REPORT
UCB/SEMM

STRUCTURAL ENGINEERING,
MECHANICS AND MATERIALS

AUTOMATED MODELING FOR STRUCTURAL ANALYSIS

by

HAZEM AN-NASHIF

Faculty Supervisor: G. H. POWELL

APRIL 1987

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA

AUTOMATED MODELING FOR STRUCTURAL ANALYSIS

HAZEM AN-NASHIF

University of California, Berkeley
Department of Civil Engineering
Berkeley, California 94720

Submitted in partial fulfillment of the requirements of the degree
of Doctor of Philosophy in Civil Engineering.

ABSTRACT

The development of Computer-Integrated Design systems for structural analysis and design has become a subject of increasing interest. An architecture for the structural analysis sub-system of such a system is proposed, and an automated modeling procedure is described.

Acknowledgement

I take this opportunity to thank my advisor Prof. Graham Powell, for his constant support and encouragement throughout this research effort. Working for Prof. Powell has been like a breath of fresh air - his openness to new ideas and encouragement of innovative techniques has been a great source of inspiration all along. I am indebted to him for his patience and positive reinforcement.

Hazem An-Nashif

TABLE OF CONTENTS

INTRODUCTION

1.

This report describes progress in one part of a research program on Computer Integrated Design (CID) for Structural Engineering. Structural Engineering CID is still in its infancy, but the success of automation in other engineering design area, particularly Very Large Scale Integrated (VLSI) circuit design, suggests that success is also possible in Structural Engineering. The goal of the research program is to create an integrated system that can perform automated structural analysis and design of frame structures in which the human designer's role is limited to supervision, expertise and major decisions.

In the 1960's, engineers began using computers to solve large structural analysis problems that were otherwise impossible to solve. Mainly, computers were used to perform numerical matrix operations such as the solution of simultaneous equations and the extraction of eigenvalues. That is, computers were used mainly as calculating machines. During the past twenty years, theory and software have improved greatly, the cost of computer hardware has been sharply reduced, and hardware performance has greatly increased. At the same time, there has been a computer science revolution, with great success in areas like data management, operating systems and logical reasoning (e.g. rule-based systems.) As a result, it is now technically feasible to perform a wide variety of engineering applications on the computer, particularly data organization and decision making. In Structural Engineering, however, the applications have remained limited, mainly to structural analysis and structural drafting, and there has been little progress in integrating together different planning, design and analysis applications via the computer.

Some major components of a Structural engineering CