts in Computer-aided Design Optimization.....	20
APPENDIX 3.	MIDAS: Management of Information for Design and Analysis of Systems.....	31
APPENDIX 4.	Data Base Design Methodology for Structural Analysis and Design Optimization.....	42
APPENDIX 5.	SMART: Scientific Database Management and Engineering Analysis Routines and Tools.....	55
APPENDIX 6.	Evaluation of the MIDAS DBMS in an Equation-Solving Environment.....	62
APPENDIX 7.	Uses of Artificial Intelligence in Design Optimization.....	72
APPENDIX 8.	Implementation of an Efficient Run-time Support System for Engineering Design Environment	94
APPENDIX 9.	Design and Implementation Issues in an Integrated Database Management System for Engineering Design Environment.....	103
APPENDIX 10.	Role of Data Base Management in Design Optimization Systems.....	112
APPENDIX 11.	An Integrated Database Management System for Engineering Applications Based on an Extended Relational Model.....	119

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution/ _____	
Availability Codes	
Dist	Avail and/or Special
A-1	

1. INTRODUCTION

A major thrust of the research was to study and develop database design and management techniques for engineering applications. Structural design and optimization was to be considered as the major area of application. Considerable progress was made in this important area of management of large numerical databases. The area is perceived to be of extreme importance. The reason is that as the computing power increases, we want to solve more complex problems. Need to integrate various disciplines to design complex systems becomes acute. We want to design efficient systems that can operate in extreme environments. All these requirements lead to computationally extensive problems. Software to solve such problems becomes quite large and complex. Data generated and managed become quite large. In addition, the flow of data is quite complex and unmanageable without a well defined database and database management system (DBMS).

In the current project, we studied the database management concepts in computer-aided design and optimization. Type of data encountered and their usage was studied. Data structures that are encountered in above application and, in general, other engineering applications were studied. Then we studied the existing database management models that had been used in the past. Based on this investigation it was determined the relational data model came close to satisfying the requirements of the engineering (numerical) data. The model can handle large relational data quite efficiently. It is, however, unsuitable for handling large matrix data types. This led to the development of what we call the "generalized relational data model." Basically, the standard relational model was generalized to handle additional data structures that are encountered in engineering applications.

A few existing database management systems, such as System R and RIM, were studied. This was done to learn about the implementation details as well as the algorithms used in them. Based on these studies and needs of engineering data, specifications for a suitable database management system were developed. Several experimental DBMS were designed and implemented to study their performance. Detailed implementation plans were developed. Various algorithms suitable for implementation of various capabilities were studied and the algorithms suitable for a DBMS to meet the specifications were selected. Several experimental codes were developed and implemented to evaluate database management concepts, algorithms and database design procedures.

This report contains all the developments outlined in the preceding paragraphs. Chapter 2 contains a list of M.S. and Ph.D. dissertations completed under the project and the reports that have been submitted previously. Chapter 3 contains a list of journal articles that have been published. Abstract of the articles are included here and full length papers are given in Appendices at the end. Chapter 4 contains a list of conference presentations and the papers that have appeared in the proceedings. Abstracts of the papers are also included.

Finally, a computer software called MIDAS/GR which stands for Management of Information for Design and Analysis of Systems /Generalized Relational Model is submitted as a part of the final report. A users manual for the system is also submitted. This experimental system has evolved as a result of several designs, redesigns and implementations. The system can handle relations, vectors, matrices and other variable size objects. The system is organized as a library. Programmers can use the library facilities from their programs. Data sets can be created and deleted while the program is running. Data can be inserted or retrieved; it can be stored in one form and retrieved in another. Thus, the system provides run-time support for data management. Using the system,

complex software can be developed and debugged in a relatively short time. With some additional effort the system can be converted into a commercial quality product.

Using this modern DBMS, creation of a software environment for engineering analysis and design is in progress. MIDAS can be used in two modes of operation:

1. It can be used to connect existing software components through a database, and
2. It can be used to develop new software that uses system facilities to manage permanent and transient data; in this mode the system provides run-time support for management of large amount of data.

Once this environment has been established, several numerical problems will be solved to evaluate the performance of the system.

2. THESES COMPLETED AND REPORTS SUBMITTED

2.1 Ph.D. Dissertations

1. T. SreekantaMurthy, "Computer-Aided Structural Design Optimization Using a Database Management System," Civil Engineering; *submitted as Technical Report No. 85.17*, November 1985.
2. G.J. Park, "Interaction Design Optimization with Modern Database Management Concepts," Mechanical Engineering; *submitted as Technical Report No. ODL-86.9*, November 1986.
3. S.S. Al-Saadoun, "A Design Optimization System with Central Database for Framed Structures," Civil Engineering; *submitted as Technical Report NO. ODL-86.21*, November 1986.

2.2 M.S. Theses

1. Y-K Shyy, "A Database Management for Engineering Applications," Mechanical Engineering; *submitted as Technical Report No. 85.23*, November 1985.
2. S. Mukhopadhyay, "A Database Management System Using A Generalized Relational Model," Computer Science; *submitted as Technical Report No. 86.27*, November 1986.
3. R. Venkatraman, "Query Optimization in Deductive Database," Computer Science; *submitted as Technical Report No. ODL-88.1*, February 1988.

2.3 Other Reports Submitted

1. SreekantaMurthy, T., Rajan, S.D., Reddy, C.P.D., Staley, D.T., Bhatti, M.A. and Arora, J.S., "Database Management in Design Optimization," *Technical Report No. CAD-SS-83.17*, October 1983.
2. SreekantaMurthy, T. and Arora, J.S., "A Survey of Database Management Systems," *Technical Report No. CAD-SS-84.19*, August 1984.
3. SreekantaMurthy, T. and Arora, J.S., "Database Design Methodology and Database Management System for Computer-Aided Structural Design Optimization," *Technical Report No. CAD-SS-84.20*, September 1984.
4. Arora, J.S., Haririan, M., Paeng, J.K., Ryu, Y.S., SreekantaMurthy, T. and Wu, C.C., "Database Design for Structural Analysis and Design Optimization," *Technical Report No. CAD-SS-84.20*, October 1984.
5. Arora, J.S. and Mukhopadhyay, S., "Specification for MIDAS/GR: Management of Information for Design and Analysis of Systems: Generalized Relational Model," *Technical Report No. CAD-SS-84.24*, December 1984.
6. Arora, J.S., "Database Design and Management in Engineering Design Optimization," *Technical Report No. ODL-85.31*, September 1985.

3. JOURNAL ARTICLES PUBLISHED

- 3.1 SreekantaMurthy, T. and Arora, J.S., "Survey of Database Management in Engineering," *Advances in Engineering Software*, Vol. 7, No. 3, pp. 126-132, 1985.

Abstract. Database management in engineering fields is becoming extremely important. A number of reports and papers have been published on this subject and computer-aided design applications. It is important to know state-of-the-art to fully utilize the benefits of this new field. The paper contains a survey of literature on database management for engineering applications. The survey is broadly classified into database management concepts and systems. Research work of various authors in related areas such as database design methodology, data models, data definition and manipulation languages, database integrity and consistency are reviewed. Various database management systems (15 in all) currently in use are reviewed and their features tabulated. Capabilities and usefulness of these systems are emphasized. The survey is intended to provide important developments in data management field to the engineering community. Also, it should be useful to those currently engaged in research on the subject.

- 3.2 SreekantaMurthy, T. and Arora, J.S., "Database Management Concepts in Computer-Aided Design Optimization," *Advances in Engineering Software*, Vol. 8, No. 2, pp. 88-97, 1986.

Abstract. This paper deals with database management concepts in computer-aided design optimization. Complex nature of engineering data and the need to organize them are emphasized. Database management concepts applicable to finite element analysis and design optimization are explained. Various aspects associated with the development of data models, such as conceptual, internal and external models are discussed. Suitability of external models, like hierarchical, network and relational, are discussed with reference to design optimization requirements. Some techniques to organize data of large matrices for efficient numerical computations are given. Concepts of normalization of data, global and local databases are described. Details of a suitable database management system are described in a separate paper.

- 3.3 SreekantaMurthy, T., Shyy, T-K. and Arora, J.S., "MIDAS: Management of Information for Design and Analysis of systems," *Advances in Engineering Software*, Vol. 8, No. 3, pp. 149-158, 1986.

Abstract. The paper describes features, system design and implementation of a database management system called MIDAS. The system has capability to organize data of both relational and numerical models, and meets several important requirements - a good data model, ability to organize large matrix data, handle various data types, simplified data definition and data manipulation languages, dynamic data definition, multiple database organization, speed of data access, and provision for temporary databases. Tabular and matrix form of data generated and used in design and analysis of system can be conveniently organized. Details of various commands of the database management system MIDAS are presented.

- 3.4 SreekantaMurthy, T. and Arora, J.S., "Data Base Design Methodology for Structural Analysis and Design Optimization," *Engineering with Computers* 1, pp. 149-160, 1986.

Abstract. A methodology to design data bases for finite element analysis and structural design optimization is described. The methodology considers three views of data

organization - conceptual, internal, and external. Tabular and matrix forms of data are included. The relational data model is used in the data base design. Entity, relation, and attributes are considered to form a conceptual view of data. First, second, and third normal forms of data are suggested to design an internal model. Several aspects such as processing iterative needs, multiple views of data, efficiency of storage and access time, and transitive data are considered in the methodology.

- 3.5 Arora, J.S., Lee, H.H. and Jao, S.Y., "SMART: Scientific Database Management and Engineering Analysis Routines and Tools," *Advances in Engineering Software*, Vol. 8, No. 4, pp. 194-199, 1986.

Abstract. Development of user-friendly and flexible scientific programs is a key to their widespread usage, extension and maintenance. The paper describes capabilities of a library called SMART that can be used in the development of such programs. The library contains many interactive programming aids and screen management utilities that can help in development of the user interfaces. A very flexible full screen data editor is available. Many other subroutines are available, such as vector and matrix operations, in-core data management, out-of-core numerical and relational database management, and graphics. The library is not advocated to be comprehensive. It is meant to show the kind of capabilities that are needed in development and maintenance of scientific programs. Such libraries can be of particular use in research on computational methods for scientific applications.

- 3.6 SreekantaMurthy, T., Shyy, Y-K., Mukhopadhyay, S. and Arora, J.S., "Evaluation of the MIDAS DBMS in an Equation-Solving Environment," *Engineering with Computers* 2, pp. 11-19, 1987.

Abstract. The paper describes an evaluation of a data base management system, MIDAS, for the solution of linear equations. A brief description of the system is given. It is used and evaluated employing skyline and hypermatrix approaches for solving large matrix equations. Performance of the two subsystems of MIDAS - MIDAS/N and MIDAS/R - is measured and compared in terms of several system parameters. Programming for efficient use of the available memory is noted. Suggestions for application programming using MIDAS are given. Major conclusions of the study are that memory management schemes, data structures, and data access methods of the DBMS play very important roles for its efficient use in dynamic environment. Such methods must be developed and implemented for large-scale engineering applications.

- 3.7 Arora, J.S. and Baenziger, G., "Uses of Artificial Intelligence in Design Optimization," *Computer Methods in Applied Mechanics and Engineering* 54, pp. 303-323, 1986.

Abstract. In this paper, basic ideas and concepts of using artificial intelligence in design optimization of engineering systems are presented. The purpose of the study is to develop an expert (knowledge-based) system that helps the user in design optimization. Two basic ideas are advocated: (1) the successful numerical implementation of algorithms needs heuristics; and (2) the optimal design process can be greatly benefited by the use of heuristics based on knowledge captured during the iterative process. Various steps in the optimization process, where artificial intelligence ideas can be of tremendous help, are delineated. Some simple rules are presented to utilize the knowledge base and raw data as it accumulates in the iterative process. A simple example is used to demonstrate some of the basic ideas.

- 3.8 Mukhopadhyay, S. and Arora, J.S., "Implementation of an Efficient Run-Time Support System for Engineering Design Environment," *Advances in Engineering Software*, Vol. 9, No. 4, pp. 178-185, 1987.

Abstract. The need for both relational and matrix data types in engineering applications has been long recognized. While matrices form mostly temporary or semi-permanent data private to a program, relations are either permanent data in public domain used by different programs or final results of a program to the end user. Though several systems came up in the last few years with various degree of facilities and level of efficiency, none however, met the requirement of versatile data structure or run-time support required in a volatile, large I/O environment of engineering database.

This paper describes implementation of a system and its evaluation using an existing users' interface. Benchmarking shows that the system is far superior to the existing ones and also incurs little overhead for DBMS calls.

- 3.9 Mukhopadhyay, S. and Arora, J.S., "Design and Implementation Issues in an Integrated Database Management System for Engineering Design Environment," *Advances in Engineering Software*, Vol. 9, No. 4, pp. 186-193, 1987.

Abstract. The need for a unified database management system for various engineering applications has long been felt. Due to the conflicting requirements, so far, most of the attempts in this field are at best partially successful. This paper presents broad perspectives of design and implementation issues of an integrated database management system. A unique feature of the system is its ability to define a unifying data model for matrices and relations. The system also provides for a unified language interface for different user groups. System architecture and a few distinguishing features are described. An evaluation of the system is presented.

- 3.10 Park, G.J. and Arora, J.S., "Role of Data Base Management in Design Optimization Systems," *J. Aircraft*, Vol. 24, No. 11, pp. 745-750, November 1987.

Abstract. To study the role of data base and data base management system, an interactive design optimization software system called IDESIGN5 is developed to solve nonlinear programming problems. Four promising algorithms are included to overcome the lack of unanimous choice of an algorithm. Tuning parameters and procedures of algorithms are implemented through extensive numerical experimentation. The interactive process consists of menu displays, advice for decisions, and applicable messages. Input data can be created interactively and the designer can change problem parameters, algorithm, and design variable data at any point of execution. If a design variable does not effect the optimization process, it can be given a fixed value interactively. Discrete variable optimization can be performed by using design variable status capability of the system. Graphics facilities are provided for decision making. The system consists of several modules that communicate with each other through a data base managed by a data base management system. Several example problems are solved in batch and interactive environments to test the system.

- 3.11 Arora, J.S. and Mukhopadhyay, S., "An Integrated Database Management System for Engineering Applications Based on an Extended Relational Model," *Engineering with Computers*, to appear, 1988.

Abstract. Engineering analysis and design of complex systems require the use of large software components. Development, maintenance, and extension of such a software system needs modern design and implementation techniques. Usually a large amount of data is generated. Flow of data is also quite complex adding further complications in

maintenance and extension of the software. A sophisticated database management system is needed to support data handling during the run-time environment as well as for the integration of various software components. Design and development of such a DBMS needs new concepts and ideas such that efficiency of calculations is not sacrificed. Degradation in efficiency due to the use of a DBMS can hinder large scale applications. The paper describes a generalized relational model to handle large matrices and tables that are encountered in numerous engineering applications. A DBMS based on the model is designed and implemented. The system supports run-time data management as well as data sharing between various software components. A preliminary evaluation of the system against some existing ones reveals the new concept and design to be quite appropriate for engineering applications. The system is very efficient and compact. Some details of design and performance of the system are given and discussed.

4. CONFERENCE PRESENTATIONS

- 4.1 SreekantaMurthy, T., Reddy, C.P.D. and Arora, J.S., "Database Management Concepts in Engineering Design Optimization," proceedings of the 25th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, May 14-16, 1984, Palm Springs, CA, (84-0967-CP) pp. 360-372.

Abstract. The paper deals with database management concepts in engineering design optimization. Complex nature of engineering data and the need to organize them are emphasized. Requirements of a good database management system (DBMS) for handling such data are presented. Various aspects associated with the development of data models, such as conceptual, internal and external models are discussed. Suitability of external models, like hierarchical, network and relational models, are discussed with reference to design optimization requirements. Some techniques to organize data of large matrices for efficient numerical computations are given. Essential components of a good DBMS, such as data definition, data manipulation, query capabilities, and memory management schemes are described. Existing DBMS for engineering applications are reviewed. Finally, details of an Engineering Database Management System (EDMS) are presented.

- 4.2 SreekantaMurthy, T., Shyy, Y-K. and Arora, J.S., "MIDAS: Management of Information for Design and Analysis of Systems," *Proceedings of the AIAA 26th Structures, Structural Dynamics and Materials Conference*, Orlando, Florida, April 1985 (85-0618-CP), pp. 85-95.

Abstract. The paper describes features, system design and implementation of a database management system called MIDAS. The system has capability to organize data of both relational and numerical models, and meets several important requirements -- a good data model, ability to organize large matrix data, handle various data types, simplified data definition and data manipulation languages, dynamic data definition, multiple database organization, speed of data access, and provision for temporary databases. Tabular and matrix form of data generated and used in design and analysis of systems can be conveniently organized using MIDAS. Details of various commands of the database management system MIDAS are presented.

- 4.3 SreekantaMurthy, T. and Arora, J.S., "A Methodology to Design Database for Finite Element Analysis and Structural Design Optimization Applications," *Proceedings of the AIAA 26th Structures, Structural Dynamics and Materials Conference*, Orlando, Florida, April 1985 (85-0743-CP), pp. 494-504.

Abstract. A methodology to design databases for finite element analysis and structural design optimization is presented. The methodology considers three views of data organization -- conceptual, internal and external. Tabular and matrix forms of data are included. Relational data model is used in the database design. Entity, relation, and attributes are considered to form a conceptual view of data. First, second and third normal form of data are suggested to design an internal model. Several aspects like processing, iterative needs, multiple views of data, efficiency of storage and access time, and transitive data are considered in the methodology.

- 4.4 Arora, J.S. and Baenziger, G., "Use of AI in Design Optimization," *Proceedings of the AIAA 26th Structures, Structural Dynamics and Materials Conference*, Orlando, Florida, April 1985 (85-0801-CP), pp. 834-846.

Abstract. In this paper, basic ideas and concepts of using artificial intelligence in design optimization of engineering systems are presented. The purpose of the study is to

develop an expert (knowledge-based) system that helps the user in design optimization. Iterative numerical algorithms must be used to accomplish this objective. It is well known that some algorithms when implemented in a computer program do not behave the way they are theoretically supposed to. Two basic ideas are advocated: (1) the successful numerical implementation of algorithms needs heuristics; and (2) the optimal design process can be greatly benefited by the use of heuristics based on knowledge captured during the iterative process. Various steps in the optimization process, where artificial intelligence ideas can be of tremendous help, are delineated. Some simple rules are presented to utilize the knowledge base and raw data as it accumulates in the iterative process. A simple example is used to demonstrate some of the basic ideas

- 4.5 Baenziger, G. and Arora, J.S., "Development of an Artificially Intelligent Nonlinear Optimization Expert System," *Proceedings of the 27th AIAA Structures, Structural Dynamics and Materials Conference*, San Antonio, TX, May 19-21, 1986, pp. 67-77.

Abstract. In the last SDM Conference some basic ideas on the use of artificial intelligence in design optimization were presented. In the present paper, the layout and design of an Artificially Intelligent Nonlinear Optimization Expert System is delineated. Implementation details of the system, including a general purpose inference engine, and the knowledge base for Nonlinear Optimization are discussed. The design of the expert system combines a modular rule based inference engine with database management and a nonlinear optimization package IDESIGN5. The inference engine has four major parts: an ordered forward-chaining inference algorithm which plans the combinations of actions required to achieve the application solution, a multi-objective backward-chaining evaluation algorithm, a knowledge base compiler, and a processor which operates the other parts by carrying out default plans and the plan developed. The most efficient and successful plans and search graphs are saved in composite form for future reuse. The system learns from statistical efficiency and success to modify the rule evaluation order and alternative plan selection. The nonlinear optimization knowledge base is rule oriented and includes control of algorithm selection, active set strategy, and reactions to solution conditions such as design variable trends, solution failure, convergence, instability, and other optimization concerns. Also included are rules that provide the basis for the interactive consultant functions of the system.

- 4.6 Park, G.J. and Arora, J.S., "Role of Database Management in Design Optimization Systems," *Proceedings of the 27th AIAA Structures, Structural Dynamics and Materials Conference*, San Antonio, TX, May 19-21, 1986, pp. 620-629.

Abstract. To study the role of a database and database management system, an interactive design optimization software system IDESIGN5 is developed to solve nonlinear programming problems (NLP). Four promising algorithms are included to overcome the lack of unanimous choice on an algorithm. Tuning parameters and procedures of algorithms are implemented through extensive numerical experimentation. The interactive process is well designed with menu displays, advice for decisions, and applicable messages. Input data can be created interactively and the designer can change problem parameters, algorithm, and design variable data at any point of execution. If a design variable does not affect the optimization process, it can be given a fixed value interactively. Discrete variable optimization can be performed by using design variable status capability of the system. Graphics facilities are provided for decision making. The trend or history of design parameters are displayed on the graphics terminal. The system consists of several modules which communicate with each other through a database managed by a DBMS. The DBMS is well exploited in managing complex data organization of the interactive system. Structural optimization problems are solved in batch and interactive

environments to test the system. IDESIGN5 has been also coupled to the finite element analysis program ADINA to solve nonlinear response optimization problems. The developed system is quite robust with diverse capabilities and wide possibilities for extension. The study shows (1) how database and DBMS has to be used in design optimization, (2) advantages and disadvantages of using a database and DBMS, and (3) what interactive capability is possible and what are its advantages. The systems like IDESIGN5 can be very useful tools in the practical design process.

- 4.7 Arora, J.S. and Baenziger, G., "A Nonlinear Optimization Expert System," Proceedings of the ASCE *Structures Congress '87*, Session on Knowledge Based Structural Design and Optimization, Orlando, FL, August 17-20, 1987 in *Computer Applications in Structural Engineering*, D.R. Jenkins (Ed.), ASCE, pp. 113-125.

Abstract. Several methods for optimum design of systems have been developed over the last twenty years. Most methods, however, work well only when used by the optimization expert. Designers who are not optimization experts have difficulties in making the algorithms and programs based on them work for their application. This indicates that the rules used by the expert in making the program work should be captured and put in the knowledge base to provide consulting help to the designers. Such an expert system should be well designed for knowledge acquisition and utilization. The paper describes the layout of such a nonlinear optimization expert system. Some of the rules used in the system are discussed. A simple application is included to demonstrate some of the capabilities.

- 4.8 Al-Saadoun, S.S. and Arora, J.S., "Interactive Design Optimization of Framed Structures," Proceedings of the ASCE *Structures Congress '87*, Session on Interactive Design and Optimization: Algorithms, Applications and Software, Orlando, FL, August 17-20, 1987, in *Computer Applications in Structural Engineering*, D.R. Jenkins (Ed.), ASCE, pp. 357-372.

Abstract. Optimum design of framed structures under multiple loading and constraint conditions requires substantial computational effort. Optimization algorithms when turned loose on such problems can actually fail resulting in wastage of human as well as computer resources. Therefore it is prudent to monitor progress of optimization algorithms for such complex design applications. The paper describes a formulation for optimum design of framed structures. The AISC code limits on element stresses, member maximum deflection, stability and slenderness ratios, width thickness ratios and nodal displacements are imposed in the design process. A procedure for optimization is developed and described. An interactive software to solve the problem is designed and implemented. It uses modular approach and modern database management concepts. The system is used to solve two design optimization problems. A postprocessor is available to select members from the AISC tables. The system is another step in making practical use of optimization a reality.

- 4.9 Arora, J.S. and Mukhopadhyay, S., "An Integrated Database Management System for Engineering Applications Based on an Extended Relational Model," Proceedings of the *Symposium on Engineering Data Base Management: Critical Issues*, (R.E. Fulton, Ed.) ASME CIE Conference and Exhibition, New York, August 9-13, 1987, pp. 49-58.

Abstract. Engineering analysis and design of complex systems require the use of large software components. Development, maintenance, and extension of such a software system needs modern design and implementation techniques. Usually a large amount of data is generated. Flow of data is also quite complex adding further complications in maintenance and extension of the software. A sophisticated database management system

is needed to support data handling during the run-time environment as well as for the integration of various software components. Design and development of such a DBMS needs new concepts and ideas such that efficiency of calculations is not sacrificed. Degradation in efficiency due to the use of a DBMS can hinder large scale applications. The paper describes a generalized relational model to handle large matrices and tables that are encountered in numerous engineering applications. A DBMS based on the model is designed and implemented. The system supports run-time data management as well as data sharing between various software components. A preliminary evaluation of the system against some existing ones reveals the new concept and design to be quite appropriate for engineering applications. The system is very efficient and compact. Some details of design and performance of the system are given and discussed.

- 4.10 SreekantaMurthy, T. and Arora, J.S. , "A Structural Optimization Program Using a Database Management System," Proceedings of the Symposium on *Engineering Database Management: Critical Issues*, (R.E. Fulton, Ed.) ASME CIE Conference and Exhibition, August 9-13, 1987, New York, pp. 59-66.

Abstract. Data management is one of the most important tasks in the development of sophisticated computer programs for optimization of practical structural systems. The paper describes and discusses the implementation of a structural optimization program which uses a relational database and a data management system. Features of the program and modules used in the program are described. Organizational structure of the database and the database design process are described. Data definition and data manipulation routines used in the optimization program are described. Several structural optimization example problems were solved to demonstrate the working of the program. Finally, the database, the data definition and manipulation routines, and the performance of the optimization program are evaluated and discussed.

APPENDIX 1

**A SURVEY OF DATABASE MANAGEMENT
IN ENGINEERING**

by

T. SreekantaMurthy and J.S. Arora

in

Advances in Engineering Software

Vol. 7, No. 3, 1985

A survey of database management in engineering

T. SREEKANTA MURTHY and J. S. ARORA

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242, USA

Database management in engineering fields is becoming extremely important. A number of reports and papers have been published on this subject and computer-aided design applications. It is important to know state-of-the-art to fully utilize the benefits of this new field. The paper contains a survey of literature on database management for engineering applications. The survey is broadly classified into database management concepts and systems. Research work of various authors in related areas such as database design methodology, data models, data definition and manipulation languages, database integrity and consistency are reviewed. Various database management systems (15 in all) currently in use are reviewed and their features tabulated. Capabilities and usefulness of these systems are emphasized. The survey is intended to provide important developments in data management field to the engineering community. Also, it should be useful to those currently engaged in research on the subject.

INTRODUCTION

Unprecedented developments in computer usage in the last five years have fostered equally impressive developments in usage of database management systems in various applications. Of particular importance is the use of database management in computer-aided design of engineering systems. The present day computer-aided design is a speedy way of designing better systems. Previously, the process was tedious and slow because of manual data generation or use of desk calculators. Development of finite element method in the mid-1950s along with modern digital computers, have made analysis and design of large systems possible. These developments, however, led to the problem of dealing with large matrices in small computer memory. Large amount of finite element data has posed a challenge to engineers to properly manage data. Furthermore, developments in design optimization algorithms posed a serious problem for designers in organizing large amounts of design data. Similarly, data management problems were encountered in other engineering disciplines.

The desire for data organization in the engineering field is not new. Engineers in pre-computer days, were systematically using set tables, charts and drawings to describe their system. With the use of small programs to generate analysis and design data, the standard methodology for data logging in tables and charts became less rigorous. This trend continued even with the computer-aided design of large

systems and brought about a total chaos in data organization. The situation was not different even within the environment of computer programs. As the computer programs for engineering were able to deal with large problems, the data storage on disk became more common. The data storage schemes were too tedious and made efficient and easy use of data almost impossible.

With use of computers in business and commercial fields, data management in accounting, inventory control, production scheduling, and other fields became important during early 1970s. An enormous amount of knowledge exists for data management in these areas. Sophisticated database management systems have been used and are constantly being improved. However, introduction of database management in the engineering field is quite recent. Due to the complex nature of information in engineering applications, the growth of database management systems has not taken place to the same extent as in business applications.

In this paper, a survey of literature on database management in engineering is given. The survey is broadly classified into database management concepts and database management systems. Research work of various authors in related areas such as database design methodology, data models, data definition and manipulation languages, database integrity and consistency are reviewed. Several database management systems that are currently in use are reviewed. Their capabilities and drawbacks are emphasized and tabulated.

DATA MANAGEMENT TERMINOLOGY, USE AND REQUIREMENTS

A number of papers have been published on database management concepts for engineering applications. The paper by Fellipa (1979) serves as an introduction to the subject of database management for scientific and engineering applications. The paper highlights the difference between the business data management and scientific data management. A comprehensive list of terminology relevant to scientific computing is given in another paper by the author (Fellipa, 1980). Since the terminology used in business DBMS is fairly new to engineers, this list serves as a starting point to get acquainted with various terminology. It is interesting to know how scientists actually use their data. The paper by Bell (1982) discusses some issues about data usage and also gives comparison between data modelling for scientific and business applications. Difficulties associated with identification of data entities, naming of entities and other related data modelling aspects are discussed. In order to bring out the difference between the use of database for business and engineering applications, Foisseau and Valette (1982) present a list of criteria. Special nature of scientific data identifies requirements of DBMS for scientific computing. Lopatka and Johnson (1978) provide a review of the requirements for a DBMS in

Accepted October 1984. Discussion closes September 1985.

supporting of engineering applications. CAD/CAM data management requirements for aircraft industry were identified by Fulton and Volgt (1976). Southall (1980) present requirements for management of engineering information in the areas of integration support, information management and computer program management. Data management requirements for IPAD and capabilities of information processor IPIP are given by Johnson, Comfort and Shull (1980).

DATABASE DESIGN METHODOLOGY

Several research studies have been conducted to find out a suitable way to design a database for engineering applications. There exists basically two different approaches to database design – one approach first generates a global schema and then derives local views from it; another approach obtains local views of different users and then integrates them to form a global view. Buchman and Dale (1979) analyze different methodologies to database design and present a framework for evaluating them. However, these methodologies appear to be not suitable for product-modelling databases. Lillchagen and Dokkar (1982) discuss the key principles needed for constructing product-modelling databases. Also, the paper depicts various methodologies for data abstraction processes leading to conceptual schemas. The number of stages required in the data abstraction process to transfer real world data to physical storage media is still not known. Grabowski, Eigner, and Raush (1978) adopted a four-stage concept to attain problem-independency as well as independence from a programming language and specific hardware. A table comparing various levels of data abstraction used by several researchers is given in another paper by the authors (Grabowski and Eigner, 1979). It has been widely accepted that the three conventional data models (hierarchical, network and relational) are not powerful enough to express all semantic structures of CAD data. Therefore, the authors suggest the use of semantic data models which serves as a theoretical tool for database designers in picturing information structures of CAD applications. The semantic model acts as an intermediate level of data abstraction between well known data models and real world information. However, the use of the semantic data model is still controversial and several research studies on this model are cited in Eberlein and Wedekind (1982).

DATABASE NETWORK

The concept of network of databases is becoming increasingly important in organizing data of several engineering applications. Databases can be considered interconnected, to form a network, through a set of programs which use them. Several researchers, Fellipa (1979), Fishwick and Blackburn (1982), and Managaki (1982) have discussed the use of database network. Data of final design and results of permanent nature which can be accessed by several applications can be stored in global database. Intermediate results, trial and error design process data needed for individual programs are stored in local databases. Database network for microcomputer, requires decomposition of a database into sub-databases. Jumarie (1982) made a preliminary study on suitability of a centralized database. The concept of master file which acts as a directory to databases and problems associated with integration of data in such databases are discussed by Czekalinski and Zgorzelki (1982).

DATA MODELS

The well-known data models – hierarchical, network and relational – have been studied by many researchers to find out their suitability for organizing engineering data. Koriba (1983) discusses applicability of ANSI/SPARC, CODASYL and the relational approach to CAD software design. The three levels of data view proposed by ANSI/SPARC are gaining wide acceptance and are likely to be incorporated in future CAD systems. On the other hand, relational approach is based on set concepts and provide a sound mathematical background. This approach provides high level of data independence, user friendly data definition and data manipulation capabilities. Relational model is becoming popular among database designers and users. Several researchers are currently working on this model. Fishwick and Blackburn (1982) discuss some advantages and disadvantages of a relational model from an engineering point of view. The paper reports the use of a relational database management system in an integrated design system. Blackburn, Storaasli and Fulton (1982) in another paper demonstrate the use of a relational database in CAD. Haskin and Lori (1982) have shown how the relational database management systems (System R) can be extended to accommodate arbitrary length of data items commonly encountered in engineering applications. The relational data model for CAD applications is emphasized by Valle (1976) stressing the flexibility aspect of relational view of data. Also the paper gives some examples related to the use of relational organization of graphic data. Some aspects of design of a relational information management system with reference to CAD and interactive data handling are also given. Foisseau and Valette (1982) propose a CAD data model based on 'type', 'object' and 'relation', and function concepts. Studies on the hierarchical model are mainly with reference to organizing large matrices. Lopez (1974) and Pahl (1981) use a hierarchical model for finite element data organization. Hypermatrix stiffness and load data are found to be most suitable to hierarchical data representation. A paper by Elliot, Kunni and Browne (1978) describes a hierarchical model of data and a DBMS system design based on it. Some practical examples on structural design and wind tunnel data management are also given in the paper. However, this system requires a precompiler to decode the data description and data manipulation commands in a source program.

DATA DEFINITION AND DATA MANIPULATION LANGUAGES

Development of suitable data definition (DDL) and data manipulation languages (DML) for engineering applications have been of interest to many authors. One of the major considerations in the design of data definition language was to keep the syntax concise and easy to use for application programmers. Several other important considerations in DDL design are described in detail by Elliot, Kunni and Browne (1978). They use special indicators in the source program code to identify the DDL and DML commands and to translate them using a precompiler to FORTRAN statements. These DDL and DML statements can be used to operate on a hierarchical data structure. Special features of DDL and DML in a relational database management system for interactive design are described by Shenoj and Patnaik (1983). In engineering design environment, database schema are continuously updated, deleted and modified. By combining DDL and DML, independence of

compilation and database can be achieved. Lafue (1979) suggested merging DDL and DML which are traditionally separate in database management systems. Thus, dynamic data definition is an important requirement of engineering database management systems. It is desirable to have a DDL and DML that support hierarchical, network and relational models. In this regard, IPIP software (Johnson, Comfort and Shull, 1980) supports both network and relational data models.

DATABASE INTEGRITY AND CONSISTENCY

Larger implications of integrated design databases are beginning to be understood. A large amount of information and an associated variety of data representations require automation of management of integrity and consistency of databases. In this regard the paper by Eastman (1978) gives some conceptual tools for organizing design systems and implementing automatic integrity management. Lafue (1978) addresses several issues related to semantic integrity and consistency maintenance of design databases. It is shown that the hierarchical structure is suitable for dealing with integrity maintenance of databases. A mechanism for effecting automatic integrity management based on validity flags is presented by Eastman and Fenves (1978). It is important to note that database support must extend considerably beyond providing passive I/O capabilities and must provide substantial assistance in integrity maintenance at various design stages. The implementation of integrity management schemes have not yet been fully realized. Entity state transaction management proposed by Kutay and Eastman (1983) allows partial integrity management schemes and is applicable to large engineering problems.

NUMERICAL DATA MANAGEMENT

The application of data management in numerical computations is fairly new. Finite element analysis and design optimization procedures require a substantial amount of matrix data processing. Data management systems require special facilities to deal with data of large matrices. A recognition of this need is made by Diani (1982) and a model is developed for numerical database arising in many scientific applications to keep track of large, sparse and dense matrices. The paper presents a generalized facility for providing data independence by relieving users of the need for knowledge of physical data organization on the secondary storage devices. Because of the limitation of core storage and to reduce the input-output operations involved in secondary storage technique, many investigations have been conducted on the efficient use of primary memory. A detailed survey by Pooch and Nieder (1973) gives various indexing techniques that can be used in dealing with sparse matrices. Darby-Dowman and Mitra (1983) describe a matrix storage schema in linear programming. Rajan and Bhatti (1983) presented a memory management scheme for finite element software. Sreekanta Murthy, Reddy and Arora (1983) describe the database management concepts that are applicable to the design optimization field.

DATABASE MANAGEMENT SYSTEMS

Several database management systems (DBMS) have been developed or are in the development stage. In the following section, 15 systems are reviewed. The capability of these systems are emphasized and important features are tabulated.

DELIGHT	- Design Language with Interactive Graphics and a Happier Tomorrow
DATHAN	- A Data Handling Program for Finite Element Analysis
EDIPAS	- An Engineering Data Management System for CAD
FILES	- Automated Engineering Data Management System
GIFTS	- GIFTS Data Management System
GLIDE	- GLIDE Language with Interactive Graphics
ICES	- Integrated Civil Engineering System
IPIP	- Information Processor for IPAD
PHIDAS	- A Database Management System for CAD
REGENT	- A System for CAD
RIM	- Relational Information Management System
SDMS	- A Scientific Data Management System
SPAR	- SPAR Database Management System
TORNADO	- A DBMS for CAD/CAM System
XIO	- A Fortran Direct Access Data Management System

DELIGHT. It stands for Design Language with interactive Graphics and a Happier Tomorrow (Nye, 1981). In its philosophy, the DELIGHT system is very close to the GLIDE system (Eastman and Henrion, 1980). DELIGHT is an interactive programming language. It has a good extension and debugging capability. It provides high-level graphic commands, a built-in editor and well-defined interface routines. A single statement, procedure or part of an algorithm can be tested without having to write and load/link a program. The system relies on virtual memory management of the operating system. It is difficult to use the system with large scale programs. Multiple users are not allowed in the system.

DATHAN. It stands for Data Handling Program. It is written mainly for finite element analysis applications (Sreekanta Murthy and Arora, 1983). The program has some basic in core buffer management scheme. It has the capability to store permanent and temporary data sets. Substructure files can be arranged quite easily with the same data set names for different substructures. Both integer and real data types can be handled. A drawback of the system is that the user has to keep track of the location from which a new data set has to begin. The system has FORTRAN data manipulation commands which are simple to use.

EDIPAS. It stands for Engineering Data Interactive Presentation and Analysis System (Heerema and van Hedel, 1983). It is a tool for data management, analysis, and presentation. The data management part provides a utility to initialize a project database, input programs to load data from files into database under user controls, and a set of routines to extract data from and load data into database in a controlled way. EDIPAS allows users to name a database, a data structure, and data entities. It allows the user to employ one or more hierarchical levels. The data is stored in entities called blocks. A data block allows matrices, single values and characteristic values as data elements. A database administration support provides initialization of database, access of users, deletion of data structures, audit database contents, and the back-up facility. The system does not have a data redefinition facility. Improvements are being made to include a redefinition facility in order that the data structures and their levels can be manipulated. Extension of an authorization provision from database level to the level of data element is being incorporated.

FILES. This is an automated engineering data management system (Lopez, 1974). It is extremely flexible with respect to the definition of a database and methods of accessing it. Information storage and retrieval may be performed using problem-oriented languages. Hierarchical data structure is provided. For example matrix type of data encountered in finite element application can be organized using hierarchical data structure. The first two levels in hierarchy may contain pointers to the third level containing actual matrix data. The program allows dynamic memory allocation. Data transfer takes place between FORTRAN common block and database. FILES has a data definition language. The System does not have data mapping language to specify mapping of data items and arrays to an external device. The data definition language (DDL) depends on the problem oriented language (POL). Therefore DDL cannot be used independently. The system requires a distinct data management compiler.

GIFTS. This is an interactive program for finite element analysis (Kamel, McCabe and Spector, 1979). It is a collection of modules in a program library. Individual modules run independently and communicate via the unified database. The database manager processes requests for opening a file, closing a file, storing data set in a file, and retrieving data set from a file. The program has a memory management scheme. Each data set is stored in a separate random access file. Paging is carried out within the working storage. A unique set of four routines are associated with a data set for opening and initializing the working storage, for reading a data set, for creating/modifying the data set, and for realizing the working storage. Drawbacks of the system are that every new data set created requires four new routines to be written. Each data set is associated with a separate common block, thereby increasing the number of common blocks in the system. The data manager is application dependent and cannot be used as a stand alone system.

GLIDE. This is a context-free database management system (Eastman and Henrion, 1980). It is designed to provide a high level facility for developing an individualized CAD system. It can be viewed as a language, a database management system, and a geometric modelling system. It allows users to define new record types known as FORM that consists of a set of attribute fields. It provides a primitive data type set to organize a database. It provides an excellent geometric modelling system or a graphic system. The drawback of GLIDE is that it does not allow multi-dimensional arrays.

ICES. Integrated Civil Engineering System is a computer system designed for solving civil engineering problems (Roos, 1966). ICES consists of a series of subsystems each corresponding to an engineering discipline. It provides a Problem Oriented Language which can be used to write subsystem programs (e.g. coordinate geometry program, stress analysis program). Command Definition Language is used by a programmer to specify the structure and required processing for each subsystem command. A Data Definition Language is used to specify the subsystem data structure. It uses its own programming language called ICETLAN (ICES FORTRAN) and has a precompiler which translates ICETLAN to FORTRAN statements.

A dynamic data structuring capability is provided in the system which helps to organize dynamic arrays in the primary memory. The hierarchical data structure is used for data modelling. Three hierarchical levels, equivalence class, members and attributes, are provided. Data is stored on secondary storage using random access files. The data

management program uses buffers to convert logical records to physical records. An identifier is supplied by the programmer which is a pointer giving the position on secondary storage of physical record. The programmer has a choice to either store data using dynamic arrays or use the data management system depending on the amount and use of the data. A drawback of the system is that it uses a pre-compiler ICETLAN to convert to the FORTRAN program instead of directly to machine language. Physical storage of data requires knowledge of address and pointers which the programmers have to give. Only three levels of hierarchy are adopted and it is difficult to extend to many levels of hierarchy.

IPIP. This is a state-of-the-art database management system satisfying engineering requirements (Johnson, Comfort and Shull, 1980). It offers a number of capabilities ranging from support for multiple schemas and data models to support for distributed processing, and data support for distributed processing, and data inventory management. An integrated software architecture supports all user interfaces: programming languages, interactive data manipulation and schema languages. IPIP supports a multiple-schema architecture of ANSI/SPARC database group. Three types of schemas – conceptual, external and internal schemas are supported. IPIP schema and data manipulation languages exhibit a high degree of integration and compatibility. The logical schema supports both the network and relational data models, and, functionally, the hierarchical data model. The internal schema of IPIP is written using the internal schema language compiler. The internal schema language overlaps that of the logical schema language to the greatest practical extent to minimize the amount of schema language with which the administrator must deal. IPIP software subcomponents consist of user interface, and data manager. The software of user interface is made up of precompilers, a query processor and compilers. Data manager software is made up of a scheduler, a message procedure interface processor, a common semantic processor, a database control subsystem, a data manipulation subsystem, a record translator, a presentation service, an access module, a resource manager and stubs.

PHIDAS. This is a data management system specially designed for handling a collection of structured data on minicomputers (Fischer, 1979). The architecture of PHIDAS is in accordance with the ANSI-2 schema. It has an external subschema based on the network model of CODASYL and an internal schema for physical tuning particularly suited for engineering database. The data description language is provided to describe schema and sub-schema. PHIDAS also has a storage structure description language. Data manipulation language is FORTRAN call statements to subroutines. A drawback of the system is that it is difficult to represent matrix type data.

RIM. This stands for Relational Information Management System (Comfort and Erickson, 1978). RIM has the capability to create and modify data element definition and relationships without compiling the schemas or reloading the database. It also provides the capability to define new types of data for use in special applications such as graphics. RIM supports three types of data: real, integer and text. Data definition and data manipulation languages are available to define or manipulate relations. The user has the capability to project, intersect, join and subtract relations. RIM has good query language. RIM's modification commands permit the user to update relation definition, change data values, attribute names, delete tuples and delete the entire relation.

Utility commands such as **LOAD**, and **EXIT** are provided to load a new database and close an existing database. A drawback of **RIM** is that it does not allow a relation having a row size more than 1024 computer words. The application oriented **FORTRAN** call statements do not have the capability to define attributes, relations, rules, etc., required in defining a schema. The system does not support management of a temporary database. Simultaneous operations on a number of databases are not possible.

REGENT. This is a system for the support of computer-aided design (Leinemann and Schlechtendahl, 1976). The main goal of the development was to provide a so-called 'system nucleus' in the sense of **ICES**. Improvement claimed for the system is that it has a powerful base language **PL/I** instead of **FORTRAN**. Interactive use has been considered in system development. The database management of **REGENT** provides facilities to compress a database, copy data between databases, and to change name and size of data elements. The database of **REGENT** is not a database in the usual sense. It is some sort of partitioned data set concept, built up using a tree structure of sequential files, but the internal structure of these files is known only to those programs that use them.

SDMS. This is a database management system developed specifically to support scientific programming applications (Massena, 1978). It consists of a data definition program to define the form of databases, and **FORTRAN** compatible subroutines to create and access data within them. Database contains one or more data sets. A data set has form of a relation. Each column of a data set is defined to be either a key or data element. The key must be a scalar. Data elements may be vectors or matrices. The element in each row of the relation forms an element set. Temporary database capability that vanishes at the end of a job is provided. A scientific data definition language provides a program-independent data structure. Both random and sequential access of data set is possible. Data elements include scalars, fixed and variable length vectors, fixed and variable-size matrices. Data element types include text, real and integer. A drawback of the system is that it does not have a query language. Generalized database load/unload is not available. Double precision data type is not allowed. The system is implemented only on **Cyber** series computers.

SPAR. The computer program is a collection of processors that perform particular steps in finite element analysis procedure (Whetstone, 1977). The data generated by each processor is stored on a database compiler that resides on an auxiliary storage device. Each processor has a working storage area that contains the input and the computed data from the processor. Allocation of spaces in the storage area is a problem dependent and is dynamically allocated during execution. Data transfer takes place directly between a specified location on disk using a set of data handling utilities. **SPAR** database complex is composed of 26 data libraries or data files. Libraries 1-20 are available for general use. Libraries 21-26 are reserved for temporary and internal use. The database manager uses a master directory to locate the table of contents which in turn is used to locate the data sets in the database. Physically, the auxiliary storage is divided into sectors of fixed size and each read/write operation begins at the beginning of a sector. Drawback of the system is that it does not provide either a hierarchical or relational data structure. Excessive fragmentation may take place if the sector size does not happen to be an integral multiple of the data that is stored.

TORNADO. This is a **DBMS** system developed for **CAD/**

CAM application (Ulfsby, Steiner and Oian, 1979). It is a **CODASYL** network system written in **FORTRAN** and is very useful for handling complex data structures. It handles variable object length and dynamic length records. The system allows different data types - integer, real, character, double precision, double integer, complex and logical data. The system has easy to use data definition language and data manipulation language. **TORNADO** system is highly portable. Data in the database can be accessed by name. There is no restriction on data set types and allows many-to-many relationships. Some drawbacks of the system are that the size of a data object defined by the system is limited by the largest integer value that can be represented in the computer, the size of the database is limited by the maximum size of a file, a multi-file version is not available and the database cannot be used by multiple users at the same time.

XIO. This is a set of subroutines that provides a generalized data management capability for **FORTRAN** programs using a direct access file (Ronald, 1978). The system allows arrays of integer, real double precision and character data storage. Both random access and sequential access of data is provided. Variable length record I/O is allowed in the system. The bit map scheme is used to identify the unused space for storage of data to minimize disk storage requirements. The program allows a restart facility using saved files following completion of a partial execution or after a program termination. The system at present is only implemented on **IBM360** or **DEC PDPII** computing systems. The system does not provide a data definition language. Nor does it provide either hierarchical or relational data structures.

Capabilities of various systems are summarized in the table.

SUMMARY AND CONCLUSIONS

A survey of database management in engineering is presented. Various topics starting from database management terminologies, requirements, data models, to advanced research topics like integrity of database, transaction management, database design methodology are covered. Many research studies on engineering data management are derived from studies on business data management area. Therefore, they do not address many problems of engineering database management. Several researchers are currently working on database management topics related to computer-aided design. But, a majority of them are related to graphic applications. There is a great scope for further research in engineering database management.

Application of database management to finite element analysis and design optimization is fairly new. Many of the general engineering database management concepts can be applied to data organization of analysis and design computations. Only a few researchers have worked on numerical database management. Many problems of data organization in engineering computing are still not solved.

Several database managements systems that are suitable for engineering data organization are reviewed. Their capabilities and limitations are described. Some of the database management systems are used in finite element analysis and design optimization problems. There is a need for good database management systems that can deal with engineering computation, as well as graphic data.

Table 1. Features of various database management systems for engineering applications

		A = Available AS = Available but specialized		H = Hierarchical F = FORTRAN		N = Network PL = PL/I		R = Relational O = Others		D = Data set			
No	DBMS	Host language	External model	Internal model	Data dynamic definition	Data manipulation language	Memory management	Application interface	Inter-active capability	Matrix data organization capability	Geo-metric data capability	Simultaneous access to multiple database facility	Multiple user Temporary database
1	DELIGHT				AS	AS	A	A	A	A	A		
2	DATHAN	F		D	A	A	A	A	A	AS			A
3	EDIPAS		H		A	A	A	A	A	AS	A		
4	FILES	F	H		AS	A	AS	A	A	A		A	A
5	GIFTS	F		D		A		A	A	AS	AS		A
6	GLIDE				AS	AS		A	A		A		
7	ICES	O	H, D		AS	AS	AS	AS					
8	IPIP		H, N, R	A	A	A		A	A		A	A	A
9	PHIDAS	F	N		A	A		A	AS		A		
10	RIM	F		R	A	A	A	A	A	A	A		A
11	REGENT	PL			AS	AS	AS	AS					
12	SDMS	F		R	A	A	A	A		A		A	A
13	SPAR	F		D	A	A	AS	A		A		A	
14	TORNADO	F	N		AS	A	A	A			A		
15	XIO	F				A	AS			A			

ACKNOWLEDGEMENT

This research is supported by the Air Force Office of Scientific Research, Grant No. AFOSR 82-0322.

REFERENCES

- Alamwala, K. A. and Mayne, R. W. Interactive computer methods for design optimization, *Computer-aided Design* 1979, 11 (4), 201
- Bell, Jean Data modelling of scientific simulation programs, *Int. Conf. on Management of Data, ACM-SIGMOD*, pp. 79-86, 1982
- Blackburn, C. L., Storaasli, O. O. and Fulton, R. E. The role and application of database management in integrated computer design, *Journal of American Institute of Aeronautics and Astronautics* 1982, pp. 603-613
- Browne, J. C. Data definition, structures, and management in scientific computing, *Proc. of ICASE Conference on Scientific Computing*, pp. 25-56, 1976
- Bryant, J. C. A data management system for weight control and design-to-cost, *NASA Conference Publication 2055*, 1978
- Buchmann, A. P. and Dak, A. G. Evaluation criteria for logical database design methodologies, *Computer-aided Design*, pp. 121-126, 1979
- Comfort, D. L. and Erickson, W. J. RIM - A prototype for a relational information management system, *NASA Conference Publications 2055*, 1978
- Czekalski, L. and Zgorzelski, M. Design database organization and access problems in large-scale machine manufacturing industry, *File Structures and Databases for CAD, IFIP*, 1982
- Daini, O. A. Numerical database management system: A model, *Int. Conf. on Management of Data ACM-SIGMOD*, 1982
- Darby-Dowman, K. and Mitra, G. Matrix storage schemes in linear programming, *SIGMAP Bulletin ACM*, No. 32, pp. 24-38, 1983
- Date, C. J. *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass., 1977
- Derwa, G. T. Advanced program weight control system, *NASA Conference Publication 2055*, 1978
- Eastman, C. M. and Henrion, M. The glide language for CAD, *J. Technical Councils of ASCE* 1980, 106 (TC1), 171
- Eastman, C. M. and Fenves, S. J. Design representation and consistency maintenance needs in engineering databases, *NASA Conference Publication 2055*, 1978
- Eastman, C. M. The representation of design problems and maintenance of their structure, *Artificial Intelligence and Pattern Recognition in Computer-aided Design, IFIP*, 1978
- Eberlein, W. and Wedekind, H. A methodology for embedding design databases into integrated engineering systems, *File Structures and Databases for CAD, IFIP*, 1982
- Elliott, L., Kunii, H. S. and Browne J. C. A data management system for engineering and scientific computing, *NASA Conference Publication 2055*, 1978
- Emkin, L. Z. ICES concepts - A modern system approach, *Computing in Civil Engineering*, pp. 89-107, 1978
- Felippa, C. A. Database management in scientific computing - I. General description, *Computers and Structures* 1979, 10, 53
- Felippa, C. A. Database management in scientific computing - II. Data structures and program architecture, *Computers and Structures* 1980, 12, 131
- Felippa, C. A. Fortran-77 simulation of word-addressable files, *Adv. Eng. Software* 1982, 4 (4), 156
- Fischer, W. E. PHIDAS - A database management system for CAD/CAM software, *Computer-aided Design* 1979, 11 (3), 146
- Fishwick, P. A. and Blackburn, C. L. The integration engineering programs using a relational database scheme, *Computers in Engng, Int. Comp. Engng Cont.*, pp. 173-181, 1982
- Foissey, J. and Vakette, F. R. A computer-aided design data model: FLOREAL, *File Structures and Databases for CAD, IFIP*, 1982
- Fulton, R. E. and Voigt, S. J. Computer-aided design and computer science technology, *Third ICASE Conf. on Scientific Computing*, pp. 57-82, 1976
- Galletti, C. U. and Giannotti, E. I. Interactive computer system functional design of mechanisms, *Computer-aided Design* 1981, 13 (3), 159
- Grabowski, H. and Eigner, M. A data model for a design database, *File Structures and Databases for CAD, IFIP*, 1982
- Grabowski, H., Eigner, M. and Rausch, W. CAD data-structures for minicomputers, *Third Int. Conf. on Computers and Engng*, pp. 530-548, 1978
- Grabowski, H. and Eigner, M. Semantic datamodel requirements and realization with a relational data structure, *Computer-aided design* 1979, 11 (3), 158
- Haskin, R. L. and Lorie, R. A. O. extending the functions of a relational database system, *Int. Conf. on Management of Data ACM*, pp. 207-212, 1982
- Haug, E. J. and Arora, J. S. *Applied Optimal Design*, John Wiley and Sons, 1979
- Heerenia, F. J. and van Hedel, H. An engineering data management system for computer-aided design, *Adv. Eng. Software* 1983, 5 (2), 67
- Jefferson, D. K. and Thomson, B. M. Engineering data management: experience and projections, *NASA Conference Publication 2055*, 1978
- Jenne, R. L. and Joseph, D. H. Management of atmospheric data, *NASA Conference Publication 2055*, 1978
- Johnson, H. R., Comfort, D. L. and Shull, D. D. An engineering data management system for IPAD, IPAD: Integrated Programs for

- Aerospace-vehicle Design, *NASA Conference Publication 2143*, 1980
- Jumarik, G. A decentralized database via micro-computers: a preliminary study, *Computers in Engineering*, Int. Comp. Engng Conf. ASME, pp. 183-187, 1982
- Kamel, H. A., McCabe, M. W. and Spector, W. W. *GIFTSS System Manual*, University of Arizona, Tucson, 1979
- Koriba, M. Database systems: their applications to CAD software design, *Computer-aided Design*, 1983, 15 (5), 277
- Kunni, T. I. and Kunni, H. S. Architecture of a virtual graphic database system for interactive CAD, *Computer-aided Design*, 1979, 11 (3), 132
- Kutay, A. R. and Eastman, C. M. Transaction management in engineering databases. Engineering Design Applications, Proceedings of Annual Meeting, Database Week, *ACM SIGMOD*, 1983
- Lafue, G. Design database and database design, *Third Int. Conf. on Computers in Engng and Building Design CAD78*, Brighton Metropole, Sussex, UK, 14-16, 1978
- Lafue, G. M. E. Integrating language database for CAD applications, *Computer-aided Design*, 1979, 11 (3), 127
- Leinemann, K. and Schlechtendahl, E. G. The Regent system for CAD, CAD Systems, *IFIP*, 1976
- Lillehagen, F. M. Towards a methodology for constructing product modelling databases in CAD, File Structures and Databases for CAD, *IFIP*, 1982
- Lopatka, R. S. and Johnson, T. G. CAD/CAM data management needs, requirements and options, *NASA Conference Publications 2055*, 1978
- Lopez, L. A. FILES: Automated engineering data management system, *Computers in Civil Engineering, Electronic Computation*, pp. 47-71, 1974
- Lopez, L. A., Dodds, R. H., Rehak, D. R. and Urzua, J. L. Application of data management to structures, *Computing in Civil Engineering*, 1978, 477
- Managaki, M. Multi-layered database architecture for CAD CAM systems, File Structures and Databases for CAD, *IFIP*, 1982
- Martin, J. *Computer Database Organization*, Prentice-Hall Inc., Englewood Cliff, NJ, 1977
- Massena, W. A. SDMS -- A Scientific Data Management System, *NASA Conference Publication 2055*, 1978
- Nye, W. DELIGHT -- Design Language with Interactive Graphics and a Happier Tomorrow, *Electronics Research Laboratory*, University of California, Berkeley, CA, 1981
- Pahl, P. J. Data management in finite element analysis, Wunderlich, W., Stein, E. and Bathe, K. J. (eds) *Nonlinear Finite Element Analysis in Structural Mechanics*, Springer-Verlag, Berlin, 1981
- Pooch, U. W. and Neider, A. A survey of indexing techniques for sparse matrices, *Computing Surveys*, 1973, 5 (2), 109
- Rajan, S. D. and Bhatti, M. A. Data management in FEM-based optimization software, *Computers and Structures*, 1983, 16 (1-4), 317
- RIM User's Guide*, Boeing Commercial Airplane Company, PO Box 3707, Seattle, Washington, 98124, 1982
- Ronald, D. P. XIO -- A Fortran direct access data management system, *NASA Conference Publication 2055*, 1978
- Roos, D. *ICES System Design*, The MIT Press, Massachusetts, 1966
- Roussopoulos, N. Tool for designing conceptual schemata of databases, *Computer-aided Design* 1979, 11 (2), 119
- Schriem, E. Functional software design and its graphical representation, *Computers and Structures* 1978, 8, 491
- Shenoy, R. S. and Patnaik, L. M. Data definition and manipulation languages for a CAD database, *Computer-aided Design* 1983, 15 (3), 131
- Somekh, E. and Kirsch, U. Interactive optimal design of truss structures, *Computer-aided Design* 1979, 253
- Southall, J. W. Requirements for company-wide management of engineering information, IPAI* Integrated Programs for Aerospace-vehicle Design, *NASA Conference Publication 2143*, 1980
- Sreekanta Murthy, T., Reddy, C. P. D. and Arora, J. S. Database management concepts in engineering design optimization, *Proc. AIAA/ASME/ASCE/AHS 25th Structures, Structural Dynamics and Material Conference*, 1984
- Sreekanta Murthy, T. and Arora, J. S. A simple database management program (DATHA), *Technical Report*, Division of Material Engng, The University of Iowa, 1983
- Sreekanta Murthy, T. and Arora, J. S. Database management concepts in design optimization, *Technical Report*, Division of Material Engng, The University of Iowa, 1983
- Sreekanta Murthy, T., Reddy, C. P. and Arora, J. S. User's manual for engineering database management system EDMS, *Technical Report*, Division of Material Engng, The University of Iowa, 1983
- Ulfsby, S., Steiner, S. and Oian, J. TORNADO: A DBMS for CAD/CAM systems, *Computer-aided Design* 1979, 193
- Valle, G. Relational data handling techniques in computer-aided design process, CAD Systems, *IFIP*, 1976
- Whetstone, W. D. *SPAR Structural Analysis System Reference Manual*, System Level II, Vol. I, NASA CR-145098-1, 1977

APPENDIX 2

**DATABASE MANAGEMENT CONCEPTS IN
COMPUTER-AIDED DESIGN OPTIMIZATION**

by

T. SreekantaMurthy and J.S. Arora

in

Advances in Engineering Software

Vol. 8, No. 2, 1986

Database management concepts in computer-aided design optimization

T. SREEKANTA MURTHY and J. S. ARORA

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, Iowa 52442, USA

This paper deals with database management concepts in computer-aided design optimization. Complex nature of engineering data and the need to organize them are emphasized. Database management concepts applicable to finite element analysis and design optimization are explained. Various aspects associated with the development of data models, such as conceptual, internal and external models, are discussed. Suitability of external models, like hierarchical, network and relational, are discussed with reference to design optimization requirements. Some techniques to organize data of large matrices for efficient numerical computations are given. Concepts of normalization of data, global and local databases are described. Details of a suitable database management system are described in a separate paper.

INTRODUCTION

In the design of complex structural and mechanical systems, the need for data management has increased considerably. Sophisticated engineering design optimization methods, use large amounts of data and require substantial computer analysis. Also, these methods use finite element and other numerical techniques during analysis. The volume of data generated during analysis are used in design sensitivity and optimization phases. Data generated and used depends directly on the number of iterations performed by the optimization algorithm. Designer needs to exercise control over the data and the program to properly guide the iterative process towards acceptable designs. Significant improvement in design capability can be achieved with effective management of data. A properly designed database and a database management system when used with interactive computer graphics will be an invaluable tool for an engineer involved in the design process.

Data management systems in business applications such as accounting, inventory control and task scheduling, are quite sophisticated. However, due to the complex nature of engineering applications, growth of their data management systems has been very slow. Although advancement in computer-aided design (graphics) has led to the development of some database systems (DBMS) and concepts,¹⁻⁵ not much has been done in development of a suitable DBMS for

engineering design optimization. Therefore, it is important to look at the role of data management in design, and to study the applicability of various database management concepts in engineering design optimization.

This paper examines various database management concepts available for design optimization. Even though most of the concepts are frequently used in business database management, they are new to the engineering community. A number of terms are used in the paper for describing these concepts.⁶⁻⁸ Since many of the terms are derived from business applications, they do not bring out clear definition when used in engineering context.⁹⁻¹¹ Therefore, they are explained with finite element analysis and design optimization examples.

Design and implementation of a good database for any application needs a sophisticated DBMS. Without such a system it is not possible to evaluate different designs for the database. Requirements of a good DBMS for design optimization must be formulated.¹² Available methods for database design and management can be evaluated in the light of these requirements. Various data models applicable for design optimization are described with examples. In a separate paper,¹² database management system having many of the desirable features for design optimization and other engineering applications is described.

NEED FOR DATABASE AND ITS MANAGEMENT IN COMPUTER-AIDED DESIGN OPTIMIZATION

In optimal design of structural and mechanical systems, we generally use nonlinear programming techniques.¹³ The design objectives and constraints for the system are described in a mathematical model. Design of a system is specified using a set of parameters called design variables. The design variables depend on the type of optimization problem. In design of aircraft components such as stiffened panels and cylinders, the design variables are spacing of the stiffeners, size and shape of stiffeners, and thickness of skin. In optimization the structural systems such as frames and trusses of fixed configuration the sizes of the elements are design variables. Thickness of plates, cross-sectional areas of bars, moment of inertia represent sizes of the elements. If shape optimization is the objective, the design variables may include parameters related to geometry of the system.

The constraints for the system are classified into the performance and size constraints. The performance constraints are on stresses, displacements, and local and overall stability requirements in the static case; frequencies and displacements in the dynamic case; flutter velocity and divergence in aeroelastic case, or a combination of these.

Accepted July 1985. Discussion closes June 1986.

The size constraints are the minimum and maximum value of design variables. In nonlinear programming, the search for the optimum design variable vector involves iterative schemes. The design variable data at the n th iteration is used to compute a direction vector and a step size along it. The direction vector involves computation of gradients of objective and constraint functions with respect to the design variables. Data belonging to equivalent design variables are grouped there by reducing the size of design variable vector.

In most problems of structural and mechanical system design, behaviour of the system can be defined using state variables, e.g., stresses, displacements, and other response variables. In such a case, state space formulation is frequently employed.¹³ Design sensitivity coefficients in terms of matrix equation are determined in state space formulation. Adjoint equations are used to define a set of variables that provide design sensitivity information. Symmetric matrix equations can be used to advantage thereby reducing the data storage requirements.

Finite element and other numerical methods are used for analysis of structural and mechanical systems. Finite element method uses data such as element number, nodal connectivity, element stiffness matrix, element mass matrix, element load matrix, assembled stiffness, mass and load matrices, displacement vectors, eigenvalues, eigenvectors, buckling modes, decomposed stiffness matrix, and the stress matrix. In general, data used in finite element and other numerical analysis procedures is quite large. Symmetry of stiffness and mass matrices is taken into account so that data storage requirement is reduced. Hypermatrix or other special schemes are generally used in dealing with large matrix equations.

For design of large structures, efficient design sensitivity analysis is particularly critical. For such structures, substructuring concept can be effectively integrated into structural analysis, design sensitivity analysis, and optimal design procedures.¹³ In this concept, one deals with small order matrices as the data can be organized substructure-wise. The degrees of freedom can be classified into boundary degrees of freedom and interior degrees of freedom. Data for the stiffness matrices corresponding to these degrees of freedom can be separately stored. Data of constraint functions corresponding to internal and boundary degrees of freedom are used in determining design sensitivity calculations. Adjoint matrix data is stored for each substructure.

Many real world problems will have features that are not explicitly contained in general optimal design formulation. Problems with peculiar features need to be treated by making minor alterations in the general algorithm. Interactive computation and graphics can be profitably employed in design optimization. At a particular iteration, the designer can study the data of design variables, constraints which are active, performance of the system, cost function, admissible direction of travel, sensitivity coefficients, etc. He can make judgement regarding suitability of a particular algorithm, change of system parameters, and redefine convergence parameters to achieve optimal design. Interactive graphics requires additional data for display of system model, results, and graphs.

Thus, for design optimization, data generated during analysis must be saved in the database. This data is used for formulation of constraints. Constraints are checked for violation. Design sensitivity analysis of violated constraints is carried out using most of the data generated during

analysis. Once design sensitivity analysis has been completed, a direction finding problem is defined and solved. Note that the size of direction finding problem at each iteration depends on the number of active constraints. Therefore, sizes of data sets change from iteration to iteration. Thus, the nature of data is quite dynamic. We should be able to dynamically create large data sets, manipulate them during the iteration, and delete some of them at the end of iteration. Useful trend information from each iteration must be saved for processing in later iterations. Note that a row of the history matrix (such as design variable values) is generated at each iteration. However, to use the trend information for a quantity (e.g., a design variable), we need to look at its value at the previous iterations. This implies that we should look at a column of the history matrix. Therefore, we should be able to create data in one form and view it another. Thus, we must have an intelligent and sophisticated DBMS. Data must be organized, saved in a database, and properly managed for design optimization.

DATABASE MANAGEMENT CONCEPTS FOR DESIGN OPTIMIZATION

The problem is how to organize data in a database, what kind of information is to be stored, what kind of database management system is suitable, and how data is manipulated and used. In this regard, sophisticated techniques are available in business data management area to deal with complex data organization problems. The techniques used in existing finite element programs, however, are primitive and difficult to use. Therefore, a study of database management concepts is made to understand various methods available for data organization and to implement them for structural design applications. The concepts are explained here with reference to finite element analysis and design optimization examples. They are, however, suitable for other engineering and scientific applications.¹⁴

Definition of various terminologies

A number of terminologies and definitions are given to facilitate descriptions in subsequent sections. More discussions can be found in refs. 6-11.

Database. A database is defined as a collection of inter-related data stored together without harmful or unnecessary redundancy to serve multiple applications. The data stored so that they are independent of programs which use them. A common and controlled approach is used in new data and in modifying and retrieving existing data within the database. The data is structured so as to provide a foundation for future application development. One system is said to contain a collection of databases if they are entirely separate in structure.⁷

Logical data structure. Data in a particular problem consists of a set of elementary items of data. An item usually consists of single element such as integer, real and character or a set of such items. The possible ways in which the data items are structured define different logical data structures.

Model. The logical structure of data.

Schema. The coded form of logical data structure is called schema.

Entity. An entity may be 'anything having reality and distinctness of being in fact or in thought'.⁸

Entity set. An entity set is a collection of entities of the same type that exist at a particular instant, e.g., set of finite elements (ELEMENTS) and set of nodes (NODES).

Property. Property is a named characteristic of an entity, e.g., element name, and element material type. Properties allow one to identify, characterize, classify and relate entities.

Property value. It is an occurrence of a property of an entity, e.g., 'element name' has property value BEAM.

Domain. A domain is the set of eligible values for a property.

- Element name = (BEAM, TRUSS, ...)
- Element material type = (STEEL, ALUMINIUM, ...)
- Length = $x \ x > 0$ and $x < 100$

Attributes. Columns of a two-dimensional table are referred to as attributes. An attribute represents use of a domain within a relation. Attribute names are distinct from those of the underlying domains; e.g.,

- Domains: NODES = $i \ i > 0$ and $i < n$
- DOFS = $j \ j > 0$ and $j < m$
- Attributes: NODE1 – First node of an element derived from domain NODES
- DOF1 – First d.o.f. derived from domain DOFS

Relation: ELEMENT (E#, NODE1, NODE2)
Attributes NODE1, NODE2 are derived from the domain NODES.

Entity key. Entity key is an attribute having different values for each occurring entity and provides unique identification of a tuple. An entity represents a compound key if it corresponds to a group of attributes. It is also called candidate key.

Functional dependence. An attribute *A* is functionally dependent on the attribute *B* of a relation *R* if at every occurrence of a *B*-value is associated with no more than one *A*-value. This is denoted as $R.B \rightarrow R.A$. As an example, consider the relation: ELEMENT (ELMT#, EL-NAME, AREA). EL-NAME is functionally dependent on ELMT#. AREA is functionally dependent on ELMT#. ELMT# is not functionally dependent on EL-NAME, because more than one element could have the same name. Similarly, ELMT# is not functionally dependent on AREA.

An attribute can be functionally dependent on a group of attributes rather than one attribute. For example, consider the relation for a triangular finite element:

CONNECTION (NODE1#, NODE2#, NODE3#, ELMT#)

Here ELMT# is functionally dependent on three nodes NODE1#, NODE2#, and NODE3#. Given any one of NODE1#, NODE2#, or NODE3# it is not possible to identify ELMT#. These functional dependencies are shown in Fig. 1.

Full functional dependency. An attribute or a collection of attributes *A* of a relation *R* is said to be fully function-



Figure 1. Functional dependencies

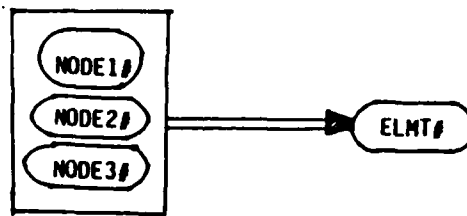


Figure 2. Full functional dependency

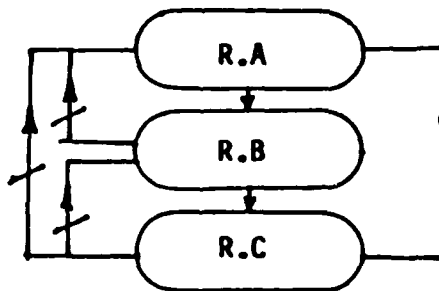


Figure 3. Transitive dependence

ally dependent on another collection of attributes *B* of *R* if *A* is functionally dependent on the whole of *B* but not on any subset of *B*. This is written as $R.B \rightarrow R.A$. In the Fig. 2, for example, ELMT# in the relation CONNECTION of a triangular finite element is fully functionally dependent on concentrated attributes NODE1#, NODE2#, and NODE3# because three nodes combined together define an element. NODE1#, NODE2#, or NODE3# alone does not identify ELMT#.

Transitive dependence. Suppose *A*, *B* and *C* are three distinct attributes or attribute collections of a relation *R*. Suppose the following dependencies always hold: *C* is functionally dependent on *B* and *B* is functionally dependent on *A*. Then *C* is functionally dependent on *A*. If the inverse mapping is nonsimple (i.e., if *A* is not functionally dependent on *B* or *B* is not functionally dependent on *C*), then *C* is said to be transitively dependent on *A* (refer to Fig. 3). This is written as

$$R.A \rightarrow R.B, R.B \rightarrow R.A, R.B \rightarrow R.C$$

Then, we can deduce that

$$R.A \rightarrow R.C, R.C \rightarrow R.A$$

For example, consider the relation

EL-DISP(ELMT#, EL-TYPE, DOF/NODE)

Here

- ELMT# \rightarrow EL-TYPE
- EL-TYPE \leftrightarrow ELMT#
- EL-TYPE \leftrightarrow DOF/NODE,
- DOF/NODE \leftrightarrow ELMT#

Therefore ELMT# \rightarrow DOF/NODE (transitively dependent)

VIEWS OF DATA AND DATA MODELS

A database can be viewed at various levels depending on the context. External view represents the data as seen by the interactive terminal users and application programmers. Conceptual view deals with inherent nature of data occurring in the real world information and represents a global view of the data. The data organization describing the physical layout is dealt at the internal level. At this level, one is concerned with efficiency and storage details. There is one more level of data organization below the internal level where the actual storage of data on a particular computer system becomes the main consideration. But, this aspect is a specialist's job and has no general guidelines. Therefore, it is not discussed here. Three levels of data – external, conceptual and internal are used to describe various views of data and are explained in the next three sections. These levels of data organization were suggested by ANSI/SPARC (Standard Planning and Requirements Committee). It is gaining wide acceptance in designing a database.

Traditionally users define the data in a database in terms of data sets. Other common approaches are through data

models – viz, hierarchical, network and relational. These data models are described in the following subsections with reference to finite element analysis and design optimization data.

Data set approach. User organizes the data using uniquely named data sets.¹⁵ Data sets are grouped to form a data library. How the contents of the data sets are managed is completely up to the application programs. Since most of the engineering data are highly unstructured, the data set offers a simple solution to describe the user's view of the data. This type of organization is quite simple, flexible and easy to use by application programmers. Further improvement in this type of organization can be done by defining ordered data sets. For example, row, column or submatrix order may be used to deal with matrix data. The data libraries formed by this approach may be classified according to project or their usage. For example, data of substructures in finite element analysis may be grouped substructure-wise, each in a separate library. This type of data modelling, however, has high redundancy. Also, it is not suitable for interactive use.

Hierarchical model. Data is organized at various levels using a simple tree structure. A tree is composed of hierarchy of elements called nodes. A detailed description of hierarchical model can be found in Date.⁶ Hierarchical structure appears to fit data of many design problems.¹⁶⁻¹⁸ To illustrate an application of a hierarchical model, consider a structure idealized by finite elements. Data pertaining to complete structure may be placed at root level. Finite elements data, node number data and material data can be placed at the next level. Such a model is shown in Fig. 4. An example of hierarchical model for hypermatrix data is shown in Fig. 5. Depending on the size of hypermatrix, it can be divided into number of submatrices and arranged at various hierarchical levels. In design optimization applications, the concept of design variable is fundamental to the development of a hierarchical model. Organizing design

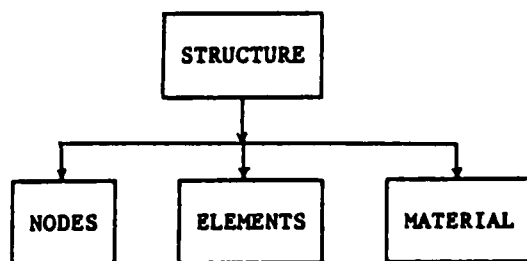


Figure 4. Hierarchical model for finite element analysis

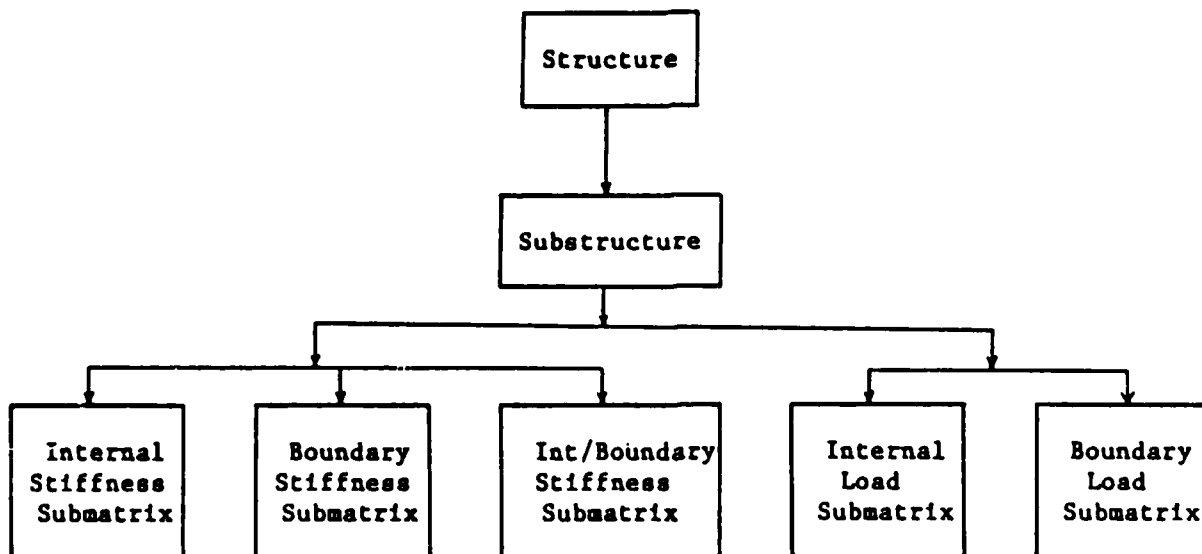


Figure 5. Hierarchical model for hypermatrix representation

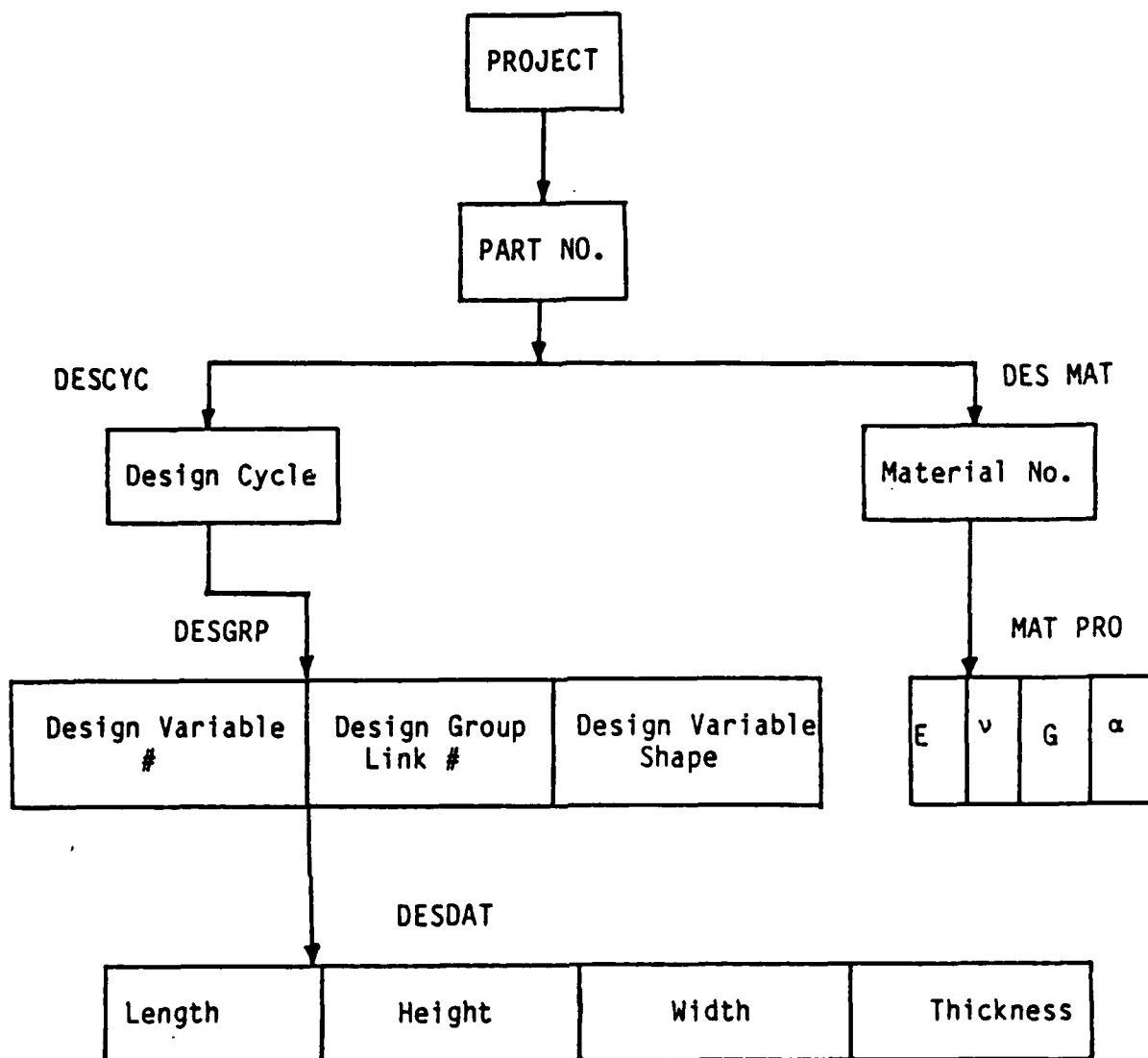


Figure 6. A hierarchical model for design variable data

variable data using a hierarchical model is described as follows. Figure 6 shows various levels of data in the hierarchical model. Root level in the model represents project name, followed by part number in the immediately next lower level. Part number has two dependent nodes having design cycle DESCYC and material number DESMAT data. At the fourth level DESGRP data consisting of design variable number, design group link, and design variable shape data are indicated. Material properties data MATPRO is also set up at the same level. Lower most level consists of detailed design data. We can see from the above examples that hierarchical model fits naturally with the usual subdivision of design data.

Network model. A collection of arbitrarily connected logical relations is called a network. A data model defined by such a network is called the network model. It is more general than the hierarchical model as it allows many-to-many relations. For example, in finite element analysis we

can form data model with items such as elements and nodes. Disadvantages of the model are in its complexity and the associated data definition language.

Relational model. A data model formed using relations is known as relation model.⁶⁻⁸ A relational data model is constructed from a tabular representation of the data. Figure 7 shows a relational model. The rows of the table are referred to as tuples. The columns are referred to as attributes. The relational model provides an easy way to represent data. The relational operations such as JOIN, INTERSECT and PROJECT can be used to form new relations. This model can provide easy access to data for the user. Also, tabular structure of the model provides a convenient way of representing engineering data that are generally in this form.

Relational model is quite appropriate for design optimization applications, since retrieval of data requires smaller preconceived paths. Applications generally require a com-

CORD

NODE NO	X	Y	Z

Figure 7. Relational model

plete set of related items simultaneously. Retrieving parts of information is not useful. In such a case the relational model which is set oriented provides a suitable way to organize the design data.

Numerical model. Most of the computations in design optimization involve operations on matrices like matrix addition, multiplications, solution of simultaneous equations, and eigenvalue calculations. The data models presented earlier are not tailored to handle matrix data effectively. It is necessary to provide a user-friendly facility for defining such numerical data and manipulating a numerical database.^{19,20} It is possible to provide such a facility by defining a new data model called numerical model. This numerical model is basically a variation of hierarchical data model having two levels of data representation. At the first level information pertaining to type and size of data is placed. The second level contains the actual numerical data. This model is shown in Fig. 8. A matrix is referenced through a user defined NAME or NUMBER. Various levels of sub-matrix organization can be defined through parameter called LEVEL. TYPE indicates the type of matrix: square, lower triangular, upper triangular, banded symmetric, banded nonsymmetric, diagonal, etc. ORDER indicates user's view of matrix storage; e.g., rows, columns or sub-matrices. For storage or retrieval of matrix data, the unit of transaction will be in terms of ORDER; i.e., row, column or full matrix. Dimension of matrix is represented by the number of ROWS or COLUMNS. PRECISION parameter specify the tolerance required while performing floating point operations. NULL parameter specifies if matrix is null or not. By checking this parameter unnecessary operations on null matrices can be eliminated, thereby saving considerable storage and execution time.

Examples of numerical model are shown in Fig. 8. Sub-matrix organization for level 1 is shown in Fig. 8a. The matrix information for level 0 can be described in the same way as shown for level 1. Symmetric banded matrix organization is shown in Fig. 8b. Note that rows or columns are of variable size. Fig. 8c shows an upper triangular matrix represented in the model.

Choice of data model. It is seen that various types of data models described in previous paragraphs could be used for design optimization applications. It is not possible to expect effective use of any one model in all situations. However, from practical considerations, we have to choose one data model for implementation purposes. Choice can be made on the merits of a data model to organize design and analysis data.

A hierarchical data model is suitable where the data to be organized occurs in a truly hierarchical fashion. The model, however, requires complex manoeuvres through a chain of pointers to access a particular data. Also, it has a fixed structure and offers little flexibility to change for alternate structures. Another drawback of the hierarchical model is the complexity of database design requiring tedious process of establishing links between data. If new kinds of data are to be added or new information must be generated from a database, it is necessary to add new links. Generally, this process requires redesigning the entire database. Network data models have similar problems, although addition of new item is much easier compared to the hierarchical data model.

Relational data model provides maximum flexibility of all the three data models. Moreover, relational data model is easier to understand. Users find it natural and familiar to

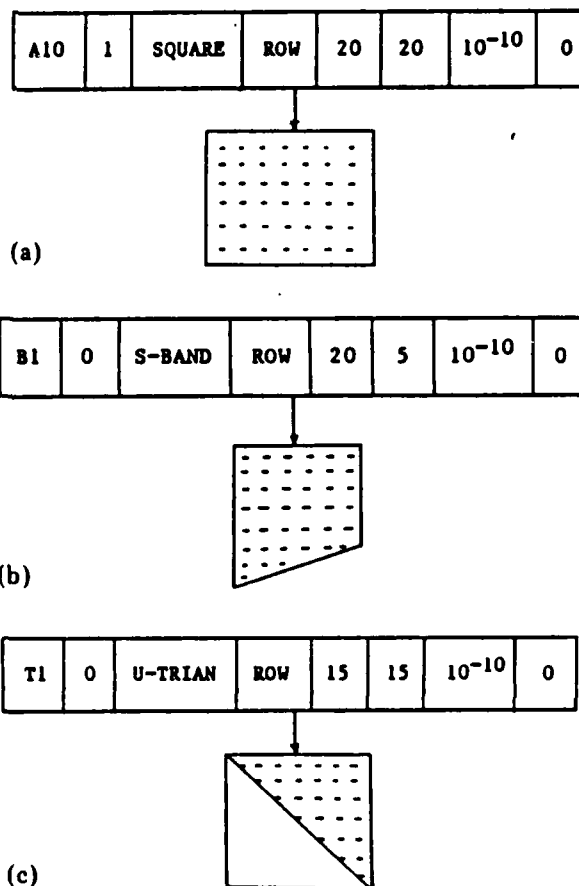


Figure 8. Examples of numerical data model: (a) A square matrix; (b) a symmetric banded matrix; (c) an upper triangular matrix

organize data in tables. A major advantage of the model is the ease with which database can be changed. As the design evolves new attributes and relations can be added, and existing ones deleted easily. The model is more appropriate for design applications, since data storage and retrieval uses a less preconceived path. It is possible to support a simple query structure using this model.

Thus, it is seen that the relational data model is appropriate for design and analysis applications. The relational data model alone, however, does not provide the capability to organize large matrix data. A numerical model which is basically a variation of the hierarchical model appears to be quite effective in representing matrix data structures. In any general design situation, coexistence of these data modelling facilities is desirable.

A conceptual data model

Conceptual model describes the structure of all types of data that need to be stored in a database. It represents real world data independent of any computer constraints and therefore provides a theoretical basis for organizing data of finite element analysis and design optimization problems. A logical approach for conceptualization of data is through information collection and analysis of data.⁶ Conceptual model can be derived either by first forming global views and then deriving local views, or by aggregating local views to form a global view. The first approach involves entity identification, relation formation, and name assignment. The other approach is based on combining segregated views of usages and information contents of individual perspectives.

Information collection is through identification of entities based on their properties. To illustrate how this may be done in design optimization, consider the entity PLATE. The entity has properties of two side dimensions, thickness, and material constant E . The associated property values are, say, 100, 50, 0.5, and 10^7 , respectively. These property values belong to a certain domain, for example, domain of property 'dimension' is $x \ 0 \leq x \leq 200$. Entity sets are formed by considering entities so identified. Attributes are associated from entity sets to domain. Finally, unique names are assigned to the entities and attributes so formulated.

Various complexities are involved in entity identification in design optimization area. Design specifications of a system are expressed quantitatively in terms of a mathematical model using design variables. They are dependent on the type of optimization problem. For example, in optimization of trusses of fixed configuration, sizes of members become design variables. If shape optimization is the objective, design variables consist of parameters related to geometry of the system. Therefore, one cannot arbitrarily form an entity set 'Design Variable'. Further, in identification of entities, time must be modelled appropriately.

Analysis of information about conceptual objects — entities, attributes and relations — leads to the formation of a conceptual model which replaces the real world information. It requires determination of functional dependencies of various kinds.^{6,7} But it is difficult to clearly identify all possible dependencies in the design data. Moreover, these dependencies could change from time to time. Finally, transformation of list of conceptual objects into irreducible units called elementary relations gives a conceptual model representing all relevant data.

Assumption that all design data can be conceptualized into a model may not hold good. This is particularly true

of many complicated design optimization procedures where identification of entities, relations, etc., is not clear. Moreover, new design optimization methods continuously evolve, leading to change in conceptual models and thereby imposing constraints on existing models. But it is important to examine the suitability of available tools and techniques for conceptualization before they can be totally rejected.

Internal model

Logical organization of data to be stored on physical storage media is described by an internal model. It is basically organization of elementary relations or parts of them and storing them as a unit to reduce number of accesses. One approach to define an internal model is by means of relations and use of normalization^{6,7} criteria to obtain relations consistent with the conceptual model. Such a collection of relations reduces redundancy, eliminates undesired anomalies in storage operations, and ensures database integrity. So far intuitive approaches have guided the design of an internal model for many existing finite element packages. With the availability of new tools for designing an internal model a more rigorous and logical approach can be used to design a database.

Relational and numerical data modelling concepts are appropriate to design an internal model for finite element analysis and optimization data. Collection of relations and matrices can be interpreted as sets, records and data items. Such sets are stored together and accessed as a unit. Details of how the data items are connected to form a record, set and finally physical blocks are part of the physical data structure. The task of physical data structure is to produce an access path for transferring logical data to physical space. Stored data mapping description encodes the access path to corresponding storage structure. This mapping provides flexibility for tuning the physical structure independent of the logical structure to achieve varying degree of efficiency and physical data independence.

External model

One of the important requirements of a database is to provide facility for data retrieval by different application programs depending on their needs. Different application programs can have different views of a database. To illustrate this consider the needs of two finite element analysis programs. Let one finite element analysis program use skyline approach to assemble and solve system equations. Another program using hypermatrix approach to perform similar task needs to use the same database. Basically, the two application programs using some common data such as geometry, material and other finite element idealization data should be able to derive them through an existing database. This aspect of catering to the needs of different application is possible through an external model.

Data structure as seen by an application program or interactive user is called an external data model. Data retrieved from actual physical storage in the database undergoes transformation till it reaches the user. Transformation involves rearrangement of data from internal level to external level into a form acceptable to the application program.

Some constraints have to be observed while designing an external model. Constraints arise while rearranging data from internal data structure to an external data structure. An important constraint is that internal data structure must be consistent with the conceptual data structure. Any retrieval and storage operation specified on the external

model must be correctly transformed into corresponding operations on the internal model and at the same time data must be consistent with the conceptual data model. Also design of the external model must fit the database management system capability.

Normalization of data

It was seen in the previous sections that data items are grouped together to form associations. An issue of concern here, is how to decide what data items have to be grouped together? In particular, using a relational model, determining what relations are needed and what their attributes should be? As database is changed, older views of data must be preserved so as to avoid having to rewrite the programs using the data. However, certain changes in data associations could force modification of programs, and could be extremely disruptive. If grouping of data items and keys is well thought of originally, such disruptions are less likely to occur.

Normalization theory^{6,7,8} provides certain guidelines to organize data items together to form relations. The theory is built around the concept of normal forms. A relation is said to be in a particular normal form if it satisfies a certain specified set of constraints. Three normal forms – 1st, 2nd and 3rd – are described below.

First normal form (1NF). A relation is said to be in the first normal form if and only if it satisfies the constraint of having atomic values.

As an example, Fig. 9 shows the relation CONN between four attributes ELMT#, E-NAME, NODES# and DOF/NODE with domains D₁, D₂, D₃ and D₄. The relation is first shown not in the 1NF and then in the 1NF.

Second normal form (2NF). A relation is in second normal form if and only if it is in 1NF and every non-key attribute is fully functionally dependent on each candidate key.

Let us see if the relation CONN of Fig. 9 in the 1NF is also in the 2NF. Consider a non-key attribute E-NAME:

ELMT#, NODES# → E-NAME

ELMT# → E-NAME

NODE# ↗ E-NAME

Therefore, ELMT#, NODES# ↗ E-NAME, i.e., E-NAME is not fully functionally dependent on (ELMT#, NODES#).

Similarly for the non-key attribute DOF/NODE:

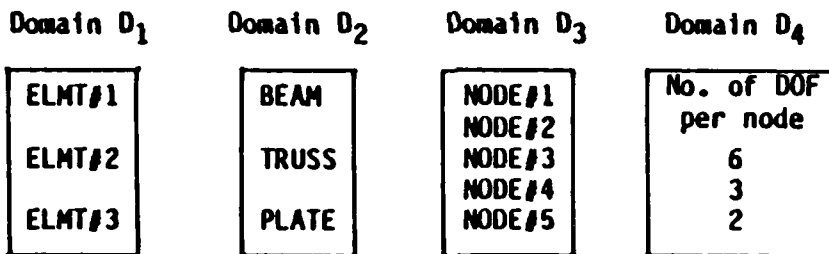
ELMT#, NODES# → DOF/NODE

ELMT# → DOF/NODE

NODE# ↗ DOF/NODE

Therefore, ELMT#, NODES# ↗ DOF/NODE. Since neither E-NAME nor DOF/NODE is fully functionally dependent on candidate key (ELMT#, NODES#), the relation CONN is not in 2NF.

Conversion of the relation CONN to 2NF consist of replacing CONN by two of its projections (refer to Fig. 10):



Key Key

CONN	ELMT#	E-NAME	NODES #	DOF/NODE
1	BEAM	1	2	6
			3	
2	TRUSS	5	3	3
			4	
3	PLATE	2	3	2
			4	
			5	

Not in 1NF

Key Key

CONN	ELMT#	E-NAME	NODES #	DOF/NODE
1	BEAM	1	6	
1	BEAM	2	6	
2	TRUSS	3	3	
2	TRUSS	5	3	
3	PLATE	2	2	
3	PLATE	3	2	
3	PLATE	4	2	
3	PLATE	5	2	

In 1NF

Figure 9. First normal form for a relation CONN

NAM-DOF ← CONN (ELMT#, E-NAME, NODES#, DOF/NODE)

ELMT-NODE ← CONN (ELMT#, E-NAME, NODES#, DOF/NODE)

Relation ELMT-NODE does not violate 2NF because its attributes are all keys.

Third normal form (3NF). A relation is in the third normal form if it is in second normal form and its every non-prime attribute is non-transitively dependent on each candidate key of the relation.

For example, consider the relation NAM-DOF (Fig. 10) to see if it is in third normal form. It still suffers from a lack of mutual independence among its non-key attributes. The dependency of DOF/NODE on ELMT#, though it is functional, is transitive (via E-NAME). Each ELMT# value determines an E-NAME value and in turn determines the DOF/NODE value. This relation is reduced further into relations NAME and DOF. These relations (Fig. 11) are in the third normal form.

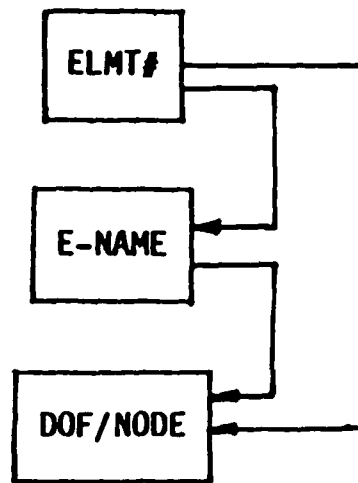
Global and local databases

Computer-aided design of complex structural systems uses several application programs during the design process. Many of these programs require common information such as geometry of the structure, finite element idealization details, material properties, loading conditions, structural stiffness, mass and load distributions, and responses resulting from the analysis runs. Also, it is common that data generated by one program is required for processing in subsequent programs in certain predetermined pattern. These data do not include transitory information such as intermediate results generated during an analysis run. The transitory information is highly unstructured and its usage pattern

NAM-DOF	ELMT#	E-NAME	DOF/NODE
	1	BEAM	6
	2	TRUSS	3
	3	PLATE	2

ELMT-NODE	ELMT#	NODE#
	1	1
	1	2
	2	3
	2	5
	3	2
	3	3
	3	4
	3	5

Figure 10. Second normal form for relation CONN



DOF	E-NAME	DOF/NODE
	BEAM	6
	TRUSS	2
	PLATE	2

NAME	ELMT#	E-NAME
	1	BEAM
	2	TRUSS
	3	PLATE

Figure 11. Third normal form for relation NAM-DOF

is known only to applications that use them. Generally, the transitory information is deleted at the end of a run. Therefore, there is a need for systematic grouping of the data.

A network of databases offers a systematic approach to support data of multiple applications.^{21,22} A network of databases consists of a global database connected to a number of local databases through program data interface. Application programs which use them may be thought of as links connecting the databases. A global database contains common information required for all applications whereas a local database contains only application dependent transitory data. Data in global database is highly structured and integrity of the database is maintained carefully. Data in a local database, however, is extremely flexible and integrity is not of importance.

The network of databases offers considerable aid in the structural design process. Any changes made to the data in global database is immediately available for use in other

applications of the system. Any new application program can be added to share the common data. The data views in global databases are clear to all applications and any modified views can be easily incorporated to suit a new application. Local databases are dependent on application programs and are highly efficient in data transfer operations since no overhead is involved in maintaining complicated data structures. It supports trial and error design process by providing scratch pad work space which can be erased from the local database at a specified design stage. Any intermediate results can be stored in a local database. Final results of a design can be transferred to a global database.

DISCUSSION AND CONCLUSION

Various concepts of database management applicable to engineering design optimization are discussed and illustrated with examples. Even though needs of database management in engineering and business applications are different, there is some commonality between them. The currently available tools and techniques for developing a good database management system have to be studied before they can be totally rejected. Also it is suggested to make use of available database management concepts and tailor them to suit our needs.

Conceptualization of design data, and internal and external models are discussed. User's view of data is described through hierarchical, network, relational and numerical models. Relational model is quite useful for optimization and general engineering applications. Numerical model is quite promising for dealing with matrix data. Coexistence of these data modelling facilities provides much flexibility to the user. Normalization of data is useful to associate various data items in an efficient and nonredundant way. Concept of global and local database provide a means to organize data at various levels in the design optimization environment. A companion paper¹² describes a database management system that is under development based on these concepts.

ACKNOWLEDGEMENTS

This research is sponsored by the Air Force Office of Scientific Research, Grant No. AFOSR 82-0322. The material of the paper is derived from presentations made by the authors at the 25th and 26th AIAA Structures, Structural Dynamics and Materials Conferences.

REFERENCES

- 1 Ulfaby, S., Steiner, S. and Olan, J. TORNADO: A DBMS for CAD/CAM systems, *Computer-Aided Design* 1979, 193
- 2 Fischer, W. E. PHIDAS - A database management system for CAD/CAM software, *Computer-Aided Design* 1979, 11 (3), 146
- 3 Cornfort, D. L. and Erickson, W. J. RIM - A prototype for a relational information management system, *NASA Conference Publications* 2055, 1978
- 4 Fulton, R. E. and Voigt, S. J. Computer-aided design and computer science technology, *Third ICASE Conf. on Scientific Computing* 1976, 57-82
- 5 Browne, J. C. Data definition, structures, and management in scientific computing, *Proc. of ICASE Conference on Scientific Computing* 1976, 25-56
- 6 Date, C. J. *An Introduction to Database Systems*, Addison-Wesley, Reading, Mass., 1977
- 7 Martin, J. *Computer Data-base Organization*, Prentice-Hall, Inc., 1977
- 8 Vetter, M. and Maddison, R. N. *Database Design Methodology*, Prentice-Hall International, 1981
- 9 Felippa, C. A. Database management in scientific computing - I. General description, *Computers and Structures* 1979, 10, 53-61
- 10 Felippa, C. A. Database management in scientific computing - II. Data structures and program architecture, *Computers and Structures* 1980, 12, 131
- 11 Sreekanta Murthy, T. and Arora, J. S. Database management concepts in design optimization, *Tech. Report No. CAD-SS-83.13*, The University of Iowa, Oct. 1983
- 12 Sreekanta Murthy, T., Shyy, Y-K. and Arora, J. S. MIDAS: Management of information for design and analysis of systems, *Adv. in Engng Software*, in press
- 13 Haug, E. J. and Arora, J. S. *Applied Optimal Design*, John Wiley & Co., 1979
- 14 Bell, Jean, Data modelling of scientific simulation programs, *Int. Conf. on Management of Data*, 2-4 June, ACM-SIGMOD, 1982, 79-86
- 15 Sreekanta Murthy, T. and Arora, J. S. A simple database management program (DATIAN), *Technical Report*, Division of Material Engng, The University of Iowa, Jan. 1983
- 16 Pahl, P. J. Data management in finite element analysis, in Wunderlich, W., Stein, E. and Bathe, K. J. (eds) *Nonlinear Finite Element Analysis in Structural Mechanics*, Springer-Verlag, Berlin, 1981, pp. 716-24
- 17 Elliott, L., Kunil, H. S. and Browne, J. C. A data management system for engineering and scientific computing, *NASA Conference Publications* 2055, 1978
- 18 Lopez, L. A., Dodds, R. H., Rehak, D. R. and Urzua, J. L. Application of data management to structures, *Computing in Civil Engineering* 1978, 477
- 19 Daini, O. A., Numerical database management system: a model, *Int. Confer. on Management of Data ACM-SIGMOD*, 1982
- 20 Rajan, S. D. and Bhatti, M. A. Data management in FEM-based optimization software, *Computers and Structures* 1983, 16 (1-4), 317
- 21 Jumarie, G. A decentralized database via micro-computers a preliminary study, *Computers in Engineering*, Int. Computer Engng Confer. ASME 1982, 4, 183
- 22 Blackburn, C. L., Storaasli, O. O. and Fulton, R. E. The role and application of database management in integrated computer-aided design, *Journal of American Institute of Aeronautics and Astronautics*, 1982, 603
- 23 Rajan, S. D. SADDLE: A computer-aided structural analysis and dynamic design language, *PhD Dissertation*, The University of Iowa, 1983

APPENDIX 3

**MIDAS: MANAGEMENT OF INFORMATION FOR
DESIGN AND ANALYSIS OF SYSTEMS**

by

T. SreekantaMurthy, Y-K Shyy and J.S. Arora

in

Advances in Engineering Software

Vol. 8, No. 3, 1986

MIDAS: management of information for design and analysis of systems

T. SREEKANTA MURTHY, Y-K. SHYY and J. S. ARORA

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242, USA

The paper describes features, system design and implementation of a database management system called MIDAS. The system has capability to organize data of both relational and numerical models, and meets several important requirements – a good data model, ability to organize large matrix data, handle various data types, simplified data definition and data manipulation languages, dynamic data definition, multiple database organization, speed of data access, and provision for temporary databases. Tabular and matrix form of data generated and used in design and analysis of system can be conveniently organized. Details of various commands of the database management system MIDAS are presented.

INTRODUCTION

Management of information has become an extremely important task in computer-aided design and analysis of engineering systems. Organization of large volume of design information is a complex task and requires careful considerations. Complexity in organizing information arises due to several reasons. First, the scientific methods of information management are still in their infancy in the engineering field when compared to sophisticated methods used in business applications. Secondly, the nature and use of engineering design information is different from business information. Thirdly, design information processing needs to consider efficiency aspects, as data is stored and retrieved a large number of times during execution of application programs. Since, a major part of design and analysis work is management of information, significant improvement in designer and application program efficiency can be achieved through better ways of managing such information.

Several systems for management of information are available for engineering applications. Systems such as FILES,¹ GIFTS,² RIM,³ SDMS,⁴ SPAR,⁵ PHIDAS⁶ and TORNADO⁷ have been used in some engineering applications. They have been developed with varying degree of sophistication and have a variety of capabilities. A study was made to find out the capabilities and usefulness of existing data management systems for design and analysis applications. It was found⁸ that use of such systems is limited to special applications for which they were developed. In particular, information programs such as FILES, GIFTS, and SPAR were used for finite element

analysis application. They are closely tied to the finite element analysis program and it is difficult to modify and extend them for design applications. On the other hand information management systems like RIM were found quite useful in integrating general engineering analysis programs, but their applicability to finite element analysis and design optimization is limited. Thus, a need of a good data (information) management system (DBMS) which can deal with organization of both design and analysis data exists.

With various requirements (discussed in the next section) in view, a database management system called MIDAS was designed. It was decided to use an existing package as much as possible. RIM is the most advanced system available for scientific database management. It supports relational data model facility. So, it was decided to see if the system can be extended to satisfy the requirements stated above. It was found difficult to extend RIM to have multiple databases, to organize large matrices, and to be efficient in handling large data sets and large memory. It essentially meant rewriting the memory management, and data definition and manipulation parts. So, it was decided to use RIM as is but add new data definition and data manipulation subroutines that could be called from a FORTRAN application program. This subsystem is called MIDAS/R which stands for MIDAS-Relational Data Management System.

A second subsystem called MIDAS/N was designed which stands for MIDAS-Numerical Data Management System. MIDAS/N supports numerical data model facility. This subsystem can handle multiple databases, small and large matrices, and small and large memory environment. This paper is intended to discuss the requirements of a DBMS, language facilities, system design and commands available in MIDAS. This will show the features required in a DBMS for engineering applications.

REQUIREMENTS OF A DBMS

It is important to lay down requirements of a good DBMS before discussing various features of MIDAS. Requirements of a DBMS for design optimization oriented applications differ considerably from requirements of business DBMS. They even differ from those for the computer-aided design (graphics) DBMS. For more discussion on the differences between business and engineering application DBMS, refs. 9 and 10 can be consulted. Before the requirements for a good DBMS are stated, types of DBMS are defined.

Context-free DBMS: Such a system is designed to work as a stand-alone package. The application program executes under the control of DBMS, i.e. DBMS acts as the main program. Thus data manager must be modified or extended when a new application is introduced.

Accepted July 1985. Discussion closes September 1986.

Application-oriented DBMS: Such systems are designed for a particular application. Their data definition, data manipulation and query languages use syntax of the application. It is either not possible to use the DBMS for other applications or it requires extensive modifications.

Application-independent DBMS: Such a system is designed based on the concept of a library of subroutines, and is not tied to any particular application. Any application software can call standard subroutines from the DBMS library to define, manipulate, and query its database.

Current trends in database management are toward designing a DBMS that meets certain standard requirements such as data independence, flexibility in data modeling, and device independence. Apart from these, some additional requirements must be met by DBMS for engineering design applications:

1. As FORTRAN is the host language for majority of the engineering applications, it is necessary to provide application interface with DBMS through standard FORTRAN statements.
2. Data model provided in DBMS must be easy to understand and apply for design application programmers and users. Data model must be flexible to suit the requirements of different applications. Relational and numerical data models are desirable.¹¹
3. Since the system will be used for design optimization in multidisciplinary environment, it must be an application-independent DBMS.
4. Design data consists of arrays and matrices to a large extent. Often matrix data are in the form of banded, submatrix, and triangular. A suitable data model, therefore, required to organize matrix data.
5. DBMS should be able to deal with various data types such as characters, short integers, long integers, single precision, double precision, and complex numbers.
6. Speed of storage and retrieval is one of the most important requirements of a DBMS in design optimization. Short access time will considerably reduce the total execution time in an iterative nature of design process.
7. Simple to use data definition and data manipulation languages are required. Applications often define data dynamically. For example, size of various matrices, length of data, etc., are not known at compilation time. They are also modified frequently. It is necessary to provide data definition capability to cater to these special needs.
8. DBMS should provide additional capability to organize the available primary memory. A suitable memory management scheme should be incorporated.
9. A good query language is required for interactive design applications. Query language should be general to cover all applications.
10. Provision for managing a temporary database will considerably help designers in evaluating trial designs and transferring the acceptable final design to a permanent database. Temporary databases will also be needed in iterative design optimization process. Thus, DBMS must be able to handle multiple databases simultaneously.

Data Definition Language (DDL)

Data definition language is a means to describe data types and logical relations among them. This definition

should identify the types of data subdivisions, assign a unique name to data types, specify the sequence of occurrence, specify the keys, assign length of data items, specify the dimension of a matrix and specify the passwords for database security. In building a data definition language for design optimization following points must be considered. First, data definition must be compatible with FORTRAN as a host language. Since data definition is continuously redefined in an application program, DDL must have feature to define data dynamically. Also provision for query of schema must be provided.

Data Manipulation Language (DML)

Data manipulation language consists of FORTRAN callable subroutines for storing, retrieving and modifying data in the database. These commands should be simple as they are frequently used in an application program. Data manipulation language commands include utility, schema information, create, and add commands. Utility commands are used for opening and closing a database, and printing error messages. Data manipulation commands' main operations are to get data from a database and store data into the database. Schema commands are useful in verifying data definition in situations where they are continuously changing.

Memory management

Efficient use of primary computer memory, is possible through judicious allocation of available space. Memory management scheme dynamically controls the allocation of available memory space. The memory is organized into pages and page sizes are assigned. The size of a page is set multiple of a physical record. The performance is better with larger page size. However, the space may be wasted if there are too many partially full pages. Small page size leads to increased page replacement activity and maintenance of large size page table. Variable length pages require more programming task. Memory management scheme should keep track of the pages in the memory. Paging scheme may adopt some kind of page replacement algorithm, for e.g. 'Least Recently Used (LRU)' algorithm. In LRU algorithm, a page counter is maintained for each page. When the page is to be replaced, the page with the highest counter value becomes the candidate. Page replacement is done when no free pages are available. Page not modified may be overwritten, instead of replacement. Memory management scheme should be developed such that the user has some control over the size of the pages. This feature helps in determining the appropriate page size while operating on large order matrices in design optimization problems. Fragmentation of large matrices in pages can be avoided by using page size in multiples of matrix size. Matrix operations such as addition, multiplication and transpose by row representation can have page size in terms of number of rows of the matrix. The algorithms that solve large order simultaneous equations by submatrix approach require at least a few submatrices to be present simultaneously in the memory. In such a case, allocation of one submatrix per page induces fewer page-faults. This leads to reduction in I/O activity of iterative algorithms and brings down the execution time.

Query language

Query language allows the user to interrogate and update a database. Even though design optimization algorithms are automated, it is necessary to provide flexibility and control

to the user for modification in actual design process. This control is useful to execute decisions which either cannot be automated or are based on designer's intuition and judgement. Requirements of a query language include data independence, simplicity, nonprocedurality, extendability and completeness. Query commands must be general and not limited to any special problem. The syntax of query should be simple enough to be understood by a non-programming user. Some typical queries are FIND, LIST, SELECT, PLOT, CHANGE, ADD, DELETE, RENAME, OPEN and CLOSE. Query of large order matrices requires special conditional clauses so that data may be displayed in parts. Formatted display of data is essential while dealing with floating point numbers.

In design optimization, a convenient query language is essential. Using the query commands, the applications programmer can formulate his design optimization problem. Cost and constraint functions can be defined by querying the database. Active set of constraints and their values can be obtained using appropriate commands. Use of query language has been recently demonstrated in structural design optimization.¹² This particular query language uses the syntax of structural design problems. Using the query language, stress and displacement constraints are formulated and checked for violation. For general design environment, we need a much more general and flexible query language.

LANGUAGE FEATURES OF MIDAS

One of the important considerations in the development of MIDAS is the language. It is developed to provide simple to use data definition language (DDL) and data manipulation language (DML) for both application programmers and interactive users of the database.¹³ These commands do not contain any reference to the storage structure of the database. Since application programmers use FORTRAN as the host language, the data definition and manipulation facilities are provided through FORTRAN subroutine call statements (commands). These commands interface the application program and MIDAS. An interactive database user can define and manipulate MIDAS database using query language.

MIDAS data definition language for application interface consists of those declarative constructs of FORTRAN needed to declare database objects: variables, arrays, FORTRAN data types, and extensions to FORTRAN to support objects not handled by FORTRAN. An extension to FORTRAN needed to define relations and matrices is provided through CALL statements. Arguments of subroutine call statements, for example, are used to specify database name, relation name, attribute name and size, etc. MIDAS data definition language allows data types — long integer, short integer, real, double precision and character.

Large matrices can be defined using DDL of MIDAS. Matrix types such as square, rectangular upper triangular, lower triangular, and hypermatrices can be defined. A matrix can contain data of integer, real or double precision data types. Matrix data definition requires specification of matrix type, data type, number of rows and columns, number of rows and columns of a submatrix, and storage order such as row, column or submatrix order. Several matrices can be defined in a database which are organized at a number of hierarchical levels.

MIDAS data definition capability has the facility to modify and delete existing data definitions. Database, rela-

tion, attributes and matrices can be renamed. Redefinition of a matrix storage order is possible. For example, a matrix stored in a row order can be redefined to a column order. Also, data types in a matrix can be redefined.

Dynamic data definition facility is one of the important features of MIDAS. This means that DDL has the facility to define relations and matrices during run time. Dynamic data definition is essential to many application programs, since it is not always possible to define relations and their attributes till a certain amount of problem related data is processed. Dynamic data definition of MIDAS also enables addition of new relations, attributes, matrices during the course of execution of design and analysis programs. A typical DDL syntax in Backus-Naur Form (BNF) for defining relation and matrix data is given in Fig. 1. A detailed list is given in ref. 13.

Data manipulation language of MIDAS offers capability to store, retrieve, modify and delete data in a database. Both relation and matrix data can be manipulated. Database name and relation or matrix name are used as a basic set of arguments in data manipulation commands. The DML has facility to store or retrieve data of a relation in two different ways depending on the convenience of the user. One way is to store data of a relation in a sequential order of row numbers and retrieve them using row numbers. A second way is to store data in a relation without any reference to row numbers. New data created at a later stage are added at the end of existing data in a relation. Retrieval of data is possible in the same order in which it was stored. Also, data of selected attributes of a relation can be stored and retrieved. The DML also has the capability to copy data of a relation from one database to another.

MIDAS data manipulation language can be used to manipulate data of large matrices. Matrices can be stored in row, column or submatrix order and can be retrieved in any of these orders. For example, a matrix stored in a row order can be retrieved in a column order. This facility is useful in matrix operations such as transpose and multiplication. The

```

<Relation Definition Statement>:: = <Reserved Procedure RELDEF>
                                   (<Relation Definition Argument Part>)
<Relation Definition Parameter Part>:: = <Database name>,
                                           <Relation name>,
                                           <Attribute names>,
                                           <Attribute types>,
                                           <Attribute row sizes>,
                                           <Attribute column sizes>,
                                           <Attribute keys>,
                                           <Relation definition error code>
<Matrix Definition Statement>:: = <Reserved Procedure MDEF>
                                   (<Matrix Definition Argument Part>)
<Matrix Definition Argument Part>:: = <Database name>,
                                       <Matrix name>,
                                       [<Row dimension of submatrix>]<null>,
                                       [<Column dimension of submatrix>]<null>,
                                       [<ROW>]<null> [<COLUMN>] [<SUBMATRIX>] [<storage order>],
                                       <Row size of matrix>,
                                       <Column size of matrix>,
                                       <Matrix element data types>,
                                       <Matrix definition error code>

```

Figure 1. Typical data definition syntax for relation and matrix definition

```

<Relation Retrieval Statement>:: = <Reserved procedure RDBSET>
                                (<Relation retrieval parameter part>)
<Relation Retrieval Parameter Part>:: = <Database name>,
                                        <Relation name>,
                                        [<Row numbers>]{<null>},
                                        <User buffer>,
                                        <Error code>
<Matrix Retrieval Statement>:: = <Reserved procedure RDBSET>
                                (<Matrix retrieval parameter part>)
<Matrix Retrieval Parameter part>:: = <Database name>,
                                        <Matrix name>,
                                        <Starting row or column or submatrix number>,
                                        <Ending row or column or submatrix number>,
                                        <Data retrieval order>,
                                        <User buffer>,
                                        <Row dimension of user buffer>,
                                        <Column dimension of user buffer>,
                                        <Error code>

```

Figure 2. Typical data manipulation syntax for retrieval of relation and matrix

DML also has commands to copy a matrix from one database to another, to compress a database, to open and close databases. A typical DML syntax in BNF to retrieve data of a relation is given in Fig. 2. A more detailed list is given in ref. 13.

MIDAS uses query language of the RIM program. The query language can be used to interactively define and manipulate relations in a database. The general syntax of the query command is

(command) (expression clause) (conditional clause)

A detailed description of these commands is given in RIM users guide.³

SYSTEM DESIGN

Database management system – MIDAS consists of two subsystems MIDAS/R and MIDAS/N. The subsystems are capable of organizing data of relational and numerical models respectively. MIDAS/R is based on modification and extension of subroutines of RIM.³ The other subsystem is specially developed to organize matrix data. The next few paragraphs summarize the important design features of both MIDAS/R and MIDAS/N subsystems.

MIDAS/R design features

Database structure of MIDAS/R is the same as the RIM database. Each database consists of three files. The first file stores relation and attribute names and their details. The second file contains the actual data. The third file contains pointers to keyed attributes. The program is written in FORTRAN77. It consists of a number of subroutines having well-defined functions. Functions of these subroutines can be generally classified into (i) command processors, (ii) input-output processor, (iii) file definition and initialization routines, (iv) memory management routines, (v) addressing and searching routines, (vi) conditional clause and rule processing routines, and (vii) security routines.

Extensions to RIM program are made to include dynamic data definition facility. The command processing routines were changed to incorporate this facility. Any new definition of a relation or an attribute during execution of a program

is stored in a file for processing. Command processing routines are used to verify syntax of the relation and attribute definition. Later, these data definitions are compiled and stored in the first file of the database.

Modification of the application interface commands are made to simplify the data storage, retrieval and modification of data. The original set of conventional data manipulation commands in RIM were found to be tedious to use in applications such as finite element analysis programs. This was because, even for a simple retrieval of data at least two or three DML calls had to be made. In the MIDAS/R program, such storage and retrieval of data are made generally using one call statement.

MIDAS/R data manipulation commands use relation and database names for data manipulation operations, whereas in RIM, users are required to assign integer numbers to refer to predefined relations of a database that are to be manipulated. The use of database and relation names in data manipulation commands reduces programming errors on the part of users which otherwise may cause reference to a wrong relation. Also, by specifying database and relation names, user is allowed to refer to a relation in another database, directly, without using commands for opening and closing of database files. Arguments of data manipulation commands of MIDAS/R for storage, retrieval and modification have provisions for specification of row number of a relation. The routines for storage and retrieval internally use the row numbers to transfer users data to specified locations in a relation. This provision of row number specification in the argument has facilitated in simplifying application program logic.

MIDAS/N design features

The second subsystem MIDAS/N is specially developed for organizing matrix data. The system is programmed in FORTRAN77. It has data definition and data manipulation subroutines to define large matrix data and to manipulate them. Rectangular, square, upper triangular, lower triangular and hypermatrices can be defined in a database.

MIDAS/N has the capability to create a number of databases. They can be temporary or permanent. An important feature is that these databases can be accessed simultaneously. A database can store data of a number of matrices of different type, such as character, short integer, long integer, real or double precision real. Each data set has several access models such as row order, column order or submatrix order. Any part of a data set can be accessed just by one call statement. Dimensions, order, and data type of a data set can be redefined dynamically. The size of a database and a matrix is unlimited, but depends only on the availability of disk space. Databases of MIDAS/N can be organized at a number of hierarchical levels and can be accessed using a path name as shown in Fig. 3.

In MIDAS/N, the available memory (called the buffer) is divided into pages. Data set is also divided into pages of the same size when it is defined or transferred into the memory buffer. MIDAS/N handles all access requests and transactions against data sets. Each transaction is in terms of pages. Once part of a data set is requested, some pages must be assigned to it. When there is no free page, some page must be freed. Page replacement strategy decides the page to be freed. MIDAS/N chooses the 'least recently used' page for replacement. For handling transactions, information about data sets and page usage must be kept and managed. Information about data sets and page usage must be kept

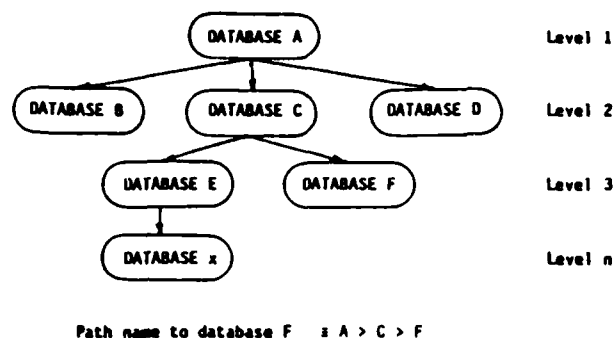


Figure 3. Hierarchical level of database organization

and managed. Information about data sets is stored in the 'data set information' table. Information about page usage is stored in the 'memory buffer information' table.

Some important considerations in designing MIDAS/N are:

- (i) The system should efficiently utilize large as well as small memory buffer.
- (ii) It should efficiently handle large and small data sets.
- (iii) The system should give flexibility to the user to store the data in one form and retrieve it in another form.
- (iv) Various commonly used data models in scientific applications should be available.
- (v) Duplication in overhead information should be avoided.
- (vi) The system should be efficiently used in personal computer as well as its larger host.
- (vii) The system should be application independent.

The system is organized into several distinct segments as shown in Fig. 4. Each segment can be modified or extended without effecting others. Each block is briefly explained in the following.

(i) *Application Program Interface (API)*. This represents the interface between application program and the database management system. This part is designed so that it does not change in the future, as any change would require changes in the application programs. It can be, however, extended to add more capabilities.

(ii) *Application Program Interface Processing System (APIPS)*. This is actually the processing part of the application program interface. Its job is to process caller's requests for validity and correctness. Once this is done, it transfers the request to a sequence of calls to the memory management system.

(iii) *Memory Management Interface (MMI)*. This is the connection between application program interface processing system and memory management system. Since any modification in this part will cause modification in APIPS and Memory Management System, it should be designed with care so that it does not need frequent modifications.

(iv) *Memory Management System (MMS)*. The job of the Memory Management System is to transfer data between the user buffer and memory buffer, calculate required logical pages, and call I/O system to read-in and update logical pages or information records.

(v) *Input/Output System (I/O System)*. This part contains a few subroutines to perform disk I/O.

(iv) *Physical data*. Information tables and data are on the physical disk.

Three information tables used by APIPS and MMS are:

(A) *APIPS information table (database information table)*. This contains information about currently opened databases. The information of each database includes its name, status, access type, logical unit number, modification index, and the names and addresses of its data sets. This table is used by APIPS only.

(B) *Common information table (data set information table)*. This contains data set information of several recently used data sets. Data set information includes data set dimensions, submatrix dimensions, data type, data model, storage order, starting address, total number of words, and left pointer of a double link. This table is shared by APIPS and MMS.

(C) *MMS information table (Memory Buffer information table)*. This contains information about each page in memory buffer and usage status of the pages. Page information includes logical page number and modification index. Usage status includes free pages, the least recently used page, and each data set connected pages. It is used by MMS only.

MIDAS/N is also in the form of a library of subroutines. Application programs can call these subroutines to perform the required functions. When the application program loads it, only the referenced subroutines are loaded. MIDAS/N has a number of user interface routines. More functions will be added and the library will grow in the future. Design of the system is such that the application program loads only a small part of MIDAS/N. This part includes a few

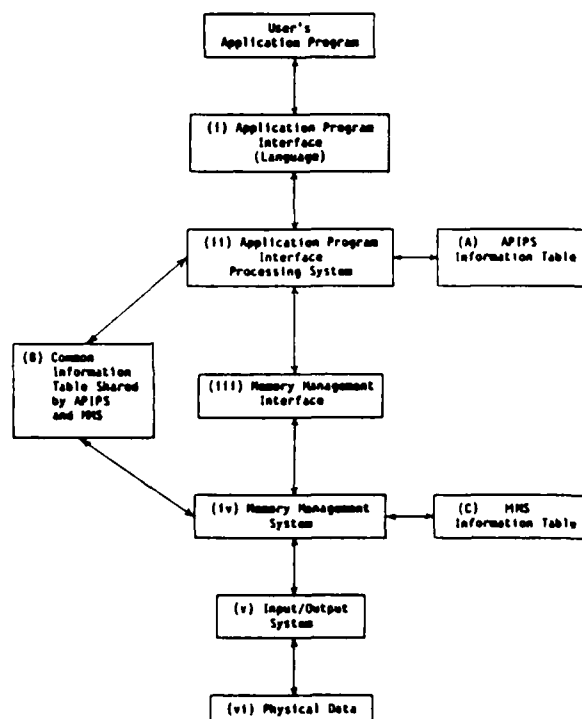


Figure 4. System Organization

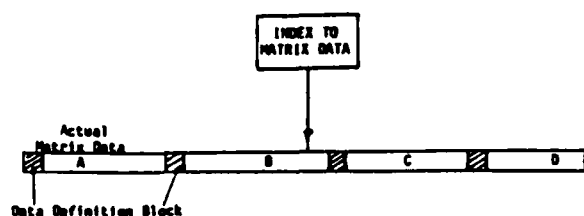


Figure 5. Physical storage structure

called AIPS routines and all the routines of MMS and I/O subsystems.

In MIDAS/N, dimensions of the memory buffer and overhead information records are defined by parameters. We call these system parameters. Changing the system parameters does not influence the organization of the database. This means that a database created by one application program can be used in others without matching their system parameters. With this property, each module of a modularized system can load MIDAS/N with the system parameters which are chosen for its efficiency. For example, the preprocessor of finite element programs can choose smaller page size with more number of pages, since this module has a lot of small data sets. For the equation solving module, it is better to choose larger page size and less number of pages because the module uses a fewer large data sets. This property is also important for a DBMS to communicate data between a personal computer (or a minicomputer) and its larger host. The reason is the available memory buffer for DBMS in a personal computer is much smaller than that in the host.

IMPLEMENTATION OF MIDAS

In the following paragraphs data definition and data manipulation routines (commands) of both MIDAS/R and MIDAS/N are given to show their capabilities. Also, matrix operation routines developed for use in analysis and design of system are given.

Data definition routines of MIDAS/R

Database initialization: CALL RDBINT

This routine initializes MIDAS/R and must be used before using any other routines of the system.

Database definition: CALL RDBDFN (NAME, STAT, IERR)

- NAME = Name of the database
- STAT = Permanent or temporary status of the database
- IERR = Error Code

A unique database can be defined using this routine. A temporary database is deleted when it is closed.

Relation Definition CALL RELDFN (NAME, RNAME, NCOL, CNAME, CTYPE, IELM, JELM, KEY, IERR)

- RNAME = Relation name
- NCOL = Number of attribute columns
- CNAME = A vector of attribute names
- CTYPE = A vector of attribute type

- IELM = A vector of row size of attributes
- JELM = A vector of column size of attribute
- KEY = A vector of key attribute indicator

A relation can be defined using this routine. A row and column intersection in a relation table can contain either a single data item, a vector or a matrix. Details of data type and layout of data in a typical relation are given in Figs. 6 and 7 respectively.

Data Set Definition: CALL RDSDFN (NAME, DSNAME, DTYPE, IELM, JELM, IERR)

- DSNAME = Name of the data set
- DTYPE = Data type (see Fig. 6)
- IELM = Row size of attribute (see Fig. 6)
- JELM = Column size of an attribute (see Fig. 6)

A data set is defined as a collection of data belonging to same data type such as single data item, vector, or matrix. In some sense, data set is a relation having only one attribute. Name of data set has to be unique in a database.

Description	TYPE	IELM	JELM
Integer	INT	1	1
Real	REAL	1	1
Double Precision	DOUBLE	1	1
Integer Vector	IVEC	n	1
Real Vector	RVEC	n	1
Double Precision Vector	DVEC	n	1
Integer Matrix	IMAT	n	n
Real Matrix	RMAT	n	n
Double Precision Matrix	DMAT	n	n
Text or Character	TEXT	n	1

NOTE: Values of IELM and JELM if 0 indicate that data is of variable length.

Figure 6. Data type and size of a relation

Attribute A Key	Attribute B Type INT	Attribute C IVEC	Attribute D IMAT
1	x	x x x x	x x x x
			x x x x
			x x x x
2	x	x x x x	x x x x
			x x x x
			x x x x
			x x x x
			x x x x
			x x x x
			x x x x
			x x x x

NOTE: For Attribute A IELM = 1, JELM = 1
 Attribute B IELM = 1, JELM = 4
 Attribute C IELM = 4, JELM = 4

Figure 7. Layout of data in a typical relation

Data Set Redefinition: CALL RDRDFN (NAME, DSNAME, DTYPE, IELM, JELM, IERR)

This routine redefines a data set using new data type, and new attribute size. Old data set definition and its data is lost.

Data definition ending: CALL RDSSEND (IERR)

After database, relations and data sets have been defined, data definition process is ended by calling this routine. During execution of this call statement, the data definition is verified and compiled internally.

Data manipulation routines of MIDAS/R

Data manipulation routines open, close, store, retrieve, modify, and delete data, rename a relation or a data set, rename an attribute and copy data sets in a database.

Open a database: CALL RDBOPN (NAME, STAT, IERR)

A database closed earlier can be opened using this routine. It must be opened before any operation is performed.

Close a database: CALL RDBEND (IERR)

A database is closed using this routine. It transfers the memory buffer data into the database and closes the files.

Store data in a relation: CALL RDSPUT (NAME, DSNAME, KROW, UBUF, IERR)

KROW = Row number
UBUF = User buffer containing data

Data can be stored into a relation from application program work area (user buffer) with this routine. Data is transferred from user buffer to the specified row of a relation. If more rows have to be stored, a FORTRAN DO loop over the row number in the application program will transfer all the required rows. More details of the routine are given in the user's manual.¹⁴

Retrieve data from a relation: CALL RDSGET (NAME, DSNAME, KROW, UBUF, IERR)

Data can be retrieved from a relation into a user buffer using this routine. Requested row is transferred from the relation to user buffer. FORTRAN DO loop over the row number is necessary if more than one row has to be retrieved. Data can be retrieved in the same order as it was stored by initializing row number as zero. Data of a relation satisfying certain condition (for example, attributes having certain values) can be retrieved into user buffer. User specifies the condition on data values that must be satisfied for retrieval by using RDSRUL routine (explained later). The details for various ways of data retrieval is given in the user's manual.¹⁴

Modify data in a relation: CALL RDSMOD (NAME, DSNAME, DROW, UBUF, IERR)

Once a database is loaded using RDSPUT, it can be modified by calling RDSMOD. This routine modifies a row of the relation. RDSGET routine must be called before calling this subroutine. The row of the relation is retrieved into user buffer. The row or its part is modified and stored back.

Delete rows of a relation: CALL RRWDEL (NAME, DSNAME, KROW, IERR)

Rows of a relation can be deleted using this routine. This is useful in elimination unwanted values in a relation.

Delete a relation: CALL RDSDEL (NAME, DSNAME, IERR)

Rename a relation: CALL RDRNAM (NAME, OLDNAM, NEWNAM, IERR)

OLDNAM = Old name of the relation
NEWNAM = New name of the relation

Rename an attribute: CALL RRNATT (NAME, DSNAME, OLDATT, NEWATT, IERR)

OLDATT = Old attribute name
NEWATT = New attribute name

Copy a relation: CALL RDSCPY (NAME1, NAME2, DSNAM1, DSNAM2, IERR)

NAME1 = Name of the database containing data
NAME2 = Name of the database where data has to be copied
DSNAM1 = Relation name containing data
DSNAM2 = Relation name where data has to be copied

Using this routine, data from one relation can be copied to another relation. Both the database and relation must have been defined before copying the data.

Condition specification for retrieval of data: CALL RDSRUL (NUM, ATNAM, COND, VALUE, BOOL, IERR)

NUM = Number of conditions
ATNAM = A vector of attribute names
COND = A vector of logical operator (EQ, GT, LT)
VALUE = A vector of attribute values
BOOL = A vector of Boolean operator (AND, OR)
IERR = Error code

As mentioned for RDSGET routine, data values satisfying certain conditions can be retrieved. The conditions are specified using RDSRUL. This routine must be executed before calling RDSGET. A maximum of ten conditions can be specified at a time. The following example illustrates use of the routine. Condition on a relaxation X:

Attribute A · GT · 15.3 · AND · Attribute B · LT · 20.1
Use NUM = 2; ATNAM(1) = 'A'; ATNAM(2) = 'B'
COND(1) = 'GT'; COND(2) = 'LT';
VALUE(1) = 15.3; VALUE(2) = 20.1;
BOOL(1) = 'AND'

Interactive commands

MIDAS/R provides interactive support for creating, updating, modifying, and deleting a database. Interactive commands are general and can be used in any application. The system provides terminal prompts for the users to respond with appropriate commands. The interactive session starts with a display of MENU and requests the user to choose one of the five options: CREATE, UPDATE, QUERY, COMMAND and EXIT. The interactive session ends with an EXIT command. The detailed commands for each of these options are entered at appropriate instant. For details of interactive commands, refer to RIM user's manual.³

Data definition subroutines of MIDAS/N

Data definition subroutines of MIDAS/N can be used to define databases and matrices. These are FORTRAN call statements and can be directly interfaced with an application program.

Database definition: CALL NDBDFN (NAME, PTHNAM, TYPE, STAT, IERR)

PTHNAM = Path name in database hierarchy
TYPE = Random or sequential access file type

This routine can be used to define a database. Path name specifies the hierarchy of databases that are stored

in a computer system file directly organized at various levels.

Renaming a database: CALL NDBRNM (OLDBN, NEWBN, PTHNAM, IERR)

OLDBN = Old database name
NEWBN = New database name

Matrix definition: CALL NDSDFN (NAME, DSNAME, ISUB, JSUB, ORDER, NROW, NCOL, DTYPE, IERR)

DSNAME = Data set (Matrix) name
ISUB = Row dimension of a submatrix if present
JSUB = Column dimension of a submatrix if present
ORDER = Order of data storage (explained below)
NROW = Row size of the matrix
NCOL = Column size of the matrix
DTYPE = Data type of data elements in matrix

A matrix (data set) can be defined by calling this routine. Order of the matrix refers to the data storage order which can be row-wise, column-wise, or submatrix-wise. In case of a triangular matrix, order is either row-wise or column-wise.

If submatrices are used, then size of a submatrix should be given.

Matrix redefinition: CALL NDSRDF (NAME, DSNAME, ISUB, JSUB, ORDER, NROW, NCOL, DTYPE, IERR)

This routine redefines a matrix in a different storage order. A matrix which is in either row, column or submatrix order can be redefined to any other order (row, column, submatrix). An upper triangular matrix can be redefined as either row or column order. Similarly a lower triangular matrix can be redefined as either row or column order. Data types can be redefined as integer, real and double precision (except characters).

Matrix renaming: CALL NDSRNM (NAME, OLDNAM, NEWNAM, IERR)

OLDNAM = Old name of a matrix
NEWNAM = New name of a matrix

Data manipulation routines of MIDAS/N

Data manipulation routines of MIDAS/N can be used to open, close, delete and compress a database, store, retrieve, delete and copy a matrix.

Open a database: CALL NDBOPN (NAME, PTHNAM, IERR)

Close a database: CALL NDBEND (NAME, IERR)

Any modifications to data in the memory buffer are transferred to the database before closing it.

Delete a database: CALL NDBDEL (NAME, PTHNAM, IERR)

Compress a database: CALL NDBCMP (NAME, IERR)

Compresses a database. Empty spaces created due to deletion or redefinition of matrices are removed by moving data in a database. This command helps in efficient utilization of disk space.

Store a matrix: CALL NDSPUT (NAME, DSNAME, NSTR, NEND, ISTR, ORDER, UBUF, IROW, ICOL, IERR)

NSTR = Starting row, column or submatrix number for storing data
NEND = Ending row, column or submatrix number for storing data
ISTR = Starting element number of each row or column

UBUF = User buffer (array name)
IROW = Row dimension of the user buffer
ICOL = Column dimension of the user buffer

This routine stores a matrix data from user buffer into a database. Full or part of a matrix can be stored and its size specified using NSTR and NEND. Row or column storage order can be used for a matrix whose order is defined as row-wise, column-wise or submatrix-wise in data definition. Submatrix storage order can only be used for a matrix defined as submatrix.

Retrieve a matrix: CALL NDSGET (NAME, DSNAME, NSTR, NEND, ISTR, ORDER, UBUF, IROW, ICOL, IERR)

A matrix can be retrieved into a user buffer from a database using this routine. Full or part of a matrix can be retrieved.

Retrieve a matrix in row order: CALL NDGETR (NAME, DSNAME, NSTR, NEND, ISTR, UBUFF, IROW, ICOL, IERR)

Retrieve a matrix in column order: CALL NDGETC (NAME, DSNAME, NSTR, NEND, ISTR, UBUFF, IROW, ICOL, IERR)

Retrieve a matrix in submatrix order: CALL NDGETM (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Matrix must have been defined in submatrix order during data definition.

Store a matrix in row order: CALL NDPUTR (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Store a matrix in column order: CALL NDPUTC (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Store a matrix in submatrix order: CALL NDPUTM (NAME, DSNAME, NSTR, NEND, ISTR, UBUF, IROW, ICOL, IERR)

Matrix should have been defined in submatrix order.

Copy a matrix: CALL NDSCOPY (NAME1, DSNAME, NAME2, IERR)

NAME1 = Name of the database containing matrix data
NAME2 = Name of the database into which matrix has to be copied

This routine copies a matrix from one database to another database.

Delete a matrix: CALL NDSDEL (NAME, DSNAME, IERR)

Matrix Operations Utilities

MIDAS/N has several routines to carry out operations on matrices stored in the database. These include matrix addition, scaling and multiplication routines. Algorithms for these utilities are developed to utilize the storage order of the data sets; i.e., if a matrix is stored in the row order in the database, an algorithm is developed to use it in that order. This is done to minimize the disk I/O and thus perform the operations efficiently. The current routines in the system are listed in the following. More routines will be added as need arises.

Multiply general matrices: CALL NMTPYx (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, IERR)

NMTPY1: Computes $AB = C$
 NMTPY2: Computes $AB^T = C$
 NMTPY3: Computes $A^T B = C$
 NMTPY4: Computes $A^T B^T = C$

Add matrices: CALL NMADDx (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, IERR)

NMADD1: Computes $A + B = C$
 NMADD2: Computes $A + B^T = C$
 NMADD3: Computes $A^T + B = C$
 NMADD4: Computes $A^T + B^T = C$

Subtract matrices: CALL NMSUBx (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, IERR)

NMSUB1: Computes $A - B = C$
 NMSUB2: Computes $A - B^T = C$
 NMSUB3: Computes $A^T - B = C$
 NMSUB4: Computes $A^T - B^T = C$

Scale of matrix: CALL NMSCLx (NAME1, DSNAME1, NAME2, DSNAME2, SCALE, IERR)

NMSCL1: Computes $A * SCALE = C$
 NMSCL2: Computes $A^T * SCALE = C$

Transpose of a matrix: CALL NMTRPZ (NAME1, DSNAME1, NAME2, DSNAME2, IERR)

Computes $A^T = C$

Multiply a matrix by a diagonal matrix: CALL NMTDGx (NAME1, DSNAME1, NAME2, DSNAME2, ARRAY, IERR)

NMTDG1: Computes $ARRAY * A = C$
 NMTDG2: Computes $ARRAY * A^T$
 NMTDG3: Computes $A * ARRAY = C$
 NMTDG4: Computes $A^T * ARRAY = C$

Rearrange rows/columns of a matrix: CALL NMSRTx (NAME1, DSNAME1, NAME2, DSNAME2, ARRAY, IERR)

NMSRT1: Rearranges rows according to the order specified in ARRAY
 NMSRT2: Rearranges columns according to the order specified in ARRAY

Equation solvers and matrix decomposition routines

MIDAS/N has several routines to decompose and solve a linear system of equations. The coefficient matrix may be stored in skyline or banded form. It may also be a full matrix. In the following, these routines are listed. Other equation solvers and eigenvalue extractors will be added at the later date.

Decompose a Symmetric Matrix by Skyline Method.
 CALL NMSKY1 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, MAXCOL, IER)

Perform Backward and Forward Substitutions to Solve a Decomposed System of Linear Equations by Skyline Method.

CALL NMSKY2 (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, NEQ, MAXCOL, IER)

Solve a System of Linear Equations by Skyline Method.
 CALL NMSKY3 (NAME1, DSNAME1, NAME2, DSNAME2, NAME3, DSNAME3, NEQ, MAXCOL, IER)

Decompose a Symmetric Banded Matrix by Cholesky's Method.

CALL NMBND1 (NAME1, DSNAME1, NEQ, MBND, IER)

Performs Backward and Forward Substitutions to Solve Decomposed System of Linear Banded Equations.
 CALL NMBND2 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, MBND, IER)

Solve System of Linear Banded Equations by Cholesky's Method.

CALL NMBAND3 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, MBND, IER)

Decompose a General Full Matrix.

CALL NMGLS1 (NAME1, DSNAME1, NEQ, IER)

Perform Backward and Forward Substitutions to Solve a Decomposed General System of Equations.

CALL NMGLS2 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

Solve System of Linear Equations.

CALL NMGLS3 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

Decompose a Full Symmetric Matrix by Modified Cholesky's Method.

CALL NMSYM1 (NAME1, DSNAME1, NEQ, IER)

Perform Backward and Forward Substitution to Solve a Decomposed Symmetric System of Linear Equations.

CALL NMSYM2 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

Solve a Full Symmetric System of Linear Equations by the Modified Cholesky's Method.

CALL NMSYM3 (NAME1, DSNAME1, NAME2, DSNAME2, NEQ, IER)

DISCUSSION AND CONCLUSIONS

A database management system MIDAS has been implemented. The system supports relational and numerical data models which are essential for organizing design and analysis data. User-friendly data definition and data manipulation commands help in improving efficiency of application programmers and designers. Dynamic data definition enables definition of relations and matrices during runtime. Multiple database capability is useful for design of systems. MIDAS appears to satisfy the requirements of a database management system for design and analysis of systems. Design of a database for structural analysis and optimization is in progress. The database will be implemented with MIDAS so that it can be evaluated along with the DBMS.

ACKNOWLEDGEMENT

This research is sponsored by the Air Force Office of Scientific Research, Grant No. AFOSR 82-0322. Material of the paper is derived from a presentation made by the authors at the 26th AIAA, Structures, Structural Dynamics and Materials Conference.

REFERENCES

- 1 Lopez, L. A. FILES: Automated engineering data management system, *Computers in Civil Engineering, Electronic Computation*, 1974, 47

- 2 Kamel, H. A., McCabe, M. W. and Spector, W. W. *GIFTSS System Manual*, University of Arizona, Tucson, 1979
- 3 *RIM User's Guide*, Boeing Commercial Airplane Company, PO Box 3707, Seattle, Washington, 98124, 1982
- 3 Massena, W. A. SDMS - A Scientific Data Management System, *NASA Conference Publication 2055*, 1978
- 5 Giles, G. L. and Haftka, R. T. SPAR data handling utilities, *NASA Technical Memorandum 78701*
- 6 Fischer, W. E. PHIDAS - A database management system for CAD/CAM software, *Computer-Aided Design*, 1979, 11 (3), 146
- 7 Ulfaby, S. Steiner, S. and Olan, J. TORNADO: A DBMS for CAD/CAM systems, *Computer-Aided Design*, 1979, 193
- 8 Sreekanta Murthy, T. and Arora, J. S. A survey of database management in engineering, *Journal of Advances in Engineering Software*, 1985, 7 (3) 126
- 9 Felippa, C. A. Database management in scientific computing - I, General description, *Computers and Structures*, 1979, 10, 53
- 10 Felippa, C. A. Database management in scientific computing - II, Data structures and program architecture, *Computers and Structures*, 1980, 12, 131
- 11 Sreekanta Murthy, T., Reddy, C. P. and Arora, J. S. Database management concepts in engineering design optimization, *Proceedings of AIAA of the 25th SDM Conference, Palm Springs, CA*, 1984
- 12 Rajan, S. D. SADDLE: A computer-aided structural analysis and dynamic design language, *PhD Dissertation*, The University of Iowa, 1983
- 13 Sreekanta Murthy, T. and Arora, J. S. Database design methodology and database management system for computer-aided structural design optimization, *Technical Report CAD-SS-84-22*, The University of Iowa, 1984
- 14 Sreekanta Murthy, T. and Arora, J. S. MIDAS user's manual, *Technical Report*, The University of Iowa, 1984

APPENDIX 4

**DATA BASE DESIGN METHODOLOGY FOR STRUCTURAL
ANALYSIS AND DESIGN OPTIMIZATION**

by

T. SreekantaMurthy and J.S. Arora

in

Engineering with Computers

1, 1986

Data Base Design Methodology for Structural Analysis and Design Optimization

T. Sreekanta Murthy and J.S. Arora

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242

Abstract. A methodology to design data bases for finite element analysis and structural design optimization is described. The methodology considers three views of data organization—conceptual, internal, and external. Tabular and matrix forms of data are included. The relational data model is used in the data base design. Entity, relation, and attributes are considered to form a conceptual view of data. First, second, and third normal forms of data are suggested to design an internal model. Several aspects such as processing, iterative needs, multiple views of data, efficiency of storage and access time, and transitive data are considered in the methodology.

1 Introduction

Data base design topic has recently caught attention among many engineering data base management enthusiasts. Designing a good data base is extremely important for successful implementation of finite element analysis and structural design optimization methods in a computer-based environment. Complexity of analysis and design data imposes severe constraint to design a good data base. Intuitive methods are still being used as no systematic approach is available to design data bases of large and complex structural design software. In view of the computer-aided design of structural systems, a good data base design methodology is very much needed.

Several reasons exist for emphasizing a systematic methodology to design a data base. First, the iterative nature of structural optimization process together with a large amount of computation make data organization a complex task. Second, the well-known finite element programs like NASTRAN, ANSYS, ADINA, and GIFTS have no centralized data base organization. Thus it becomes extremely difficult and cumbersome for general users to ex-

tract intermediate data generated by the programs for structural optimization use. Out-of-core data organization in the programs were based on intuition, since few scientific data base design techniques for large engineering programs were available at the time the programs were developed. Third, the designer needs control over the program and data to obtain optimum design of structures. Finally, a good data base will enable addition of new optimization and other programs without extensive modification of the data base or programs.

In this paper a methodology to design data bases for finite element analysis and structural design optimization applications is presented. The methodology aims to replace the intuitive design of data bases by a systematic method. Also, the methodology provides a basis for integrating several application programs through a common data base.

2 Background

Advocates of engineering data base management software have realized that their system will not be of much use to organize data without a well-designed data base. This situation has drawn the attention of some investigators to find possible ways of designing a data base. The paper by Koriba [1] describes several approaches followed in business applications and their suitability to CAD applications. Most commonly known approaches are ANSI/SPARC, CODASYL, relational, hierarchical, and network. Among them, the ANSI/SPARC approach, which recognizes three levels of data views (conceptual, internal, and external), provides a generalized framework and basis for a good data base design.

Hierarchical approach has been tried by Lopez [2] and Pahl [3] for finite element analysis application. Data base design using this model, however, is tedious, since much time is spent working out con-

Reprint requests: J.S. Arora, Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242.

nections between various data items. Penalty for faulty data base design is high as some data may become totally inaccessible due to poor design. Moreover, any modification of a hierarchical data base is difficult. If new data items need to be added, new links must be established. This sometimes requires restructuring of the entire data base.

The relational data model has several advantages [4]. It has simple tabular structure that is easy to use. Data base design is simple as no links or pointer are used in relational model. Data base can be easily modified using a simple set of data manipulation commands. The model is particularly suitable for engineering applications since a relation can be regarded as a two-dimensional array. Various columns can be referred to by names or column numbers and rows by their numbers. These types of data structures are quite natural in engineering applications. Fishwick and Blackburn [5] used a relational data model with the finite element program SPAR and optimization package PROSSS. But the use of the model was limited to interfacing these programs by using a relational data base management system.

Data base design follows some well-defined steps. The basic problem is that once all the data items have been identified, how should they be combined to form useful relations. The first step is the extraction of all the characteristics of the information that is to be represented in the data base. Analysis of the information and their integration into one conceptual model is the second step. The conceptual data model obtained by this process is abstract. It is independent of any computer restraint or data base management software support. In order for the conceptual model to be useful, it must be expressed in terms that are compatible with a particular DBMS by considering efficiency of storage space and access time. An internal model is developed for this purpose which is compatible with the conceptual data model. Finally, the data base design requires accommodation of different users of the data base by providing an external data model. The systematic process by which one traverses the different steps of data base design and performs the mapping from one level of abstraction to the next is called a data base design methodology.

Several methodologies are used in design of data bases for business applications. Suitability of some of the methodologies to computer-aided design applications has been investigated by Buchman and Dale [6]. The investigators analyzed three existing methodologies—Bubenko's methodology,

Kahn's methodology, and Smith and Smith's methodology, with reference to applications in engineering design of a chemical plant. A list of criteria for evaluating the methodologies is given. Salient characteristics of these methodologies are outlined here. In Bubenko's methodology, entities are identified and classified from query and transaction descriptions. A strong point of this method is the provision for two levels of abstractions. Grouping of entities, however, is highly intuitive and application dependent. Kahn's approach has characteristics of separating the data base design problem into two perspectives: information structure perspective, which describes the interconnection of information, and the usage perspective, which deals with the processing requirement of information. The method requires design of two schema—one processing and the other information oriented—which are merged at the end of data base design to get a conceptual model. This methodology merges local views into global view by aggregating entities. Designers' intuition is required in this methodology to form nonredundant entities and relations. The Smiths' methodology ignores information analysis and considers only abstract objects of interest. An object can be viewed as an entity, attribute, or relation depending solely on the viewpoint of the user. The abstraction step used in this method is highly intuitive. Grabowski and Eigner [7] pointed out the necessity for a semantic model construction in the CAD application. They described three available semantic models, using the example of a geometric model of a line: (1) based on binary association, (2) based on entity and attribute association, and (3) expanded relational model.

The methodologies described earlier are at the research state in the business data management field and are not suitable for engineering applications. Also, many of them do not discuss details of the procedure or implementation aspect and hence cannot be directly used for designing an engineering data base. Therefore it is necessary to adopt good features and guidelines provided by existing methodologies and arrive at a suitable one to design data bases for finite element analysis and structural design optimization applications.

The proposed methodology to design data bases considers several features and requirements of finite element analysis and structural design optimization applications. Some important features of data considered in the methodology are—tabular structure, matrices, static information, operational information, multiple views of data for different ap-

plication, and iterative changes in data. Tabular structure of data can be conveniently organized using a relational data model, whereas large matrix data needs a different approach [4]. A simpler approach to design a data base is by considering static and operational information separately for an initial design and later merging them to arrive at a final design. Multiple views of data are necessary to accommodate theoretical, implementational, and user's requirements. Conceptual, internal, and external views of data suggested by ANSI/SPARC is considered to accommodate multiple views of data. The methodology uses entity set, relationship set, and attributes to form syntactic basic element of the conceptual model.

In summary, the methodology given in the paper considers the following aspects: (1) three views of data—conceptual, internal, and external as suggested by ANSI/SPARC; (2) entity set, relationship set, and attributes to form syntactic basic elements of the conceptual model; (3) relational data model; (4) matrix data; (5) processing requirements; and (6) normalization of data for relational model.

3 Methodology to Develop a Conceptual Data Model

Analysis of the data used in finite element analysis and structural design optimization is necessary to develop a conceptual data model. In the analysis the information in use or needed later is identified, classified, and documented. This forms the basis for a conceptual data model to represent structural design data and design process as a whole. In the following sections familiarity with terminologies of relational model [4, 8, 9] is assumed.

The following steps are proposed to develop a conceptual data model:

1. Identify all the conceptual data objects of structural analysis and design optimization.
2. Data identified is stored in a number of relations. The data reduced to elementary relations representing inherent association of data.
3. More elementary relations are derived from the ones formed in Step 2. This step uncovers more relationships between basic data collected in Step 2.
4. Redundant and meaningless relations obtained in Step 3 are removed to obtain a conceptual data model.

The conceptual model obtained by this process

is abstract, representing the inherent nature of structural design data and is independent of any computer restraint or data base management software support. These steps are now discussed in detail.

3.1 Identification of Conceptual Data Objects

The following steps are proposed to identify the conceptual data objects used in structural design. Entity sets and attributes are considered to be the syntactic basic elements of the model. Domain definition is extended to include vectors and matrices. An attribute value can be the relation name or null.

Step 1. Identify each type of entity and assign a unique name to it.

Step 2. Determine the domains and assign unique names to them. This step identifies the information that will appear in the model, such as attributes.

Step 3. Identify the primary key for each type of entity depending on the meaning and use.

Step 4. Replace each entity set by its primary key domains. Determine and name relations corresponding to the association between the primary key domain and other domains. This step gives a collection of relations forming a rough conceptual data model.

Example

We consider a sample structural design problem to describe these steps.

Step 1. The following entity sets can be identified for the structure:

STRUCTURE (S), BEAM (B), TRUSS (T),
MEMBRANE-TRI (TRM), MEMBRANE-QD
(QD), NODE (N), ELEMENT (E)

Step 2. We can identify the following domains:

STRUCTURE#	Structure identification number (integer)
B#	Beam element identification number (integer)
T#	Truss element identification number (integer)
TR#	Triangular membrane element identification number (integer)
QD#	Quadrilateral membrane element identification number (integer)
NODE#	Node number (integer)
E#	Element number (integer)
EL-TYPE	Element type (BEM2, BEM3, TRS2, TRS3, TRM2, TRM3, QDM2, QDM3)

MATID	Material identification code, for example, {STEEL-1, STEEL-2, ALUM-5, COMP-1}. It also refers to a relation or table of material properties; for example, STEEL-1 refers to relation STEEL and material subtype 1
MATPRO	Material property {E, μ , G, . . . }
CSID	Cross-section-type identification code; for example, {THICK-1, THICK-2, RECT-1, CIRC-5, ISEC-6, LSEC-15}. It also refers to a relation of cross-sectional details. For example, RECT-1, refers to a relation RECT and a cross-section subtype 1
CSPRO	Cross-sectional property {H, W, T, R, . . . }
DOF#	Degrees of freedom numbers
LOAD-TYP	Load-type {CONCENTRATED, DISTRIBUTED, TEMPERATURE, ACCELERATION}
X	X coordinate (real)
Y	Y coordinate (real)
Z	Z coordinate (real)
DESCRIPTION	Description (characters)
VEC	Vectors {integer, real, and double precision vectors}
MATX	Matrices {integer, real, and double precision matrices}
VECID	Vector identification code = {x·y x = vector description, y = number}; for example, FORCE-5, LOAD-10
MAXID	Matrix identification code = {x·y x = matrix description, y = number}; for example, EL-STIFF-10, EL-MASS-5

Step 3. The following entity keys are identified

STRUCTURE#	for entity set structure
B#	for entity set beam
T#	for entity set truss
TR#	for entity set TRM
QD#	for entity set QD
E#	for entity set element

Step 4. In the association between entity sets and domain the entity sets from step 1 are replaced by their primary keys. Attribute names are derived from domain names to provide role identification. The following relations are identified:

For entity set TRM

TRM (TR#, E#, EL-TYP, MATID, E, NODE1#, NODE2#, NODE3#, CSID, T, LOAD-TYP, LOAD#, VECID, VEC, MAXID, MATX)

A triangular membrane element is identified by TR#. Element number E# uniquely identifies the finite elements of a structure. Attributes NODE1#, NODE2#, and NODE3# are derived from domain NODES. Similarly, E is the role name for domain MATPRO. CSID identifies the cross-section property T. Vectors and matrices associated with the element are identified through VECID and MAXID, respectively. Similarly, the relations TRUSS, BEAM, QDM are obtained.

3.2 Reduction to Elementary Relations

In the previous section we described a method to identify entities, domains, and relations to produce a rough conceptual model of the structure. Our idea is to develop a conceptual model that contains all the facts and each fact occurring only once. In order to produce a conceptual data model, we transform the rough model into a better model by using a set of elementary relations [8]. Using the concept of functional dependencies, full functional dependencies, and transitive dependencies [4, 9] we can establish rules for reducing a relation to an elementary relation. The following steps are identified to form elementary relations:

Step 1. Replace the original relations by other new relations to eliminate any (nonfull) functional dependencies on candidate keys.

Step 2. Replace the relations obtained in step 1 by other relations to eliminate any transitive dependencies on candidate keys.

Step 3. Go to step 5 if the:

- a. relation obtained is all key
- b. relation contains a single attribute that is fully functionally dependent on a single candidate key.

Step 4. Determine the primary key for each relation that may be a single attribute or a composite attribute. Take projections of these relations such that each projection contains one primary key and one nonprimary key.

Step 5. Stop when all elementary relations are obtained.

Example

To see how these steps are used, we consider the relation TRM that was given earlier and reduce it to elementary relations:

Table 1. Transitive closure for elementary relations

Derived relations	Dependencies	Composition	Semantically meaningful
ER15	$E\# \rightarrow EL-TYP$	$E\# \rightarrow TR\# \rightarrow EL-TYP$	YES
ER16	$E\# \rightarrow NODE1\#$	$E\# \rightarrow TR\# \rightarrow NODE1\#$	YES
ER17	$E\# \rightarrow NODE2\#$	$E\# \rightarrow TR\# \rightarrow NODE2\#$	YES
ER18	$E\# \rightarrow NODE3\#$	$E\# \rightarrow TR\# \rightarrow NODE3\#$	YES
ER19	$E\# \rightarrow MATID$	$E\# \rightarrow TR\# \rightarrow MATID$	YES
ER20	$E\# \rightarrow CSID$	$E\# \rightarrow TR\# \rightarrow CSID$	YES
ER21	$E\# \rightarrow LOAD-TYP$	$E\# \rightarrow TR\# \rightarrow LOAD-TYP$	YES
ER22	$E\# \rightarrow MAXID$	$E\# \rightarrow TR\# \rightarrow MAXID$	YES
ER23	$TR\# \rightarrow E$	$TR\# \rightarrow MAXID \rightarrow E$	NO
ER24	$TR\# \rightarrow T$	$TR\# \rightarrow CS-TYP \rightarrow T$	NO
ER25	$TR\# \rightarrow VEC$	$TR\# \rightarrow VECID \rightarrow VEC$	NO
ER26	$TR\# \rightarrow MATX$	$TR\# \rightarrow MATXID \rightarrow MATX$	NO

Step 1.

ER1 (TR#, E#) ER2 (TR#, EL-TYP)
 ER3 (TR#, NODE1#) ER4 (TR#, NODE2#)
 ER5 (TR#, NODE3#) ER14 (E#, TR#)

Step 2.

ER6 (TR#, MATID) ER7 (MATID, E)
 ER8 (TR#, CSID) ER9 (CSID, T)
 ER10 (TR#, VECID) ER11 (VECID, VEC)
 ER12 (TR#, MAXID) ER13 (MATXID, MATX)

Step 3. The preceding relations contain a single attribute, so go to step 5.

Step 4. Skip

Step 5. ER1 to ER13 are elementary relations.

The steps can be applied to the rest of the relations identified earlier to get a set of elementary relations for the sample structural problem.

3.3 Determination of Transitive Closure

While deriving a large number of relations for obtaining a conceptual data model it is possible that some relations might have been missed. In general, one can derive further elementary relations from any incomplete collection of such relations. To explain in a simple way how such additional relations can be derived, consider two relations ER1(A,B) and ER2(B,C), which imply functional dependencies: $A \rightarrow B$ and $B \rightarrow C$. Taking the product of these functional dependencies, we get $A \rightarrow C$. Therefore, from suitable pairs of elementary relations representing functional dependencies, further elementary relations can be derived. Deriving all such relations from the initial collection of elementary relations yields a transitively closed collection of elementary relations called transitive closure [8].

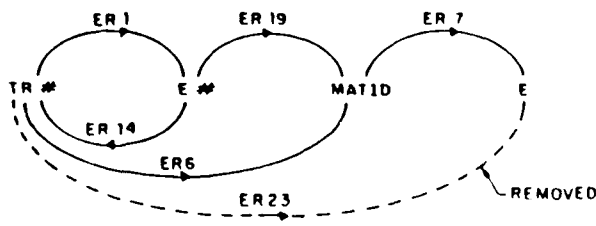
There are problems associated in interpreting relations in transitive closure. For example, consider relations ER1 (TR#, MATID) where $TR\# \rightarrow MATID$, and ER7 (MATID, E) where $MATID \rightarrow E$. Transitive closure for this set yields the relation ER (TR#, E) which implies TR# identifies E. This relation, however, does not represent true information since material property E is dependent only on the material number and not on the element number. The relation could be wrongly interpreted. Therefore such semantically meaningless dependencies must be eliminated. It is possible to determine transitive closure, by using directed graphs and the connectivity matrix [10].

The transitive closure for the example produces additional dependencies as given in Table 1. We have eliminated meaningless dependencies from the list.

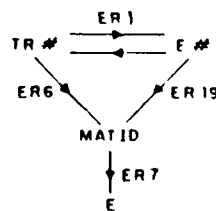
3.4 Determination of Minimal Covers

We need to remove redundant elementary relations to provide a minimal set of elementary relations. A minimal cover is the smallest set of elementary relations from which transitive closure can be derived [8]. The following points are noted: (1) minimal cover is not unique, (2) deriving several alternative minimal covers from a transitive closure guarantees that every possible minimal cover is found, and (3) we can select a minimal cover that best fits the structural design process needs.

An example of finding a set of minimal cover from the transitive closure derived in previous sections is given in Fig. 1. A set of minimal cover for this transitive closure is {ER1, ER14, ER6, ER7} and {ER1, ER14, ER19, ER7}, out of which one set may be chosen to suit our requirement.



(I) Transitive Closure



(II) Rearrangement

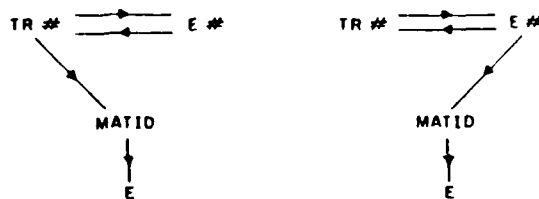


Fig. 1. Digraph representation of minimal cover

The preceding procedure can be applied to other transitive closures derived in the previous section. Thus we can get further sets of minimal covers. Each minimal cover is a nonredundant list of elementary relations and is an appropriate conceptual model of the structural design data.

4 Data Base Design to Support an Internal Model

An internal model deals with the logical organization of data to be stored on physical storage devices. An approach to build an internal model by means of an *n*-ary relations is given here. The *n*-ary relations have to be consistent with the conceptual model and have to follow certain rules. Any storage and update operations (insert, modify, delete) must not lead to inconsistency in data. To avoid anomalies in storage and update operations (insert, modify, delete) must not lead to inconsistency in data.

To avoid anomalies in storage and update operations, we adopt a normalization procedure [9]. In the following discussion we describe a methodology to support an internal model based on normalization procedures.

An example of an element stiffness matrix is considered here to describe the methodology. Consider the conceptual model given by the following elementary relations:

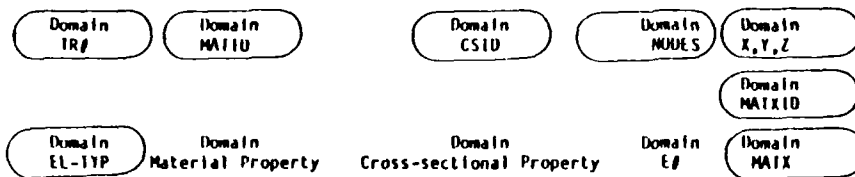
- ER1 (TR#, EL-TYP)
- ER2 (TR#, NODE1#)
- ER3 (TR#, NODE2#)
- ER4 (TR#, NODE3#)
- ER5 (TR#, MATID)
- ER6 (MATID, E)
- ER7 (TR#, CSID)
- ER8 (CSID, T)
- ER9 (NODE#, X)
- ER10 (NODE#, Y)
- ER11 (NODE#, Z)
- ER12 (TR#, MATXID)
- ER13 (MATXID, MATX)
- ER14 (E#, NODE#)
- ER15 (TR#, E#)

Data needed for the generation of element stiffness matrix is derived from various domains and represented in a single relation TRM-D as shown in Fig. 2. Our main intention is to get all the data required for generation of stiffness matrices for triangular membrane elements in one access or a minimum number of accesses. It is observed that the relation in Fig. 2 is not in the first normal form. Therefore this unnormalized relation should be replaced by a semantically equivalent relation in INF as shown in Fig. 3. The advantage of INF over the unnormalized relation is that operations required for application programs are less complicated and easy to understand.

To check consistency of this model, first we identify the key attributes. Candidate keys are compound, consisting of (E#, NODE#) and (TR#, NODE#). The primary key is selected as (TR#, NODE#). Secondary keys are TR#, E#, MATID, CSID, NODE#, and MATXID. These key attributes of the relation are consistent with those in the elementary relation. Second, we need to identify whether all the attributes in the internal model and dependencies between them are consistent with the conceptual model. It can be observed that attributes NODE1#, NODE2#, and NODE3# do not appear in the relation. Therefore these three attributes should be included in the relation. The relation TRM-D is now written as:

TRM-D (TR#, E#, EL-TYP, E, CSID, T, NODES#, NODE1#, NODE2#, NODE3#, X, Y, X, MATXID, MATX)

The functional dependencies reflected by elementary relations ER1 to ER15 are satisfied in the internal model with the values shown in Fig. 3. There-



TR#	E#	EL-TYP	MATID	E	CSID	T	NODE#	X	Y	Z	MATXID	MATX
1	15	TRM3	STEEL*5	10 ⁸	THICK*6	0.1	5 7 8	1. 3. 2.	5. 4. 6.	7. 8. 3.	SIF*1	{:::}
2	16	TRM3	ALUM*4	0.9x10 ⁷	THICK*4	0.2	12 14 18	5. 7. 3.	9. 4. 5.	8. 1. 8.	SIF*2	{:::}

Fig. 2. A tentative internal model

TR#	E#	EL-TYP	MATID	E	CSID	T	NODE#	X	Y	Z	MATXID	MATX
1	15	TRM3	STEEL*5	10 ⁸	THICK*8	0.1	5	1.	5.	7.	SIF*1	{:::}
1	15	TRM3	STEEL*5	10 ⁸	THICK*8	0.1	7	3.	4.	8.	SIF*1	{:::}
1	15	TRM3	STEEL*5	10 ⁸	THICK*8	0.1	8	2.	6.	3.	SIF*1	{:::}
2	16	TRM3	ALUM*4	0.9x10 ⁷	THICK*4	0.2	12	5.	9.	8.	SIF*2	{:::}
2	16	TRM3	ALUM*4	0.9x10 ⁷	THICK*4	0.2	14	7.	4.	1.	SIF*2	{:::}
2	16	TRM3	ALUM*4	0.9x10 ⁷	THICK*4	0.2	18	3.	5.	8.	SIF*2	{:::}

Fig. 3. Relation TRM-D in 1NF

fore at this instant the internal model is consistent with the conceptual model. However, it would be no longer consistent if arbitrary changes in the values of the table are made. Also, note that many values in the relation TRM-D are redundant. These inconsistencies and redundancies occur because of the anomalies in the 1NF. Thus it is not desirable to use the relation in Fig. 3 to represent the internal model. Modification to 2NF is necessary to avoid the anomalies [9] in the storage operations. The relation TRM-D should be converted into a set of semantically equivalent relations as follows:

- TRM-D1 (TR#, E#, EL-TYPE, NODE1#, NODE2#, NODE3#, MATID, E, CSID, T, MATXID, MATX)
- TRM-D2 (NODE#, X, Y, Z)
- TRM-D3 (E#, NODE#)

The preceding three relations TRM-D1, TRM-D2, and TRM-D3 are all in 2NF because the first two relations do not possess any compound candidate

key and the third relation has all keys. Note that by splitting the relation TRM-D no information is lost and the relations are still consistent with the conceptual model. However, TRM-D1 relation is still not satisfactory since it can lead to anomalies in storage operations. Modification of the relation is necessary to 3NF to avoid anomalies in storage operation. Nonkey attributes must be nontransitively dependent on candidate keys to avoid these anomalies. It can be observed from the relation TRM-D1 of Fig. 4 that attributes E, T, and MATX are transitively dependent on TR# through MATID, CSID, and MATXID, respectively. Removing these transitive dependencies, we get the following relations:

- TRM-D4 (TR#, E#, EL-TYP, NODE1#, NODE2#, NODE3#, MATID, CSID, MATXID)
- TRM-D5 (MATID, E)
- TRM-D6 (CSID, T)
- TRM-D7 (MATXID, MATX)

TRM-01

TR#	E#	EL-TYP	NODE1#	NODE2#	NODE3#	MATID	E	CSID	T	MATXID	MATX
1	15	TRM3	5	7	8	STEEL*5	10^8	THICK*8	0.1	SIF*1	{...}
2	16	TRM3	12	14	18	ALUM*4	0.9×10^7	THICK*4	0.2	SIF*2	{...}

TRM-02

NODE#	X	Y	Z
5	1.	5.	7.
7	3.	4.	8.
8	2.	6.	3.
12	5.	9.	8.
14	7.	4.	1.
18	3.	5.	8.

TRM-03

E#	NODE#
15	5
15	7
15	8
16	12
16	14
16	18

Fig. 4. Relations in 2NF

The preceding three relations together with TRM-2 and TRM-D3 constitute the internal model for element stiffness matrix generation purpose. This internal model is consistent with the conceptual model identified earlier. Also, note that the number of relations in the internal model is only 6 as compared to 15 elementary relations in the conceptual model.

In summary, the following steps are necessary to derive an internal model that is consistent with the conceptual model. Normalization procedures have to be adopted at each step to reduce redundancy and to eliminate undesired anomalies in storage operation. This ensures integrity of the stored values in the data base. At each step unsatisfactory relations are replaced by others.

Step 1. Form relations with attributes derived from a set of domains.

Step 2. Eliminate multiple values at the row-column intersection of the relation table. Vectors and matrices are considered to be single data items for this step.

Step 3. The result of Step 2 is the relations in the 1NF. Take projections of 1NF relations to eliminate any nonfull functional dependencies and get relations in the 2NF.

Step 4. Take projection of relations obtained in Step 3 to eliminate transitive dependencies to form relations in the 3NF. Thus a set of relations in the 3NF is the internal model.

5 Some Aspects to Accommodate an External Model

One of the important requirements of a data base is to provide facility for data retrieval by different application programs depending on their needs. Different application programmers can have different views of a data base. Data structure as seen by an application program or interactive user is called an external data model. Transformations are required involving rearrangement of data from the internal level to the external level into a form acceptable to the application program. Some constraints have to be observed while designing an external model. Constraints arise while rearranging data from internal data structure to an external data structure. Any retrieval and storage operations specified on the external model must be correctly transformed into corresponding operations on the internal model, and at the same time, the internal model must be consistent with the conceptual data model. An example of how an external model is derived from an internal model is given subsequently.

Suppose a particular user would like to know the coordinates of nodes of each triangular finite element for a generation of element stiffness matrices. This means that the external model:

EL-CORD (TR#, E#, EL-TYPE, X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)

has to be provided for that particular user. Note that the external view EL-CORD contains data items from two different relations—TRM-D4, TRM-D2. Therefore a procedure is required to transform the internal data model (relations TRM-D4, TRM-D2) to the external data model (relation EL-CORD). This can be done using JOIN and PROJECT operations [9] as follows:

TRM-A (TR#, E#, EL-TYP, NODE1#) ←
TRM-D4

TRM-B (TR#, E#, EL-TYP, NODE2#) ←
TRM-D4

TRM-C (TR#, E#, EL-TYP, NODE3#) ←
TRM-D4

TRM-D (TR#, E#, EL-TYP, X1, Y1, Z1)
= TRM-A*TRM-D2

TRM-E (TR#, E#, EL-TYP, X2, Y2, Z2)
= TRM-B*TRM-D2

TRM-F (TR#, E#, EL-TYP, X3, Y3, Z3)
= TRM-C*TRM-D2

EL-CORD (TR#, E#, EL-TYPE, X1, Y1, Z1,
X2, Y2, Z2, X3, Y3, Z3)
= TRM-D*TRM-E*TRM-F

NOTE: ← indicates PROJECT; * indicates JOIN

It can be seen from the algorithm that we did not modify the original relations TRM-D4 and TRM-D2 to retrieve the data required for a particular inquiry. The relations TRM-D4 and TRM-D2 are still consistent with the conceptual model. Therefore pure retrieval operations for rearrangement of data does not cause any inconsistency in data values.

Now, consider the reverse process of transforming external data structure to internal data structure. Suppose a particular user wants to insert the nodal coordinates of a finite element using the external view EL-CORD. Here relation EL-CORD has the only key TR# and has no reference to the node number to which the element is connected. Insertion is not consistent with the conceptual model because it requires the coordinates of nodes which are dependent on keys NODE#. This restriction is also reflected in the internal model—TRM-D2 that requires NODE# as key values for insertion. Therefore the transformation of relation EL-CORD into the internal model is not possible. From this example it follows that there are restrictions for rearranging data from external model to internal model.

6 Methodology to Incorporate Matrix Data into a Data Base

In finite element analysis and structural design optimization, we encounter the problem of storage of large order matrices. This data is unique to the application, so no attempts have been made to design such data bases in the business data base management area. Consequently, there is a need for the development of a new generalized user friendly technique to deal with large order matrices. For the purpose of our discussion, matrices are grouped into five types and are referred by the type number: (1) square matrix *A*, (2) banded matrix *A*, (3) hypermatrix *H*, (4) skyline matrix *S*, and (5) sparse matrix *P*. A methodology is proposed in this section consisting of conceptual, internal and external views of large matrices.

Conceptually, a matrix is a two-dimensional array of numbers. These numbers appear in a certain pattern; for example, square, sparse, symmetric, diagonal, banded, lower triangular form, upper triangular form, unitary form, tridiagonal form, hypermatrix form, and skyline form. A matrix is uniquely identified by a name. Rows and columns of the two-dimensional array are used for identification of data elements in the matrix. A conceptual view of a matrix can be represented by the following elementary relations:

ER1 (NAME, MATRIX TYPE)
ER2 (NAME, NUM-OF-ROWS)
ER3 (NAME, NUM-OF-COLUMNS)
ER4 (NAME, ROW, COLUMN,
DATA-ELEMENT-VALUE)
ER5 (NAME, NUM-OF-HYPER ROWS)
ER6 (NAME, NUM-OF-HYPER COLUMNS)
ER7 (NAME, HYP-ROW, HYP-COLUMN,
ROW, COLUMN, DATA-ELEM-VALUE)
ER8 (NAME, BAND-WIDTH)
ER10 (NAME, SUB-MAT-ROW-SIZE)
ER11 (NAME, SUB-MAT-COLUMN-SIZE)
ER12 (NAME, VECTOR OF
SKYLINE-HEIGHT)
ER13 (NAME, HYP-ROW, HYP-COLUMN,
NULL-OR-NOT)

The attributes of these elementary relations are self-descriptive. These elementary relations completely define a matrix and provide the conceptual structure of the matrix.

An internal (storage) structure for large order matrices has to be developed that is consistent with the conceptual structure. Storage schemes have to be developed based on efficiency and processing considerations. The special nature of the matrix, that is, sparse, dense, or symmetric, should be used to provide storage efficiency. We can classify various matrix types considered in the previous section into two basic types—sparse and dense. Note that banded or diagonal matrices are not to be mistaken as sparse. Many possible storage schemes are available to store dense and sparse matrices. Conventional storage schemes—row-wise, column-wise, submatrix-wise are useful for storing dense matrices. Choice among these storage schemes should be based on consideration of several aspects—storage space, processing sequence, matrix operation, page size, flexibility for data modification, ease of transformation to other storage schemes or user's views, number of addresses required to locate rows or submatrices, and availability of data base management system support. These aspects are considered in detail now.

Storage space. The row storage scheme can be used for square, banded, and skyline matrix types. However, this scheme is not appropriate for the hypermatrix. Symmetric, triangular, and diagonal properties of a square matrix can be used in saving storage space if the variable length of rows is used. Similar schemes can be used for banded and skyline matrices to store data elements that appear in a band or skyline column. Submatrix storage can be used for all matrix types. The submatrix is most appropriate for hypermatrix data. Both schemes have disadvantages when zero elements within a row or submatrix have to be stored.

Processing sequence. Row storage requires that the assembly of matrices, storage, and retrieval be made only row-wise. This becomes inefficient if row-wise processing cannot be made. The submatrix approach is suitable for all types of processing sequence—row-wise, column-wise, or in any arbitrary order.

Matrix operations. Operations such as transpose, addition, multiplications, and solutions of simultaneous equations are frequently carried out at various stages of structural design. The row storage scheme is highly inefficient for matrix transpose when column-wise storage is required. During multiplication of two matrices A and B , a column of B can be obtained only by retrieving all of the rows of B . Therefore, the row storage scheme becomes inappropriate for such an operation. However, the

submatrix storage scheme does not impose any such constraints in matrix operation, thus providing a suitable internal storage scheme.

Page size. A page is a unit or block of data stored or retrieved from memory to disk. For a fixed page size, only a number of full rows or a number of full submatrices together with fractional parts of them can be stored or retrieved at a time. It is clear that fragmentation of rows or submatrices takes place depending on the size of rows or submatrices. Large row size will overlap more than one page in memory and cause wastage of space. The submatrix scheme has the advantage of providing flexibility in choosing the submatrix size to minimize fragmentation of pages.

Flexibility for data modification. For modification of rows of a matrix, both row and submatrix storage schemes are suitable. But the row scheme would be more efficient than the submatrix storage scheme. For modification of a few columns of a matrix, the row storage scheme requires a large number of I/O.

Transformation to other schemes. The submatrix storage scheme requires a minimum number of data access to transform to the column-wise storage scheme.

Address required. The submatrix storage requires a fewer number of addresses to locate data than the row storage scheme, provided submatrices are reasonably large.

Thus for internal storage of large order matrices in a data base, the preceding aspects should be carefully considered. It appears that both submatrix and row storage schemes can be appropriate for various applications.

In order that the internal storage scheme be consistent with the conceptual model, we need to store additional information about the properties of the matrix. That additional information is given by the elementary relations ER1, ER2, ER3, ER5, ER6, ER9 to ER13. These can be combined and stored in a relation. A typical relation required for the internal storage of submatrices is shown in Fig. 5.

So far we have considered schemes for internal organization of large matrices. Since different users view the same matrix in different forms—banded, skyline, hypermatrix, triangular, or diagonal, it is necessary to provide external views to suit individual needs. The unit of transactions on various views of a matrix may be row-wise, column-wise, submatrix-wise, or data element wise. If the internal scheme is submatrix-wise, the external view need

NAME	HYP-ROW NO.	HYP-COLUMN NO.	NULL or NOT	SUBMATRIX
A	1	1		[...]
A	1	2		[...]
...	

Fig. 5. Relations for matrix storage

not be submatrix wise. Therefore transformations are necessary to convert the internal matrix data into the form required by a particular user.

Next, we consider the sparse matrix storage scheme. Several storage schemes have been suggested by Pooch [11] and Danini [12]. They are the bit-map scheme, address map scheme, row-column scheme, and threaded list scheme. Out of these, the row-column scheme is simple and easy to use. Also, the row-column scheme can be easily incorporated into the relational model. Therefore, this scheme can be considered for storing sparse matrices encountered in design sensitivity analysis.

The row-column storage scheme consists of identification of row and column numbers of non-zero elements of a sparse matrix and storing them in a table. This scheme provides flexibility in the modification of data. Any nonzero value generated during a course of matrix operation can be stored or deleted by simply adding or deleting a row in the stored table. The external view of the row-column storage scheme can be provided through suitable transformation procedures.

7 Processing, Efficiency, and Other Considerations

In the following paragraphs special consideration needed for satisfying processing, efficiency, and iterative needs of finite element analysis and structural design optimization are discussed. Many procedures in structural design, such as element stiffness matrix routines, generate huge amounts of data. Generally, it is not preferable to store such data in a data base at the expense of disk space and data transportation time. This inefficiency can be avoided by storing only a minimum amount of data needed to generate the required information (element stiffness matrix). In general, a data model can be replaced by (1) an algorithm that generates the user-requested information, and (2) a set of (mini-

imum) data that will be used by an algorithm to generate user-requested information.

A network of data bases offers considerable aid in iterative structural design process. It consists of a global data base connected to a number of local data bases through a program data interface. A global data base can be used to store common information required for all applications, whereas a local data base contains only application-dependent transitory data. Local data bases are dependent on application programs and are highly efficient in data accessing, since no overhead is involved in supporting complicated data structures. It supports iterative design process by providing temporary work space that can be erased at the end of an iteration. Any intermediate data generated by applications can be stored in a local data base.

A number of parameters pertaining to structural problem definition and applications generally require storage in a data base. Users should have complete flexibility in organizing such parameters in a relation because they are dependent on problem and application.

Special consideration is needed in naming relations belonging to different substructures. One option is to name uniquely all relations of various substructures. Another way is to store all relations in a data base that is used only for storing data of a particular substructure.

8 Conclusions

A methodology to design a data base for a finite element analysis and structural design optimization is described. The methodology adopts several good features of available data base design techniques. The data base design using three views of data considers both theoretically possible and implementational aspects of data organization. The relational data model is shown to be quite useful in providing a clear and simple picture of data for designing a data base. Special characteristics of analysis and design data such as vectors, matrices, and processing of data are accommodated in the methodology. Examples used for discussing the methodology are relevant and can be easily extended to the actual data base design. The methodology is expected to provide a good start in arriving at a systematic approach to design data bases for computed-aided analysis and the design of structural systems. The design of such a comprehensive data base is in progress [13]. A software system to process such a data

base has been designed. It is being implemented with a specially designed data base management system called MIDAS [14]. The data base design and the system will be evaluated and the results reported in the near future.

Acknowledgment

This research is sponsored by the Air Force Office of Scientific Research, Grant No. AFOSR 82-0322. Material of the paper is derived from a presentation made by the authors at the 26th AIAA, Structures, Structural Dynamics and Materials Conference, April 15-17, 1985, Orlando, Florida.

References

1. Koriba, M. (1983) Database systems: Their applications to CAD software design. *Comput-Aid. Des.* 15(5) 277-288
2. Lopez, L.A. (1974) FILES: Automated engineering data management system. *Comput. Civ. Eng. Electr. Comput.*, ASCE 47-71
3. Pahl, P.J. (1981) Data management in finite element analysis. In: *Nonlinear Finite Element Analysis in Structural Mechanics*. (Eds. W. Wunderlich, E. Stein, K.J. Bathe) Berlin: Springer-Verlag, pp. 714-716
4. Sreekanta Murthy, T., Arora, J.S. (1986) Database management concepts in computed-aided design optimization. *Adv. in Eng. Software*, to appear, April
5. Fishwick, P.A., Blackburn, C.L. (1982) The integration engineering programs using a relational database scheme. *Comput. Eng. Int. Comput. Eng. Conf.*, pp. 173-181
6. Buchmann, A.P., Dale, A.G. (1979) Evaluation criteria for logical database design methodologies. *Comput-Aid. Des.* 121-126, Vol. 11, No. 3, May
7. Grabowski, H., Eigner, M. (1982) A data model for a design database. *File Structures and Databases for CAD*. In: *Proceedings of the International Federation of Information Processing*, pp. 117-144
8. Vetter, M., Maddison, R.N. (1981) *Database Design Methodology*. Englewood, NJ: Prentice-Hall International
9. Date, C.J. (1977) *An Introduction to Database Systems*. Reading, MA: Addison-Wesley
10. Sreekanta Murthy, T., Arora, J.S. (1984) Database design methodology and Database Management System for Computed-Aided Structural Design Optimization. Technical Report CAD-SS-84.20, Optimal Design Laboratory, College of Engineering, The University of Iowa
11. Pooch, U.W., Nieder, A. (1973) A survey of indexing techniques for sparse matrices. *Comput Surv.* 5(2), 109-133
12. Daini, O.A. (1982) Numerical database management system: A model. *Int. Conf. on Data Management*, Association for Computing Machinery, Special Interest Group on Management of Data, 192-199
13. Arora, J.S., Al-Saadoun, S.S., Haririan, M., Sreekanta Murthy, T., Wu, C.C. (1985) Preliminary design of a general purpose structural design optimization system (S-DOS). Report No. ODL 85.8, Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA, July
14. Sreekanta Murthy, T., Shyy, Y-K., Arora, J.S. (1986) MIDAS: Management of information for design and analysis of systems. *Adv. in Eng. Software*, to appear, April

APPENDIX 5

**SMART: SCIENTIFIC DATABASE MANAGEMENT AND
ENGINEERING ANALYSIS ROUTINES AND TOOLS**

by

J.S. Arora, H.H. Lee and S.Y. Jao

in

Advances in Engineering Software

Vol. 8, No. 4, 1986

System initialising routine

TINIT*: Defines the terminal type and output unit number for the system. The subroutine must be called to initialise the system.

String processing routines

These routines may be used to manipulate strings of characters (see also *string table processing routines*).

LTRIM: Returns the trimmed length of a string.
LNBLK: Returns the location of the nonblank portion of a string.
TSHFTL: Shifts a string by N positions to the left.
TSHFTR: Shifts a string by N positions to the right.
TJUST: Left justifies characters in a string.
TJUSTR: Right justifies characters in a string.
TINSRS: Inserts characters into a string.
TREMV: Removes characters from a string.
TCOPY: Copies data or a string.

String table processing routines

A variety of processes may be set up using these string table routines. The string table is a user defined array of character strings which the routines manipulate.

TSEPAR: Separates a string into substrings which are separated by delimiters ',', '=' or blank(s).
LCTAB1: Sequentially searches for the specified string in a string table and returns its location. The search can be over the entire table or specified rows.
LCTAB2: This subroutine is a binary search counterpart of LCTAB1. It searches the specified string in a string table and returns its location. The search can be over the entire table or specified rows.
TSORTA: Sorts a string table into the alphabetical order.
TINSRT: Inserts a string into the string table after a specified location. The size of the table is increased by one.
TREMT: Removes an element from a string table by location.
TINSRG: Inserts data into a data table by location.
TREMG: Removes data from a data table by location.

Number-string conversion routines

Simple conversion routines and more complex expression evaluating routines are included here.

TCNVRT: Converts a string which may be an arithmetic expression to its value.
TSETC: Sets a constant value for calculating the result of an arithmetic expression.
TGETC: Gets a value of a constant from the constant table.
TDELC: Removes a constant from the constant table.
TSAVEC: Saves the constant table to a file.
TLOADC: Loads the constant table from a file.
TNUMST: Converts a string to a number of specified data type. The string can be an arithmetic expression.
TSTNUM: Converts a given number of specified data type to a string. The format can be also specified.
TDTYPE: Encodes data type into useable form.
LFIELD: Returns string format field length (for use with TSTNUM).
TZERO: Assigns a zero value to a variable.

Screen control routines

Given that the terminal type is initialised by TINIT, certain terminal screen functions can be controlled by these routines.

TCURSR*: Moves the cursor to the specified position.

TCLEAR*: Clears the screen.

TBELL: Outputs a bell sound.

TINSTR: Moves the cursor to the specified position and reads a string.

TOUSTR: Moves the cursor to the specified position and displays a string.

TOUNUM: Moves the cursor to a specified position and displays data.

TYESNO: Performs the interactive YES/NO query and returns the code.

On-line reference display routines

Provides a menu and tests the user response.

TMENU: Displays a menu contained in an array and performs the interactive selection. The entire or partial menu in the array can be displayed.

Full screen data edit routines

Several routines are provided to allow easy editing of data from arrays or tables.

TEDIT1: Displays a description of data items contained in an array and performs data input or editing interactively. An entire list or a partial list of data descriptions, contained in the array can be displayed.
TEDIT2: Displays a relation table and performs data input or editing interactively.
TEDITM: Displays a matrix and performs data input or editing interactively.
TEDITC: Edits the constant table for number-string conversion routines.

Data output routines

TPRINT1: Prints a one-dimensional array of data items to a file unit. This is a hard-copy counterpart of subroutine TEDIT1.
TPRINT2: Prints a relation table to a file unit. This is a hard-copy counter-part of subroutine TEDIT2.
TPRINTM: Prints a matrix to a file unit. This is a hard-copy counter-part of subroutine TEDITM.

Sorting routines

Several sorting routines are available based on different methods. These routines are for numeric data of various types which may be specified for each routine.

TSRT1: Straight insertion sort.
TSRT2: Binary insertion sort.
TSRT3: Straight selection sort.
TSRT4: Bubble sort.
TSRT5: Shaker sort.
TSRT6: Heap sort.
TSRT7: Quick sort.

3. VECTOR AND MATRIX OPERATIONS ROUTINES

This part contains various vector and matrix operation subroutines. Capabilities for elementary vector and matrix operations, symmetric, banded, and sparse matrix decomposition, equation solvers, eigenvalue extractors and updating of factorised matrices are available. Single as well as double precision routines are available. The subroutines are carefully designed to operate on vectors rather than individual elements. They can be easily installed on vector computers. All routines in this part start with the letter 'Z'. The second

letter for each subroutine name is either 'S' or 'D'; 'S' implies single precision and 'D' implies double precision.

Vector operations

ZSMAX OR ZDMAX: Finds the largest positive element in a vector starting from a specified position. It also returns the position of the element.

ZSMAXA OR ZDMAXA: Finds the absolute maximum element in a vector starting from the specified location. It also returns the position of the element.

ZSDOT OR ZDDOT: This is a function subprogram to calculate the dot product of two vectors.

ZSSOTX OR ZDSOTX: Sorts a vector according to the specified order. The original vector is saved.

ZSSCLX OR ZDSCCLX: Scales a vector by a constant. The original vector may be saved.

ZSAADDV OR ZDADDV: Adds two vectors to produce a third vector. The original vectors may be saved.

ZSSUBV OR ZDSUBV: Subtracts one vector from another to produce a third vector. The original vectors may be saved.

ZSVINI OR ZDVINI: Initialises a vector to zero.

ZSUNIT OR ZDUNIT: Computes the length of a vector and normalizes it to a unit vector.

ZSNORM OR ZDNORM: Computes the Euclidean norm of a vector.

ZSCROS OR ZDCROS: Computes the cross product of two vectors and stores it in a matrix.

ZSXPCY OR ZDXPCY: Scales a vector by a constant and adds the resulting vector to another vector to produce a third vector. The original vectors may be saved.

ZSIND OR ZDIND: Generates (in an integer vector) indices according to the ascending order of elements in a vector.

Matrix operations

ZSMINT OR ZDMINT: Initialises a matrix to zero.

ZSMSCL OR ZDMSCL: Scales a matrix by a constant. The original matrix is destroyed.

ZSMADD OR ZDMADD: Adds two matrices to produce a third matrix. The original matrices may be saved.

ZSMSUB OR ZDMSUB: Subtracts two matrices to produce a third matrix. The original matrices may be saved.

ZSATAU OR ZDATAU: Takes the transpose of a matrix and pre-multiplies the original matrix by it. Columns of the original matrix are assumed to be unit vectors. All diagonal elements are set to unity. If any off-diagonal element is close to unity, then the corresponding two column vectors are nearly parallel.

ZSATAG OR ZDATAG: Takes the transpose of a matrix and pre-multiplies the original matrix by it. Columns of the matrix are assumed to be general vectors.

ZSATAI OR ZDATAI: Takes the transpose of a matrix and pre-multiplies the original matrix by it. The order of multiplication of columns of the matrix is specified in a vector. Columns of the original matrix are used in that order for multiplication.

ZDAAT OR ZDAAT: Calculates the transpose of a matrix and post-multiplies the original matrix by it.

ZSMATM OR ZDMATM: Multiplies two matrices to generate a third matrix.

ZSTRAN OR ZDTRAN: Computes the transpose of a matrix and stores it in a second matrix.

ZSAMX OR ZDAMX: Multiplies a matrix by a vector and stores it in another vector.

ZSAMXI OR ZDAMXI: Multiplies a matrix by a vector according to the order of columns specified in an integer vector.

ZSATX OR ZDATX: Takes the transpose of a matrix and multiplies it by a vector to generate another vector.

ZSSOTA OR ZDSOTA: Sorts columns of a matrix according to the indices in an integer vector.

ZSAMD OR ZDAMD: Post-multiplies a matrix by a diagonal matrix. The diagonal matrix is in vector form. The original matrix may be saved.

ZSDMA OR ZDDMA: Pre-multiplies a matrix by a diagonal matrix. The diagonal matrix is in vector form. The original matrix may be saved.

ZSDMX OR ZDDMX: Multiplies a diagonal matrix by a vector. The diagonal matrix is in vector form. The original vector may be saved.

ZSCOPY OR ZDCOPY: Copies a matrix into another matrix.

ZSCUTL OR ZDCUTL: Copies an upper triangular matrix to the lower part.

ZSCLTU OR ZDSCLTU: Copies a lower triangular matrix to the upper part.

ZSCOMP OR ZDCOMP: Compacts a two-dimensional array. The subroutine can be used to change the dimension of a two-dimensional array. For example, an $NA \times NC$ matrix containing useful data in the $M \times N$ upper left part ($M < NA$ and $N < NC$) can be compacted by the subroutine. The new dimension of the matrix will be $M \times N$. The space ($NA * NC - M * N$) is released.

ZSEXPN OR ZDEXPN: Expands the dimension of a two-dimensional array. The purpose of this subroutine is opposite to that of the compression subroutine ZSCOMP. An $M \times N$ matrix is expanded to $NA \times NC$ where $NA > M$ and $NC > N$.

ZSLTMV OR ZDLTMV: Post-multiplies a lower triangular matrix, stored in one-dimensional form, by a vector.

ZSTLMV OR ZDTLMV: Post-multiplies the transpose of a lower triangular matrix, stored in one-dimensional form, by a vector.

ZSUTMV OR ZDUTMV: Post-multiplies an upper triangular matrix, stored in one-dimensional form, by a vector.

ZSTUMV OR ZDTUMV: Post-multiplies the transpose of an upper triangular matrix, stored in one-dimensional form, by a vector.

ZSLDLP OR ZDLPLP: Updates the Cholesky factorisation of a matrix for the addition case by a method given by ref. 2.

ZSLDLM OR ZDLML: Updates the Cholesky factorisation of a matrix for the subtraction case by a method given in ref. 2.

ZSLTMD OR ZDLTMD: Post-multiplies a lower triangular matrix, stored in one-dimensional form, by a diagonal matrix stored in vector form. It returns a lower triangular matrix in one-dimensional form.

ZSUTMD OR ZDUTMD: Post-multiplies an upper triangular matrix, stored in one-dimensional form, by a diagonal matrix in vector form. It returns an upper triangular matrix in the one-dimensional form.

ZSLTXR OR ZDLTXR: Solves a linear system of equations whose coefficient matrix is lower triangular matrix stored in the one-dimensional form ($Lx = b$). The right-hand side vector is destroyed.

ZSTLXR OR ZDTLXR: Solve the linear system of equations whose coefficient matrix is the transpose of the lower triangular matrix stored in one-dimensional form ($L^T x = b$). The right-hand side vector is destroyed.

ZSUTXR OR ZDUTXR: Solves the linear system of equations whose coefficient matrix is upper triangular matrix stored in one-dimensional form ($Ux = b$). The right-hand side vector is destroyed.

ZSTUXR OR ZDTUXR: Solves the linear system of equations whose coefficient matrix is the transpose of an upper triangular matrix stored in one-dimensional form ($U^T x = b$). The right-hand side vector is destroyed.

ZSBUMX OR ZDBUMX: Post-multiplies on upper triangular banded matrix by a vector. The banded part of the matrix is squeezed into a rectangular array. The diagonal elements are squeezed to the first column.

ZSBLMX OR ZDBLMX: Post-multiplies the transpose of an upper triangular banded matrix by a vector. The banded part of the matrix is squeezed into a rectangular array. The diagonal elements are squeezed to the first column.

ZSBMUX OR ZDBMUX: Post-multiplies a symmetric banded matrix by a vector. Only the upper part of the banded matrix needs to be input. The diagonal elements are squeezed to the first column.

ZSBNOR OR ZDBNOR: Normalises each column of a matrix with respect to a positive definite symmetric banded matrix.

ZSDPGE OR ZDDPGE: Decomposes a general square matrix into lower and upper triangular matrices.

ZSFSGE OR ZDFSGE: Performs forward substitution to solve the unit lower triangular system of linear equations.

ZSBSGE OR ZDBSGE: Performs backward substitution to solve the upper triangular system of linear equations.

ZSSLGE OR ZDSLGE: Solves a general system of linear equations.

ZSDPGV OR ZDDPGV: Decomposes a general matrix with a global pivot strategy.

ZSDFSGV OR ZDFSFGV: Performs forward substitution to solve the unit lower triangular system of linear equations with a global pivot strategy.

ZSBSGV OR ZDBSGV: Performs backward substitution to solve the upper triangular system of linear equations with a global pivot strategy.

ZSSLGV OR ZDSLGV: Solves a linear system of equations with a global pivot strategy.

ZSGINV OR ZDGINV: Inverts a matrix and computes its determinant.

ZSSYDU OR ZDSYDU: Decomposes a symmetric full matrix into an upper triangular matrix leaving the lower triangular portion undisturbed.

ZSSYFS OR ZDSYFS: Performs forward substitution to solve the unit lower triangular system of linear equations obtained from the subroutine ZSSYDU or ZDSYDU.

ZSSYBS OR ZDSYBS: Performs backward substitution to solve the upper triangular system of linear equations obtained from the subroutine ZSSYDU or ZDSYDU.

ZSSYDD OR ZDSYDD: Decomposes a symmetric full matrix into $U^T D U$.

ZSSYSL OR ZDSYSL: Solves a symmetric linear system of equations by decomposition.

ZSSYIN OR ZDSYIN: Inverts a symmetric full matrix.

ZSCBDU OR ZDCBDU: Decomposes a symmetric banded matrix stored in squeezed rectangular form.

ZSCBFS OR ZDCBFS: Performs forward substitution to solve the decomposed symmetric banded system of equations obtained from the subroutine ZSCBDU or ZDCBDU.

ZSCBBS OR ZDCBBS: Performs backward substitution to solve the decomposed symmetric banded system of equations obtained from the subroutine ZSCBDU or ZDCBDU.

ZSCBDD OR ZDCBDD: Decomposes a symmetric banded squeezed matrix into upper triangular and diagonal matrices.

ZSCBSL OR ZDCBSL: Solves a symmetric banded system of linear equations.

ZSVBDU OR ZDVBDU: Decomposes a symmetric variable band width matrix.

ZSVBFS OR ZDVBFS: Performs forward substitution to solve the decomposed symmetric variable band width system of linear equations obtained from the subroutine ZSVBDU or ZDVBDU.

ZSVBBS OR ZDVBBS: Performs backward substitution to solve the decomposed symmetric variable band width system of linear equations obtained from the subroutine ZSVBDU or ZDVBDU.

ZSVBDD OR ZDVBDD: Decomposes a symmetric variable band width matrix into upper triangular and diagonal matrices.

ZSVBSL OR ZDVBSL: Solves a symmetric variable band width linear system of equations.

ZSJACB OR ZDJACB: Solves a generalised eigenproblem using the generalised JACOBI iteration method with the option of selecting output environment.

ZDEIGN: Solves for the smallest eigenvalues and the corresponding eigenvectors in the generalised eigenproblem using the subspace iteration method.

ZDEIGR: Solves for the smallest eigenvalues and corresponding eigenvectors in the generalised eigenproblem using the subspace iteration method.

ZDEIGJ: Solves a generalised eigenproblem using the generalised JACOBI iteration method without the option of selecting output environment.

4. IN-CORE DATA MANAGEMENT SYSTEM (IDMS)

A set of FORTRAN subroutines is described to handle the in-core data management for engineering applications. The in-core data management system (IDMS) performs the chores of dynamic partitioning of arrays for application programs. The programmer can easily manage an in-core array. The data in the array can be INTEGER*2, INTEGER*4, REAL*4, REAL*8 or CHARACTER*n, where n is an arbitrary positive integer. Each 'Data Set' (a group of data of the same data type) is in the form of a matrix. All data sets are stored in a single memory block and managed by IDMS. The memory block is declared by the user as a named COMMON block. The named COMMON 'CORE' is compulsory.

By using IDMS subroutines the programmer can easily perform the following in-core data management:

1. Define a data set (reserve the memory for a data set)
2. Change the dimension of a data set
3. Change the format of a data set
4. Delete a data set from the memory block
5. Duplicate a data set
6. Load (or save) the in-core data from (or to) a file
7. Get information about memory block or a specified data set
8. Get error codes and error messages
9. Store (or retrieve) data to (or from) a data set
10. Perform operations on the data sets
11. Perform full screen editing of the data set
12. Print a data set to an output device.

Errors are detected by IDMS (it is optional) and stored in memory. This memory is updated soon after each call to IDMS. The user can get error messages and error codes after each call or optionally, can have them displayed at the terminal.

General routines

XINIT: Turns on the IDMS, creates a map for data management, and initialises it.
 XDEF: Defines a data set for storing data.
 XCNAME: Changes the name of a data set.
 XCDIM: Changes the dimension of a data set.
 XCFMT: Changes the output format of a data set.
 XDEL: Deletes a data set from the memory block.
 XCOPY: Copies a data set.
 XSAVE: Saves the memory block in a file.
 XLOAD: Loads the memory block from a file.

Information request routines

XERROR: Returns the current error message.
 XERINF: Returns all the information about data set 'DSNAME'.
 Function LOCAT: Returns the location (address in the block) of a data set.
 XNAME: Return names of the data sets.
 XCORE: Returns status of memory allocation.

Data access routines

XGET: Gets an element from a data set.
 XPUT: Puts an element into a data set.
 XGETS: Gets a portion (submatrix) of data set from memory block to another array.
 XPUTS: Puts a portion (submatrix) of data set from an array to the memory block.
 XEDITM: Performs full screen edit of the data set.
 XPRNTM: Prints a data set to an output device.

5. DATABASE MANAGEMENT SYSTEM MIDAS

MIDAS is an advanced database management system capable of organising and managing large amounts of data for engineering applications. Detailed description of the system, its design, capabilities and commands are given by Sreekanta Murthy *et al.*⁴ Therefore only a summary of its capabilities is described here.

MIDAS consists of two subsystems MIDAS/N and MIDAS/R. These subsystems are capable of organising data of numerical and relational models respectively. MIDAS/N is specially developed to organise matrix data. This subsystem can handle multiple databases, small and large matrices of various type, and small and large memory environment. Memory management is used in MIDAS/N to efficiently utilise the available computer memory which is divided into a number of pages of fixed size. Number of pages and page size can be varied by changing a few parameters. Least recently used page replacement strategy is adopted in the memory management system.

MIDAS/R is based on modification and extension of the RIM system,^{1,3} which stands for MIDAS - Relational Data Management System. The traditional relational model is extended in RIM such that its attributes can be variable length vectors and matrices. This is done to organise matrix type data usually encountered in engineering applications. A user interface has been added to the system so that it can be used with FORTRAN programs.

Both subsystems of MIDAS provide simple to use data definition and data manipulation languages for application programmers. Data definition language consists of sub-routine CALL statements to define relations and matrices. Integer, real, double precision and character data types are allowed. Large matrices such as square, rectangular, upper triangular, skyline and hyper-matrices can be defined.

Dynamic data definition is possible. Data manipulation language offers capability to store, retrieve, modify and delete data in the database.

Using MIDAS/R, relations can be stored and retrieved in row order, and matrices can be stored and retrieved in submatrix order. Subroutines for conditional search on various attributes are available. Traditional relational algebra commands are also available. The subsystem can be used for interactive manipulation of data and queries.

MIDAS/N is more suitable for storing and manipulating large matrices that can be created in one order and manipulated in another. Their dimensions can be changed during run time. Several subroutines are available to manipulate matrices stored in the database. Matrix decomposition and equation solving subroutines are available.

6. GRAPHICS UTILITIES

Graphics utilities are provided for basic graphic operations such as drawing graphs on a terminal. One or several curves can be drawn simultaneously, and graph axes and labels can be provided by using several routines specifically written for this purpose.

The graphics routines are written for and are specific to certain terminal types, which include Tektronix (under 4027 mode), HP2648A terminals, and Apollo workstations.

There are three types of routines - first level, second level and device driver routines. First level routines can be called directly by the user to plot curves, axes and labels. The second level routines are more basic and are called by the first level routines. The device driver routines are for specific graphics terminals.

Several co-ordinate systems are used depending on the level of the routine. Included are user's co-ordinates, world co-ordinates (which range from 0-100) and device co-ordinates (in terms of pixels). Each level of routines scales the co-ordinates provided by the calling routine to its own co-ordinate system.

First level routines

First level routines provide complete graph and curve development capability for the application programmer.

GPLOTV: Draws a curve for y versus x for the data provided.
 GPLOTM: Draws multiple curves for y versus x for the data provided.

Second level routines

Second level routines provide the basic component routines used by the first level routines and also some basic graphics utilities for the application programmer.

GAREA: Defines display boundaries for the device and draws x and y axes and labels for the axes.
 GCVTRT: Converts a real number array of tic marks into a character array.
 GDRAW: Draws a curve according to the input data.
 GEND: Ends graphic operations.
 GFDGT: Computes the number of digits needed to display the scale on the axes.
 GFIND: Finds the minimum and maximum values in a vector.
 GHTEXT: Displays a character string horizontally starting from the point (x, y).
 GOUTLN: Draws the outline of the entire graphics area.
 GRANGE: Separates the differences between the maximum and minimum values on the axes into intervals for tic marks.

GVTEXT: Displays a character string vertically starting from point (x, y).

GXINDX: Draws the x-axis tic marks and labels them.

GYINDX: Draws the y-axis tic marks and labels them.

Device driver routines

The device driver routines provide the most basic routines specific to terminal types listed above.

GUERAS: Clears the graphics screen and returns the terminal to the text mode.

GUTEXT: Displays a character at a point.

GUMOVE: Moves the pen to a point.

GUPEN: Draws a line from the current pen position to a point.

GUSET: Sets parameters for character size, spacing and angle, and for line type.

GUSTRT: Initialises the graphics terminal.

GUWNO: Computes scale factors to convert from world co-ordinates to device co-ordinates.

7. DISCUSSION AND CONCLUSIONS

In this paper, capabilities of a library of subroutines, called SMART, for scientific program development are described. The library has routines for interactive program development, data entry and editing, in-core data management, out-of-core data management, vector and matrix operations for in-core or out-of-core data, linear algebra and graphics. The library has been used in the development of several structural analysis and design optimisation programs for educational and research purposes. It has considerably facilitated in their development and debugging.

The library has been carefully designed for efficient numerical operations. For example, most of the matrix operations, equation solvers and eigenvalue extractors use lower level kernals for various vector operations. The *kernals* are also written in FORTRAN77. They can be, however, written in the machine language to enhance efficiency. In addition, the subroutines are written in such a form that they can be easily vectorised automatically by compilers. Thus the library can be installed on vector computers without any additional effort. Current implementation is on PRIME supermini-computer and Apollo workstations.

Plans also exist for addition of subroutines for various finite elements and basic finite element operations. It is concluded that more SMART libraries are needed to support scientific software development in academic research as well as commercial environment.

ACKNOWLEDGEMENT

This work is based on a project sponsored by the Air Force Office of Scientific Research, No. AFOSR 82-0322.

REFERENCES

- 1 Comfort, D. L. and Erickson, W. J. RIM - A Prototype for a Relational Information Management System, *NASA Conference Publication 2055*, 1978, 183-196
- 2 Gill, P. E., Murray, W. and Saunders, M. A. Methods for computing and modifying the LDV factors of a matrix, *Math. Comput.* 1975, 29 (12), 1051-1077
- 3 *RIM User's Guide*. Boeing Commercial Airplane Company, PO Box 3707, Seattle, Washington 98124, 1982
- 4 Sreekanta Murthy, T., Shyy, Y.-K. and Arora, J. S. MIDAS: Management of Information for Design and Analysis of Systems, *Advances in Engineering Software*, 1986, 8 (3), 149-158

APPENDIX 6

**EVALUATION OF THE MIDAS DBMS IN AN
EQUATION-SOLVING ENVIRONMENT**

by

T. SreekantaMurthy, Y-K Shyy, S. Mukhopadhyay and J.S. Arora

in

Engineering with Computers

2, 1987

Evaluation of the MIDAS DBMS in an Equation-Solving Environment

T. Sreekanta Murthy, Y-K. Shyy, S. Mukhopadhyay, and J.S. Arora

Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, IA 52242

Abstract. The paper describes an evaluation of a data base management system, MIDAS, for the solution of linear equations. A brief description of the system is given. It is used and evaluated employing skyline and hypermatrix approaches for solving large matrix equations. Performance of the two subsystems of MIDAS—MIDAS/N and MIDAS/R—is measured and compared in terms of several system parameters. Programming for efficient use of the available memory is noted. Suggestions for application programming using MIDAS are given. Major conclusions of the study are that memory management schemes, data structures, and data access methods of the DBMS play very important roles for its efficient use in dynamic environment. Such methods must be developed and implemented for large-scale engineering applications.

1 Introduction

In the last few years, data base management systems (DBMS) have started to play an important role in the computer-aided design and analysis of engineering systems. Several data base management systems (DBMS) are available for engineering applications. Systems such as MIDAS [1], FILES [2], GIFTS [3], RIM [4], SDMS [5], SPAR [6], PHIDAS [7], and TORNADO [8] have been developed with a varying degree of sophistication and have a variety of capabilities. A survey [9] of existing data base management systems was made to find out their capabilities and usefulness for design and analysis applications. It was found that use of many of the systems is limited to those applications for which they were developed. Thus, there was a need to have a good DBMS that could deal with data organization of design and analysis applications. A data base management system called MIDAS [1] was designed and implemented based on data base man-

agement concepts [10] developed for engineering applications.

It is important to evaluate the capabilities and usefulness of a data base management system before implementing it in an engineering application. Such a system should meet several requirements [1] so that it can be used effectively. Speed of data storage and retrieval is one of the most important requirements given in Ref. 1. Short access time will considerably reduce total execution time in an iterative design process. Therefore, a data base management system should be evaluated to see if it satisfies the requirements of an engineering application.

In this paper, an evaluation of the data base management system MIDAS is given. The evaluation is made by using it in several programs for an out-of-core solution of linear equations. Such systems of equations are encountered in many engineering applications, such as static and dynamic analysis of structures and mechanical systems, heat transfer applications, fluid mechanics, eigenvalue analysis, and others. MIDAS has two subsystems that use different memory management, data access methods, and data models. The performance of the subsystems is measured by noting several parameters with a view toward evaluating memory management, data models, and access methods. The paper gives details of the performance and describes methods of developing applications by using MIDAS. It suggests efficient ways of integrating a DBMS in engineering applications.

2 Description of MIDAS

A detailed description of MIDAS is given in Refs. 1, 11, and 12. MIDAS stands for Management of Information for Design and Analysis of Systems. It consists of two subsystems MIDAS/N and

Reprint requests: J.S. Arora, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, IA 52242.

MIDAS/R. These subsystems are capable of organizing data by using numerical and relational models [10], respectively. MIDAS/N is specially developed to organize matrix data. This subsystem can handle multiple data bases, small and large matrices of various type, and small and large memory environments. Memory management is used in MIDAS/N to utilize efficiently the available computer memory, which is divided into a number of pages of fixed size. The number of pages and page size can be varied by changing a few parameters. The least recently used page replacement strategy is adopted in the memory management system.

MIDAS/R is based on modification and extension of the RIM^{4,13} system. This modification was made to see if the system can be extended to satisfy the requirements [1] of engineering applications. It was found difficult to extend RIM to have multiple data bases, organize large matrices, and be efficient in handling large data sets and large memory. It essentially meant rewriting the memory management, data definition, and data manipulation parts. Thus, it was decided to use RIM as is but to add new data definition, data manipulation, and memory management subroutines. This subsystem is called MIDAS/R, which stands for MIDAS-Relational Data Management System. The traditional relational model is extended in RIM so that its attributes can be variable length vectors and matrices. This is done to organize matrix-type data that is usually encountered in engineering applications.

Both subsystems of MIDAS provide simple-to-use data definition and data manipulation languages for application programmers and interactive users. Data definition language consists of subroutine CALL statements to define relations and matrices. Integer, real, double precision, and character data types are allowed. Large matrices such as square, rectangular, upper triangular, skyline, and hypermatrices can be defined. A dynamic data definition [1] is possible. Data manipulation language offers capability to store, retrieve, modify, and delete data in a data base. Both relation and matrix data can be manipulated. Using MIDAS/R, relations can be stored and retrieved in row order and matrices can be stored and retrieved in submatrix order. Using MIDAS/N, matrices can be stored and retrieved in row, column, or submatrix order.

3 Evaluation of MIDAS

The solution of a large number of simultaneous equations is one of the most common application

programs that can use a DBMS. The data base management system MIDAS is evaluated by using it in an out-of-core equation-solving application. The two subsystems—MIDAS/N and MIDAS/R—are separately used for solving equations. This enables us first to evaluate performance of the two subsystems separately and later to compare them with each other. This way the memory management schemes, data models, and access methods of the two subsystems can be evaluated. The approaches employed for solving equations and performance of the subsystems are given in the following subsections.

3.1 Skyline and Hypermatrix Approaches for Solving Equations

Sparsity in assembled matrix equations is used to advantage in reducing storage space and computation time. Of the three common approaches—banded, skyline [14], and hypermatrix [15] storage schemes, the later two are known to be more effective in conserving computer resources. Therefore, these approaches are selected for storing large matrices. Both approaches use a Gaussian elimination scheme to decompose the coefficient matrix. Forward and backward substitutions are carried out to obtain the solution of the equations. Thus, theoretically, the number of arithmetic operations is the same.

Computer programs for solving symmetric equations using skyline and hypermatrix approaches were developed. The skyline approach uses the left-hand side (LHS) coefficient matrix in skyline form. Each column of the LHS coefficient matrix is of variable length, storing only nonzero values. Each column, however, includes zero values within it. The right-hand side (RHS) and solution for unknowns are stored in a column vector. A numerical data model [10] consisting of two levels of matrix data organization is used with MIDAS/N. The first level contains details about skyline height and addresses of diagonal elements; the second level contains actual matrix data. A relational model is used to represent skyline data with MIDAS/R. A variable length vector is used as an attribute of a relation. Two separate programs were developed for solving equations, one using the MIDAS/N, and the other using the MIDAS/R subsystem.

The computer program for solving equations employing the hypermatrix approach uses the LHS coefficient matrix in partitioned form. A large matrix is divided into a number of submatrices. Only

Table 1. Performance of MIDAS/N

Page size	Number of pages	Half-bandwidth: 10				Half-bandwidth: 30			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	33 ^a	340	127	16262	125	1394	480	38688
		(248) ^b	(22698)	(1001)	(11487)	(486)	(39545)	(1326)	(22939)
1024	20	31	70	30	16262	115	337	120	38688
		(211)	(6013)	(339)	(11487)	(411)	(10370)	(435)	(22939)

^a Data without parentheses is for skyline approach.

^b Data in parentheses is for hypermatrix approach.

Number of equations: 1000.

the upper symmetric portion of the LHS matrix is stored. Similarly, the RHS matrix is also divided into a number of submatrices. The solution vectors are also stored in submatrix form. Null submatrices in the LHS and RHS are not stored or manipulated. Partially full submatrices, however, store zero coefficients within them. Again, a numerical data model is used for matrix data organization with MIDAS/N. The first level contains nonnull submatrix numbers and their sizes; the second level contains actual matrix data. The relational data model is used with MIDAS/R. Two more programs were developed; one using MIDAS/N, and the other using MIDAS/R.

Thus, in total, four computer programs are developed for solving equations:

1. SKYSOL solves the equations using MIDAS/R by the skyline approach.
2. SKYSOLB [12] solves the equations using MIDAS/N by the skyline approach.
3. HYP SOL uses MIDAS/R to solve equations by the hypermatrix approach.
4. HYPMDN uses MIDAS/N to solve equations by the hypermatrix approach.

Two sets of large equations were solved using the four computer programs SKYSOL, SKYSOLB, HYP SOL, and HYPMDN. The first set of equations has 1000 unknowns and a half-bandwidth of 10. The second set of equations has 1000 unknowns and a half-bandwidth of 30. For the hypermatrix approach, a submatrix size of 10 is used. These equations were solved with different memory management schemes. Performance of the programs was measured by noting the central processing unit time (CPU), the number of reads made on physical data base (NREAD), the number of writes made on physical data base (NWRITE), and the number of calls made to the data base management subroutines (NCALL). Detailed performance of the programs is discussed in the following sections. All cal-

culations were performed on a PRIME 750 computer.

The primary focus of the present investigation is on the evaluation of subsystems of MIDAS and not on the algorithms for the solution of equations. However, comparisons of the two algorithms on the selected set of systems of equations will be made.

3.2 Performance of MIDAS/N

The two computer programs SKYSOLB [12] and HYPMDN using MIDAS/N were used to solve the two sets of equations. The results of CPU, NREAD, NWRITE, and NCALL are given in Table 1. The equations are solved for two cases of page size and the number of pages as follows: (1) 20 pages of 256 short integer words and (2) 20 pages of 1024 short integer words.

The following points are observed from the table. As page size is increased from 256 words to 1024 words, the CPU time for solving equations using skyline and hypermatrix approaches reduces, but not to a considerable extent. The number of reads and writes for the larger page size is much less than that for the small page size. The number of writes is smaller than the number of reads, due to the replacement of unmodified pages by new data causing more reads than writes. Note that the number of calls to MIDAS/N routines remains the same for both page sizes, as expected. The number of calls to MIDAS/N increased as bandwidth increased.

The skyline approach for solving equations of 1000 unknowns with a half-bandwidth of 10 and a page size of 256 words took 33 s, whereas the hypermatrix approach with the same memory management scheme took 248 s. The hypermatrix approach is about 8 times slower than the skyline approach. In solving the second set of equations with a band-

Table 2. Performance of MIDAS/R

Page size	Number of pages	Half-bandwidth: 10				Half-bandwidth: 30			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	229 ^a	16103	218	12984	3139	65602	530	32594
		(2721) ^b	(92312)	(564)	(12082)	(8208)	(275060)	(1157)	(24117)
1024	20	119	13970	74	12984	1576	31417	191	32594
		(1623)	(50678)	(619)	(12082)	(3718)	(115788)	(1327)	(24117)
20480	2	51	303	65	12984	1298	27481	145	32594
		(928)	(26644)	(495)	(12082)	(1312)	(27497)	(2598)	(24117)

^a Data without parentheses is for skyline approach.

^b Data in parentheses is for hypermatrix approach.

Number of equations: 1000.

width of 30, the skyline approach used 125 s, whereas the hypermatrix approach used 486. In this case, the hypermatrix approach is about 4 times slower than the skyline approach. We see from the result that as the bandwidth increases, the difference in CPU time between skyline and hypermatrix approaches reduces, because the percentage of nonzero elements within the nonnull submatrices reduces considerably as the bandwidth increases. Arithmetic operations on nonzero elements reduces with the lower percentage of nonzero elements, thus reducing CPU time. It is believed that as the bandwidth increases and large submatrix size is used, the large difference in CPU times between the two approaches will reduce substantially. Note, however, from Table 1 that the data organization with hypermatrix approach is such that NREAD and NWRITE are quite large as compared to those with the skyline approach. It appears the hypermatrix approach will be less efficient as compared to the skyline approach. Definite conclusions, however, cannot be given because relative efficiency is a function of the programming habits and data organizations.

3.3 Performance of MIDAS/R

The two computer programs SKYSOL and HYP-SOL using MIDAS/R were used to solve the two sets of equations. A preliminary investigation showed RIM's memory management to be inefficient. This fact is also discussed in more detail in Section 3.6. Therefore, in the results reported here, the new memory management routines of MIDAS/R are employed. These routines reside a layer above RIM's memory management system and utilize its lower-level routines for physical read

and write operations. The new memory management system has its own buffer whose size, number of pages, and page size can be changed. The results of CPU, NREAD, NWRITE, and NCALL are given in Table 2. The table shows results for three different cases of page size and number of pages as follows: (1) 20 pages of 256 short integer words, (2) 20 pages of 1024 short integer words, and (3) 2 pages of 20480 short integer words.

The following points are observed from the table. As the page size is increased CPU time for solving equations using skyline and hypermatrix approaches reduces. The number of reads (NREAD) for large page size is less than that for the small page size. NWRITE also has a similar trend except for the case of hypermatrix approach with a page size of 1024 words, where it increases when page size is increased. The number of writes is much smaller than the number of reads. This may be due to the hashing scheme used in the MIDAS/R addressing and searching routines. The MIDAS/R program makes a large number of reads to search and locate the required data. Further, note that the number of calls to MIDAS/R routines remains the same for all page sizes, as expected. The number of calls to MIDAS/R increases as bandwidth is increased.

The skyline approach for solving the first set of equations with a page size of 256 words took 229 s, whereas the hypermatrix approach for solving the same equations with the same memory management scheme took 2721 s. The hypermatrix approach is about 10 times slower than the skyline approach. In solving the second set of equations with a bandwidth of 30, the skyline approach used 3139 CPU s, whereas the hypermatrix approach used 8208 CPU s. In this case, the hypermatrix approach is about 3 times slower than the skyline approach. Thus, we see from the results that as bandwidth is increased,

Table 3. Comparison of MIDAS/N and MIDAS/R using skyline approach

Page size	Number of pages	Half-bandwidth: 10 Skyline height: 10				Half-bandwidth: 30 Skyline height: 30			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	33 ^a (229) ^b	340 (16103)	127 (218)	16262 (12984)	125 (3139)	1394 (65602)	480 (530)	38688 (32594)
1024	20	31 (119)	70 (13970)	30 (74)	16262 (12984)	115 (1576)	337 (31417)	120 (191)	38688 (32594)

^a Data without parentheses is for MIDAS/N.

^b Data in parentheses is MIDAS/R.

Number of equations: 1000.

the difference in CPU time between skyline and hypermatrix approaches again reduces. The remarks given at the end of Section 3.3 regarding relative efficiency of the two approaches also apply to the preceding data.

3.4 Comparison of MIDAS/N and MIDAS/R Using Skyline Approach

The efficiency of MIDAS/N and MIDAS/R is compared in solving equations using the skyline approach. The two computer programs SKYSOL and SKYSOLB were employed to solve the two sets of equations. The comparison of results is given in Table 3. Equations are solved for two cases of page size and the number of pages as follows: (1) 20 pages of 256 words and (2) 20 pages of 1024 words.

The following points are observed from the table. MIDAS/N used 33 s of CPU time for solving 1000 equations with a half-bandwidth of 10 and a page size of 256 words, whereas MIDAS/R took 229 s for solving the same equations. Thus, we see that MIDAS/R is about 8 times slower in this case. For a page size of 1024 words, MIDAS/R is about 4 times slower than MIDAS/N. The number of writes in MIDAS/R is about 2 times that of MIDAS/N, and the number of reads in MIDAS/R is also very high as compared to MIDAS/N. We could attribute this high value of reads to the nature of addressing and searching employed in MIDAS/R. MIDAS/N uses an indexing (direct addressing) scheme to locate the data, whereas MIDAS/R uses a hashing scheme, which uses a large number of reads and searches to locate the required data. The number of calls to MIDAS/N, on the other hand, was higher than in MIDAS/R because MIDAS/N uses addresses of diagonal elements stored in a single row vector to locate the columns of the skyline matrix. MIDAS/N

is required to locate this address before accessing the skyline vector. MIDAS/R does not require such addresses, since skyline vectors are stored in variable length rows, thereby using a fewer number of calls to MIDAS/R. In the case of the second set of equations with a half-bandwidth of 30, similar trends of CPU, NREAD, NWRITE, and NCALL are observed.

Therefore, from the preceding results, we see that memory management, data model, and data access method in MIDAS/R are inefficient as compared to MIDAS/N.

3.5 Comparison of MIDAS/N and MIDAS/R Using Hypermatrix Approach

The efficiency of MIDAS/N and MIDAS/R is compared in solving equations using the hypermatrix approach. The two computer programs HYP SOL and HYPMDN were employed to solve the two sets of equations. The comparison of results is given in Table 4. Equations are solved for different cases of page size and number of pages.

The following points are observed from the table. MIDAS/N uses 248 s of CPU time to solve 1000 equations with a half-bandwidth of 10 and page size of 256 words, whereas MIDAS/R takes 2721 s of CPU time to solve the same set of equations. Thus, MIDAS/R is about 10 times slower than MIDAS/N in this case. For a page size of 1024 words, MIDAS/R is about 8 times slower than MIDAS/N. The number of reads in MIDAS/R is about 4 times more than in MIDAS/N for a page size of 256 words. On the other hand, the number of writes in MIDAS/R is about 2 times less than in MIDAS/N for the same size. For a page size of 1024 words, the number of reads in MIDAS/R is about 8 times those of MIDAS/N, whereas the number of writes in

Table 4. Comparison of MIDAS/N and MIDAS/R using hypermatrix approach

Page size	Number of pages	Half-bandwidth: 10				Half-bandwidth: 30			
		CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
256	20	248 ^a	22698	1001	11487	486	39545	1326	22939
		(2721) ^b	(92312)	(564)	(12082)	(8208)	(275060)	(1157)	(24117)
1024	20	211	6013	339	11487	411	10370	435	22939
		(1623)	(50678)	(619)	(12082)	(3718)	(115788)	(1327)	(24117)

^a Data without parentheses is for MIDAS/N.

^b Data in parentheses is MIDAS/R.

Number of equations: 1000; Submatrix size: 10*10.

Table 5. Performance of memory management (MM) of MIDAS/R

Total RIMs	Memory of built-in MM	CPU	With additional MM interface		CPU
			Page size	No. of pages	
1	9216	1597	256	20	229
			1024	20	120
			2048	20	82
2	20480	1543	256	20	228
			1024	20	116
			2048	20	79
3	81920	1514	256	20	222
			1024	20	112
			2048	20	75
4	163840	1608	256	20	218
			1024	20	116
			2048	20	81

Number of equations: 1000; Half-bandwidth: 10.

MIDAS/R is about 2 times those in MIDAS/N. The number of calls in MIDAS/R are slightly higher than in MIDAS/N. This is because MIDAS/R uses two call statements—RDSGET and RDSMOD—to modify data, whereas MIDAS/N uses only one call statement—RDSPUT—to modify data. Similar trends of CPU, NREAD, NWRITE, and NCALL are observed for the second set of equations.

Therefore, from these results we can again see that MIDAS/R is inefficient, as compared in MIDAS/N.

3.6 Performance of Memory Management of MIDAS/R

As mentioned earlier, the MIDAS/R program is based on modifications and extensions of the RIM program [4]. A new memory management interface

was added to provide improvements in performance of the RIM program. Here a comparison of the performance of RIM's built-in memory management system and the performance of MIDAS/R with the additional memory management is made. The CPU times to solve 1000 equations with a SYKSOL program are used to compare the performance. Table 5 shows the CPU time for various sizes of RIMs built-in memory management schemes and additional memory management schemes. Observe from the table that the CPU times does not vary considerably by increasing the total memory of the built-in memory management scheme of RIM. This indicates that the RIM program is not capable of utilizing the large memory of the computer effectively. By introducing the additional memory management scheme in MIDAS/R, we see that CPU time reduces considerably. As page size increases from 256 to 2048 words, CPU time reduces from 228 to 79 s for Case 2 of 20480 words of memory. Similar trends are observed for other sizes of memory management buffer. Therefore, we see that memory management plays a very important role in the dynamic environment of engineering applications. Also as observed in Table 5, memory management of the RIM program can be substantially improved.

4 Programming for Efficient Use of Available Memory

The computer programs for solving equations described in Section 3 use only a minimum of workspace, which is just sufficient to hold the needed columns of the skyline matrix or the submatrices in the hypermatrix approach. In such a case, the efficient use of available memory depends completely on the memory management of the DBMS. However, every call to DBMS is associated with certain

Table 6. Performance of MIDAS/N with additional workspace using skyline approach

Workspace REAL*8	Page size: 256				Page size: 1024			
	CPU	NREAD	NWRITE	NCALL	CPU	NREAD	NWRITE	NCALL
25	35.5	340	127	16270	33.3	70	30	16270
25*	(33.1)	(340)	(127)	(16262)	(30.7)	(70)	(30)	(16262)
100	17.6	340	127	5203	15.1	70	30	5203
200	7.7	339	127	843	6.3	70	30	843
800	5.8	340	127	167	4.8	70	30	167
1000	5.7	340	128	134	4.7	70	30	134
4000	5.4	364	136	34	4.5	70	31	34
10000	5.3	342	130	12	4.3	61	33	12

* With SKYSOLB using the minimum workspace of 25 REAL*8 words.

Number of equations: 1000; Half-bandwidth: 10; Skyline height: 10; Number of pages: 20.

overhead which increases as the number of calls to DBMS increases, leading to increased execution time. Therefore, alternative schemes are necessary to improve the efficiency of application programs that use a DBMS.

One possible way to make efficient use of available memory is by providing a sufficiently large workspace in the application program itself. The application programs should be developed with additional logic to use the workspace as much as possible before doing any transaction on the data base using the DBMS. This way, it is possible to reduce the number of calls to the DBMS, which in turn reduces the I/O and CPU times.

Using the approach described previously, another equation solver called SKYSOLA [12] was developed employing MIDAS/N. Available workspace, which is larger than the minimum required, is assigned to various subroutines of SKYSOLA. Additional program logic is incorporated to make use of the workspace in the best possible manner. The program is used for solving equations of 1000 unknowns and a half-bandwidth of 10. The performance of the system is measured for several sizes of workspace. The results are given in Table 6. Note that the data manipulation language of MIDAS/R is such that a program similar to SKYSOLA cannot be developed with it. MIDAS/R must read one record of a relation at a time, so there is no advantage in providing extra workspace in the application program.

The following points are observed from the table. For a minimum work space of 25, the program SKYSOLA uses more CPU than SKYSOLB for a page size of 256 words. This is because additional logic in SKYSOLA uses some CPU time. As the

workspace is increased from 25 to 10000 words, CPU reduces from 35.5 s to 5.3 s. Also, the number of calls to MIDAS/N reduces from 16270 to 12. This reduction in CPU time and number of calls to MIDAS/N is quite significant. Note that the increase in the workspace beyond 800 words does not reduce the CPU time significantly. It remains almost constant at around 5 s. Similar results are obtained for a page size of 1024 words. Note that the number of reads and writes for a page of 1024 words is much less than those for a page size of 256 words. Also CPU time is less for a page size of 1024 words. From these observations, we can conclude that by using workspace efficiently in the application program, CPU time can be reduced. However, we also note that beyond a certain amount of workspace, there may not be further significant reduction in the CPU time.

5 Suggestions for Application Programming Using MIDAS

It is useful for an engineering application programmer to be familiar with the internal structure of the DBMS. In particular, the programmer should be aware of the memory management scheme used and understand the effect of changing system parameters. The choice of a page size and the number of pages can have an effect on the efficiency of an application. Several points are noted that will permit the use of MIDAS and other such systems effectively:

1. A larger page size is suitable when data is accessed sequentially. It is not suitable when data is accessed in a random manner. In the

latter case, the entire data from a page may not be used before replacing it, or before going to the next page. Also, a larger page size may lead to internal fragmentation of data, if page management is not properly designed.

2. A large number of pages is beneficial to a program using several data sets simultaneously or when data is retrieved randomly.
3. If the number of pages is too small compared to the number of data sets currently used, then page replacement activity may increase (thrashing).
4. For better efficiency, the page size should be the same as that of the operating system (i.e., the unit of transfer). The reason is that if the page size is smaller than the operating system's page size, then each call to read the data may cause the operating system to transfer more data into the memory. A larger page size should be chosen as a multiple of the operating system's page size.
5. Use as much workspace as possible in the application program to reduce the CPU time. Also, minimize the number of calls to the DBMS. A general-purpose DBMS has a certain amount of overhead that can be avoided by efficiently utilizing the workspace available in the application program. The DBMS is still essential for organizing a large amount of data in complex applications. The use of extra workspace in an application program, however, complicates its logic to manage it. Also, the application program may become difficult to maintain.

6 Discussion and Conclusions

The data base management system MIDAS is evaluated by organizing the data for and solving sets of linear equations. Its subsystems, MIDAS/N and MIDAS/R, are used with skyline and hypermatrix approaches. The following conclusions are drawn from the evaluation:

1. MIDAS/N is more efficient than MIDAS/R in solving large sets of equations. This is primarily due to the data models, data access methods, and memory management schemes in the two subsystems. The relational data model used in MIDAS/R is inappropriate for representing large matrix data in a runtime environment.
2. Memory management and data access methods of the DBMS play an important role in

engineering applications. Proper methods for these steps must be developed and implemented for efficient use of a DBMS in the dynamic environment.

3. Memory management of MIDAS/N can effectively use large memory. Further improvements are possible by eliminating sequential search used at several places.
4. Memory management of MIDAS/R (RIM [4]) is inefficient for engineering applications. It can be substantially improved.
5. Application programs should be carefully designed to minimize the number of calls to the DBMS by properly using the available workspace. This makes the application program slightly more complex.

The present study has helped in refining the requirements of a DBMS that has to be used in the dynamic runtime environment. Many complex engineering applications fall into this category. The refined specifications for the system are given in Ref. 16. Enhanced data definition facility, memory management scheme, and storage layout are proposed. A unified data definition facility for matrices and relations, while retaining their distinct features, is developed. The memory management scheme is designed to handle large as well as small buffers efficiently. The memory and physical storage is organized in terms of pages. A page can contain data from one or more data sets. This scheme essentially rules out the possibility of internal fragmentation in large pages. These enhancements will be implemented into MIDAS and evaluated in the future.

Acknowledgment

Research is sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR 82-0322. The U.S. government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright notation thereon.

References

1. Sreekanta Murthy, T.; Shyy, Y-K.; Arora, J.S. (1986) MIDAS: Management of information for design and analysis of systems. *J. Adv. Eng. Soft.* 8(3), 149-158
2. Lopez, L.A. (1974) FILES: Automated engineering data management system. *J. of Str. Div., Am. Soc. of Civ. Eng.*, Vol. 101, No. ST4, April, pp. 661-676
3. Kamel, H.A.; McCabe, M.W.; Spector, W.W. (1979) GIFTS 5 System Manual, University of Arizona, Tucson
4. RIM User's Guide. (1982) Boeing Commercial Airplane Company, P.O. Box 3707, Seattle, WA

5. Massena, W.A. (1978) SDMS—A scientific data management system. NASA Conference Publications 2055
6. Giles, G.L.; Haftka, R.T. (1978) SPAR data handling utilities. NASA Technical Memorandum 78701
7. Fischer, W.E. (1979) PHIDAS—A database management system for CAD/CAM Software Comput.-Aided Des. 11(3), 146-150
8. Ulfsby, S.; Stiener, S.; Oian, J. (1979) TORNADO: A DBM for CAD/CAM systems. Comput.-Aid. Des. 11, 193-197
9. Sreekanta Murthy, T.; Arora, J.S. (1985) A survey of database management in engineering. J. Adv. Eng. Soft. 7(3), 126-132
10. Sreekanta Murthy, T.; Arora, J.S. (1986) Database management concepts in computer-aided design optimization. J. Adv. Eng. Soft. 8(2), 88-97
11. Sreekanta Murthy, T.; Shyy, Y.-K.; Arora, J.S. (1985) MIDAS: Management of information for design and analysis of systems. In: Proceedings of the 26th American Institute of Aeronautics and Astronautics—Structures, Structural Dynamics, and Materials Conference, pp. 85-95. Orlando, FL
12. Shyy, Y.-K.; Mukhopadhyay, S.; Arora, J.S. (1985) A Database Management System for Engineering Applications. Technical Report No. ODL-85.23, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, June
13. Comfort, D.L.; Erickson, W.J. (1978) RIM—A prototype for a relational information management system. NASA Conference Publication 2055, pp. 183-196
14. Bathe, K.-J.; Wilson, E.L. (1976) Numerical Methods in Finite Element Analysis. Englewood Cliffs, NJ; Prentice-Hall
15. von Fuches, G.; Roy, J.R.; Schrem, R. (1972) Hypermatrix solution of large sets of symmetric positive-definite linear equations. Comput. Meth. Appl. Mech. Eng. 1, 197-216
16. Arora, J.S.; Mukhopadhyay, S. (1984) Specification for MIDAS-GR Management of Information for Design and Analysis of Systems: Generalized Relational Model. Technical Report No. CAD-SS-84.24, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, December

APPENDIX 7

**USES OF ARTIFICIAL INTELLIGENCE IN
DESIGN OPTIMIZATION**

by

J.S. Arora and G. Baenziger

in

Computer Methods in Applied Mechanics and Engineering

54, 1986

USES OF ARTIFICIAL INTELLIGENCE IN DESIGN OPTIMIZATION*

Jasbir S. ARORA and G. BAENZIGER

*Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242,
U.S.A.*

Received 5 March 1985

Revised manuscript received 15 July 1985

In this paper, basic ideas and concepts of using artificial intelligence in design optimization of engineering systems are presented. The purpose of the study is to develop an expert (knowledge-based) system that helps the user in design optimization. Two basic ideas are advocated: (1) the successful numerical implementation of algorithms needs heuristics; and (2) the optimal design process can be greatly benefited by the use of heuristics based on knowledge captured during the iterative process. Various steps in the optimization process, where artificial intelligence ideas can be of tremendous help, are delineated. Some simple rules are presented to utilize the knowledge base and raw data as it accumulates in the iterative process. A simple example is used to demonstrate some of the basic ideas.

1. Introduction

Artificial intelligence (AI) is the science of creating intelligent behavior on computers. Computer programs having built-in intelligence are called expert or knowledge-based systems. They perform a task normally done by experts or specialists in a field, and in doing so they use captured, heuristic knowledge. General introduction to the field of artificial intelligence can be found in articles by Waldrop [1-3]. An excellent introduction to the subject from the engineering point of view is presented by Dym [4]. In-depth discussion of artificial intelligence and expert systems can be found in a book by Rich [5] and books edited by Hayes-Roth, Waterman, and Lenat [6], and Barr and Feigenbaum [7].

The principles of AI can be applied to almost any field. Expert systems have been developed [4] in the fields of medicine for diagnosis, finite element analysis as a modeling consultant, locomotive repairs, hydrology for estimating input parameters for watershed simulation, seismic-damage assessment, design of high-rise buildings, VLSI design in electrical engineering and others. Use of AI is particularly suitable in the field of engineering design, where rules of thumb (heuristics) have been developed over the years through successful design and redesign of similar systems.

*This research is sponsored by the Air Force Office of Scientific Research, Grant No. AFOSR 82-0322. The material of the paper is derived from a presentation made by the authors at the 26th AIAA Structures, Structural Dynamics and Materials Conference, April 1985.

The AI concepts can also be used in optimal design of systems. So far, methods of optimization and associated programs have been used mostly by the experts in the field. The limited use of the methodology by general designers has resulted in failure more often than in success. We need to examine why the experts have been able to apply techniques to almost any design problem but the general designers have not. The main reason is that experts can observe the behavior of the algorithm, diagnose failures and erratic behavior, and fix them by adjusting certain parameters. The question then is: can we capture the experts' knowledge, develop appropriate rules and implement them in an expert system? The answer is yes. The optimal design process is iterative where the same set of calculations is performed repeatedly. Data from each iteration can be saved to provide knowledge about the problem helpful in finding an optimal design. We will use the data that the experts use to monitor the progress of the algorithm and develop rules required to adjust parameters.

Another reason for using AI in optimal design is that no one algorithm can efficiently solve all classes of problems. We can monitor characteristics of the optimization problem and develop rules to switch back and forth between algorithms. Even if an algorithm is theoretically guaranteed to solve a class of problems, its numerical implementation needs heuristics (due to round-off, truncation, etc.) to achieve convergence.

The purpose of the paper is manifold: (1) to elaborate on why AI techniques are needed in the design-optimization process; (2) to describe how AI techniques can be used; (3) to discuss some general concepts about the architecture of the expert system (the objective would be to lay out a plan for development of such a system); and finally, (4) to exemplify the ideas through a primitive expert system and experience with it.

2. The expert system: an overview

The development of more powerful software in engineering has been accompanied by an increasing complexity and detail in program control structures. Increasingly, the decision-making process has been incorporated into programs to help the user operate the software.

It is only with the advent of AI that the study of human intelligence has been approached with the intent of incorporating aspects of that intelligence in engineering programs. Our goal is to apply knowledge of optimization towards the improvement of the design process. We must not only incorporate that knowledge, but also knowledge about intelligence itself (how we solve problems, operate the process, and make decisions).

Considerable effort is required in the planning stages of development to avoid problems later. Consideration is given to the scope of the problem modeled, the type of knowledge involved, the form in which the knowledge is represented, and the type of inference engine used.

Expert systems are designed to deal with a particular application, capturing an expert's knowledge and implementing rules based on it. Since expertise comes in narrow fields, an expert system can be developed only for a particular application. The scope is limited, in the present case, to the control of nonlinear optimization algorithms. By limiting the scope, the knowledge base is maintained at a workable size, the decision process is faster, and the form of the knowledge is reduced to a few specific representations.

The rules developed and used by an expert can be of two types: (1) based on a well-defined

algorithm and easily coded in conventional programs (algorithmic rules); and (2) based on experience and heuristics but not part of a conventional algorithm (nonalgorithmic rules). It turns out that both types of rules can be used in an expert system for general design optimization.

2.1. Components of knowledge-based systems

An expert system is likely to have the following four components: (1) input/output facility; (2) knowledge acquisition facility; (3) knowledge base; and (4) inference engine. The input/output facility can include interactive queries, reports, and graphics. We will see later in an example how this is accomplished. The knowledge acquisition essentially consists of identifying the data that need to be collected and converted to a usable form. The process of knowledge acquisition may be broken down into phases [4]: (1) identification; (2) conceptualization; (3) formalization; (4) implementation; and (5) testing.

The process of developing an expert system is, to a large extent, the development of an applied knowledge base in a form which may be used efficiently by an inference engine. The knowledge base contains the knowledge of an expert about control of optimization algorithms. Our application, a numerically intensive design process, deals with aspects of optimization such as numerical precision, rates of convergence, recognition of convergence, active sets, feasible and infeasible designs, and constraint violations. The numerical results of an algorithm include new design points, function and gradient values, directions, and step sizes from successive iterations.

Two types of knowledge bases can be used in the design-optimization process: (1) an existing knowledge base about a class of design problems which is accumulated over a period of time, e.g., for design of aircraft structures, bridges, roads, etc.; and (2) a generated knowledge base that is accumulated for a particular design problem during the iterative process. Primarily, the second type is considered here, although both types of knowledge can be useful for the expert system.

The inference engine is at the heart of the system and is based upon new techniques developed in the artificial intelligence field. These techniques are rapidly evolving into a coherent set of search and deduction strategies.

2.2. Domain of application

To develop an expert system we must precisely define its domain of application. In this paper an expert system for the following general nonlinear programming problem is discussed:

$$\begin{aligned}
 (P) \quad & \text{minimize} \quad f(\mathbf{b}), \quad \mathbf{b} \in \mathbb{R}^k, \\
 & \text{subject to} \quad g_i(\mathbf{b}) = 0 \quad i = 1, \dots, n, \\
 & \quad \quad \quad g_i(\mathbf{b}) \leq 0, \quad i = n + 1, \dots, m.
 \end{aligned}$$

Here \mathbf{b} is a design variable vector, $f(\mathbf{b})$ is a cost function, $g_i(\mathbf{b})$ are constraints, n is the number of equalities, and m is the total number of constraints. Many design-optimization problems represented in the above model possess the following special characteristics which influence development of an expert system:

- (1) Most functions in the problem P implicitly depend on design variables b . The function evaluations are tedious and time-consuming.
- (2) Evaluation of gradients of such functions needs special algorithms.
- (3) Function and gradient evaluation need special-purpose programs that can be quite complicated and large.
- (4) During each design-optimization cycle, ninety percent of the computational effort is spent in function and gradient evaluation. Therefore, the information generated during each iteration should be utilized to the maximum extent possible. One of the objectives of the expert system will be to accomplish this task.

3. Need for AI in design optimization

There are many facets of optimization which might incorporate the strategies of an expert system. Many of the processes already include some measure of the expert's knowledge, but the complexity of the software does not allow the level of application possible. We will discuss some instances in the design-optimization process where artificial intelligence is needed.

3.1. Optimization algorithms

Many optimization algorithms have been developed [8–14] to solve the problem P . Among these, the recursive quadratic programming method of Pshenichny [8, 9, 12], the cost function bounding method of Arora [13], feasible directions methods [9, 10], a hybrid method [14], and multiplier methods [9, 11] appear to be quite promising. All methods have certain common calculations during each iteration. They all need: (1) cost and constraint function values; (2) gradients of cost and constraint functions; (3) solution of a subproblem (although each method defines it differently); (4) step-size determination after a direction of design change has been determined; and (5) some mechanism to enforce global convergence.

Of the five methods listed above, the first four fall into the class of primal methods and the multiplier methods fall into the class of transformation methods. Some methods, like the recursive quadratic programming method [8, 9, 12] and the hybrid method [14], accumulate second-order information about the problem. This can lead to superlinear convergence. Careful implementation of the methods is needed, however. Each method has its own peculiarities and needs its own rules. All of the above methods use active-set strategies implying that only a subset of the constraints is used in defining the subproblem at each iteration. If the active set is not properly identified, convergence difficulties can occur. Both active set identification and convergence problems can be dealt with in the expert system.

3.2. Numerical problems with algorithms

If numerical algorithms are not carefully implemented in computer software they may not behave the way they are theoretically supposed to. One reason for this is the occurrence of truncation and round-off errors. In addition, many algorithms are proved to converge when certain parameters tend to zero or infinity. The rate at which the parameters go to their limiting value can influence the numerical behavior of the algorithm. To avoid division by zero the

use of range parameters must always be considered. Optimization algorithms are not different from any other numerical algorithm and encounter the above-mentioned difficulties. Thus, heuristics can be used to accomplish robust implementation of algorithms.

3.3. Providing expert performance for the designer

The level of knowledge required to successfully use optimization software has limited its use to the experts themselves. The average designer is unable to grasp the heuristic insight into the problem solution. Attempts made by the designer to duplicate the efforts of the expert in using the software can result in frustration. Effective use of the software by nonexperts requires that many of the details of the solution process be incorporated into the software and removed from the designer's responsibilities. This requires software capable of setting parameters, making decisions, and recognizing critical conditions in the process. The software must act as a consultant to the designer and as a control system for the optimization algorithms.

Several instances where the software system can provide expert performance for the designer are described in the following:

(1) The system can identify discrepancies in the gradient calculation of cost and constraint functions and report the findings to the designer so that he can fix them.

(2) It is possible that no feasible design exists for the problem. The system can identify flaws in the problem formulation by identifying the constraints that cannot be satisfied.

(3) The system can recognize characteristics of the problem and select appropriate algorithms to solve it. For example, it could classify *the* problems as (i) linear, (ii) unconstrained, (iii) mildly nonlinear, and (iv) highly nonlinear. This can be accomplished by capturing appropriate knowledge and developing rules to use it. Proper algorithms should be available to deal with each class of problems.

(4) Another area where the system can provide expert performance is the identification of design variables that have the most or the least influence on the design process. This can be done by comparing the sensitivity coefficients of all functions with respect to various design variables. This information can be of extreme importance to the designer. He can give 'fixed-design' status to variables that have the least influence, and optimize the ones that are most critical. Overall efficiency of the optimal design process can be improved.

3.4. Making the optimization process more efficient

The optimization process requires many finely tuned components to successfully find solutions to large or complex nonlinear programming problems. Even with the parameters and controls finely tuned, the process may still take considerable time and computational power to find a solution.

The expert is often able to see, after some iterations, the ways in which the algorithm has been inefficient in its search for the solution. The patterns of data available during the search can be used to help guide the optimization process in a direction which would lead to the solution much faster than if unaided.

The selection of the starting point may be improved by preliminary analysis of the problem, use of gradients of functions and experience with the optimization algorithms available. If an algorithm performs better with a feasible or infeasible starting point, then selection of the

starting point should reflect that fact. Often, the algorithm converges very slowly from a specific region in the design space. It may be better in those cases to select a new starting point than to continue with the computations in progress.

Knowledge of the design space and the functions involved can be incorporated into the process and used to aid in the solution search. Some regions of the solution space may be undefined. Some may be restricted due to bounds on the solution space which are used to limit the solution to physical reality. These limitations can be recognized by the system and efficiently dealt with. In addition, algorithms may be more effective in some circumstances than in others. An algorithm which has been used up to the current design point may no longer be as effective as another algorithm available in the system.

Another way to improve efficiency of the iterative process is to use the trend information to adjust parameters. For example, the expert system can observe trends in each design variable and advise appropriate actions. Some design variables may have changed very little for this last few iterations. They may be given 'fixed-design' status. Others may be increasing or decreasing continuously. The expert system can extrapolate to predict their optimum values.

Certain strategies incorporated in the nonlinear programming algorithms have a very sensitive nature. For instance, to help limit the volume of computations, not all constraints are considered at every iteration. It is a matter of judgement as to when a constraint should be considered and when it need not be considered. This has been accomplished in the past by collecting the set of active constraints based on a parameter, ϵ , indicating how close the current design point is to violating a particular constraint. The method achieves the desired goal but is inefficient due to its inflexibility, as functions are often added and removed in successive iterations. This fact can be recognized and used to keep particular constraints in the active set during successive iterations to calculate better search directions.

3.5. Solving difficult problems

Convergence in nonlinear optimization is often difficult to achieve. Round-off errors and other numerical pitfalls in the iterative processes can occur and limit the success of the search. These effects may, in many ways, be reduced by the design of the optimization algorithm itself, but cannot be eliminated entirely. The type of behavior characteristic of these effects can be monitored and dealt with during the optimization process.

Different algorithms can be used at different stages of the search to expedite convergence. For instance, an approximate method which requires fewer computations can be used initially to speed up the search. Later, when convergence becomes a problem with the approximate method, a more exact method can be used.

4. Architecture of the expert system

Once the type and form of the knowledge have been catalogued, consideration must be given to the means to manipulate, interpret, and evaluate it. The system must have an input/output facility capable of dealing with input from both the user and the expert, tracing the behavior and decisions of the system during testing by the knowledge engineer, and printing, permanently or temporarily, the intermediate results, key decisions, conclusions, and output data for the user. The system must have recyclable logical relations which it uses to

operate inferential strategies. The system should provide consulting support to the user like an expert in the field would. For example, the system may respond to failure of an algorithm by listing various options for the user to try, just like the expert would.

A basic purpose of the expert system would be to solve the optimization problem P as efficiently as possible, and in so doing guarantee an answer which can be any of the following:

- (1) An optimal solution exists and the system guarantees to find it.
- (2) A solution does not exist and system reports this fact.
- (3) An optimum exists but the system cannot find it due to errors in the input data, problem formulation, or other parameters. It simply reports the best solution for a given formulation.

In the following paragraphs, we will first discuss general facilities and features needed of an expert system. Then we will define a general architecture for the system.

4.1. Interface with the optimization algorithms

The algorithms involved in engineering related nonlinear optimization are modular in nature and in operation. Usually they are based on iteration after iteration of the same computation-intensive search, where a single direction and step size are selected. Certain information is involved in each iteration. After determining a new design point, the iteration is completed and control is returned to the main program. Certain information computed during the iteration is saved as a history of computations and is available to evaluate the condition of the problem solution. Part of the knowledge base must be devoted to the interpretation of the data and to the translation of the patterns into conditions that the control portion of the system uses to make decisions. Thus, two aspects of the problem have been identified; pattern matching of the gross data to control conditions and decision making (selection of control parameters) based on those conditions. Both aspects require heuristic knowledge, but there are enough differences in the tasks to consider separating not only the processes but also the heuristic knowledge. This may be an advantage in more ways than expected. The advantages include: decrease in the size of the knowledge base searched and consequently the decrease in the overall time required; an organizational simplification in the knowledge collection phase; and the possibility that the processing of the gross data can occur as the data are produced, not waiting for the algorithm computations to be completed.

4.2. Default parameters

The system developed must have default parameters for several reasons. It is possible that, for a given problem, many detectable conditions may be observed simultaneously or that none may occur at all. The system initially cannot be aware of the behavior of the solution search until the functions and gradients have been computed for one or two iterations. The default algorithm selection should be based on initial information given and any previous experience with problem solutions of a similar nature. For flexibility the defaults must also allow user input. This would allow a knowledgeable user to experiment with the initial phases of the solution search.

4.3. A rule-based system

What is a rule-based system?

A rule-based system is one that deals with simple logical relationships between data and between rules to extend the available knowledge to determine knowledge that is not available.

The logical relations generally deal with the existence or truth of a condition in its domain and represent the heuristic information provided by the expert.

Why a rule-based system?

The knowledge of engineering problem solving and engineering problems are appropriately modeled by the simple logical relationships of the rule-based system. These problems involve primarily numerical manipulation and not creative development. The rule base in this type of problem solving is less extensive and better defined than in the creative design of the architect or artist and does not tend to be as cumbersome [15]. Specific algorithms, solution techniques, and parameters may be decided precisely so that specific conditions or behaviors of the problem solution may be dealt with. The process to determine the specific conditions, however, is somewhat different. The conditions result from different sources, some of which require processing large quantities of information. Observing the behavior of the problem requires a pattern-matching search rather than a decision-making search. The difference is subtle but the orientation of the search may be quite different. The logical decision-making process is more suited to a forward chaining or data driven process, whereas the pattern-matching process, with large volumes of data, is more suited to a goal-oriented backward chaining search. The larger extent of search required in the latter case and the numerical nature of the data involved cause some problems for the rule-based system and the data representation.

Critical issues for the rule-based system

The rules must be flexible enough to deal with quantitative relationships as well as Boolean ones. Tests as to whether a specific rule is applicable or true may require that numerical equalities and inequalities be tested.

The rule-based system must also be able to deal with incomplete information or an unknown evaluation. There may be insufficient information to determine a condition, yet this does not mean the condition does not exist. The continuing iterative process also requires the ability to negate conditions that were previously affirmed because the conditions of the solution search may be transitory.

Other types of systems

Other types of systems are available as inference engines for expert systems. They may incorporate syntactic, frame, hierarchical, or other organizations and processes. They are specially suited to only certain applications and disciplines. To the extent that these forms of organization are suitable to the engineering problem solving at hand, they may be incorporated with minor variations in the rule-oriented system, though at some loss of efficiency.

4.4. Pattern matching

Gross data interpretation is a pattern-matching process which must be incorporated into the optimization control program. For large multivariable nonlinear optimization problems the identification of the behavior and conditions occurring in the problem solution are obscured by the quantity of interpretable information. This process may be a continuous one that is performed in parallel with the actual optimization algorithm chosen. Certain discontinuities in

the algorithms used, in the search behavior observed, or in the solution space itself make the pattern-matching task more difficult and perhaps place the greatest limitation on the form of the system.

4.5. Dealing with incorrect and probabilistic information

The success of the solution search is occasionally impeded by insufficient or incorrect information. The lack of sufficient data is usually obvious and may be dealt with early by halting the program or by using defaults. Incorrect information, however, is often obscure and may never be discovered. The system requires decisions based on correct information to arrive at a solution efficiently. In the problem at hand, only the original problem conditions and occasional intermediate parameters are provided by the user. The algorithms themselves are often able to determine errors in the input data. In addition, the system developed must attempt to substantiate other aspects of the user input and internal decision-process conditions.

Certain aspects of the solution search are probabilistic. The probability that a combination of rules determines a condition or selects a parameter which is appropriate depends on the certainty of the heuristic knowledge, predictability of the solution space or judgement about the parameters involved. Selection of starting points falls in this category as does algorithm selection, convergence, and other aspects. These may be dealt with, in part, by numerical relationships. The redeeming aspect of our application is its iterativeness and regenerativeness. It is possible to test the decisions made and return to choose a new or modified decision. We can backtrack as required when performance does not meet our expectations. This ability does not eliminate the need to consider the certainty of the decisions. In large problems, backtracking might involve significant computational waste.

4.6. Transparency and debugging

There are some opponents to the inclusion of expertise in software. The primary reasons are that there is no way of determining whether the software includes appropriate principles, whether the application of the knowledge is appropriate, and whether the software is computationally and operationally correct. This is a significant problem for the computer scientist and expert. It is dealt with, to some extent, by the improvement of software documentation, by writing a readable software code, by reporting an audit trail, and also by doing extensive testing and debugging during development. There is no way to make software perfect, but it is possible to relieve many of the apprehensions involved by making the processes transparent. By providing run-time explanation and tracing of what is happening, what processes are being used, and what knowledge has been applied to make indicated decisions, the developer may document the software's computations, inform the user of what is going on, and perhaps even provide him with an education in the particular field of expertise.

The types of information to be retained and presented depend on the user of the system and should be separated into various aspects of the process. One aspect is decision making, which, due to the logic-oriented rule base, may provide step-by-step relationships between the conditions observed and the decisions made. Another aspect is the observed behavior or conditions themselves, which may be explained by the specific data or trend information used and retained in

the history of the computations. Yet another aspect is the algorithm itself and its principles and computations. The development of traces for these aspects for explanation to the user also aids in the process of debugging. The logical errors are more obvious if the information is provided in the less cryptic form required of such process traces.

4.7. Concurrent programming and decision making

Certain aspects of the expert system may be operated in parallel. While the algorithm is being performed on the data, available data can be processed in the pattern-matching and decision process. Certain decisions must be left until the output data is complete, such as algorithm selection or decisions involving the design point and the step determined in the latest iteration. The real-time performance can be quicker if parallel computations can be made. This does not mean the computer power expended will be less. That depends more on the degree of additional efficiency provided by the system's heuristic ability. The efficiency of the process may not always be of paramount importance. Our goal, in addition to making the optimization process more efficient, is to make it easy to find solutions to problems which were difficult to solve without the more sophisticated control system.

4.8. The system architecture

The expert system must be organized in a modular form. A possible organization of the system is shown in Figs. 1 and 2. Fig. 1 shows an overall organization of the expert system. It has a module that controls the use of various algorithms. It has an interaction facility as well as an output facility. Its inference engine is simple, using rules of the expert and the knowledge base accumulated during the optimization process to decide which algorithm to use. Fig. 2 shows an overall organization for the implementation of an algorithm. Various components needed to carry out functions of the algorithm are shown as (1) input/output facility; (2) access to user-provided subroutines or programs defining the optimization problem (function and gradient evaluation facility); (3) access to a sophisticated data base management system; (4) interactive graphics and query facility; (5) access to knowledge and data bases; (6) knowledge acquisition facility; (7) inference engine for the algorithm; and (8) rules for intelligent implementation.

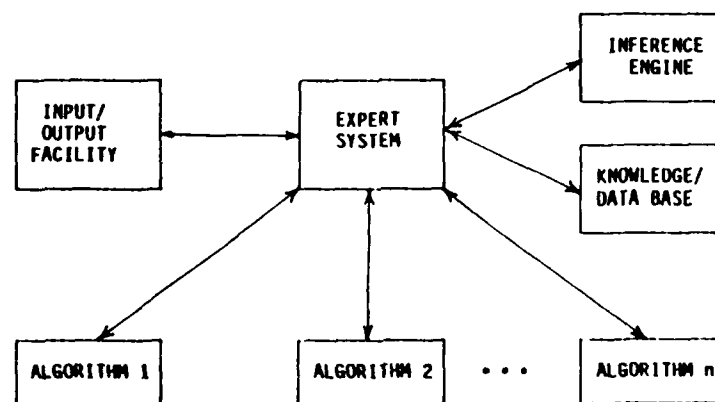


Fig. 1. Overall organization of the expert system for design optimization.

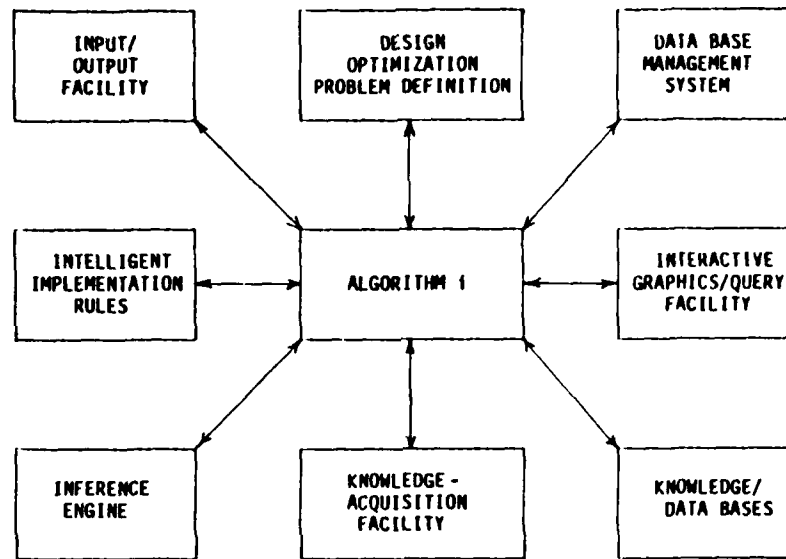


Fig. 2. Organization for implementation of Algorithm *i*.

5. Incorporating expert knowledge into the system

5.1. Form of the knowledge

The form of the knowledge inherent to our problem is a mathematically oriented rule structure. The mathematical nature of engineering problem solving makes the inclusion of numerical relationships crucial to the application of rules. It makes the rule base much more flexible but also makes the mechanics of the system much more complex. The concept requires the development of variable data identities and values. This implies the development of an inference engine that acts also as a compiler or an interpreter. Variables are not useful unless accompanied by relationships (e.g. $>$, $<$, $=$, etc.) and further parameters to specify specific data from data sets. The iterative nature of optimization search requires that data be identified with a specific iteration or algorithm. The same identification applies to the association with constraints and design variables.

5.2. Sample optimization rules

In an attempt to illustrate some of the criteria to be dealt with, a small sample of the rules envisioned are included below. The application to which they relate is small compared to the entire problem at hand so one can see the tremendous task involved in planning and incorporating the knowledge for the entire system. A few simple variables are abbreviated for simplification. Many of the variables require further parameters to identify the specific datum to which they refer. Algorithm data variables (VAR-) identify data, condition flags (FLAG-) identify Boolean knowledge about the problem circumstance, and action clauses (ACT-) provide the

initiative to perform a non-rule-oriented function or algorithm. The rules are generally of the form 'IF preconditions. . . THEN conclusions. . .'.

The issue represented here is that of active constraint instability. In the optimization search the direction of search sometimes must be normal to a particular constraint. When this occurs the algorithms are at odds between improving the objective function value and satisfying the constraint. Zig-zagging can occur with the constraint becoming active and then inactive in alternate steps. These behavioral characteristics can be used to modify the search beneficially by either always including the constraint in the active set or by leaving it out of the set (ignoring it). Computationally, either of these alternatives may be better than the zig-zag behavior indicated.

The first rules below provide tests for such a condition. When the behavior is exhibited, the rules would signal the condition by setting a flag, followed by relating the condition to other conditions and eventually, by performing some appropriate action. Actually some rules set flag conditions and others reset them (NOT FLAG-). The second set of rules relate the condition to some appropriate action. Depending on the inference-engine architecture, the action may be dealt with in many ways. The simple mode envisioned here is that, whenever an action clause becomes true, the engine immediately performs the appropriate function or algorithm associated with it. After the action is complete the clause is reset (a sort of single shot analogy).

Simple rule variables

CN	Constraint-Number
IN	Iteration-Number
CIN	Current-Iteration-Number
DV	Design-Variable-Number

Rules to identify conditions

```

IF    VAR-Constr-Value (CN, CIN) <  $\epsilon$ -Active-Criterion
      VAR-Constr-Value (CN, CIN-1) >  $\epsilon$ -Active-Criterion
      VAR-Constr-Value (CN, CIN-2) <  $\epsilon$ -Active-Criterion
THEN NOT FLAG-Constr-Activity-Stable (CN)

IF    VAR-Constr-Value (CN, CIN) >  $\epsilon$ -Active-Criterion
      VAR-Constr-Value (CN, CIN-1) <  $\epsilon$ -Active-Criterion
      VAR-Constr-Value (CN, CIN-2) >  $\epsilon$ -Active-Criterion
THEN NOT FLAG-Constr-Activity-Stable (CN)

IF    VAR-Constr-Value (CN, CIN) >  $\epsilon$ -Active-Criterion
      VAR-Constr-Value (CN, CIN-1) >  $\epsilon$ -Active-Criterion
THEN FLAG-Constr-Activity-Stable (CN)

IF    VAR-Constr-Value (CN, CIN) <  $\epsilon$ -Active-Criterion
      VAR-Constr-Value (CN, CIN-1) <  $\epsilon$ -Active-Criterion
THEN FLAG-Constr-Activity-Stable (CN)

IF    FLAG-Constr-Activity-Stable (CN)
THEN NOT FLAG-Constr-Removed-from-Active-Set (CN, CIN-1)
      NOT FLAG-Constr-Added-to-Active-Set (CN, CIN-1)

```

Choosing appropriate actions

```

IF      NOT FLAG-Constr-Activity-Stable (CN)
        NOT FLAG-Nonlinear-Constr (CN)
        VAR-Constr-Value (CN, CIN) <  $\epsilon$ -Active-Criterion
THEN    ACT-Remove-Constr-from-Active-Set (CN)
        FLAG-Constr-Removed-from-Active-Set (CN, CIN)

IF      NOT FLAG-Constr-Activity-Stable (CN)
        NOT FLAG-Nonlinear-Constr (CN)
        VAR-Constr-Value (CN, CIN) >  $\epsilon$ -Active-Criterion
THEN    FLAG-Constr-Removed-from-Active-Set (CN, CIN)

IF      NOT FLAG-Constr-Activity-Stable (CN)
        FLAG-Nonlinear-Constr (CN)
        VAR-Constr-Value (CN, CIN) >  $\epsilon$ -Active-Criterion
THEN    ACT-Add-Constr-to-Active-Set (CN)
        FLAG-Constr-Added-to-Active-Set (CN, CIN)

IF      NOT FLAG-Constr-Activity-Stable (CN)
        FLAG-Nonlinear-Constr (CN)
        VAR-Constr-Value (CN, CIN) <  $\epsilon$ -Active-Criterion
THEN    FLAG-Constr-Added-to-Active-Set (CN, CIN)

IF      NOT FLAG-Constr-Activity-Stable (CN)
        FLAG-Constr-Added-to-Active-Set (CN, CIN-1)
THEN    ACT-Add-Constr-to-Active-Set (CN)
        FLAG-Constr-Added-to-Active-Set (CN, CIN)

IF      NOT FLAG-Constr-Activity-Stable (CN)
        FLAG-Constr-Removed-from-Active-Set (CN, CIN-1)
THEN    ACT-Remove-Constr-from-Active-Set (CN)
        FLAG-Constr-Removed-from-Active-Set (CN, CIN)

```

This is a simplified version of the rules required for the problem. The rules relate only the epsilon-active criteria to the conditions. Additional knowledge would also be incorporated to deal with the details of simply active and violated constraint activity. Other criteria might also be involved.

Some discussion must be made of the use of the word NOT in the rule clauses. The purpose of the word NOT is twofold: to select the negation of a clause as the precondition and to negate or reset the flag as a conclusion. In all Boolean logic systems the concept of negation is required. Here it is extended to affect the condition flags as switches.

Conditions can exhibit an inhibitory effect as well as a supportive one. The presence of the nonlinear condition, which must be established as a precondition, effects the application of the rules. Its presence is a logical mechanism to inhibit one rule and support another.

Certain rules to identify active constraints at the optimum have also been discussed in [16-18]. They are based on local monotonicity analysis of cost and constraint functions. The rules utilize Kuhn-Tucker necessary conditions, Lagrange multipliers, and gradients of cost and constraint functions. A production system based on these rules and certain global rules (based on designer's knowledge about the specific problem) has been demonstrated in [18]. Example problems clearly show advantages of using expert's knowledge in the design-optimization process.

5.3. Relating rules

Also associated with the knowledge about instability or zig-zagging of the epsilon activeness of a constraint is the recognition that a trend is also occurring in the design variables as the search moves along the constraint. This condition can be cited by adding to the conclusions (THEN clauses) of the rules above or by adding a separate rule as follows:

```
IF     NOT FLAG-Constr-Activity-Stable (CN)
THEN  FLAG-Possible-Design-Trend
```

This rule may be used to initiate evaluation of design-variable trend locating rules which determine the design variables which exhibit the inferred trend. These rules might include the following:

Trend condition rules

```
IF     FLAG-Possible-Design-Trend
      VAR-Design-Change (DV, CIN) > 0
      VAR-Design-Change (DV, CIN-1) > 0
THEN  FLAG-Design-Trend (DV)

IF     FLAG-Possible-Design-Trend
      VAR-Design-Change (DV, CIN) < 0
      VAR-Design-Change (DV, CIN-1) < 0
THEN  FLAG-Design-Trend (DV)

IF     FLAG-Possible-Design-Trend
      VAR-Design-Change (DV, CIN) ~ 0
      VAR-Design-Change (DV, CIN-1) ~ 0
THEN  FLAG-Static-Design-Variable-Trend (DV)

IF     FLAG-Design-Trend (DV)
      VAR-Design-Change (DV, CIN) > VAR-Design-Change (DV, CIN-1)
      VAR-Design-Change (DV, CIN-1) > VAR-Design-Change (DV, CIN-2)
THEN  FLAG-Increasing-Nonlinear-Trend (DV)

IF     FLAG-Design-Trend (DV)
      VAR-Design-Change (DV, CIN) ~ VAR-Design-Change (DV, CIN-1)
      VAR-Design-Change (DV, CIN-1) ~ VAR-Design-Change (DV, CIN-2)
THEN  FLAG-Linear-Trend (DV)

IF     FLAG-Design-Trend (DV)
      VAR-Design-Change (DV, CIN) < VAR-Design-Change (DV, CIN-1)
      VAR-Design-Change (DV, CIN-1) < VAR-Design-Change (DV, CIN-2)
THEN  FLAG-Decreasing-Nonlinear-Trend (DV)
```

Flexibility can be built into the rule base if care is taken by making the rules independent. The independent rule structure can be a two-edged sword, however, and can trap the unwary in an inflexible system. For example, take the above relations between the unstable activity of a constraint and the design-variable trends. The latter is instigated by a simple flagged condition as a precondition. If the rules were incorporated as is, then the search for trends in variables would occur only if a constraint was unstable. We, of course, do not mean to limit the search for trends in this way. We must work with the rule base until all aspects of the rules

relate appropriately. In the above case, parallel rules without the flagged condition could be added or some more elaborate rules developed. A caution should be made about making rules too independent. If, for example, we eliminated the flag requirement from the preconditions of the rules searching for trends in design variables, we would lose a link between the two conditions. This link is as much a part of the heuristic knowledge as the observance of trends. There are therefore some trade-offs involved in the formulation of the rule base. Often the means of evaluating rules include the occurrence of such links to develop a train of thought.

To a certain extent these rules resemble the logic of many of the standard computer-programming languages. In the expert system, however, they will not be dealt with in the same fixed way. Knowledge about goals, search, and rule evaluation are applied to perform the logical manipulations in an organized and efficient manner. The order of the performance of the rule evaluations can make a significant difference in the conclusions reached. The problem could be avoided by exhaustive rule application to find conditions followed by control decisions but this is not always possible. In identifying conditions the quantity of data available would prohibit an exhaustive search.

5.4. Data base requirements

The data base management systems (DBMS) must have the capacity to deal with the new knowledge including data and rules. The system may use both in-core and fixed disks to access any volume of information with reasonable access time. The DBMS must be designed for use with an expert system. Access times must be sufficiently quick and flexible enough to deal with both a single datum out of large volumes of data and a more complicated rule structure. Such a system called MIDAS is being developed and tested [19].

These are not minor issues in the performance of the expert system. The addition of the inferential computations to the optimization computations being already performed imposes an additional burden on the DBMS. Delays in the flow of information may produce unreasonable real-time performance problems.

6. An interactive design-optimization system

An interactive design-optimization system called IDESIGN [20] has been developed over the past four years. It is a fixed rule-base expert system written in FORTRAN77. It is used here to illustrate how some of the ideas and rules discussed previously have been implemented and extended through the use of expert knowledge.

The architecture of the system is shown in Fig. 3. Only in-core data management is used in the current version. Various rules are imbedded throughout the system. Although the system is designed using small subroutines, it is not modular in the true sense. The system is being redesigned and implemented with MIDAS [20].

The user has to provide two types of information to use the system. First, he must prepare four subroutines to describe the optimization problem, which calculate cost function, constraint functions, and gradients of cost and constraint functions. The controlling program calls appropriate subroutines to calculate these quantities whenever desired. Second, the user must provide the input data which defines bounds on design variables, equality and inequality

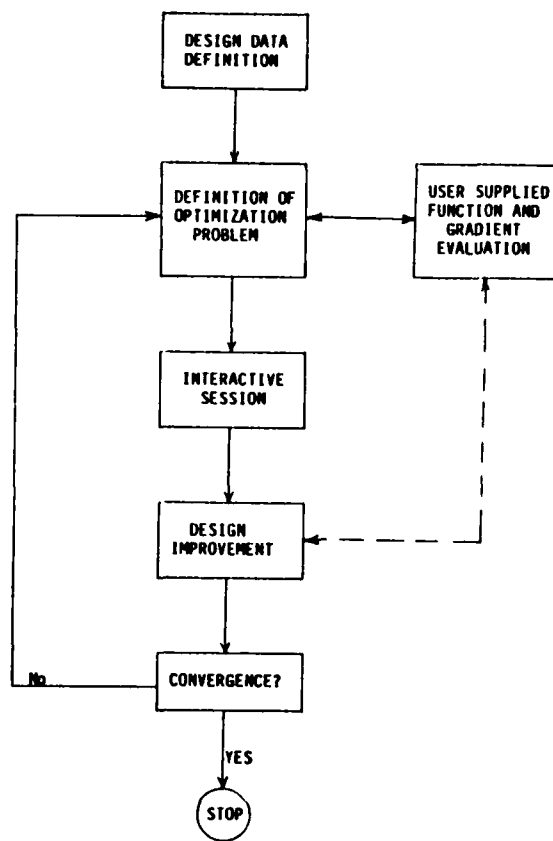


Fig. 3. Structure of interactive design system IDESIGN.

constraints, starting design estimate, convergence parameter and other control information. The input data can be provided interactively or read from a file. The system has its own editor, so the input data can be edited.

Three optimization algorithms are available in the system: the cost-function bounding method [13], a constrained variable metric method [14], and a hybrid method [14]. The program can be run in an interactive mode as well as in a batch mode. A wide variety of interactive capabilities are available. Interactive graphics is also available to observe and use the trend information. We will describe some of the capabilities with the help of a simple spring design problem [13] formulated as

$$\begin{aligned}
 &\text{minimize} && f = (n + 2)Dd^2, \\
 &\text{subject to} && g_1 = 1.0 - D^3n/(71875d^4) \leq 0, \\
 & && g_2 = (4D^2 - dD)/(12566(Dd^3 - d^4)) + 1/(5108d^2) - 1 \leq 0, \\
 & && g_3 = 1 - 140.45d/(D^2n) \leq 0, \\
 & && g_4 = (D + d)/1.5 - 1 \leq 0,
 \end{aligned}$$

where d = wire diameter, D = coil diameter, and n = number of coils. The constraints on

design variables are

$$0.05 \leq d \leq 0.20, \quad 0.25 \leq D \leq 1.3, \quad 2 \leq n \leq 15.$$

6.1. Automatic gradient checking

The system has a built-in procedure to check gradients using the expressions provided by the user. It first determines the appropriate increment in design variables for use in the finite difference procedure. Then the gradients evaluated with finite differences and with analytical expressions are compared. If they differ by less than $\delta\%$ ($\delta = 1$), then the gradient expressions provided by the user are assumed to be correct. Otherwise, a warning message is issued identifying the function having possible errors, and the user is given the option of either continuing or aborting the run.

The user can also omit gradient expressions in the subroutines and use the system's capability to automatically evaluate them at each iteration.

6.2. Starting point improvement

The system provides consulting support to improve the starting point without operating any optimization algorithm. This is particularly useful when the starting point is infeasible. To perform this consulting duty, the system checks sensitivity coefficients of all constraints with respect to each design variable. When sensitivity coefficients with respect to a design variable for all violated constraints have the same sign, that variable can be changed to decrease all violations. The system uses this knowledge to suggest all possible changes. The user can select a suitable change. A typical interactive session to improve the starting design for the spring problem is shown in Fig. 4. It can be seen that in just two changes suggested by IDESIGN, all constraints have been corrected without a severe penalty on the cost function. Note that there were substantial changes to the design. The final design reported in Fig. 4 is not too far from the optimum. If the program were allowed to run without consulting help, it would have taken 12 iterations to reach the optimum.

The example shows the use of the knowledge of the expert to reduce the number of iterations in obtaining a solution.

6.3. Use of trend information

The system stores histories of design variables, constraints, maximum violation, cost function, and the convergence parameter. It can plot these histories at a graphics terminal. With these the user can make design decisions interactively. For example, if a design variable is moving in one direction, it can be extrapolated. If a constraint is never violated, it may be dropped from further consideration. If the cost function is not improving significantly, the process may be terminated. If a design variable has not changed in several iterations it may be fixed at the current value. Further development of the system, including interpolation functions and rules, will be made to provide more consulting support to the user.

In addition to the above, we can add to the system the capacity to determine whether a variable is very sensitive or not sensitive at all. Insensitive variables can be initially kept fixed and only the sensitive ones optimized. Currently, IDESIGN only displays the sensitivity coefficients for various constraints in the form of bar charts at a graphics terminal.

CURRENT DESIGN VARIABLES ARE:

1 0.5000E-01 2 0.1300E+01 3 0.2000E+01
 COST FUNCTION VALUE = 0.1300E-01
 MAX. VIOLATION = 0.2488E+01

The design can be improved by increasing any of the following variables:

Var. No.	Current Value	Suggest Value	Current cost	New Cost
1	0.5000E-01	0.6210E-01	0.1300E-01	0.1929E-01
3	0.2000E+01	0.1500E+02	0.1300E-01	0.5525E-01

The design can be improved by decreasing any of the following variables:

Var. No.	Current Value	Suggest Value	Current cost	New Cost
2	0.1300E+01	0.3218E+00	0.1300E-01	0.3218E-02

Which variables do you want to change?

Variable No: 2

New Value: 0.3218

Constraints might be all satisfied.

Do you want to modify again? N

CURRENT DESIGN VARIABLES ARE:

1 0.5000E-01 2 0.3218E+00 3 0.2000E+01
 COST FUNCTION VALUE = 0.3218E-02
 MAX. VIOLATION = 0.8516E+00

The design can be improved by increasing any of the following variables:

Var. No.	Current Value	Suggest Value	Current cost	New Cost
3	0.2000E+01	0.1500E+02	0.3218E-02	0.1368E-01

Which variables do you want to change?

Variable No: 3

New Value: 15

Some constraints are still violated.

Do you want to modify again? Y

No further improvement can be made without new sensitivity coeff.

Still want to modify? Y

The design can be improved by increasing any of the following variables:

Var. No.	Current Value	Suggest Value	Current cost	New Cost
1	0.5000E-01	0.5019E-01	0.1368E-01	0.1370E-01
3	0.1500E+02	*	0.1368E-01	*

The design can be improved by decreasing any of the following variables:

Var. No.	Current Value	Suggest Value	Current cost	New Cost
2	0.3218E+00	0.3174E+00	0.1368E-01	0.1363E-01

Which variables do you want to change?

Variable No: 2

New Value: 0.3174

Constraints might be all satisfied.

Do you want to modify again? N

CURRENT DESIGN VARIABLES ARE:

1 0.5000E-01 2 0.3174E+00 3 0.1500E+02
 COST FUNCTION VALUE = 0.1349E-01
 MAX. VIOLATION = 0.0000E+00

Fig. 4. Interactive session with IDESIGN to improve the starting point for the spring design problem.

6.4. Algorithm selection

Three algorithms are available in the system. The user can specify his selection in the input data. The cost function bounding algorithm uses only first-order information whereas the constrained variable metric method generates approximate second-order information (Hessian of the Lagrange function) and uses it in defining the direction. The method has a local superlinear rate of convergence which means that, once the design point is in a certain neighborhood (domain of convergence) of the optimum, the algorithm converges quite rapidly. This is generally characterized by the fact that the step size is unity in the neighborhood of the solution. The algorithm can be slow outside the domain of convergence because the approximate second-order information is not very accurate due to variations in the active set. To overcome this difficulty, a hybrid method has been developed which starts with the first-order algorithm and switches to the constrained variable metric method once the design is in the proper neighborhood. Heuristics have been used to determine when to make the switch. Some of the rules used are: (1) small difference between upper and lower bounds on the optimum cost; (2) convergence parameter smaller than a specified value; (3) no change in the active set for several iterations; and (4) small changes in the cost function for several iterations. These rules have worked well for some 120 test problems.

6.5. Automatic restart

When the constrained variable metric method is used, it is sometimes advantageous to restart the procedure, (i.e., set the Hessian to identify). As evidence of this situation, the spring design problem, when started with a design of (1, 2, 3), takes 158 iterations without the restart option and 75 iterations with it. This clearly indicates that the restart option can speed up the convergence. The question then is, what data and knowledge should be used to decide when to restart the algorithm? Again, several heuristic rules have been developed and implemented: (1) changes in the active set; (2) a large condition number for the approximate Hessian matrix; and (3) a small step size.

6.6. Robust implementation of algorithms

Special care has been exercised to implement algorithms so that they are numerically robust. For example, the subproblem is normalized at each iteration such that all the gradients have a unit norm. This has worked extremely well in numerical tests. Whenever something goes wrong with the solution procedure, the system either takes a corrective action or provides information for the user to correct the problem. Many rules have been used to accomplish robust implementation of algorithms.

7. Discussion and conclusions

In this paper, some basic ideas on development of an expert system for general design-optimization applications are presented. The need for the use of AI concepts and some specific instances where they can be helpful in the design-optimization process are discussed. A general architecture of an expert system for the application is developed. Advantages of such a system are demonstrated with the help of a primitive expert system called IDESIGN.

AD-A193 325

DATABASE DESIGN AND MANAGEMENT IN ENGINEERING
OPTIMIZATION(U) IOWA UNIV IOWA CITY OPTIMAL DESIGN LAB
J S ARORA FEB 88 ODL-88.2 AFOSR-TR-88-8366

2/2

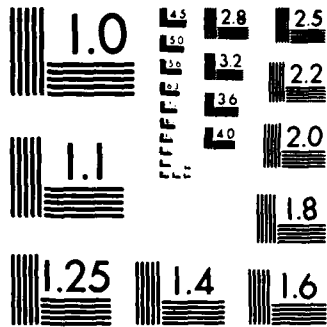
UNCLASSIFIED

AFOSR-82-0322

F/G 12/7

NL

FILE
DATE
BY



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

It appears that the possible advantages of including AI techniques in the optimum design process are significant. The expense of developing an expert system and knowledge base, which will transform the state-of-the-art optimization algorithms into an expertly controlled system, is worthwhile. The efforts required include the development or adaptation of an inferential generator to manipulate the expert knowledge and the accumulation and refinement of that knowledge. Both tasks involve considerable time and effort to be certain that the performance of the resulting system is both capable and efficient.

The system must be based on the most up-to-date developments and strategies in the AI field and, since the field is changing rapidly, the system must also provide for future developments. We envision the capacity of the system, in the not too distant future, as being able to develop or plan an attack strategy for a particular problem and to learn from its success and failure so that it can apply that knowledge to subsequent problems. Our experience with this and other aspects of how we solve problems and streamline production should be directed to the advancement and improvement of the performance of design-optimization systems.

References

- [1] M.M. Waldrop, Artificial intelligence (I): Into the world, *Science* 223 (1984) 803.
- [2] M.M. Waldrop, The necessity of knowledge, *Science* 223 (1984) 1279.
- [3] M.M. Waldrop, Artificial intelligence in parallel, *Science* 225 (1984) 608.
- [4] C.L. Dym, Expert systems: new approaches to computer-aided engineering, in: *Proceedings of the 25th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference*, Palm Springs, CA (May 1984) 99-115.
- [5] E. Rich, *Artificial Intelligence* (McGraw-Hill, New York, 1983).
- [6] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, eds., *Building Expert Systems* (Addison-Wesley, Reading, MA, 1983).
- [7] A. Barr and E.A. Feigenbaum, eds., *The Handbook of Artificial Intelligence* (Kaufman, Los Altos, CA, 1981).
- [8] J.S. Arora and A.D. Belegundu, Structural optimization by mathematical programming, *AIAA J.* 22 (6) (1984) 854-856.
- [9] A.D. Belegundu and J.S. Arora, A study of mathematical programming methods for structural optimization; Part I: Theory; Part II: Numerical aspects, *Internat. J. Numer. Meths. Engrg.* 21 (1985) 1583-1624.
- [10] G.N. Vanderplaats, *Numerical Optimization Techniques for Engineering Design: with Applications* (McGraw-Hill, New York, 1984).
- [11] G.V. Reklaitis, A. Ravindran and K.M. Ragsdell, *Engineering Optimization: Methods and Applications* (Wiley, New York, 1983).
- [12] O.K. Lim and J.S. Arora, Extensions of Pshenichny's algorithm for engineering design optimization, *Comput. Meths. Appl. Mech. Engrg.*, to appear.
- [13] J.S. Arora, An algorithm for optimum structural design without line search, in: E. Atrek, H. Gallagher, K.M. Ragsdell and O.C. Zienkiewicz, eds., *New Direction in Optimum Structural Design* (Wiley, New York, 1984) Ch. 20.
- [14] P.B. Thanedar and J.S. Arora, An efficient hybrid optimization method and its role in computer-aided design, in: *Proceedings CAD/CAM Robotics and Automation Conference*, University of Arizona, Tucson, AZ, February 1985.
- [15] Artificial intelligence and pattern recognition in computer-aided design, in: *Proceedings International Federation of Information Working Conference* (North-Holland, Amsterdam, 1978).
- [16] S. Azram and P. Papalambros, An automated procedure for local monotonicity analysis, *Trans. ASME J. Mech., Transmissions Automated Design* 106 (1984).

- [17] S. Azram and P. Papalambros, A case for a knowledge-based active set strategy, *Trans. ASME J. Mech., Transmissions Automated Design* 106 (1984).
- [18] H.L. Li and P. Papalambros, A production system for use of global optimization knowledge, *Trans. ASME J. Mech., Transmissions Automated Design*, to appear.
- [19] T. SreekantaMurthy, Y.-K. Shyy and J.S. Arora, MIDAS: management of information for designs and analysis of systems, Paper No. 85-0618-CP, AIAA 26th Structural Dynamics and Materials Conference, Orlando, FL, April 1985.
- [20] J.S. Arora, P.B. Thanedar and C.H. Tseng, User's manual for program IDESIGN 3.3, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, IA, December 1984.

APPENDIX 8

**IMPLEMENTATION OF AN EFFICIENT RUN-TIME SUPPORT
SYSTEM FOR ENGINEERING DESIGN ENVIRONMENT**

by

S. Mukhopadhyay and J.S. Arora

in

Advances in Engineering Software

Vol. 9, No. 4, 1987

Implementation of an efficient run-time support system for engineering design environment

S. MUKHOPADHYAY and J. S. ARORA

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, Iowa 52242, USA

The need for both relational and matrix data types in engineering applications has been long recognized. While matrices form mostly temporary or semi-permanent data private to a program, relations are either permanent data in public domain used by different programs or final results of a program to the end user. Though several systems came up in the last few years with various degree of facilities and level of efficiency, none however, met the requirement of versatile data structure or run-time support required in a volatile, large I/O environment of engineering database.

This paper describes implementation of a system and its evaluation using an existing users' interface. Benchmarking shows that the system is far superior to the existing ones and also incurs little overhead for DBMS calls.

INTRODUCTION

As the complexity of data grew, the conventional DBMS posed major handicaps. The design of MIDAS has evolved due to the necessity of secondary storage management in engineering applications, in particular structural design optimization.^{1,2} It was felt that such a system should support both relational and numerical data models with an integrated data definition and manipulation facility. Such facilities would lead to ease in programming (involving large data), and make such programs less error prone, and more efficient.

Requirements of a DBMS for engineering application took shape in the last few years through work on MIDAS: Management of Information for Design and Analysis of Systems.^{3,4} The biggest shortcoming of the current system, MIDAS/N, is that it cannot handle relations. A cursory look in the design of MIDAS/N showed various drawbacks in its design which lead to internal searches at critical DBMS calls. It was decided to develop a completely new system with extended facilities. The new system is MIDAS/GR.^{5,6}

There are seven modules which form the core of MIDAS/GR access method. Task of each module is clearly defined and they interact with each other through well defined interface. In the lowest level is I/O library which invokes operating system routines for transfer of data between disk and main memory. This module provides a machine independent interface to memory management

module for data transfer. Memory management module manages MIDAS/GR buffer. It partitions the buffer into pages of fixed size. All I/O at this level is in terms of pages and using I/O library. It uses stack management module to implement least recently used (LRU) page replacement policy.

The coordinating module in access method is the data management module. It provides facilities of data definition and data manipulation. It uses memory management module to read/write appropriate data. There are four modules which are designed to aid data management in specific tasks. They are page management, list management and segment management, and index management. Page management formats and manages all access to a relation page. List management module maintains all system lists in alphabetically increasing order of names. Segment management allocates and frees segmented memory (file is treated as a contiguous byte address space). Index management takes care of all operations involving indices. Figure 1 shows the schematic view of the organization of different modules.

BUFFER ORGANIZATION

MIDAS/GR buffer is logically divided into two pools: Block Buffer (BB) and Page Buffer (PB). Block Buffer contains all administrative informations, e.g. cursor table, page tables, page page tables (for large data objects), etc. Page Buffer contains the data pages and page table pages (for large data objects). Figure 1 shows the coarse configuration of Block Buffer and Page Buffer.

When an object is opened, an entry is allocated in the cursor table. It has two pointers: pointer to a master record which contains all attributes of the data object and the other to the page table.

A page table contains entry for each page. They are the addresses of the page in disk and in page buffer, address of the page entry in LRU stack, and a mark bit to recognize if the page is modified. For each page in the page buffer, there is an entry in the stack. This entry contains the address of the entry in page table. Logically there are two stacks: stack of clean pages and stack of dirty pages.

The top most entry represents the most recently used page and bottom most entry represents the least recently used page. Figure 2 shows the detailed configuration of MIDAS/GR buffer.

MEMORY MANAGEMENT

Selection of memory management mechanism plays key role in system performance. However, there is no unique method which will perform best in all situations. There are

Accepted April 1987. Discussion closes December 1987.

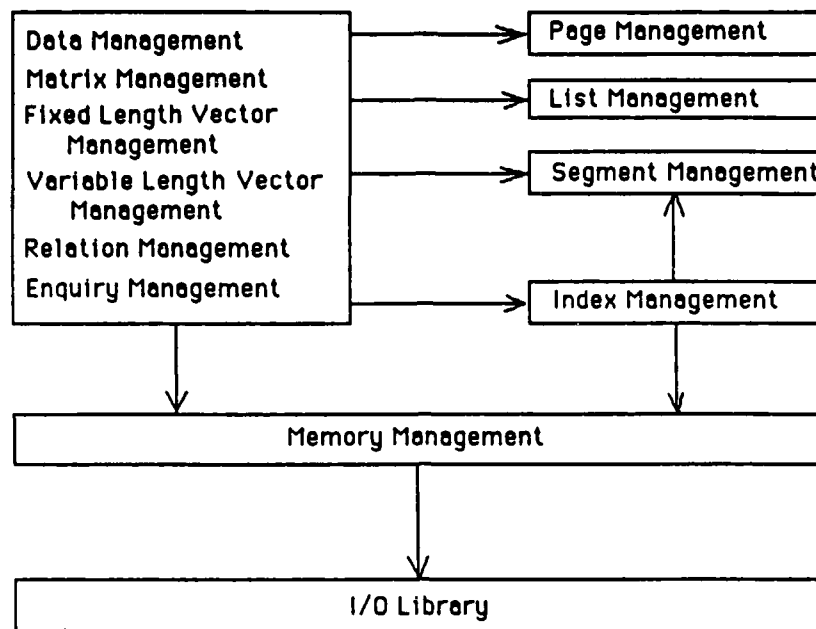


Figure 1. Organization of MIDAS/GR access module

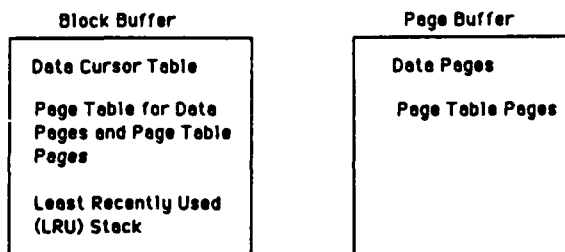


Figure 2. Block buffer and page buffer

three different memory management schemes used in MIDAS/GR. Each is efficient in its context: Management of Segmented Secondary Storage, Management of Segmented Main Memory, Management of Paged Primary Memory.

Management of Segmented Secondary Storage: Currently secondary storage is accessed using operating system's file management facilities. A file is considered a linear address space at page level, i.e. unit of access (address and transfer) is a page. Since we allow dynamic allocation and deallocation of space (objects are created, expanded and destroyed dynamically), there will be lots of holes in the file. One solution is to compact every time memory is deallocated; compaction is, however, much slower process, whereas it is much more economical to search a well organized hole list.

Since disk access is slow, hole list is organized in the header of the file and retained in the main memory after it is read first time. Figure 4 shows the structure of the hole list. It is arranged in increasing order of the address of available space. Since this list contains sufficient information, no disk access is necessary for allocation and deallocation purposes.

Management of Segmented Main Memory: A page of variable length record is a linear address space at record

level. When any information in a page is accessed, the whole page is in memory. Therefore, one does not incur any extra overhead if control information is kept along with the used and available space.

A modified version of Boundary Tag method with improved first fit algorithm (using rover counter, Knuth 1973⁷) is used for allocation and deallocation of space. In this method only one byte at the beginning of a block is used. This reduces storage overhead (which is at a premium in a page), and also allows us to use a very simple algorithm for space liberation, as there is no search or collapsing of adjacent free blocks involved. Even space reservation involves little search as allocation is distributed over the page using a rover counter. However, allocation procedure is slightly complex, and runs little longer as collapsing of adjacent free blocks is done at this time. Figure 5 shows the configuration of free and used space.

Management of Paged Primary Memory: This mechanism is used to manage the page buffer. Page buffer is a collection of page frames which are equal to the physical slots on disk. Each opened object has a page table which points to the page frame belonging to it. Available page frames in page buffer are linked together.

These pages are managed in a least recently used basis. LRU policy is implemented using two stacks: stack of dirty pages and stack of clean pages. Top most entry of the stack is the most recently used page and the bottom most one is the least recently used page. Every time a page is accessed, its position in the stack is moved to the top. They are implemented using double links. They dynamically share same memory. Total space occupied by two stacks together equal to the number of page frames in the page buffer.

Pages are simply read into the page buffer till it is full. Once it is full and a new page has to be read in, the system reuses the page frame of the least recently used clean page. If there is no clean page in page buffer then the content of the least recently used dirty page is written back to disk and the page frame is reused.

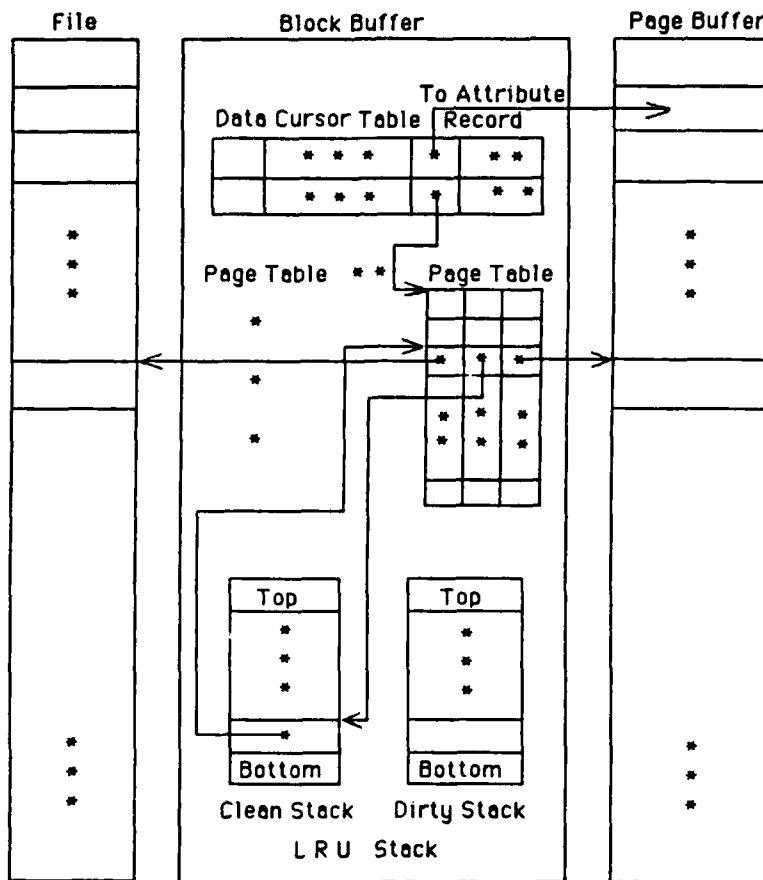


Figure 3. MIDAS/GR buffer configuration

Address of Available Space	Length of Available Space
*	*
*	*
*	*

Figure 4. Hole list for storage management

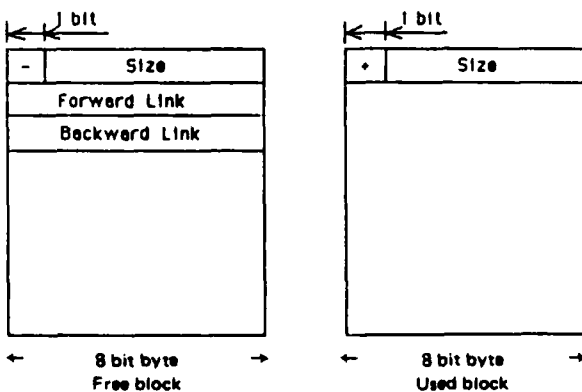


Figure 5. Configuration of free and used space

DATA MANAGEMENT

The data language is an English keyword-oriented syntax for query, as well as for data definition, data manipulation, and control. It provides facilities ranging from simple queries to complex data manipulation intended for professional programmers. The same language may be embedded in a host language program or may be used as a stand alone system. The syntax of the language is given in the companion paper Mukhopadhyay and Arora.⁵ Examples of different operations are given in the following.

A. Data definition language

1. Create a two dimensional 1000 * 1000 temporary matrix named STIFFNESS, with double precision elements and band width 20.
 CREATE TEMPORARY MATRIX STIFFNESS (1000, 1000) BANDED 20: (DOUBLE PRECISION);
2. Create a permanent relation known only to this user named XCOMP with attributes NODE_NO, COORD, FORCE, MOMENT.
 CREATE PERMANENT PRIVATE VECTOR XCOMP:
 NODE_NO (INTEGER, KEY),
 COORD (REAL, NONULL),
 FORCE (REAL),
 MOMENT (REAL);

3. Create a permanent shared relation ELEMENT with attributes ELM_NO, MAT_NO.
CREATE PERMANENT SHARED VECTOR ELEMENT:
ELM_NO (INTEGER, KEY),
MAT_NO (INTER);
4. Create a relation NODE with attributes NODE_NO, COORD, ELM_NO.
CREATE VECTOR NODE:
NODE_NO (INTEGER, KEY),
COORD (VEC (3): REAL),
ELM_NO (INTEGER, NONNULL);
5. Create an index INODE on relation NODE with unique key using attribute NODE_NO.
CREATE UNIQUE IMAGE INODE ON NODE
(NODE_NO);
6. Create a link LELM from relation ELEMENT to relation NODE where the relations are clustered (in the same pages), ordered by NODE_NO.
CREATE CLUSTERING LINK LELM
FROM ELEMENT (ELM_NO)
TO NODE (ELM_NO)
ORDER BY NODE_NO;
7. Destroy relation NODE.
DROP VECTOR NODE;

B. Query language

1. Retrieve all the elements from the matrix STIFFNESS where row number greater than 5 and column number less than 70.
RETRIEVE *
FROM STIFFNESS
WHERE ROW > 5
AND COL < 70;
2. Retrieve unique NODE_NO, FORCE pair from relation XCOMP where NODE_NO less than 50 and FORCE greater than 5, and order them in ascending order of NODE_NO.
RETRIEVE UNIQUE NODE_NO, FORCE
FROM XCOMP
WHERE NODE_NO < 50
AND FORCE > 5
ORDER BY NODE_NO ASC;

C. Data manipulation language

1. Retrieve all data fields from relation ELEMENT2 where ELM_NO greater than 50, and insert ELM_NO, MAT_NO fields to the relation ELEMENT.
INSERT INTO ELEMENT (ELM_NO, MAT_NO):
(RETRIEVE *
FROM ELEMENT2
WHERE ELM_NO > 50);
2. Insert (10, 2.2, 5.7, 1.32, 3) data to the relation NODE.
INSERT INTO NODE (NODE_NO, COORD, ELM_NO):
(10, 2.2, 5.7, 1.32, 3);
3. Delete (set to zero) all the elements of the matrix STIFFNESS where row greater than 100 and column greater than 100.
DELETE STIFFNESS
WHERE ROW > 100
AND COL > 100;
4. Delete the tuple from the relation NODE where NODE_NO is 5.
DELETE NODE
WHERE NODE_NO = 5;

5. Delete all the tuples from the relation ELEMENT which has 3 nodes.
DELETE ELEMENT X
WHERE (RETRIEVE CNT (*)
FROM NODE
WHERE ELM_NO = X.ELM_NO) = 3;
6. Update all the elements of the row number 30 to value 25 of the matrix STIFFNESS.
UPDATE STIFFNESS
SET 25
WHERE ROW = 30;
7. Update COORD value to 2.54 times in all tuples where ELM_NO is 2, 10, 19, or 25 in relation NODE.
UPDATE NODE
SET COORD = COORD * 2.54
WHERE ELM_NO IN (2, 10, 19, 25);

EVALUATION

Application programs have been accessing MIDAS using a well defined interface. The specification of this interface is given in MIDAS/N User's Manual.⁸ The same interface is implemented on top of MIDAS/GR access method. As a result MIDAS/GR has become accessible to all applications that have been using MIDAS.

For the purpose of benchmarking MIDAS/GR and comparing it with MIDAS/N, an application program is chosen. This program solves a system of equations using the skyline method. The reason for choosing this program is that in most engineering problems it is necessary to solve a system of equations, and the performance of the application depends critically on the performance of the equation solver.

The equation solver is flexible in its use of the main memory. While running in a large system, it can take advantage of large memory and make requests for data in bulk, therefore, making less calls to MIDAS; this leads to better performance. Whereas in a smaller system, it uses little core memory and makes repeated calls to MIDAS for data transfer; this usually leads to lower performance of the system. The management of workspace within an application, therefore, has profound effect on overall system performance. So, the size of workspace is included as one of the parameters along with the size of the problem, while evaluating MIDAS/GR.

For benchmarking MIDAS/GR and for comparing its performance with MIDAS/N, different system and run-time environments are chosen. It is felt that among system parameters page size and number of pages in the buffer should make most noteworthy effect on the system performance. Among run-time parameters workspace size and number of equations are chosen for variation. The values of the parameters are chosen as follows:

Parameter	Values		
Page size	1 KB	4 KB	16 KB
Number of pages	16 nos.	64 nos.	256 nos.
Workspace size	8 KB	32 KB	128 KB
Number of equation	1000, 5000, 10 000, 20 000, 30 000, 40 000		
Half-band width	100	-	-

All tests are run on a DN460 Apollo computer with 2 MB of primary memory.

Tables 1-3 show variation of cpu time with buffer size (page size * number of pages), for different page sizes. The computation time shows the expected trend (i.e. cpu time goes down as buffer size goes up) in most cases, except where the buffer size is very large. For large buffer (1 MB and beyond) the result is very close but slightly erratic. The reason is that currently the buffer is implemented in virtual memory. For large buffer, some portion of it may be allocated on disk, which may degrade system performance.

Tables 4-6 shows the effect of page size on cpu time. The results seem to favor larger page size over the smaller one, except for small workspace (8 KB), where optimal page size lies between 1 KB and 16 KB. It may be noted, however, that this result may not show the trend reliably, as the unit of transfer and allocation still remains unchanged (1 KB). A larger page may be fragmented over the disk; then it will require more than one seek to transfer a

Table 1. Buffer size (in KB) vs. cpu time (in seconds) for 1 KB page
Workspace = 32 KB. No. of equation = 10 000

Buffer size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
16	11 497	21 628	1.88
64	9 515	18 257	1.92
256	8 623	12 627	1.46

Table 2. Buffer size (in KB) vs. cpu time (in seconds) for 4 KB page
workspace = 32 KB. No. of equation = 10 000

Buffer size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
64	9 670	18 362	1.90
256	8 423	12 252	1.46
1 024	8 440	12 331	1.46

Table 3. Buffer size (in KB) vs. cpu time (in seconds) for 16 KB page
Workspace = 32 KB. No. of equation = 10 000

Buffer size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
256	8 413	12 172	1.45
1 024	8 457	12 254	1.45
4 096	8 384	12 247	1.46

Table 4. Page size (in KB) vs. cpu time (in seconds) for 8 KB workspace

Buffer size = 256 KB. No. of equation = 10 000

Page size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
1	8 798	12 777	1.45
4	8 622	12 439	1.44
16	8 763	12 286	1.40

Table 5. Page size (in KB) vs. cpu time (in seconds) for 32 KB workspace

Buffer size = 256 KB. No. of equation = 10 000

Page size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
1	8 623	12 627	1.46
4	8 423	12 252	1.46
16	8 413	12 172	1.45

Table 6. Page size (in KB) vs. cpu time (in seconds) for 128 KB workspace

Buffer size = 256 KB. No. of equation = 10 000

Page size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS : GR
1	3 519	3 968	1.13
4	3 402	3 777	1.11
16	3 390	3 725	1.10

Table 7. Workspace size (in KB) vs. cpu time (in seconds) for 16 KB buffer

Page size = 1 KB. No. of equation = 10 000

Workspace size	CPU time in MIDAS/GR	CPU time in MIDAS/N	MIDAS/N : GR
8	11 709	21 793	1.86
32	11 497	21 628	1.88
128	3 468	4 001	1.15

Table 8. Workspace size (in KB) vs. cpu time (in seconds) for 64 KB buffer

Page size = 1 KB. No. of equation = 10 000

Workspace size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
8	9 711	21 440	2.21
32	9 515	18 257	1.92
128	3 527	4 010	1.14

Table 9. Workspace size (in KB) vs. cpu time (in seconds) for 256 KB buffer

Page size = 1 KB. No. of equation = 10 000

Workspace size (KB)	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
8	8 798	12 777	1.45
32	8 623	12 627	1.46
128	3 519	3 968	1.13

page. This can be corrected partly by clustering relevant pages, and fully by changing the unit of allocation and transfer to the new page size.

Tables 7-9 show variation in cpu time with workspace. Cpu time goes down as the size of workspace increases;

though it is an intuitively correct result, this trend is influenced to a good extent by the algorithm of the equation solver. Larger the workspace, it performs less operations and makes less calls to DBMS; this in turn reduces cpu time. Therefore, the results show the variation of cpu time with respect to workspace size in a magnified scale.

Tables 10-12 show increase in cpu time as the size of the problem (i.e. number of equations) increases. The increase is quite reasonable; it increases about ten times for ten-fold larger problem. Cpu time, however, decreases as the size of the buffer increases.

Table 13 compares cpu time using MIDAS/GR with large workspace and cpu time without using any database management system (data kept in virtual memory).

In general, MIDAS/GR is about twice as fast as MIDAS/N. Exceptions are the cases where workspace is very large, since large workspace leads to little use of DBMS as most of the data reside in main memory. Even in such cases MIDAS/GR is superior to MIDAS/N. The study shows that proper design of the DBMS is extremely important for engineering applications.

Table 10. Number of equations vs. cpu time (in seconds) for 64 KB buffer

Page size = 4 KB. Workspace = 32 KB

No. of equations	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
1 000	878	1 639	1.87
5 000	4 933	9 560	1.94
10 000	9 670	18 362	1.90
20 000	20 316	37 360	1.84
30 000	30 465	55 562	1.82
40 000	40 641	77 935	1.92

Table 11. Number of equations vs. cpu time (in seconds) for 256 KB buffer

Page size = 4 KB. Workspace = 32 KB

No. of equations	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
1 000	772	1 124	1.46
5 000	4 320	6 076	1.41
10 000	8 423	12 252	1.46
20 000	18 111	25 199	1.39
30 000	26 324	37 099	1.41
40 000	35 296	49 462	1.40

Table 12. Number of equations vs. cpu time (in seconds) for 1 024 KB buffer

Page size = 4 KB. Workspace = 32 KB

No. of equations	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
1 000	754	1 092	1.45
5 000	4 316	6 116	1.42
10 000	8 440	12 331	1.46
20 000	18 188	25 286	1.39
30 000	26 439	40 211	1.52
40 000	35 424	49 791	1.41

Table 13. Number of equations vs. cpu time (in seconds) for 1 28 KB workspace

Page size = 1 KB. Buffer size = 16 KB

No. of equations	CPU time in MIDAS/GR (s)	CPU time w/o DBMS (s)	Penalty for DBMS use (%)
1 000	322	281	14.6
5 000	1 729	1 502	15.1
10 000	3 468	3 023	14.7
20 000	7 013	6 193	13.2
30 000	10 551	9 241	14.2
40 000	14 099	12 228	15.3

Another interesting observation relates to the use of DBMS itself. It is a popular belief in scientific field that one must make great sacrifice in efficiency to use DBMS (for other gains). Our result shows that with DBMS (MIDAS/GR) the program runs only about 14% longer than that without DBMS (database is in virtual memory). Though this small increase in cpu time is acceptable in most cases, we believe that a good database management system should, in fact, improve efficiency. The reason for MIDAS/GR not performing to its specification is known to us. The current shortcomings of MIDAS/GR and the remedies are discussed in next section. With those remedies incorporated, we are sure that MIDAS/GR will be unbeatable even in efficiency.

CURRENT LIMITATIONS

Limitations of MIDAS/GR arise mostly from two sources: disk management and memory management. Currently MIDAS/GR uses the operating system's file management facility to store and access data in disk. This leads to related data scattered all over the disk. The operating system does not support the clustering of disk pages. Consequently it takes longer time to access data. Also, the unit of transfer (between disk and main memory) is fixed by the operating system's page size. Current page size is 1 KB which is quite small for engineering applications.

Similarly, the memory management module has no control over primary memory of the computer. It defines its buffer in virtual memory, and transfers pages between virtual memory and virtual disk (disk + operating system buffer for disk) using operating system's facilities. This mechanism can lead to inefficiency under certain circumstances, particularly when running large programs. Consider a dirty page in the buffer which is actually in disk. To replace this page, it has to be read and then written to the disk. This is useless disk access overhead.

Current memory management policy is to replace least recently used clean page if available, otherwise replace least recently used dirty page. This could lead to inefficiency under certain circumstances. Consider a situation when all the pages in the buffer are dirty. Now, if MIDAS/GR receives only 'read requests', all these requests will be serviced by a single page of the buffer. The disadvantage is, if the same page is requested more than once (not consecutively) then each request may force an actual disk read.

FUTURE PLANS

I/O Management: Implementation of disk management and memory management modules on top of operating system's

facilities has two inherent disadvantages: duplication of effort with larger program execution time due to indirect handling, and inability to rise over the vagaries of a general purpose operating system. To improve the system time, it is important to be able to store logical database pages in proper physical slots of the disk; particularly, clustering of related pages is important to reduce disk access time; similarly, it is important to ensure that MIDAS/GR buffer defined in main memory remains there at all times.

In other words, specific operating systems tasks should be replaced altogether by MIDAS/GR routines. Since MIDAS/GR memory management and disk management policies are designed to suit specific application, programs would run much more efficiently in the new set up.

Salvage Program: In current implementation, if an application program using MIDAS/GR crashes unexpectedly, it may force the database into inconsistent state, or even make it inaccessible. A salvage program is required to bring the database to a consistent state. The salvage program scans through the system catalog and deletes entries whenever it finds their presence leads to inconsistency. Therefore, consistency is achieved at the expense of deleting some data object.

Lock Subsystem: It is important to provide concurrent access to the database to optimize its use, and also to facilitate parallel execution of related modules in application program. This, however, is not possible without a concurrency control mechanism. We, therefore, plan to implement a lock subsystem to regulate concurrent access to the database without jeopardising its integrity.

Recovery Mechanism: In any working environment, it is important that a job is able to continue after an interruption without losing any previously done work. The interruption can be in many forms: crash due to error in application program, crash due to DBMS error, job abort due to wrong DBMS call, job abort due to non-availability of data, deadlock etc. After the program is restarted it should be possible to redo the already done work and continue from that point; it is even better if the program is allowed to restart automatically (when it is not crash due to error). On the other hand, if it is not possible to continue after a program crashes, all the work done by the program must be undone.

We plan to implement a log subsystem which keeps trail of activities on selected data objects. Such data objects can be brought to consistent state after a fail; while other data objects will be marked as changed or inconsistent after an unexpected failure, and the same will be notified to the next user.

Data Language: MIDAS/GR provides a unified approach to define and manipulate various data models (relational and numerical) through a data sub-language. This language may be used from a host programming language or as a stand alone utility.

The advantage of having the same language for programmers and terminal users is the ease of communication and use. Our immediate plan is to write an interpreter for the terminal users, since it is comparatively easy to implement. Subsequently, we plan to develop a compiler to process the application programs using MIDAS/GR.

So far in our design we have omitted a few important aspects of database management system: authorization, integrity and distributed database. Authorization, though a crucial issue in business application, we felt, can be postponed in our work. This, however, should not be construed as that security is not important in engineering applications.

Our next objective is to design a simple and efficient authorization mechanism based on password. Entries of this subsystem will be in hierarchical form, e.g. organization, project, programmer; where children entities by default inherit their parents' authority.

The problem of integrity is the problem of ensuring that the data in the database is accurate, i.e. the problem of guarding the database against invalid updates. An integrity subsystem should monitor the transactions, specifically update operations, and detect integrity violations. In the event of a violation, it should take appropriate action, for example, rejecting the operation, reporting the violation, or even correcting the error.

Distributed database management system is our answer to efficient parallel processing over a network. As we know, current emphasis on software design in engineering applications is to incorporate as much parallelism in the algorithm as possible. Parallel algorithms are developed at various levels of granularities. At a coarser level, parallelism is in terms of programs, where independent modules (connected through a database) run in parallel.

To achieve true parallelism, different modules may run on different nodes in a network. It will be efficient if the database is distributed over the network, instead of a central location servicing all requests, which will lead to bottleneck in the network. A distributed version of MIDAS/GR can solve this problem. In our context of local area network, we feel, it may still be advantageous to keep the public domain of the database in a central location, while the private ones can be distributed to appropriate nodes.

CONCLUSION

The need for both relational and matrix data types in engineering applications has been long recognized. While matrices form mostly temporary or semi-permanent data private to a program, relations are either permanent data in public domain used by different programs or final results of a program to the end user. Several systems have been proposed in the last few years⁴ with various degree of facilities and level of efficiency; none, however, meets the requirement of versatile data structure or run-time support required in a volatile, large I/O environment of engineering database.

We believe that MIDAS/GR is unique in its design and capabilities. Its biggest strength lies in its ability to define a unifying data structure, using primitive (integer, real, double precision) and structured (vector, matrix, string, record) data types. Relation is just one of the many user defined data types (vector of records). Because of its general approach, matrices can be defined as easily as any other data types (say, relation), and it can have its own composite data elements, and qualifiers (sparse, banded, etc.). Similarly, relations can have non-atomic attribute values (i.e. a matrix or a vector).

Data objects are broadly grouped in two categories: static and dynamic, e.g. a matrix is a static object, whereas a relation is a dynamic object. Distinct storage organization is adopted for static and dynamic data objects to optimize access time and storage utilization; this distinction is, however, transparent to end users who view data through their own definition.

ACKNOWLEDGEMENT

Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under

Grant Number AFOSR 82-0322. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

REFERENCES

- 1 Rajan, S. D. and Bhatti, M. A. Data management in FEM-based optimization software, *Computers and Structures* 1983, 16 (1-4), 317
- 2 SreekantaMurthy, T. and Arora, J. S. Database management concepts in computer-aided design optimization, *Advance in Engineering Software* 1986, 8 (2), 88
- 3 SreekantaMurthy, T., Shyy, Y. K. and Arora, J. S. MIDAS-management of information for design and analysis of systems, *Advances in Engineering Software* 1986, 8 (3), 149
- 4 SreekantaMurthy, T. and Arora, J. S. Computer-aided structural optimization using a database management system, *Technical Report No. ODL-85.17*, Optimal Design Laboratory, College of Engineering, University of Iowa, 1985
- 5 Mukhopadhyay, S. and Arora, J. S. Design and implementation issues in an integrated database management system for engineering design environment, *Advances in Engineering Software* 1987, 9 (4), 000
- 6 Mukhopadhyay, S. A database management system using generalized relational model, *Master's Thesis in Dept. of Computer Science* (May), The University of Iowa, Iowa City, Iowa 52242, 1986
- 7 Knuth, D. E. *Art of Computer Programming*, Vol. 1, *Fundamental Algorithms*, Reading, Mass., Addison-Wesley, 1973
- 8 Shyy, Y. K., Arora, J. S., Mukhopadhyay, S. et al. *MIDAS/N User Manual*, Optimal Design Laboratory, College of Engineering, University of Iowa, 1984

BIBLIOGRAPHY

- Arora, J. S. and Mukhopadhyay, S. Specification for MIDAS-GR management of information for design and analysis of system: generalized relational model, *Technical Report No. CAD-SS-84.24*, Optimal Design Laboratory, College of Engineering, University of Iowa, 1984
- Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D. et al. System R: Relational approach to database management, *ACM Trans. on Database Systems* 1976, 1 (2), 97

APPENDIX 9

**DESIGN AND IMPLEMENTATION ISSUES IN AN
INTEGRATED DATABASE MANAGEMENT SYSTEM
FOR ENGINEERING DESIGN ENVIRONMENT**

by

S. Mukhopadhyay and J.S. Arora

in

Advances in Engineering Software

Vol. 9, No. 4, 1987

Design and implementation issues in an integrated database management system for engineering design environment

S. MUKHOPADHYAY and J. S. ARORA

Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, Iowa 52242, USA

The need for a unified database management system for various engineering applications has long been felt. Due to the conflicting requirements, so far, most of the attempts in this field are at best partially successful. This paper presents broad perspectives of design and implementation issues of an integrated database management system. A unique feature of the system is its ability to define a unifying data model for matrices and relations. The system also provides for a unified language interface for different user groups. System architecture and a few distinguishing features are described. An evaluation of the system is presented.

INTRODUCTION

Complexity of software for engineering applications has grown exponentially over the years. This is particularly true for the design optimization software where several software components must be integrated to have a viable design tool. The software components are user interface, model generator, model analyzer, design problem definition module, design sensitivity analysis module, optimizer, model update module and the post-processor. Such a large software system generates a huge amount of data. It is clear that for a highly flexible and efficient system, a very sophisticated and advanced database management scheme is needed. Large numerical as well as relational databases must be handled in an efficient and integrated manner. Thus, a very sophisticated database management system (DBMS) is needed that can cater both the advanced users (system programmers) as well as the casual users. This and a companion paper¹ describe design and implementation of such a DBMS.

In a recent paper,² design and implementation of a DBMS called MIDAS was reported. MIDAS stands for Management of Information for Design and Analysis of Systems. The system has two distinct subsystems: MIDAS/R and MIDAS/N. MIDAS/R handles relational data and is an extended version of the well known DBMS called RIM,³ and MIDAS/N handles numerical data sets, such as various types of matrices. It was hoped that with the two subsystems, an application software, such as the one for design

optimization of large systems could be developed using both the relational and the numerical data models. Whereas some application programs have been developed with MIDAS as the DBMS, it has become clear that the use of two separate subsystems is not convenient and efficient. The two subsystems have their own architecture, memory management and user interface. They do not communicate with each other and their use in an application program essentially amounts to using two independent DBMS. This is highly undesirable.

Thus the issue of an integrated database management is re-examined. Design and implementation of a new database management system is reported in this and the companion paper. The system is unique in its design concepts. The strength of the system lies in its ability to define matrices and relations using a unified data model. It also provides facilities which were not hitherto available in a single DBMS in an organized fashion. These facilities are made possible mainly by the development of a new data model, which is termed as the Generalized Relational Model. Accordingly the system is named MIDAS/GR - Management of Information for Design and Analysis of Systems/Generalized Relational model.

The system provides efficient run-time support to various engineering applications handling large numerical and relational data. It also provides a central storage area for various groups in a design department. For example, groups working on structural geometry, structural layout, cost estimation, material control, project planning, etc. need to share data and interact with one another. The system provides necessary interface and facilities for such purpose.

Detailed design of the new system is substantially different from its predecessors described by SreekantaMurthy, Shyy and Arora.² The paper describes broad perspectives of design and implementation issues of the new integrated database management system. The concept of a generalized relational model is described. System architecture and data models are explained. Other issues such as storage, index and link organizations are also discussed. Details of implementation and system evaluation are given in the companion paper.¹

The paper also contains some results of running applications using the new system. The system is almost twice as fast compared to its predecessor. Results are also comparable in efficiency, if a program is run without DBMS (smaller problem that fits in operating systems virtual memory). Appendix contains an abridged version of the higher level data language provided by MIDAS/GR in BNF notation.

Accepted April 1987. Discussion closes December 1987.

REVIEW OF LITERATURE

There is a functional separation between business and scientific computing. It is therefore, not surprising that commercial database management systems are unsuitable in scientific applications. Important similarities, however, exist between the two types of systems in terms of data language, storage management, access method, concurrency control, and recovery mechanism. This aspect is discussed at length by Felippa.^{4,5} Database management concepts in computer-aided design optimization are discussed by SreekantaMurthy and Arora.⁶

SreekantaMurthy and Arora⁷ studied a number of existing database management systems for engineering applications. The survey includes systems such as DELIGHT, DATHAN, EDIPAS, FILES, GIFTS, GLIDE, ICES, IPIP, PHIDAS, REGENT, RIM, SDMS, SPAR, TORNADO, and XIO. Mukhopadhyay⁸ referred to System R⁹ and INGRES.¹⁰ Felippa¹¹ presented a new system called NICE.

None, however, is found suitable to meet the challenge of integrated engineering environment. Most of the systems (e.g. MIDAS/N, GIFTS, etc.) support only numerical databases and provide run-time support, while others (e.g. RIM, NICE) provides global database for different user groups to share data. The latter systems (e.g. RIM, INGRES, etc.) have tried to extend relational model to incorporate matrices or vectors. This artificial definition of matrices leads to difficulty in organization of numerical data. This also makes direct access to individual data elements impossible.

GENERALIZED RELATIONAL MODEL

Relations are important for sharing data among different users. Whereas most physical models in business applications can be represented conveniently in database as relations, engineering applications require both matrix and relational data types. Most large matrices form temporary or semi-permanent data private to a program. Relations are either permanent data in public domain used by different users or final results of a program to the end user. Therefore a new scheme is needed to represent both relations and matrices in a unified way for integrated engineering applications.

Engineers and scientists are well versed with the use of matrices and vectors. It is natural for them to imagine a relation as simply a two dimensional array whose each column has unique definition. Thus the concept of a matrix is generalized and the resulting model is termed as the generalized relational model. This scheme is supported by primitive and structured data types. Using these data types users can define their own data models.

The novelty of the new approach is that the relation is derived from a matrix. In all previous attempts (RIM, INGRES), one tried to extend relation for matrix data type. This led to clumsy and inefficient handling of numerical data. In MIDAS/GR the basic data type is matrix. Matrix can be one dimensional (vector) or two dimensional, and it can have elements with composite data structure. Relations are derived from the matrices as vector of record.

SYSTEM ARCHITECTURE

The overall architecture of MIDAS/GR is described by its two main components. The lower level component is Data Storage Interface (DSI), and the upper level component is called Data Language Interface (DLI). DSI is an internal interface which handles access to single data elements. It

manages space allocation, storage buffers, transaction consistency, system recovery, etc. It also maintains indexes on selected fields of relations and pointer chains across relations.

The DSI has been designed so that new data objects or new indices can be created at any time, or existing ones destroyed, without quiescing the system and without dumping and reloading the data. This facilitates gradual database integration and returning of access paths. One can redefine data objects, i.e. change dimensions of matrices or add new fields to relations. Existing programs which execute DSI operations on data aggregates remain unaffected by the addition of new fields.

DSI has many functions which can be found in other systems, both relational and nonrelational, such as the support of index and pointer chain structures. The areas which have been emphasized and extended in the DSI included integrated data definition and manipulation facilities for numerical (matrices) and relational data models, dynamic definition of new data types and access paths, dynamic binding and unbinding disk space, and crash resistance and recovery.

Data Language Interface (DLI) is the external interface which can be called directly from a programming language. The high level data language is embedded within the DLI and is used as the basis for all data definition and manipulation. In addition, the DLI maintains the catalogs of external names, since the DSI uses only system generated internal names.

DATA MODEL

The basic data models in MIDAS/GR are matrices (two dimensional) and vectors. Vectors are of two types: variable length and fixed length. A fixed length vector can also be considered as a degenerate matrix with only one row. A special type of variable length vector is the relation (vector of record). Figure 1 shows relationships among different data models.

Matrix and fixed length vectors are static objects, i.e. their sizes are statically determined at the time of their creation. This essentially means that they can have elements only of fixed size. In contrast, variable length vectors are variable in nature. Not only is their size determined dynamically (depending only on the number of elements defined), also their elements may be of variable length.

There are four structured data types available in the system: Record, Vector, Matrix, String. The type record gives the most general method to obtain structured data type. It joins elements of arbitrary, possibly themselves structured types into, a compound type. Vectors and matrices are in contrast homogeneous structures. They consist of all components of the same type, called the base type. The base type in turn may be a structured type. This opens up the possibility of defining a number of special data types. For example, a matrix of complex numbers, where complex number is a record of two real numbers.

A vector of record may in general be treated as a relation where each component of the vector is an occurrence of the record, and each component of the record is an attribute of the relation.

If a matrix is defined as matrix of matrices, the matrix is considered to be made up of a number of submatrices. Matrices may be accessed rowwise, columnwise, or submatrixwise.

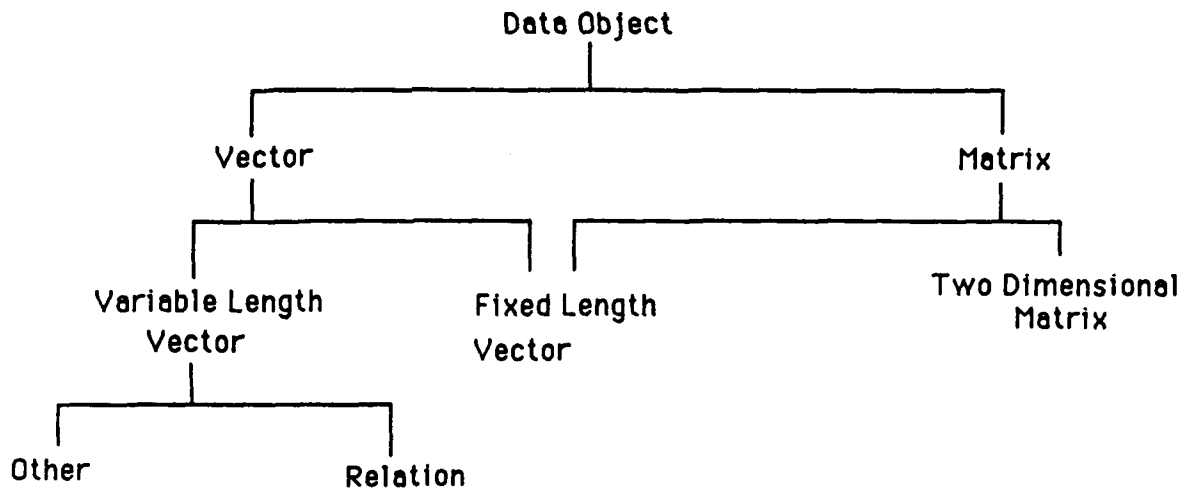


Figure 1. Data model of MIDAS/GR

String is a special structured data type designed to deal with a sequence of characters. String may be used as base type to define other structured data types.

There are four primitive types available in the system. They are Integer, Real, Double precision, and Character.

STORAGE ORGANIZATION

The storage interface provides one potentially infinite linear address space. However, it is preferable to partition a large database into areas. Such a partition allows for smaller addresses to be used. It improves flexibility for controlling access to the database, and provides a means of factoring out some common attributes of a collection of data. It is also useful for selectively saving and restoring information.

The database consists of a set of disjoint areas, each of which constitutes a linear address space. These areas are used for storing user data, access path structure, internal catalog information, and intermediate results. All the elements of a data object must reside within a single area; however, a given area may contain several data objects, indices, etc.

Areas are classified in three major types, depending on the combination of functions supported and overhead incurred: Public Area, Private Area, and Temporary Area. Public Area contains shared data that can be simultaneously accessed by multiple users. Private Area contains data that can be used by only one user at a time (or data that is not shared at all). Temporary Area contains only temporary data which is lost as soon as the program terminates.

Data in public and private areas are recoverable (i.e. data will not be lost in the event of a failure), but not the one in temporary area. This reduces the overall overhead, as the overhead associated with full support of concurrent sharing needed for public data area, can be avoided for private and temporary data.

The addressing of a particular location in an area could be done at the byte level, by using a relative address from the beginning of the area. However, such a continuous space must be stored on auxiliary storage in records. We therefore, decompose the address space into logical pages, knowing that these pages will be stored on disk in physical slots of identical size. A page is, therefore, unit of access (address and transfer).

Two distinct storage organizations are adopted for fixed length data objects and variable length data objects. For fixed length objects, all data elements are defined at the time of creation. Therefore, full storage allocation is made as soon as it is created. Whereas for variable length objects, size of the object at any time depends on the number of elements defined. Therefore, allocated storage is expanded dynamically as more and more elements are defined.

For variable length objects, data aggregates may be stored in a page and addressed using the page number and the offset of the data aggregate from the beginning of the page. This strategy has the disadvantage that a data aggregate must remain in a fixed location within a page; this leads to internal fragmentation within a page, as the data aggregates are deleted. Figure 2 shows the organization of a page of variable length object. This is a hybrid scheme, which combines the speed of a byte address pointer with the flexibility of indirection. This organization was originally proposed by Astrahan *et al.*⁹ for their experimental DBMS System R.

Each individual record has a numeric identifier, called DaId (Da for data aggregate). Each DaId is a concatenation of a page number, along with a byte offset from the bottom of the page. The offset denotes a special entry or 'slot' which contains the byte location of the data aggregate in that page. This technique allows efficient utilization of space within data pages, since space can be compacted and data aggregates moved with only local changes to the pointers in the slots. The slots themselves are never moved from their positions at the bottom of each data page, so the existing DaIds can still be employed to access the data aggregates. Since the position of the byte offset at the bottom of the page is fixed, internal movements of the data aggregates within a page does not change its address. This provides easy way to compact data aggregates within a page. A variable length object is made up of a collection of such pages.

A matrix is a fixed length object, i.e. its elements are created or deleted all at once. Therefore, we do not have the problem of internal fragmentation within a page for such objects. Also elements in a matrix are accessed both at random and in sequence. Therefore, it is important that the logically contiguous elements are stored physically close to each other. The elements may be stored rowwise, column-

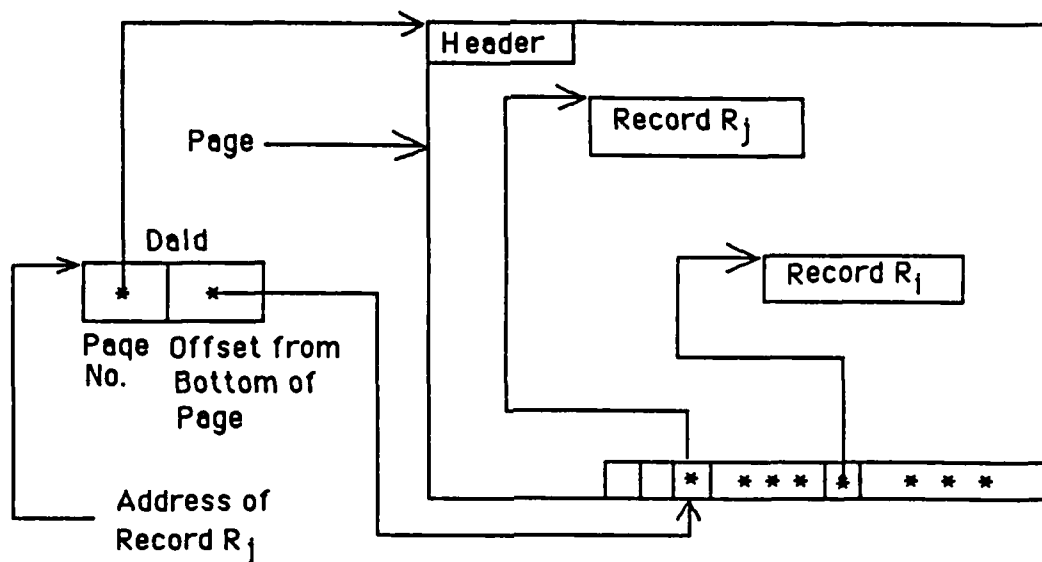


Figure 2. Page organization of variable length object

wise or submatrixwise. Our previous experience shows that if the request for data storage or retrieval varies from that of storage order, I/O is extremely slow; in worst case it may require reading or writing a page for every data element.

McKeller and Coffman¹² show a technique that allows quick access to matrices rowwise, columnwise and submatrixwise. Matrices are stored submatrixwise. Each submatrix contains a fixed number of rows and columns, such that the size of the submatrix is not greater than a page size. It is possible to choose number of rows and columns of submatrices in such a way that minimizes wastage of storage space. Figure 3 shows storage organization for such a data model. Fixed length vectors are stored similarly where number of row is one.

Data objects are divided into two categories – small and large. The mapping between logical and physical pages of the data object is defined by using page table. For large data objects the page table becomes quite large. Therefore, the page table of the large object is defined as small fixed length vector and stored in the database as a permanent object. This technique leads to very small page table in the virtual memory at the expense of one level of indirection for each page access.

INDEX ORGANIZATION

An index is a logical ordering with respect to values in one or more sort fields. Indices combined with scans provide the ability to scan data objects along a value ordering. Also, an index provides associative access capability. The DLI can rapidly fetch data aggregate from an index by keying on the sort field values. The DLI can also open a scan at a particular point in the index, and retrieve a sequence of data aggregates with a given range of sort values.

A new index can be defined at any time on any combination of fields; only restriction being that fields must be atomic. Furthermore, each of the fields may be specified as ascending or descending order. Once defined, an index is maintained automatically by the DSI. An index can also be dropped at any time.

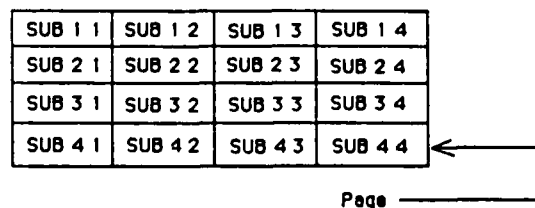


Figure 3. Storage organization of fixed length object

The DSI maintains indices through the use of a multi-page index structure. An internal interface is used for associative or sequential access along an index, and also to delete or insert index entries when data aggregates are deleted, inserted or updated. The parameters passed across this interface include the sort field values along with the DaId of the given data aggregate.

Many techniques for organizing an index have been proposed. Knuth¹³ provides a survey of the basics. While no single scheme can be optimum for all applications, a technique of organizing indices called B-tree¹⁴ has become widely used. The B-tree is, *de facto*, the standard organization for indices in a database system. For reasons of efficiency in both random and sequential access, we have adopted a variation of B-tree, called B*-tree.¹⁵ In a B*-tree all keys reside in the leaves. The upper levels, which are organized as a B-tree, consist only of an index. Leaf nodes are linked together; this allows easy sequential processing.

Each index is composed of one or more pages. A new page can be added to an index when needed as long as one of the pages within the area is marked available. Each page is a node and contains an ordered sequence of index entries. For nonleaf nodes, an entry consists of a (sort value, pointer) pair. The pointer addresses another page in the same structure, which may be either a leaf page or another nonleaf page. In either case the target page contains entries for sort values greater than the given one. For the leaf nodes, an entry consists of (sort value, DaId) pair. For the same sort value, the order of the entries are not defined. The leaf

pages are chained in a doubly linked list, so that sequential access can be supported from leaf to leaf.

LINK ORGANIZATION

A link is an access path which is used to connect data aggregates in one or two data objects. The DLI decides which data aggregates will be on a link and determines their relative position. The DSI maintains internal pointers so that newly connected data aggregates are linked to previous and next twins; previous and next twins are linked to each other when a data aggregate is disconnected.

A unary link involves a single data object and provides a partially defined ordering of data aggregates. Unary links can be used to maintain ordering specification (not value ordered) of data aggregates, which are not supported by DSI. It also provides an efficient access path through all data aggregates of an object without the time overhead of an internal page scan.

The more important access path is a binary link. It provides a path from single data aggregates (parents) in one object to sequences of data aggregates (children) in another object. The DLI determines which data aggregates will be children under a given parent, and the relative order of children under a given parent. A data aggregate may be parents and/or children in an arbitrary number of different links. The only restriction is that a given data aggregate can appear only once within a given link.

The main use of binary links is to connect child data aggregates to a parent data aggregate based on value matching in one or more fields. With such a structure the DLI can access data aggregates in one object based on the matching field in a data aggregate in a different object. This function is specially important for supporting relational join operations, and also for supporting navigational processing through hierarchical and network models of data. A striking advantage is gained over indices when the children are clustered on the same page as the parent. Another important feature is that links provide reasonably fast associative access without the use of an extra index.

The links are maintained in the DSI by storing DaIds in the prefix of data aggregates. New links can be defined at any time. When a new link is defined, a portion of the prefix is assigned to hold the required entries. An existing link can be dropped at any time. When this occurs, each data aggregate in the corresponding data object(s) is accessed by DSI in order to invalidate the existing prefix entries and make space available for subsequent link definition.

CONCURRENCY CONTROL

Concurrency is introduced to improve system response and utilization. But if several transactions are scheduled concurrently then the inputs of some transaction may be inconsistent even though each transaction in isolation is consistent.

If the database is read only then no concurrency control is needed. However, if transactions update shared data then their concurrent execution needs to be regulated. If all transactions are simple, and all data are in primary storage, then there is no need for concurrency. However, if any transaction runs for a long time or does I/O then concurrency control will be needed to improve responsiveness and utilization of the system.¹⁶

Concurrency must be regulated by some facility which regulates access to shared resources. Locking is introduced

to eliminate three forms of inconsistency due to concurrency: Lost Update, Dirty Read, Un-repeatable Read.

A lock is associated with each object in the database.¹⁷ Whenever using the object, transaction acquires the lock and holds it until the transaction is complete. Even when the object does not exist, it is locked to ensure that no other transaction will create such an object before the transaction terminates. In this case, the non-existence of the object is locked; such non-existent objects are called phantoms.

Locks are dichotomized as share mode locks (S-lock), which allow multiple readers of the same object and exclusive mode locks (X-lock) which reserve exclusive access to an object. Responsibility for requesting and releasing locks can either be assumed by the user or be delegated to the system. Our approach is to use automatic lock protocols which ensure protection from inconsistency, while still allowing the user to specify alternative lock protocols as an optimization.

Lock manager has two basic calls: Lock and Unlock. Their formats are as follows:

LOCK (lock), (mode), (control):
Locks a specified object.

UNLOCK (lock): Releases the specified lock.

(lock) is the resource name. (mode) is one of the modes specified. (control) can be either WAIT, in which case the call is synchronous and waits until the request is granted or is cancelled by the deadlock detector, or (control) can be TEST in which case the request is cancelled if it can not be granted immediately.

Since there are so many locks, it only allocates those with non-null queue headers (i.e. free locks occupy no space). Setting a lock consists of hashing the lock name into a table. If the header already exists the request enqueues on it, otherwise the request allocates the lock header and places it in the hash table. When the queue of a lock becomes empty, the header is deallocated.

A deadlock exists when every member of a subset of concurrent transactions is locked out of a resource or facility by one or more members of the subset; and each transaction must have that facility or resource in order to continue processing.^{18,19} As the user of a database system must have the possibility of enlarging his exclusive access rights in increments, deadlocks are unavoidable.

One easy solution to deadlock is to time out. Time out causes waits to be denied after some specified interval. As the system becomes congested, more and more transactions timeout. Also, timeout puts an upper limit on the duration of a transaction. In general, the dynamic properties of timeout make it acceptable for a lightly loaded system, but inappropriate for a congested system.

Deadlock prevention is achieved by requesting all locks at once, or requesting locks in specified order, or never waiting for a lock etc. In database context, deadlock prevention is a bad deal because one rarely knows in advance, what locks are needed, and consequently one locks too much in advance. Therefore, a database management system generally requires deadlock detection mechanism. Resolution of deadlock essentially becomes another source of backup.

RECOVERY MANAGEMENT

For recovery purposes, data models are classified in two groups: fixed length data objects and variable length data

objects. Fixed length data objects are matrices and fixed length vectors. In general, they are temporary and volatile in nature, i.e. they come into existence when some design program is run, and remain in the database while the design or analysis process is in progress. In structural analysis and optimization, they are typically stiffness matrix, damping matrix, mass matrix or iteration matrix of design variables and constraints.

Variable length data objects are variable length vectors and relations. In most cases, they are less volatile in nature and form permanent data in many scientific projects. They typically represent geometrical data, material property, general configuration data (structural, piping, etc.). Not only that these data change little during the life time of a project, they form part of the read only database after the project is over.

It is uneconomical and unnecessary to provide recovery facilities to fixed size objects, as they are either temporary or can be derived from other permanent data. Moreover, because of their large size, heavy I/O and volatile nature, recovery can be more expensive than to regenerate them, as may be required in worst case.

For recovery several methods are considered. Shadow mechanism²⁰ was originally considered for its simplicity.²¹ However, it consumes large amount of disk space to hold the shadow pages. In fact, considering the large I/O characteristics of scientific databases, this may require even 100% shadow overhead. Also, shadow mechanism is inefficient in a database which is concurrently accessed by a number of users.

The alternative mechanism adopted is 'write ahead log (WAL)' protocol.²² WAL requires that log records be written on secondary storage ahead of the corresponding updates. Further, it requires that undo and redo be restartable, i.e. attempting to redo a done page will have no effect and attempting to undo an undone page will have no effect either. This allows restart to fail and retry as though it were a first attempt.

In the event of a system failure which causes a loss of disk storage integrity, it must be possible to continue with a minimum of lost work. Such situations are handled by periodically making copy of the database state and keeping it in an archive. This copy and a log of subsequent activity can be used to reconstruct the current state. The archive mechanism periodically dumps a transaction consistent copy of the database to magnetic tape.

Although performing a checkpoint takes only a short time, dumping the entire database is a lengthy operation. Moreover, it is desirable that dump be taken when the system is in a quiesced state, such as at the time of controlled system shutdown. Gray²³ describes a 'fuzzy' dump mechanism which allows the database to be dumped while it is in operation.

EVALUATION

Currently, application programs access MIDAS using a well defined interface. The specification of this interface is given in MIDAS/N User's Manual.²⁴ The same interface is implemented on top of MIDAS/GR access method. As a result MIDAS/GR has become accessible to all applications that have been using MIDAS.

For the purpose of benchmarking MIDAS/GR and comparing it with MIDAS/N, an application program is chosen. This program solves a system of equations using the skyline method. The reason for choosing this program is that in

most engineering problems it is necessary to solve a system of equations, and the performance of the application depends critically on the performance of the equation solver.

Tables 1 and 2 gives some idea about the performance of MIDAS/GR. More detailed results are contained in the companion paper.¹ All tests are run on a DN460 Apollo computer with 2 MB of primary memory.

In general, MIDAS/GR is about twice as fast as MIDAS/N. Another interesting observation relates to the use of DBMS itself. It is a popular belief in the scientific field that one must make great sacrifice in efficiency to use DBMS (for other gains). Our result shows that with DBMS (MIDAS/GR) the program runs only about 10-12% longer than that without DBMS (database is in the virtual memory). This small increase in cpu time is acceptable in most cases.

At this time, MIDAS/GR does not manage its own disk or primary memory. The increased run-time is attributed to MIDAS/GR's dependence on the Operating System for memory management and disk space allocation. This leads to duplication of effort in memory management and inefficiency in allocation of disk space. We believe that MIDAS/GR will, in fact, improve efficiency in future when it manages its disk and buffer directly.

CONCLUSION

A properly designed DBMS can have enormous impact on the architecture of future software for complex engineering applications. As more computational power becomes available, the range of applications and complexity of the software increases. Extensions, maintenance and debugging of the software can be considerably facilitated by the use of a proper database and DBMS. Thus, a properly designed and implemented DBMS will be an invaluable tool for future software developments to solve complex-interdisciplinary engineering analysis and design problems.

Table 1. MIDAS/GR vs. MIDAS/N

Page size = 4 KB, Buffer size = 64 KB, Workspace size = 32 KB

No. of equations	CPU time in MIDAS/GR (s)	CPU time in MIDAS/N (s)	Ratio MIDAS/N : GR
1 000	878	1 639	1.87
5 000	4 933	9 560	1.94
10 000	9 670	18 362	1.90
20 000	20 316	37 360	1.84
30 000	30 465	55 562	1.82
40 000	40 641	77 935	1.92

Table 2. MIDAS/GR vs. Virtual Memory

Page size = 16 KB, Buffer size = 256 KB, Workspace size = 128 KB

No. of equations	CPU time in MIDAS/GR (s)	CPU time w/o DBMS (s)	Penalty for DBMS use (%)
1 000	314	281	11.7
5 000	1 680	1 502	11.9
10 000	3 390	3 023	12.1
20 000	6 831	6 193	10.3
30 000	10 289	9 241	11.3
40 000	13 747	12 228	12.4

Though the design of MIDAS is directly influenced by the current structural optimization applications, it possesses features which make it amenable to more general engineering applications. The main reason is that most of the needs of structural optimization typically represent requirements of many other engineering applications.

The new system presents an overall upgrade in all respects. Major advantages lie in data definition facility, memory management scheme, and storage layout. Unlike earlier systems, it has unified data definition for matrices and relations, while retaining their distinct features.

The improvement in memory management is implicit in faster program execution. It has reduced overhead on DBMS calls; therefore, application programmers need not burden themselves with the management of large work spaces. Also, buffer can be extended with the addition of new memory, resulting in improved performance.

APPENDIX A. DATA LANGUAGE SYNTAX

The following is a shortened version of the BNF syntax for MIDAS/GR data language. In this notation, square brackets [] indicate optional constructs, and the braces { } indicate repeating groups of zero or more items.

```
statement ::= query
           | dml-statement
           | ddl-statement
           | control-statement

query ::= query-expr [ ORDER BY ord-spec-list ]
query-expr ::= query-block | ( query-expr )
query-block ::= retrieve-clause [ INTO target-list ]
              FROM from-list [ WHERE boolean-expr ]
retrieve-clause ::= RETRIEVE [ UNIQUE ] ret-expr-list
                | RETRIEVE [ UNIQUE ] *
ret-expr-list ::= ret-expr { , ret-expr }
ret-expr ::= expr | var-name * | obj-name *
target-list ::= var-name { , var-name }
from-list ::= obj-name [ var-name ] { , obj-name [ var-name ] }
            [ { query-block } ]
ord-spec-list ::= field-spec [ direction ] { , field-spec }
               [ direction ]
field-spec ::= field-name
            | obj-name.field-name
            | var-name.field-name
direction ::= ASC | DESC
where-clause ::= WHERE boolean-expr
boolean-expr ::= boolean-term
              | boolean-expr OR boolean-term
              | boolean-expr EXOR boolean-term
boolean-term ::= boolean-factor
              | boolean-term AND boolean-factor
boolean-factor ::= ( NOT | boolean-primary
boolean-primary ::= predicate | ( boolean-expr )
predicate ::= expr comparison expr
            | expr comparison obj-spec
            | < field-spec-list > [ NOT ] IN full-obj-spec
            | obj-spec comparison full-obj-spec
full-obj-spec ::= obj-spec | literal
```

```
obj-spec ::= query-block
          | ( query-expr )
          | constant
field-spec-list ::= field-spec { , field-spec }
expr ::= arith-term
       | expr add-op arith-term
arith-term ::= arith-factor
            | arith-term mult-op arith-factor
arith-factor ::= [ add-op ] primary
primary ::= field-spec
         | fn ( ( UNIQUE ) expr )
         | fn ( * )
         | constant
         | ( expr )
comparison ::= comp-op | IN | NOT IN
comp-op ::= = | < > | > | >= | < | <=
add-op ::= + | -
mult-op ::= * | /
fn ::= identifier
constant ::= quoted-string | number | ROW | COL | NULL
          | USER | DATE
name ::= ( creator . ) identifier
creator ::= identifier
field-name ::= identifier { ( subscript-list ) }
subscript-list ::= subscript { , subscript }
subscript ::= constant | var-name
integer ::= number
dml-statement ::= assignment
               | insertion
               | deletion
               | update
assignment ::= ASSIGN TO receiver : query-expr
receiver ::= obj-name { ( field-name-list ) }
insertion ::= INSERT INTO receiver : insert-spec
insert-spec ::= query-expr
             | literal
field-name-list ::= field-name { , field-name }
deletion ::= DELETE obj-name [ var-name ] [ where-clause ]
update ::= UPDATE obj-name [ var-name ] set-clause-list
        [ where-clause ]
set-clause-list ::= set-clause { , set-clause }
set-clause ::= SET [ field-name = ] expr
            | SET [ field-name = ] ( query-expr )
ddl-statement ::= create-obj
               | create-image
               | create-link
               | define-view
               | drop
               | comment
create-obj ::= CREATE [ perm-spec ] [ shared-spec ] obj-defn
perm-spec ::= PERMANENT | TEMPORARY
shared-spec ::= SHARED | PRIVATE
obj-defn ::= matrix-defn | vector-defn
matrix-defn ::= MATRIX matrix-name ( integer , integer )
             [ qualifier ] : field-defn-list
matrix-name ::= identifier
qualifier ::= SPARSE | UPPER | LOWER | TRIDG | Banded integer
field-defn-list ::= field-defn { , field-defn }
```

```

field-defn ::= ( field-name ) ( type [ , NONULL ] [ , KEY ] )
type ::= primitive-type | structured-type ( : primitive-type )
primitive-type ::= INTEGER | REAL
                | DOUBLE PRECISION
                | CHARACTER
structured-type ::= STR ( Integer )
                | STR ( * )
                | VEC ( Integer )
                | VEC ( * )
                | MAT ( Integer, Integer )
vector-defn ::= VECTOR vector-name : field-defn-list
vector-name ::= identifier
create-image ::= CREATE [ Image-mod-list ] IMAGE image-name
              ON obj-name ( ord-spec-list )
Image-mod-list ::= Image-mod ( , Image-mod )
Image-mod ::= UNIQUE | CLUSTERING
create-link ::= CREATE [ CLUSTERING ] LINK link-name
              FROM obj-name ( field-name-list )
              TO   obj-name ( field-name-list )
              ( ORDER BY ord-spec-list )
drop ::= DROP system-entity name
comment ::= COMMENT ON system-entity name : quoted-string
          | COMMENT ON FIELD obj-name.field-name
          : quoted-string
system-entity ::= MATRIX | VECTOR | VIEW | IMAGE | LINK

```

ACKNOWLEDGEMENT

Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant Number AFOSR 82-0322. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

REFERENCES

- 1 Mukhopadhyay, S. and Arora, J. S. Implementation of an efficient run-time support system for engineering design environment, *Advances in Engineering Software* 1987, 9 (4), 000
- 2 SreekantaMurthy, T., Shyy, Y. K. and Arora, J. S. MIDAS-management of information for design and analysis of systems, *Advances in Engineering Software* 1986, 8 (3), 149
- 3 Comfort, D. L. and Erickson, W. J. RIM-A prototype for a relational information management system, *NASA Conference Publication 2055* 1978
- 4 Felippa, C. A. Database management in scientific computing - I. General description, *Computers Structures* 1979, 10, 53
- 5 Felippa, C. A. Database management in scientific computing - II. Data structures and program architecture, *Computers Structures* 1980, 10, 53

- 6 SreekantaMurthy, T. and Arora, J. S. Database management concepts in computer-aided design optimization, *Advances in Engineering Software* 1986, 8 (2), 88
- 7 SreekantaMurthy, T. and Arora, J. S. A survey of database management in engineering, *Advances in Engineering Software* 1985, 7 (3) 126
- 8 Mukhopadhyay, S. A database management system using generalized relational model, *Mater's Thesis in Dept. of Computer Science*, The University of Iowa, Iowa City, Iowa 52242, May 1986
- 9 Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D. et al. System R: Relational approach to database management, *ACM Trans. on Database Systems* 1976, 1 (2) (June), 97
- 10 Stonebraker, M., Wong, E., Kreps, P. et al. Design and implementation of INGRESS, *ACM Trans. on Database Systems* 1976, 1 (3), 189
- 11 Felippa, C. A. Architecture of a distributed analysis network for computational mechanics, *Computers and Structures* 1981, 13, 405
- 12 McKellar, A. C. and Coffman, E. G. Jr. Organizing matrices and matrix operations for paged memory systems, *Comm. of the ACM* 1969, 12 (3), 153
- 13 Knuth, D. E. *Art of Computer Programming*, Vol. 3, *Fundamental Algorithms*, Reading, Mass., Addison-Wesley, 1973
- 14 Bayer, R. and McCreight, E. Organization and maintenance of large ordered indexes, *Acta Informatica* 1972, 1, 173
- 15 Comer, D. The ubiquitous B-tree, *Computing Survey* 1979, 11 (2), 121
- 16 Munz, R. and Krenz, G. Concurrency in database systems - A simulation study, *Proc. ACM SIGMOD Int. Conf. on Management of Data* (August), pp. 111-120, 1977
- 17 Gray, J. N., Lorie, R. A. and Putzolu, G. R. Granularity of locks in a shared data base, *Proc. First International Conference on Very Large Data Bases* (September), pp. 428-451, 1975
- 18 Macri, P. P. Deadlock detection and resolution in a CODASYL-based data management system, *Proc. ACM SIGMOD Int. Conf. on Management of Data* (June), pp. 45-49, 1976
- 19 Coffman, E. G. Jr., Elphick, M. J. and Shoshani, A. System deadlocks, *ACM Computing Survey* 1971, 3 (2), 67
- 20 Lorie, R. A. Physical integrity in a large segmented database, *ACM Trans. on Database Systems* 1977, 2 (1), 91
- 21 Arora, J. S. and Mukhopadhyay, S. Specification for MIDAS-GR management of information for design and analysis of system: generalized relational model, *Technical Report No. CAD-SS-84.24*, Optimal Design Laboratory, College of Engineering, University of Iowa, 1984
- 22 Gray, J. N., McJones, P., Blasgen, M. et al. The recovery manager of the System R data manager, *ACM Computing Survey* 1981, 13 (2), 223
- 23 Gray, J. N. Notes on data base operating systems, *Research Report RJ2188* (February), IBM Research Lab., San Jose, Cal. 95193, 1978
- 24 Shyy, Y. K., Arora, J. S., Mukhopadhyay, S. et al. *MIDAS/N User Manual*, Optimal Design Laboratory, College of Engineering, University of Iowa, 1984

BIBLIOGRAPHY

- Shyy, Y. K., Mukhopadhyay, S. and Arora, J. S. A database management system for engineering applications, *Technical Report No. ODL-85.23*, Optimal Design Laboratory, College of Engineering, University of Iowa, 1985
- SreekantaMurthy, T., Shyy, Y. K., Mukhopadhyay, S. and Arora, J. S. Evaluation of database management system MIDAS in equation solving environment, *Engineering with Computers* 1987, 2 (1), 11

APPENDIX 10

**ROLE OF DATA BASE MANAGEMENT IN
DESIGN OPTIMIZATION SYSTEMS**

by

G.J. Park and J.S. Arora

in

J. Aircraft

Vol. 24, No. 11, 1987

Role of Data Base Management in Design Optimization Systems

G. J. Park* and J. S. Arora†
University of Iowa, Iowa City, Iowa

To study the role of data base and data base management system, an interactive design optimization software system called IDESIGN5 is developed to solve nonlinear programming problems. Four promising algorithms are included to overcome the lack of unanimous choice of an algorithm. Tuning parameters and procedures of algorithms are implemented through extensive numerical experimentation. The interactive process consists of menu displays, advice for decisions, and applicable messages. Input data can be created interactively and the designer can change problem parameters, algorithm, and design variable data at any point of execution. If a design variable does not effect the optimization process, it can be given a fixed value interactively. Discrete variable optimization can be performed by using design variable status capability of the system. Graphics facilities are provided for decision making. The system consists of several modules that communicate with each other through a data base managed by a data base management system. Several example problems are solved in batch and interactive environments to test the system.

Introduction

THE nonlinear programming problem (NLP) in the finite-dimensional space is to minimize a cost function $f(b)$ for $b \in S$, where S is a subset of R^n defined as

$$S = \{b: g_i(b) = 0; i = 1, m'; g_i(b) \leq 0; i = m' + 1, m;$$

$$b_{iL} \leq b_i \leq b_{iU}; i = 1, n\}$$

and $g_i(b)$ are constraints, b a design variable vector, and b_{iL} and b_{iU} the smallest and largest values allowed for the i th design variable, respectively.

The problem has been treated in the nonlinear programming literature quite extensively. In that material, it is usually assumed that the explicit form of the function $f(b)$ and $g_i(b)$ is available. However, if we allow the functions to be implicit, then several classes of structural and mechanical system design problems can be described by the model.¹

A number of nonlinear programming algorithms have been developed and used for optimal design.²⁻⁴ However, software development for design optimization is lagging behind these advances.⁷⁻¹¹ It is well known that some algorithms when implemented in a computer program do not behave in the way they are theoretically supposed to. Considerable expertise and numerical experimentation is needed to properly implement an algorithm. In addition, no single algorithm can efficiently solve all classes of problems. Instead, a powerful software system with various capabilities and facilities is desired to minimize the difficulties. So far, software for design optimization is either not available or it is tedious to use. Most of the existing programs are not interactive. This means that the program does not allow any user control over the iterative pro-

cess. Graphics capabilities is almost unheard of in design optimization. The software should have these facilities and options for various controls and decision making. Also, as range of applications of optimization expands, more complex data are generated which must be handled properly through a data base management system (DBMS). It should be possible to easily refine existing capabilities and add new ones. To allow various options, several good algorithms should be implemented.

To study the role of an organized data base and a data base management system, a general purpose software system called IDESIGN5 is introduced in this paper. It is an interactive design optimization system incorporating modern data base management concepts. The system has the foregoing capabilities and is an extension of an earlier program called IDESIGN3.¹⁰ That program does not use any data base management system and has no out-of-core calculations. The new system is designed using modern software development and data base management concepts.

Selection of Algorithms for IDESIGN5

Although many algorithms are available for solving nonlinear optimization problems, it is difficult to find an algorithm that solves all classes of the problems efficiently. In IDESIGN5, several good algorithms have been selected. These are Pshenichny's algorithm (LINRM),² cost function bounding algorithm (CFB),⁴ modified Pshenichny's algorithm (PLBA),³ and a hybrid method.⁶ These algorithms are selected based on their generality, robustness, and efficiency. They have performed reasonably well on a wide range of applications.

The detailed procedures of the selected algorithms have been published in the literature.¹⁻⁴ All algorithms roughly implement the following steps:

- 1) Set $k=0$; estimate starting design variables; initialize parameters.
- 2) At the k th iteration, identify a potential constraint set. Define and solve a subproblem for the search direction.
- 3) Check the convergence criteria. If the criteria are satisfied, the process is stopped.
- 4) Check the condition for the progress of the algorithm and calculate a step size, update the Hessian of Lagrangian if necessary.
- 5) Update the design variables. The process is then restarted from step 2.

Presented as Paper 86-0991 at the AIAA/ASME/ASCE/AHS 27th Structures, Structural Dynamics and Materials Conference, San Antonio, TX, May 21-23, 1986; received April 4, 1986; revision received March 15, 1987. Copyright © American Institute of Aeronautics and Astronautics, Inc., 1987. All rights reserved.

*Graduate Research Assistant, Optimal Design Laboratory, College of Engineering (currently Assistant Professor, CAD/CAM Center, Purdue University at Indianapolis, IN).

†Professor, Civil and Mechanical Engineering, Optimal Design Laboratory, College of Engineering. Member AIAA.

Role of Data Base in Design Optimization

IDESIGN5 implements the major steps of the selected algorithms in separate modules. A module is a program that can be compiled and executed independently. The sequence of execution of modules is controlled by the main module. The modules communicate with each other through the data base. Interaction is allowed during the execution of the system from the beginning to the end. The designer can control the progress by changing the problem data and algorithm at any point.

Interactive Capabilities in IDESIGN5¹²

Design optimization programs require information about the problem to be solved. This information includes: 1) input data, such as number of design variables, number of constraints, etc.; 2) cost and constraint functions; and 3) gradients of cost and constraint functions. If the gradient expressions are not supplied, the system produces them by the finite-difference method. When there is a mistake in the input data or problem definition, errors will occur in the solution procedure. The system gives explicit error messages to guide the designer to correct mistakes.

The system allows for interactive data entry. A menu is displayed for selection of proper data segment and entry. The system is protected from user's mistakes. If a data mismatch is found, messages are given in detail. The procedure is simple, so even a beginner can easily follow it.

It is extremely useful and important to monitor the optimal design process through the interactive session. Histories of the cost function, constraint functions, design variables, maximum constraint violation, and convergence parameter can be monitored. When the histories are graphically displayed, they are of great help in interactive decision making as well as for gaining insight into the design process. Design sensitivity coefficients of the cost function and potential constraints are displayed in the form of normalized bar charts. This information shows relative sensitivity of the design variables. If the design process is not proceeding satisfactorily (there could be inaccuracies or errors in the problem formulation and modeling), it is necessary to stop it and check the formulation of the problem. This will save human as well as computer resources. Also, if one algorithm is not progressing satisfactorily, then a switch can be made to another one or the process can be restarted from any previous design. The system can give suggestions for design changes by using the design sensitivity coefficients. It is also possible to utilize the software in a batch environment. In this case, the system uses default values for the parameters found to be best through experience and numerical experimentations.

Specification of the Status of Design Variables

In the optimization process, some design variables have little effect on the final solution of the problem. Also, in many cases, some design variables reach their optimum value and have little effect on the iterative process after that point. Thus, we need to identify such design variables and fix them. In this way, the problem is defined by only a few important variables and its size is reduced. When a fixed variable becomes important in the design process, it can be released to the active group. The status of design variables can be defined interactively in IDESIGN5.

The capability to specify the status of the design variables can be exploited in practice to obtain a discrete optimum solution for the problem. When we design a structural or mechanical system, the member sizes are the design variables. Members must be selected from an available set, but design variables are considered continuous in the optimization process. This is one of the reasons that optimization has not been extensively used in practical engineering design. The difficulty can be overcome by defining the status of design variables. If a design variable becomes close to an available size, it can be

fixed to that value interactively. This also means that the designer can use his intuition in selecting discrete member sizes. This is very important because an expert designer's experience is considered to be extremely valuable and reliable in the engineering design community. Until the optimum is found, design variables can be assigned fixed values nearest to the ones available. The graphical display of various data can help the designer in decision making.

An important point is *when or how* to determine the status of a design variable. The ideal solution would be to automatically determine the status using a sophisticated algorithm. This can be done by pattern recognition of the trend, just as an expert designer would do. However, such a solution requires knowledge of engineering capabilities that is a topic of future research. In IDESIGN5, status of a design variable can be changed only interactively.

Role of Data Base in IDESIGN5

In the design optimization process, large amounts of data must be generated and manipulated. Specifically, the interactive process needs enormous amount of data, so a proper data base and a data base management system (DBMS) are essential. In addition, since IDESIGN5 is composed of independent modules, a DBMS is needed to communicate between the modules. A data base for IDESIGN5 is designed and managed by a DBMS, as described in the next section. This systematic generation and handling of large amount of data, along with the modular structure, has greatly facilitated the coding and debugging of the system.

Data Base Management in IDESIGN5

The data base is a centralized collection of data accessible to several application programs and optimized according to a data base definition schema. The data base management system (DBMS) is a software program handling all access requests and transactions against the data base.

In practical design optimization, finite-element and other numerical methods are adopted during analysis of structural and mechanical systems. In general, the amount of data used in finite-element analysis is quite large and various schemes are used to reduce the storage of data. In addition, the formulation of constraints and the design sensitivity analysis are carried out using most of the data generated during the analysis phase. Interactive computations and graphics also require additional data for the display of the system model and intermediate results. Therefore, the data must be organized and saved properly in a data base for efficient design optimization.

Implementation of Data Base in IDESIGN5

In IDESIGN5, a data base management system called MIDAS/N is used for handling the data.¹³ MIDAS/N stands for the management of information for design and analysis of systems/numerical model. MIDAS/N is an application-independent data base management system and uses a different approach to deal with data management problems of scientific and engineering computing. It is designed specifically to handle numerical data. A data base of MIDAS/N is stored in a file. It can be a direct or a sequential access file. The status of a data base can be temporary or permanent. Each data base contains several data sets, which can be either one- or two-dimensional arrays (vectors or matrices). The matrix can be rectangular or triangular. The data type of a data set can be character, short integer, long integer, real, or double precision real. Each data set has several access models, such as row, column, or submatrix. Any part of a data set can be accessed by just one call statement. The dimension of the data set can be redefined dynamically.

At the beginning of each module in IDESIGN5, the necessary data are retrieved from the data base. Generated data are stored in the data base during the execution of the

module. Connection with the data base needs additional CPU time; IDESIGN5 is designed to minimize the additional time. Usually, design optimization data are in the form of matrices or vectors. One matrix is defined as a data set, and several vectors having the same dimensions are included in a data set. For interactive graphics, the history of the design data is stored in the data base and retrieved when it is used.

Design of a Data Base

Collection of Information

To design a data base for IDESIGN5, the flow of the optimization data for various algorithms must be studied. Various vectors, matrices, and control parameters that are either input or generated during the solution process must be identified. All of the data must be collected and properly grouped to define the data base. In every module, these data are retrieved from the data base and stored in a common array for further processing. Table 1 lists the data for all the modules. Data are stored or retrieved using the variable names shown in the table. The variable names start with different letters according to their dimensions. One or several optimization data are stored in a data set.

Data Set Definition

By the characteristics of the optimization data, the data sets can be defined. One data set represents a matrix or several vectors having the same dimensions. If several vectors are included in a data set, they are stored like a matrix. The list of data sets is shown in Table 2. In some cases, a data set is stored by row but retrieved by column, or vice versa. These are also shown in Table 2. A permanent data base FPERM is defined at the beginning of the execution of the system and used in each module.

Design of IDESIGN5

Structure of IDESIGN5

A general-purpose optimization software program can have an overall structure as shown in Fig. 1. Each block can be composed of several modules. Figure 2 shows the structure of IDESIGN5 and the sequence in which the modules are ex-

ecuted. Each block is a program that can be compiled and executed independently. As shown in Fig. 2, the modules have two libraries, IDESIGN5.LIB and SMART.¹⁴ IDESIGN5.LIB contains some subroutines common to many IDESIGN5 modules. SMART is a library containing several general-purpose interactive aids and subroutines for matrix and other standard operations. Some modules are used in one algorithm, while others are shared by many. When the modules are combined into one system, a main program controls the flow of progress based on the decisions made in every module. When each module is used independently, the operating system capabilities are used for controlling the flow of the calculations.

Each module has a buffer array for vector and matrix data and it is distributed to each subroutine according to the needs. A common array is declared for the problem parameters used in all modules. At the beginning of a module, the problem parameters are retrieved from the data base.

Definition of Optimization Problem

In this block of Fig. 1, the designer has to provide two types of information to use the system: input data for the problem

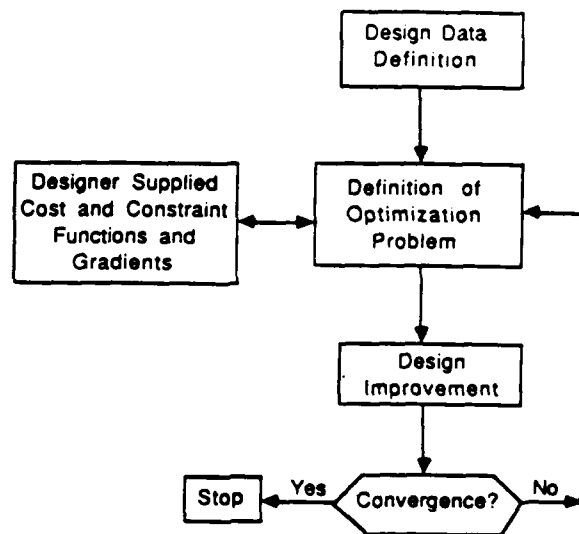


Fig. 1 Overall structure of a design optimization program.

Table 1 List of data

Data description	Type	Dimension ^a	Variable name
Cost function gradient	Vector	nv	ACOGR
Values of design variables	Vector	nv	ADEVL
(Hessian matrix)* design change	Vector	nv	AHXDB
Increment of design variable	Vector	nv	AINCT
Gradient of Lagrangian at previous iteration	Vector	nv	ALOLD
Lower bound of design variable	Vector	nv	ALOWR
Upper bound of design variable	Vector	nv	AUPPR
Lagrange multipliers of QP subproblem	Vector	nv + nc	CLMQP
Constraint function value	Vector	nc	CONFN
Lagrange multipliers	Vector	nc	CORLM
Constraint gradient lengths	Vector	nc	CSENM
History of constraint functions	Matrix	itrs x nc	DCOFN
History of convergence parameter	Vector	itrs	DCONP
History of cost function	Vector	itrs	DCOST
History of design variables	Matrix	itrs x nv	DEIVL
History of maximum constraint violation	Vector	itrs	DMAXV
Active constraint set	Vector	nc	LC
Design variable status	Vector	nv	NFIX
Active set of previous iteration	Vector	nc	NPREC
Constraint sensitivity matrix	Matrix	nv x nc	TSENM
Hessian matrix	Vector	nv x (nv + 1) + 2 + nv	UPPAT

^a nv = number of design variables, nc = number of constraints, itrs = number of iterations.

Table 2 Structure of data base FPERM

Data set name	Included data (variable name)	Dimension ^a	Order of creation	Order of use
SDINV	ACOGR, ADEVL, AHXDB, AINCT, ALOWR, AUPPR, NFIX, NORDR	nv x 9	Col	Col
SDINC	CONFN, CORLM, CSENM	nc x 3	Col	Col
SDINQ	CLMQP	nv + nc	Col	Col
SSENM	TSENM	nv x nc	Col	Col, row
SUPDT	UPPAT	nv(nv + 1) + 2 + nv	Col	Col
SHISA	DCOND, DCOST, DMAXV	itrs x 3	Col	Col
SHISB	DEIVL	itrs x nv	Col	Row
SHISC	DCOFN	itrs x nc	Col	Row
SINTG	LC, NPREC	nc x 2	Col	Col
SSYST	System parameters	300	Col	Col

^a nv = number of design variables, nc = number of constraints, itrs = number of iterations.

and definition of the design problem. The input data for a problem can be provided interactively from a terminal through menu displays or from an input file. The definition of the design problem in Fig. 1 is provided by the designer through four independent modules. These are COSTFN, CONSTFN, COSTGR, and CONSTGR, as shown in Fig. 2. They contain the cost function, constraint functions, cost function gradient, and constraint function gradients, respectively. These modules can be very simple or very complex depending on the application. The modules calculate function values and gradients and store them in the data base. They can be written using any utilities of the computer. If the analytic expressions for the gradients are not available, IDESIGN5 can automatically calculate the gradients by the finite-difference method. In this case, the modules for gradient calculations need not be supplied.

Modules Suitable for Many Algorithms

The modules shared by every algorithm are as follows.

Input

This module is the first one executed in IDESIGN5 system. All input data for the design problem are defined. Input data consist of five groups: 1) initial terminal session—file definitions and terminal type; 2) problem definition—numbers of design variables and constraints, etc.; 3) parameter definition—convergence parameters, printing code, etc.; 4) design variable data—initial design, lower and upper bounds and status of design variables; and 5) algorithm selection. Each group of data is defined using a menu display. Between two consecutive menus, the designer can change the data already defined or the program can be terminated. Also, HELP facility is available at any point of process. Various menus used during the interactive session are given in Park and Arora.¹²

Interactive

By designer's option, the interactive module can be executed during any iteration. Features of this module are similar to the INPUT module. The designer can control the progress of the problem-solving procedure and problem data can be changed at any point in the execution. Proper menus are displayed for

changing the algorithm or the current data. To help the designer's decision, graphical displays for various data are available. The histories of parameters are plotted. Relative sensitivities for cost and constraint function gradients are shown as bar charts. Advice for design-variable change can be given using linear extrapolation. When the design point is in the infeasible region, the relation between the correction of constraint violation and cost function improvement is used to advise the designer by linear extrapolation of the sensitivity information.

Active

The potential constraint subset is defined in this module. The potential constraints identified here are used to define a subproblem that determines the search direction in the design space.

Subsolver

Normalized⁴ subproblems are solved for the search direction using the subroutine QPSOL¹³ in this module. Fixed design variables are dropped.

Modules for Problem Definition

As mentioned earlier, the COSTFN, COSTGR, CONSTFN, and CONSTGR modules should be prepared for cost function, cost function gradient, constraint functions, and constraint function gradients, respectively. Any data in the data base can be used, and the produced data can be stored in the data base.

Modules for LINRM and PLBA Algorithms

These modules are used in LINRM and PLBA methods and the hybrid method after a switch to the PLBA algorithm is made.

RQPSTEP

In this module, step size is determined by a one-dimensional search that requires evaluation of the cost and constraint functions. These are calculated using the designer-supplied modules.

RQP

Data that the subproblem will use to determine a search direction are generated.

HESSUPT

The Hessian matrix of the Lagrange function is updated in the PLBA algorithm in the factorized form. Through internal options, the Hessian can be set to an identity matrix and a direct update can be obtained when the full matrix is updated. When the condition number of the Hessian matrix is over a certain positive large number, it is set to the identity matrix.

RQPUP

From the output of the subsolver, the design increment vector is determined and convergence criteria are checked.

Modules for CFB Algorithm

These modules are used in the CFB algorithm and hybrid method before switching.

CFBSTEP

Here, the maximum constraint violation is calculated. From the second iteration onward, the step size determined in the CFBUP module is adjusted to remain within the bounds on the design variables.

CFB

Here, the data for the subproblem are calculated according to the current design conditions.

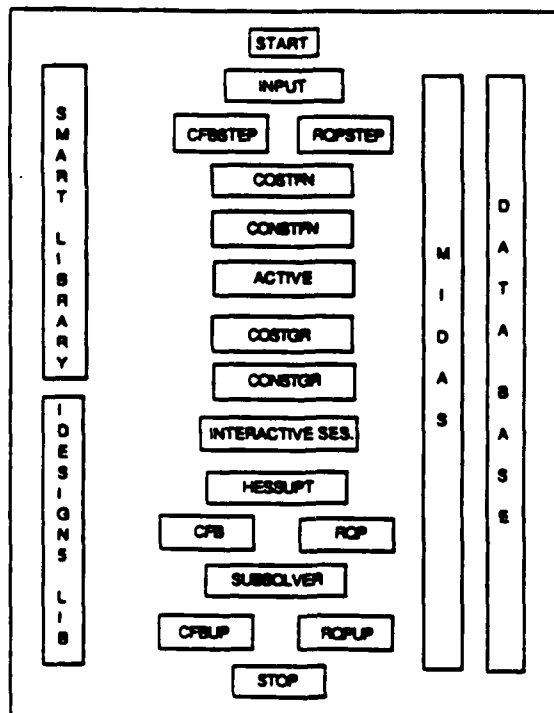


Fig. 2 Structure of IDESIGN5 and its modules.

Table 3 Interactive execution of 25 bar truss design problem^a

Iteration	Interactive change			
1	MVC = 2.7207			
	Algorithm = R3			
2	MVC = 0.4780			
	PLBA algorithm is used from this iteration			
7	No.	Value	Given value	Status change
	1	0.01	1.62	Fixed
	2	1.4831	1.80	Fixed
	4	0.0076	1.62	Fixed
	7	2.20765	3.13	Fixed
10	3	2.0093	2.88	Fixed
	7	3.13		Released
12	6	1.9958	2.38	Fixed
13	5	1.085928	1.80	Fixed
15	Optimum is found			
	MVC = 0.5693E - 06			
	ND = 0.3034E - 07			
	Cost function = 0.7744E + 03			
	CPU for data base = 29.015			
	Net CPU = 49.796			
	Total CPU = 78.811			
	No. of calls for function evaluation = 25			
	No. of total gradient evaluations = 163			
	Design variable values: 1.62, 1.8, 2.88, 1.62, 1.8, 2.38, 3.6295			

^aMVC = maximum constraint violation, ND = norm of direction vector.

CFBUP

The design increment vector is calculated and the convergence criteria are checked.

Sample Applications

Performance of nonlinear programming algorithm can be ascertained only by numerical experiments requiring collection and implementation of the test problems. In IDESIGN5, the procedures of algorithms and tuning parameters are implemented through extensive numerical experiments.¹²

Many numerical example problems have been solved using the algorithms available in IDESIGN5. These include 115 test problems from the mathematical programming literature, small-scale engineering design problems, structural design problems such as trusses and frames, dynamic response optimization, and nonlinear response optimization. Details of all these applications are given in Ref. 12. Here we describe some details of applications to trusses and the interactive use of IDESIGN5.

The seven trusses, given in Refs. 16 and 17, are optimized for various constraint conditions, such as stress, displacement, and natural frequency. Combination of trusses and constraint conditions yields 15 problems. The formulation, design data, and sensitivity analysis have been discussed in the literatures.^{3,4,17} The number of design variables are 7-47, state variables 8-150, and constraints 18-629. Multiple loading cases are treated.

For analysis of the structure and gradient calculation, a computer program TRUSSOPT is employed. TRUSSOPT is composed of four subroutines, each of which is attached to a designer-supplied module in IDESIGN5. TRUSSOPT does not use a data base management system.

Results for each problem in a summary form are given in Ref. 12. The data collected for each problem include the number of iterations, function evaluations, and gradient evaluations, CPU times for data base management and total execution, and optimum point. For the purpose of comparison, results of the PLBA algorithm in IDESIGN3 are also included. IDESIGN3 is the previous version of IDESIGN5 that does not use a data base management system. CFB fails

on three problems and LINRM exceeds the maximum iteration limit in six problems. As expected, the performance of the hybrid and PLBA methods is superior to others. This was also observed in small-scale problem applications. The results of PLBA and the hybrid methods are similar to those of IDESIGN3. If we compare IDESIGN5 to IDESIGN3, the number of iterations increases in five problems and decreases in eight problems. The number of calls for function evaluations increases in six problems and decreases in eight problems. The total CPU time of IDESIGN5 is generally larger than that for IDESIGN3. Two main reasons cause the increase in IDESIGN5: 1) we need computing time for access to the data base and 2) the data are redefined at the beginning of each module. For example, the performances of PLBA and IDESIGN3 are exactly the same in the 25 bar truss with stress constraints except the computer time. For the 47 bar truss with all constraints, the number of calls for function evaluations with PLBA is greater than that of IDESIGN3. However, the total CPU time of IDESIGN5 is less than that for IDESIGN3. This is because the QP solver (QPSOL) in IDESIGN5 is more efficient than the QP solver (VEO4AD in the Harwell library) in IDESIGN3. Therefore, the algorithm and the software for solving the subproblem are also very important. When the structure becomes larger, the CPU time for the data base comprises a smaller percentage of the total CPU time. This is because the analysis needs a greater portion of the computing time and TRUSSOPT does not use a data base and DBMS.

Interactive Use of IDESIGN5

The 25 bar truss given in Refs. 16 and 17 is interactively optimized with all constraints. In practical optimization, the design variables are usually member sizes that must be selected from the available sections. Thus, discrete variable optimization should be employed. The present structure is designed to have double-angle sections given in the AISC code. The area of each member is considered as a design variable. The areas available in the AISC code are 1.62, 1.80, 1.99, 2.13, 2.38, 2.62, 2.63, 2.88, 2.93, 3.09, 3.13, 3.38, 3.47, 3.55, 3.63, 3.84, 3.87, 3.88, 4.18, 4.22, 4.49, and 4.59. The design process summarized in Table 3 is explained as follows:

Iteration 1

Maximum violation of constraints is very large, so constraint correction step R3 of CFB algorithm is chosen to correct constraints.

Iteration 2

Maximum violation is corrected; PLBA algorithm is selected from this iteration to the end.

Iteration 7

The design variables that are not changing during the iterative process are assigned values closest to those available. The history of the design variables is shown on a graphics terminal for this decision.

Iteration 10

Design variable 3 is fixed and design variable 7 is released. The sensitivity bar charts are exploited for this decision; that is, a sensitive design variable is released to the active status.

Iteration 12

Design variable 6 is fixed.

Iteration 13

Design variable 5 is fixed.

Iteration 15

The optimum is found. All the design variables have fixed values except the seventh. After optimum is found, design

variable 7 is given the value 3.84. With these fixed values, there are no violated constraints. Therefore, these values can be taken as the final solution.

At the final point, all the design variables have discrete values. The cost function ($0.774E+03$) is higher than that ($0.5907E+03$) obtained earlier.¹² The reason is the lower bound used there is 0.01, while the smallest available size is 1.62. The optimum cost function with 1.62 as the lower bound and continuous design variables is $0.7161E+03$ with the design as 1.62, 2.1478, 2.3341, 1.62, 1.9737, and 3.4309. Thus, the cost function increases by 8.1% when discrete design variables are used.

Use of IDESIGN5 with ADINA

The IDESIGN5 system has been also used to optimize truss structures with nonlinear response. A finite-element-method software system called ADINA is used for nonlinear analysis.¹⁸ Different modules are developed for cost and constraint functions and sensitivity information. The formulation, analysis method, and design data are given in detail by Haririan et al.¹⁹ Both geometric and material nonlinearities are included and many problems are solved. Thus, coupling of IDESIGN5 to a commercial analysis software has been successfully accomplished.

Discussion and Conclusions

A software system can be evaluated only by numerical tests. Numerous problems have been solved in batch as well as interactive environment, indicating that IDESIGN5 is a reliable software system. The conclusions based on the study are as follows:

1) A software system IDESIGN5 having diverse optimization capabilities has been developed. A data base for the system is designed and a data base management system is used to handle the data base. Procedures to integrate the data base management capabilities are developed and demonstrated. Use of a DBMS facilitated development of the system considerably.

2) The system can be readily expanded or combined with other systems. The combination of IDESIGN5 and ADINA illustrates this capability.

3) The algorithms in IDESIGN5 are implemented by numerical experimentation. Various problems have been effectively solved. In terms of the performance, PLBA and hybrid methods show better results.

4) Interactive execution shows good performance for the optimization process. Interactive change of problem parameters and design variable data can be exploited in problem solving. Experience with IDESIGN5 shows that a design optimization software must support interactive and graphics facilities for practical applications.

5) Discrete optimization is obtained by interactive definition of the status of the design variables. Expert designer's intuition can be thus exploited, making the system extremely useful in practical design optimization.

6) Data base and data base management systems have important roles in a powerful and user-friendly interactive design optimization system. The data base facilities can be used to collect raw data from which knowledge can be derived. Such knowledge can then be incorporated into an expert system that can act as a consultant and trouble-shooter during the design optimization process. This will be a useful extension of the system.²⁰

7) Data base generation and management needs additional computational effort. However, as the problem size increases, the CPU time for data management decreases relative to the total CPU time.

Acknowledgment

This research is sponsored by the U.S. Air Force Office of Scientific Research, Air Force Systems Command under Grant AFOSR82-0322. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes, notwithstanding any copyright notation thereon.

References

- Arora, J. S. and Thandar, P. B., "Computational Methods for Optimum Design of Large Complex Systems," *Computational Mechanics*, Vol. 1, No. 2, 1986, pp. 221-242.
- Belegundu, A. D. and Arora, J. S., "A Recursive Quadratic Programming Method with Active Set Strategy for Optimal Design," *International Journal for Numerical Methods in Engineering*, Vol. 20, No. 5, 1984, pp. 803-816.
- Lim, O.K. and Arora, J. S., "An Active Set RQP Algorithm for Engineering Design Optimization," *Computer Methods in Applied Mechanics and Engineering*, Vol. 57, 1986, pp. 51-65.
- Arora, J. S., "An Algorithm for Optimum Structural Design Without Line Search," *New Directions in Optimum Structural Design*, edited by E. Atrek et al., Wiley, New York, 1984, Chap. 20.
- Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design: With Applications*, McGraw-Hill, New York, 1984.
- Thanedar, P. B., Arora, J. S., and Tseng, C. H., "A Hybrid Optimization Method and Its Role in Computer-Aided Design," *Computer and Structures*, Vol. 23, No. 3, 1986, pp. 305-314.
- Gabriele, G. A. and Ragsdell, K. M., "Large Scale Nonlinear Programming Using the Generalized Reduced Gradient Method," *Journal of Mechanical Design*, Vol. 102, No. 3, 1980, pp. 566-573.
- Vanderplaats, G. N., Sugimoto, H., and Sprague, C. M., "ADS-1: A New General Purpose Optimization Program," AIAA Paper 83-0831, 1983.
- Balling, R. J., Parkinson, A. R., and Fret, J. C., "OPT-DES.BYU: An Interactive Optimization Package with 2D/3D Graphics," *Recent Experiences in Multidisciplinary Analysis and Optimization*, edited by J. Sobieski, NASA CP 2327, 1984.
- Arora, J. S., Thanedar, P. B., and Tseng, C. H., "User's Manual for IDESIGN," Version 3.5, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, Tech. Rept. ODL-86.6, 1986.
- Atrek, E., Callagher, R. H., Ragsdell, K. M., and Zienkiewicz, O. C. (eds.) *New Directions in Optimum Structural Design: Proceedings of International Symposium on Optimum Structural Design*, 1981, Wiley, New York, 1984.
- Park, G. J. and Arora, J. S., "Interactive Design Optimization with Modern Database Management Concepts," Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, Rept. ODL-86.9, 1986.
- SreekantaMurthy, T., Shyy, Y. K., and Arora, J. S., "MIDAS: Management of Information for Design and Analysis of Systems," *Advances in Engineering Software*, Vol. 8, No. 3, 1986, pp. 149-158.
- Arora, J. S., Lee, H. H., and Jao, S. Y., "SMART: Scientific Database Management and Engineering Analysis Routines and Tools," *Advances in Engineering Software*, Vol. 8, No. 4, Oct. 1986, pp. 194-199.
- Gill, P. E., Murray, W., Saunders, M. S., and Wright, M. H., *User's Guide for QPSOL*, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University, Stanford, CA, Rept. SOL 84.6, 1984.
- Thanedar, P. B., Arora, J. S., Tseng, C. H., Lim, O. K., and Park, G. J., "Performance of Some SQP Algorithms on Structural Design Problems," *International Journal for Numerical Methods in Engineering*, Vol. 23, No. 12, 1986, pp. 2187-2203.
- Haug, E. J. and Arora, J. S., *Applied Optimal Design: Mechanical and Structural Systems*, Wiley-Interscience, New York, 1979.
- Bathe, K. J., "On the Current State of Finite Element Methods and Our ADINA Endeavours," *Advances in Engineering Software*, Vol. 2, No. 2, 1980, p. 59.
- Haririan, M., Cardoso, J. B., and Arora, J. S., "Use of ADINA for Design Optimization of Nonlinear Structures," *Computers and Structures*, Vol. 26, No. 1/2, 1987, pp. 123-134.
- Arora, J. S. and Baenziger, G., "Uses of Artificial Intelligence in Design Optimization," *Computer Methods in Applied Mechanics and Engineering*, Vol. 54, 1986, pp. 303-323.

APPENDIX 11

**AN INTEGRATED DATABASE MANAGEMENT SYSTEM FOR
ENGINEERING APPLICATIONS BASED ON AN
EXTENDED RELATIONAL MODEL**

by

J.S. Arora and S. Mukhopadhyay

To Appear in

Engineering with Computers, 1988

ABSTRACT

Engineering analysis and design of complex systems require the use of large software components. Development, maintenance, and extension of such a software system needs modern design and implementation techniques. Usually a large amount of data is generated. Flow of data is also quite complex adding further complications in maintenance and extension of the software. A sophisticated database management system is needed to support data handling during the run-time environment as well as for the integration of various software components. Design and development of such a DBMS needs new concepts and ideas such that efficiency of calculations is not sacrificed. Degradation in efficiency due to the use of a DBMS can hinder large scale applications. The paper describes a generalized relational model to handle large matrices and tables that are encountered in numerous engineering applications. A DBMS based on the model is designed and implemented. The system supports run-time data management as well as data sharing between various software components. A preliminary evaluation of the system against some existing ones reveals the new concept and design to be quite appropriate for engineering applications. The system is very efficient and compact. Some details of design and performance of the system are given and discussed.

INTRODUCTION

Management of information has become an extremely important task in computer-aided design and analysis of engineering systems. Organization of large volumes of design information is a complex task and requires careful consideration [1]. Several systems for management of information are available for engineering applications [2]. Programs such as FILES [3], RIM [4], SDMS [5], PHIDAS [6] and TORNADO [7] have been used in some engineering applications.

These systems have been developed with varying degree of sophistication and have a variety of capabilities. A study was made to find out the capabilities and usefulness of the existing data management systems for design and analysis applications [2]. It was found that the use of such systems is limited to special applications for which they were developed. It is difficult to modify and extend them for design applications. Similarly, systems like RIM were found quite useful in integrating general engineering analysis programs; but their applicability for finite element analysis and design optimization is limited [8]. Thus, a need for a good data management system which can deal with organization of both design and analysis data exists [9]. This need for integrated database management is further discussed in the next section.

The paper describes broad perspectives of research issues in the design and implementation of a new integrated database management system. Architecture and storage organization of such a system are discussed. The new concept of a generalized relational model is introduced which forms the basis for design of a new database management system (DBMS). The system is named MIDAS/GR - Management of Information for Design and Analysis of Systems/Generalized Relational model. The paper also contains some results of running applications using the new system. The system is almost two times faster than MIDAS/N [9,10], an existing DBMS handling only numerical data. Results are also comparable in efficiency when the program is run without the DBMS (smaller problem that fits in operating systems virtual memory).

NEED FOR INTEGRATED DATABASE MANAGEMENT

Databases can be broadly classified into two types: tabular and numerical. A tabular database contains data sets that look like tables (relations). Such databases can be quite large and usually contain permanent type of data.

Usually the amount of data processed at one time, and the disk I/O are quite small. These include databases for such industries as banking, airline, transportation, etc. Several DBMS for processing these databases have been developed. They generally have interactive facilities for query, insertion, deletion and update of the database.

Numerical databases contain matrix type of data sets. These databases are usually temporary and can be very large. Usually the amount of data processed and the disk I/O can be very large. Such databases are generated during engineering analysis and design optimization, numerical analysis, and other scientific applications. As noted earlier, several DBMS of varying sophistication to process numerical databases have been developed.

Most scientific and engineering applications need and use both tabular and numerical databases. For example, in structural design and optimization applications, the finite element model and the analysis results can be represented in a tabular database which is permanent or semi-permanent. However, during analysis and iterative design optimization, a large amount of numerical data is generated and processed. Such data must be stored and processed using numerical databases and a suitable DBMS.

Business oriented DBMS cannot be used for engineering applications for a variety of reasons. First, they do not support engineering data types; this precludes their interaction with almost all technical programs. Second, engineering data exists in several forms; their integration is a difficult problem. Finally, most engineering application programs need a local database on which an enormous amount of computation is carried out involving the entire database; this means the DBMS must provide efficient run-time support to a large I/O application.

Also, DBMS developed to handle only numerical databases cannot be used for applications requiring integrated databases. The reason is that it is not only inefficient to use such systems but also the programming becomes quite clumsy due to non-availability of proper data structures. Thus, it is important to use a DBMS in engineering applications that is capable of handling various data types.

The design of the new system [11-14] overcomes all the three problems and facilitates implementation of a unified database management system for such diverse applications as analysis and design of structural and mechanical systems, generation of production information, inventory control, project scheduling, etc. This will allow sharing of common data between business and engineering applications.

GENERALIZED RELATIONAL MODEL

Relations are important for sharing data among different users. Whereas most physical models in business applications can be represented conveniently in database as relations, engineering applications require both matrix and relational data types. Most large matrices form temporary or semi-permanent data private to a program. Relations are either permanent data in public domain used by different users or final results of a program to the end user. Therefore a new scheme is needed to represent both relations and matrices in a unified way for integrated engineering applications.

Engineers and scientists are well versed with the use of matrices and vectors. It is natural for them to imagine a relation (table) as simply a two dimensional array whose each column has unique definition. Thus the concept of a matrix is generalized for relations and the resulting model is termed as the generalized relational model. This scheme is supported by primitive and

structured data types. Using these data types users can define their own data models.

The novelty of the new approach is that the relation is derived from a matrix. In all previous attempts, e.g. RIM [4], one tried to extend relation for matrix data type. This led to clumsy and inefficient handling of numerical data. In MIDAS/GR the basic data type is matrix. Matrix can be one dimensional (vector) or two dimensional, and it can have elements with composite data structure. Relations are derived from the matrices as vectors of records.

DATA MODELS OF MIDAS/GR

The basic data models in MIDAS/GR are matrices (two dimensional) and vectors. Vectors are of two types : variable length and fixed length. A fixed length vector can also be considered as a degenerate matrix with only one row. A special type of variable length vector is the relation (vector of records). Figure 1 shows relationships among different data models.

Matrix and fixed length vectors are static objects, i.e., their sizes are statically determined at the time of their creation. This essentially means that they can have elements only of fixed size. In contrast, variable length vectors are dynamic in nature. Not only is their size determined dynamically (depending only on the number of elements defined), their elements may also be of variable length.

There are four structured data types available in the system : Record, Vector, Matrix, String. The type record gives the most general method to obtain structured data type. It joins elements of arbitrary, possibly themselves structured types, into a compound type. Vectors and matrices are in contrast homogeneous structures. They consist of all components of the same type, called the base type. The base type in turn may be a structured

type. This opens up the possibility of defining a number of special data types. For example, a matrix of complex numbers, where complex number is a record of two real numbers.

A vector of records may in general be treated as a relation where each component of the vector is an occurrence of the record, and each component of the record is an attribute of the relation. Since, a record can have a component of structured data type, a relation may have non-atomic attribute values.

If a matrix is defined as matrix of matrices, the matrix is considered to be made up of a number of submatrices. Matrices may be accessed rowwise, columnwise, or submatrixwise.

String is a special structured data type designed to deal with a sequence of characters. String may be used as base type to define other structured data types.

There are four primitive types available in the system. They are Integer, Real, Double Precision, and Character.

UNIQUE FEATURES OF MIDAS/GR

MIDAS/GR is the first database management system designed to handle large numerical databases with relational facilities. At the heart of the system lies the new concept of generalized relational model. The new concept allows us to unify numerical and relational data models into an integrated system.

The system provides efficient run-time support to various application programs handling large numerical and relational data. It also provides a central storage area for various groups in a design department. For example, groups working on structural geometry, structural layout, cost estimation, material control, project planning, etc. need to share data and interact with

one another. The system provides necessary interface and facilities for such purpose.

Apart from the above, the system provides following facilities which together make it a unique system:

Data Structure

- a) The system provides integrated data definition facility, using primitive and structured data types. Relation is just one of the many user defined data types.
- b) Because of its general approach, the system defines matrix as easily as any other data type (say relation). As a result, matrix can be treated as a distinct entity and it can have its own composite data elements.
- c) It provides several qualifier (sparse, banded, etc) to define special types of matrices frequently used in engineering applications. This allows efficient handling of storage without direct involvement from the user. Moreover, this provides uniform data structure for such matrices; this effectively facilitates development and maintenance of system routines to manipulate them.

Data Language

- a) It provides unifying data language. It treats relation and matrices similarly. It is possible to make query (or other data manipulation operations) on individual elements of a matrix.
- b) It provides language with same syntax for both terminal users and for those using them from a programming language. This leads to ease of communication between two classes of users.
- c) It provides a language which is non-procedural, i.e., its statements are statements of intent on the part of the user. This makes search optimization feasible. Also for a very large class of queries users need

not resort to loops or branching. This simplicity in turn means more productivity.

Data Independence

The system provides multiple view of the same stored record. This is made possible by maintenance of multiple access paths. Apart from physical sequence, it maintains indexes and links (binary and unary) for direct access and for sequential access in a different value ordering. This leads to data independence.

Concurrent Usage

It allows several users to use the system concurrently, doing retrieval, update and other operations, without conflicting with each other.

Recovery and Restart

It allows database to recover from crash, either undoing everything upto a check point, or redoing upto a check point and then restarting from that point.

SYSTEM ARCHITECTURE

The overall architecture of MIDAS/GR is described by its two main components. The lower level component is Data Storage Interface (DSI), and the upper level component is called Data Language Interface (DLI), as shown in Figure 2. DSI is an internal interface which handles access to single data elements. It manages space allocation, storage buffers, transaction consistency, system recovery, etc. It also maintains indexes on selected fields of relations and pointer chains across relations.

The DSI has been designed so that new data objects or new indices can be created at any time, or existing ones destroyed, without quiescing the system and without dumping and reloading the data. One can redefine data objects,

i.e., change dimensions of matrices or add new fields to relations. Existing programs which execute DSI operations on data aggregates remain unaffected by the addition of new fields.

DSI has many functions which can be found in other systems, both relational and nonrelational, such as the support of index and pointer chain structures. The areas which have been emphasized and extended in the DSI include integrated data definition and manipulation facilities for numerical (matrices) and relational data models, dynamic definition of new data types and access paths, dynamic binding and unbinding disk space, and crash resistance and recovery.

Data Language Interface (DLI) is the external interface which can be called directly from a programming language. The high level data language is embedded within the DLI and is used as the basis for all data definition and manipulation.

Figure 2 shows the major modules of MIDAS/GR in current implementation. In the lowest level is I/O library which invokes operating system routines for transfer of data between disk and main memory. Memory management module manages MIDAS/GR buffer. It partitions the buffer into pages of fixed size. All I/O at this level is in terms of pages and using I/O library. Data management module provides facilities of data definition and manipulation. Index management module provides facilities of index definition and manipulation. Relation management module uses data management and index management modules to provide facilities of relation definition and manipulation. Page management formats and manages all access to a relation page. Segment management allocates and frees segmented memory (file is treated as a contiguous address space). Hash management manages a hash table for locks; and lock management allocates and deallocates locks. Lock management is used

by all the modules and also by the user programs to protect their respective resources.

The address space of MIDAS/GR is divided into pages. Broadly there are two types of pages : those containing administrative information and the others containing user data. Data pages are in turn of two types : for variable length objects and for fixed length objects. Two distinct storage organizations are adopted for fixed length data objects and variable length data objects. For fixed length objects, all data elements are defined at the time of creation. Therefore, full storage allocation is made as soon as it is created. Whereas for variable length objects, size of the object at any time depends on the number of elements defined. Therefore, allocated storage is expanded dynamically as more and more elements are defined.

Data objects are divided into two categories - small and large. The mapping between logical and physical pages of the data object is defined by using page table. For large data objects the page table becomes quite large. Therefore, the page table of the large object is defined as small fixed length vector and stored in the database as a permanent object. This technique leads to very small page table in the virtual memory at the expense of one level of indirection for each page access.

MIDAS/GR buffer is logically divided into two pools : Block Buffer (BB) and Page Buffer (PB). Block Buffer contains all administrative informations, e.g., cursor table, page tables, page page tables (for large data objects), etc. Page Buffer contains the data pages and page table pages (for large data objects). Configuration of the buffer is shown in Figure 3.

When an object is opened, an entry is allocated in the cursor table. It has two pointers : pointer to a master record which contains all attributes of the data object and the other to the page table. A page table contains entry

for each page. They are the addresses of the page in disk and in page buffer, address of the page entry in LRU stack (for page replacement), and a mark bit to recognize if the page is modified. For each page in the page buffer, there is an entry in the stack. This entry contains the address of the entry in page table. Logically there are two stacks : stack of clean pages and stack of dirty pages. The top most entry represents the most recently used page and bottom most entry represents the least recently used page.

ALGORITHMS FOR IMPLEMENTATION

MIDAS/GR is designed as a number of cooperating modules. They interact with each other through well defined interfaces. Each module is designed carefully to perform a specific job. Existing literature is studied to determine the most suitable algorithm and data structure relevant to a module. Research issues related to these modules and relevant algorithms are described in the following.

I/O Library

At the lowest level is the I/O library. It treats database as a linear address space at the byte level. This module uses operating system routines for transfer of data between the disk and the main memory. It provides a machine independent interface to memory management module for data transfer. The module will be replaced subsequently with a new disk management module when it is ready.

Segmented Storage Management

Since we allow dynamic allocation and deallocation of space (objects are created, expanded and destroyed dynamically), there will be lots of holes in the file. One solution is to compact every time memory is deallocated;

compaction, however, is a much slower process, whereas it is much more economical to search a well organized hole list.

Since disk access is slow, hole list is organized in the header of the file and is retained in the main memory after it is read first time. The list contains the addresses of the available spaces and its length; and it is arranged in an increasing order of addresses. Since this list contains sufficient information, no disk access is necessary for allocation and deallocation purposes.

Paged Buffer Management

To reduce redundant I/O, it is important to have a well managed buffer in the main memory. Several memory management schemes are studied [15]. It is felt that a paging mechanism with the least recently used replacement policy will be the most suitable choice. Accordingly, the buffer is divided into a collection of page frames which are equal to the physical slots on the disk.

LRU policy is implemented using two stacks : stack of dirty (modified) pages and stack of clean pages. Top most entry of the stack is the most recently used page and the bottom most one is the least recently used page. Every time a page is accessed, its position in the stack is moved to the top. Pages are simply read into the page buffer till it is full. Once it is full and a new page has to be read in, the system reuses the page frame of the least recently used clean page. If there is no clean page in page buffer then the content of the least recently used dirty page is written back to disk and the page frame is reused.

To achieve further efficiency, the system allows a page to be pinned in the buffer. In that case the page will not be removed until it is unpinned. This facility is useful when the system has fore knowledge about activities of certain pages. Also the system allows two levels of page table. In that case

an entry in the page table points to a page which in itself is a part of the next level of page table. This leads to smaller page table for large objects, at the expense of indirection in data page access.

Stack Management

This module manages the two stacks for the implementation of the least recently used replacement policy. For reasons of efficiency, they dynamically share the same memory. Total space occupied by two stacks together equals the number of page frames in the page buffer. The entries of each of the stacks are connected together by double links.

Matrix Management

A matrix is a fixed length object, i.e., its elements are created or deleted all at once. Therefore, the problem of internal fragmentation within a page is avoided for such objects. Also elements in a matrix are accessed both at random and in sequence. Therefore, it is important that the logically contiguous elements are stored physically close to each other.

McKellar and Coffman [16] show a technique that allows quick access to matrices rowwise, columnwise and submatrixwise. Matrices are stored submatrixwise. Each submatrix contains a fixed number of rows and columns such that the size of the submatrix is not greater than a page size. It is possible to choose the number of rows and columns of submatrices in such a way to minimize the wastage of storage space. Figure 4 shows storage organization for such a data model. Fixed length vectors are stored similarly where the number of rows is one.

Relation Management

Figure 5 shows the organization of a page of variable length object. This organization was originally proposed by Astrahan, et al. [17] for their

experimental DBMS System R. Each individual record has a numeric identifier, called DaId (Da for data aggregate). Each DaId is a concatenation of a page number, along with a byte offset from the bottom of the page. The offset denotes a special entry or 'slot' which contains the location of the data aggregate in that page.

This technique allows efficient utilization of space within data pages, since space can be compacted and data aggregates moved with only local changes to the pointers in the slots. The slots themselves are never moved from their positions at the bottom of each data page, so the existing DaIds can still be employed to access the data aggregates. Since the position of the byte offset at the bottom of the page is fixed, internal movements of the data aggregates within a page does not change its address. This provides easy way to compact data aggregates within a page. A variable length object is made up of a collection of such pages.

Relation Page Management

A relation page is divided into a number of physical records. Allocation and deallocation of space takes place with respect to physical records. A modified version of Boundary Tag method with improved first fit algorithm using rover counter [18] is used for allocation and deallocation of space.

In this method only one byte at the beginning of a block is used. This reduces storage overhead which is at a premium in a page. It also allows the use of a very simple algorithm for space liberation, as no search or collapsing of adjacent free blocks is involved. Even space reservation involves little search as allocation is distributed over the page using the rover counter. However, allocation procedure is slightly complex, and runs little longer as collapsing of adjacent free blocks is done at this time. Figure 6 shows the configuration of free and used space.

Index Management

An index is a logical ordering with respect to values in one or more sort fields. Indices provide the ability to scan data objects along a value ordering. Also, an index provides associative access capability. The DLI can quickly fetch data aggregate from an index by keying on the sort field values. The DLI can also open a scan at a particular point in the index, and retrieve a sequence of data aggregates with a given range of sort values.

Many techniques for organizing index have been proposed. Knuth [19] provides a survey of the basics. While no single scheme can be optimum for all applications, a technique of organizing indices called B-tree [20] has become widely used. The B-tree is, de facto, the standard organization for indices in a database system. For reasons of efficiency in both random and sequential access, we have adopted a variation of B-tree, called B⁺-tree [21]. In a B⁺-tree all keys reside in the leaves. The upper levels, which are organized as a B-tree, consist only of an index. Leaf nodes are linked together allowing easy sequential processing.

Sort Management

Sorting is required at various phases of database management, e.g., while creating an index on an existing data object, or data is returned in an order different from that of retrieval. So, it is necessary to have an efficient general purpose sorting system.

Depending on the amount of data the system chooses either array sort (data contained entirely in the main memory), or file sort (data is spread in main memory and disk). Array sort is based on Quick sort algorithm [22,23]. File sort is a superior version of Polyphase sort [24]. In our algorithm, we combined an array sort method, called Heap sort [25], and used it in the

distribution phase of initial runs. As a result these runs always have the length of approximately the size of the available main memory.

Lock Management

Two types of locks are implemented - shared and exclusive. Shared lock allows a number of processes to read an object concurrently. Exclusive lock allows a single writer to modify an object. The algorithms, based on Courtois et al. [26] are considered efficient and are implemented using semaphore facilities of the operating system.

Hash Management

This module provides facility for name resolution. A number of methods are studied for hash addressing, collision handling and table layout. Finally an algorithm based on scatter index table [27] is chosen. The data area is chained together by double links. This leads to overall quicker performance at the expense of slight space overhead.

EVALUATION

Currently, application programs access MIDAS using a well defined interface. The specification of this interface is given in MIDAS/N User's Manual [28]. The same interface is implemented on top of MIDAS/GR access method. As a result MIDAS/GR has become accessible to all applications that have been using MIDAS.

For the purpose of benchmarking MIDAS/GR and comparing it with MIDAS/N, an application program is chosen. This program solves a system of equations using the skyline method. The reason for choosing this program is that in most engineering problems it is necessary to solve a system of equations, and the performance of the application depends critically on the performance of the equation solver.

Table 1 and 2 gives some idea about the performance of MIDAS/GR. More detailed results are contained in Refs. 13 and 14. All tests are run on a DN460 Apollo computer with 2 MB of primary memory.

In general, MIDAS/GR is about twice as fast as MIDAS/N. Another interesting observation relates to the use of DBMS itself. It is a popular belief in the scientific field that one must make great sacrifice in efficiency to use DBMS (for other gains). Our result shows that with DBMS (MIDAS/GR) the program runs only about 10-12% longer than that without DBMS (database is in the virtual memory). This small increase in cpu time is acceptable in most cases.

At this time, MIDAS/GR does not manage its own disk or primary memory. The increased run-time is attributed to MIDAS/GR's dependence on the Operating System for memory management and disk space allocation. This leads to duplication of effort in memory management and inefficiency in allocation of disk space. We believe that MIDAS/GR will, in fact, improve efficiency in future when it manages its disk and buffer directly.

It is noted here that MIDAS/N has been recently evaluated against RIM [8]. Parameters such as the number of reads, number of writes, number of calls to DBMS, and the CPU time were measured for the two systems. MIDAS/N was determined to be far superior to RIM for dynamic run-time support environment. Main drawbacks of RIM for such applications are in data models, data access methods and the memory management scheme. Using the results of that study, we can safely conclude that MIDAS/GR is far superior to RIM as a run-time support system.

CONCLUDING REMARKS

In addition to the MIDAS/N user interface, MIDAS/GR supports a very powerful and flexible user language developed in the BNF notation. The

language has numerous data definitions and data manipulation facilities for vectors, matrices and relations. It can be used from the terminal on an ad hoc basis, or from a programming language (currently C and FORTRAN). The language is described in detail in the MIDAS/GR user's manual [29].

Creation of an efficient computer-aided analysis and design environment needs run-time support of an integrated database management system. A properly designed DBMS will have enormous impact on the architecture of future software for complex engineering applications. As more computational power becomes available, the range of applications and complexity of the software increases. Extensions, maintenance and debugging of the software can be considerably facilitated by the use of a proper database and DBMS. Thus, a properly designed and implemented DBMS, such as MIDAS/GR, will be an invaluable tool for future software developments to solve complex - interdisciplinary engineering analysis and design problems.

ACKNOWLEDGEMENT

Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant Number AFOSR 82-0322. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

REFERENCES

1. SreekantaMurthy, T., and Arora, J.S., "Database Management Concepts in Computer-Aided Design Optimization," Advances in Engineering Software, Vol. 8, No. 2, 1986, pp. 88-97.
2. SreekantaMurthy, T., and Arora, J.S., "A Survey of Database Management in Engineering," Advances in Engineering Software, Vol. 7, No. 3, 1985, pp. 126-133.
3. Lopez, L.A., "FILES: Automated Engineering Data Management System," Computers in Civil Engineering, Electronic Computation, 1974, pp. 47-71.

4. Comfort, D.L., and Erickson, W.J., "RIM-A Prototype for a Relational Information Management System," NASA Conference Publication 2055, 1978.
5. Massena, W.A., "SDMS: A Scientific Data Management System," NASA Conference Publication 2055, 1978.
6. Fischer, W.E., "PHIDAS - A Database Management System for CAD/CAM Software," Computer-Aided Design, Vol. 11, No. 3, 1979, pp. 146-150.
7. Ulfby, S., Steiner, S., and Oian, J., "TORNADO: A DBMS for CAD/CAM Systems," Computer-Aided Design, 1979, pp. 193-197.
8. SreekantaMurthy, T., Shyy, Y.K., Mukhopadhyay, S., and Arora, J.S., "Evaluation of Database Management System MIDAS in Equation Solving Environment," Engineering with Computers, Vol. 2, 1987, pp. 11-19.
9. SreekantaMurthy, T., Shyy, Y.K., and Arora, J.S., "MIDAS - Management of Information for Design and Analysis of Systems," Advances in Engineering Software, Vol. 8, No. 3, 1986, pp. 149-158.
10. Shyy, Y.K., Mukhopadhyay, S., and Arora, J.S., "A Database Management System for Engineering Applications," Technical Report No. ODL-85.23, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, IA 52242, 1985.
11. Arora, J.S. and Mukhopadhyay, S., "Specification for MIDAS-GR Management of Information for Design and Analysis of System : Generalized Relational Model," Technical Report No. CAD-SS-84.24, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, IA 52242, 1984.
12. Mukhopadhyay, S. and Arora, J.S., "A Database Management System using Generalized Relational Model," Technical Report No. ODL-86.27, Optimal Design Laboratory, College of Engineering, University of Iowa, Iowa City, IA 52242, 1986.
13. Mukhopadhyay, S. and Arora, J.S., "Design and Implementation Issues in an Integrated Database Management System for Engineering Design Environment," Advances in Engineering Software, Vol. 9, No. 4, 1987, pp. 186-193.
14. Mukhopadhyay, S. and Arora, J.S., "Implementation of an Efficient Run-time Support System for Engineering Design Environment," Advances in Engineering Software, Vol. 9, No. 4, 1987, pp. 178-185.
15. Denning, P., "Virtual Memory," Computing Surveys, Vol. 2, No. 3, 1970, pp. 153-189.
16. McKellar, A.C. and Coffman, E.G., Jr., "Organizing Matrices and Matrix Operations for Paged Memory Systems," Comm. of the ACM, Vol. 12, No. 3, March 1969, pp. 153-165.
17. Astrahan, M.M., Blasgen, M.W., Chamberlin, D.D., et al., "System R : Relational Approach to Database management," ACM Trans. on Database Systems, Vol. 1, No. 2, 1976, pp. 97-137.

18. Knuth, D.E., Art of Computer Programming, Vol I: "Fundamental Algorithms," Reading, Mass., Addison-Wesley, 1973.
19. Knuth, D.E., Art of Computer Programming, Vol. III: "Sorting and Searching," Reading, Mass., Addison-Wesley, 1973.
20. Bayer, R. and McCreight, E., "Organization and Maintenance of Large Ordered Indexes," Acta Informatica, Vol. 1, 1972, pp. 173-189.
21. Comer, D., "The Ubiquitous B-tree," Computing Survey, Vol. 11, No. 2, June 1979, pp. 121-137.
22. Hoare, C.A.R., "Quicksort", Comp. Journal, Vol. 5, No. 1, 1962, pp. 10-15.
23. Hoare, C.A.R., "Proof of a Recursive Program: Quicksort," Comp. Journal, Vol. 14, No. 4, 1971, pp. 391-95.
24. Gilstad, R.L., "Polyphase Merge Sorting - An Advanced Technique," Proc. AFIPS Eastern Jt. Comp. Conf. Vol. 18, 1960, pp. 143-48.
25. Williams, J.W.J., "Heapsort," Comm. of the ACM, Vol. 7, No. 6, 1964, pp. 347-48.
26. Courtois, P.J., Heymans, F., and Parnas, D.L., "Concurrent control with readers and writers," Comm. of the ACM, Vol. 14, 1971, pp. 667-668.
27. Morris, R., "Scatter Storage Techniques", Comm. of the ACM, Vol. 11, No. 1, Jan. 1968, pp. 38-44.
28. Shyy, Y.K., Arora, J.S., Mukhopadhyay, S., et al., "MIDAS/N User Manual", Optimal Design Laboratory, College of Engineering, University of Iowa, 1984.
29. Mukhopadhyay, S. and Arora, J.S., "MIDAS/GR User's Manual," Optimal Design Laboratory, College of Engineering, The University of Iowa, Iowa City, IA 52242, 1987.

Table 1 : MIDAS/GR VS. MIDAS/N

Page Size = 4 KB

Buffer Size = 64 KB

Workspace Size = 8 KB

Band width = 100

No. of Eqns.	CPU Time in MIDAS/GR (Seconds)	CPU Time in MIDAS/N (Seconds)	Ratio MIDAS/N : GR
1000	899	1811	2.01
5000	5444	10,437	1.92
10,000	9905	20,194	2.04
20,000	22,781	40,945	1.80
30,000	34,827	62,050	1.78
40,000	45,251	81,329	1.80

Table 2 : MIDAS/GR VS. Virtual Memory
Page Size = 16 KB
Buffer Size = 1024 KB
Workspace Size = 128 KB

No. of Eqns.	CPU Time in MIDAS/GR (Seconds)	CPU Time W/o DBMS (Seconds)	Penalty for DBMS use %
1000	301	281	7.1
5000	1601	1502	6.6
10,000	3376	3023	11.7
20,000	6839	6193	10.4
30,000	10,319	9241	11.7
40,000	13,785	12,228	12.7

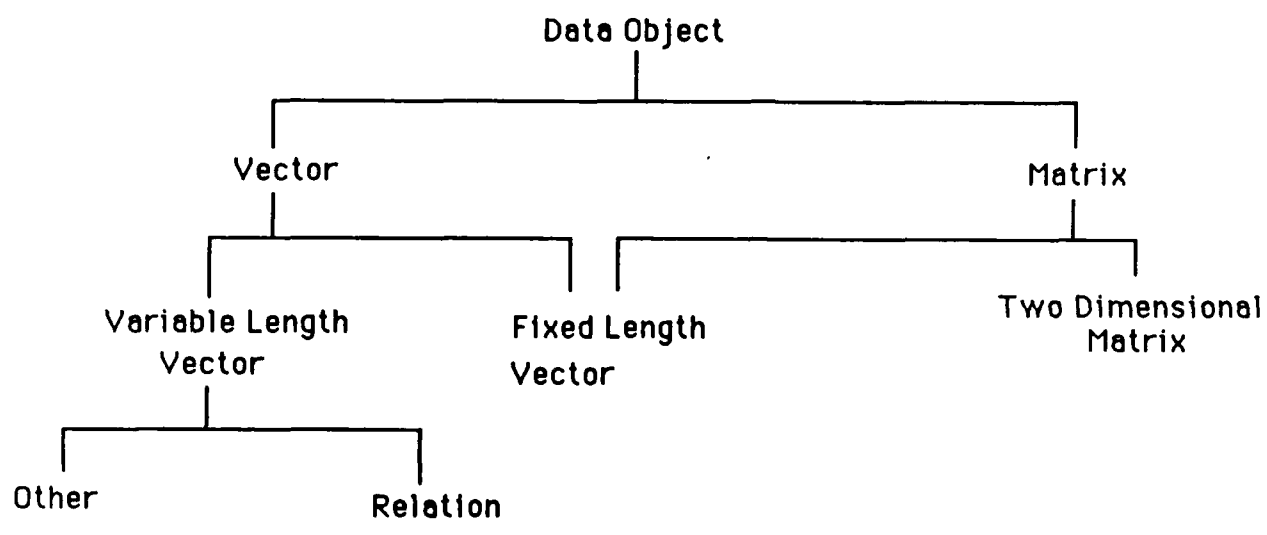


Figure 1. Data Model of MIDAS/GR

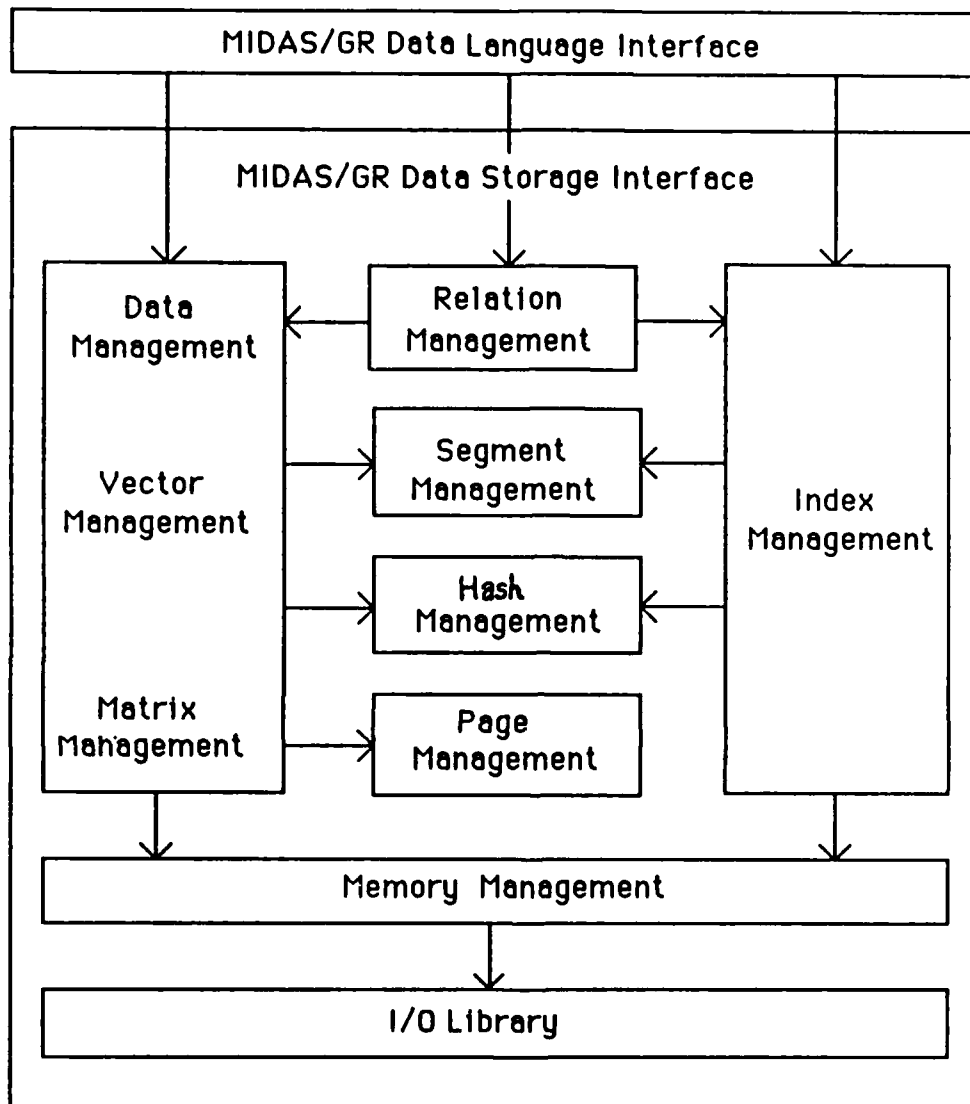


Figure 2. Organization of MIDAS/GR

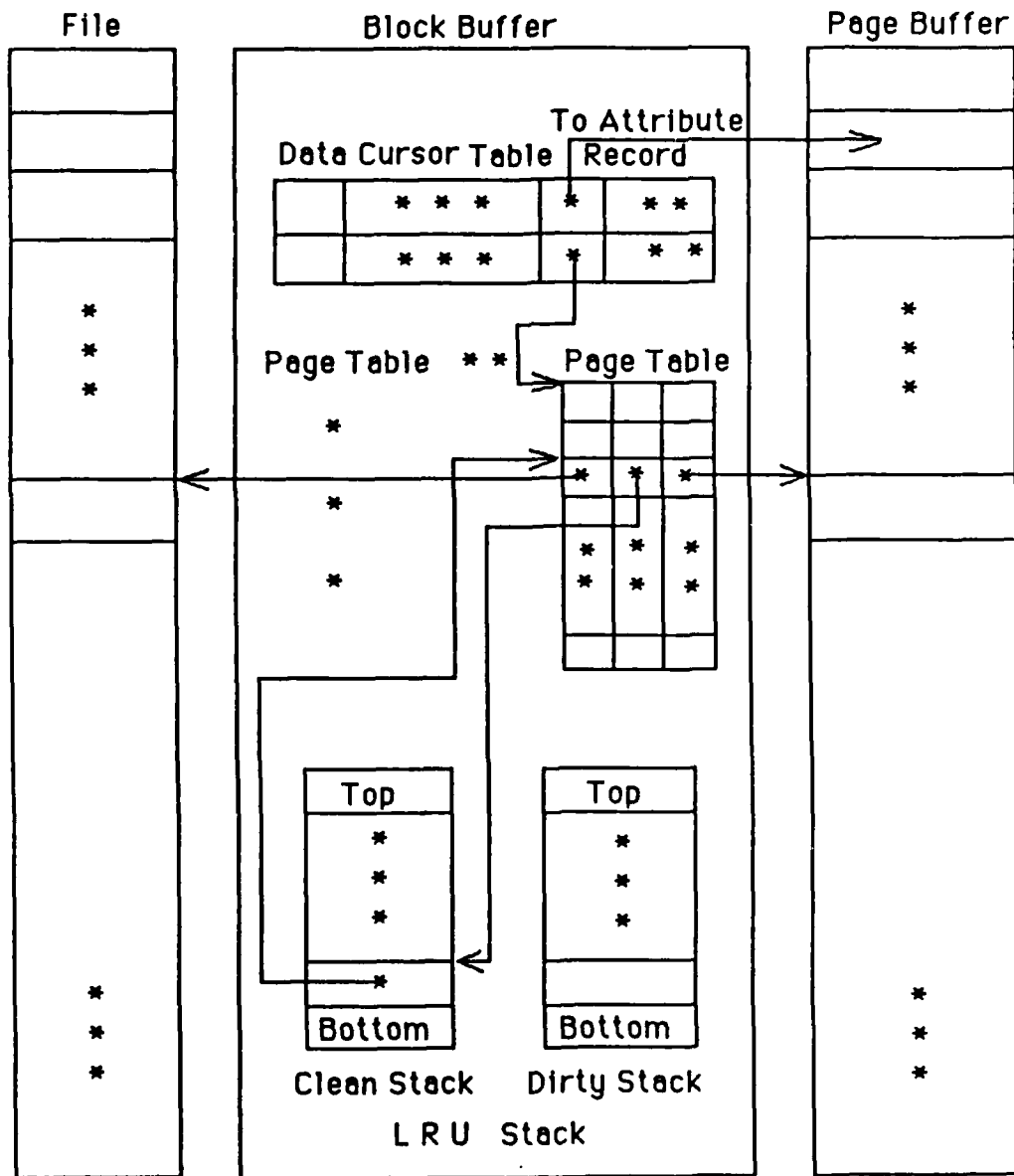


Figure 3. MIDAS/GR Buffer Configuration

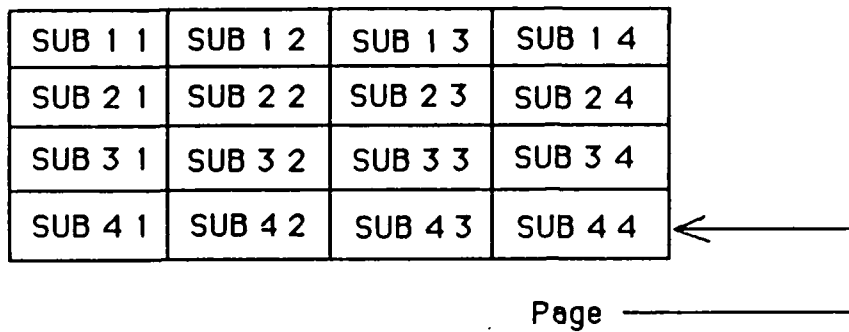


Figure 4. Storage Organization of Fixed Length Object

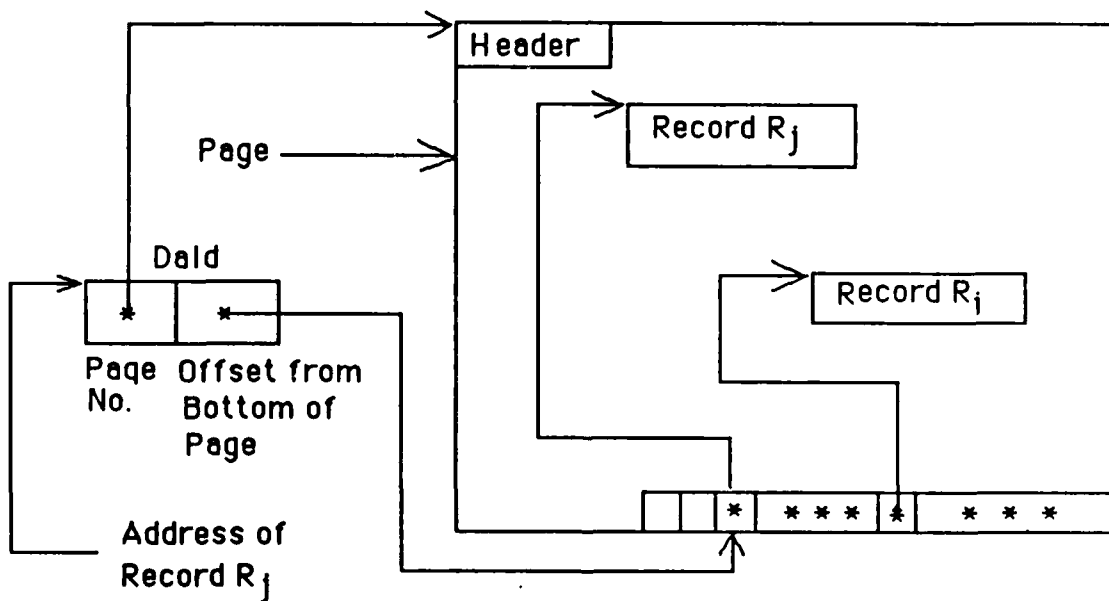


Figure 5. Page Organization of Variable Length Object

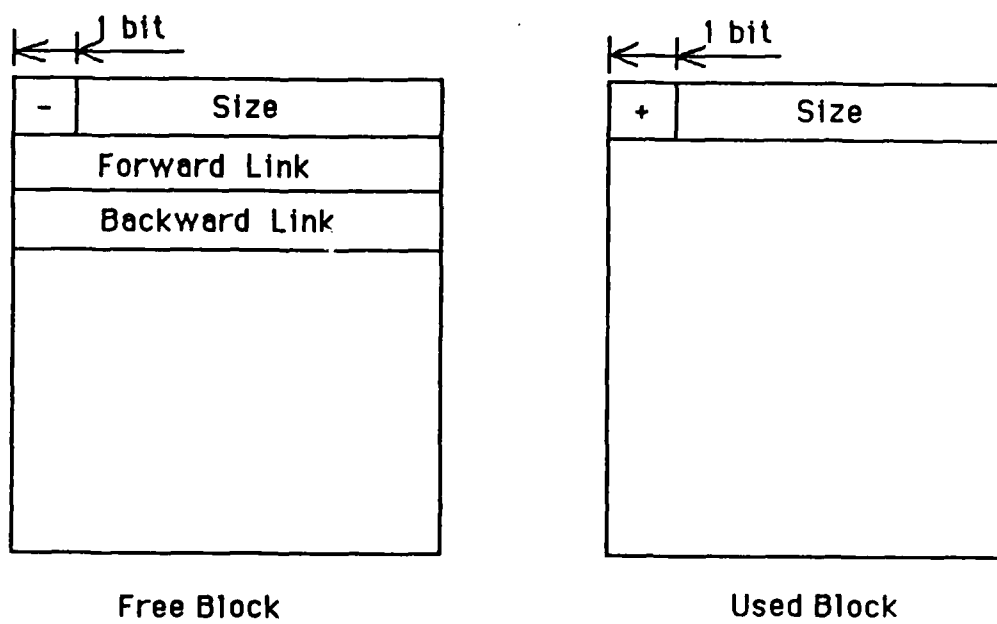


Figure 6. Configuration of Free and Used Space

END

DATE

FILMED

6-1988

DTIC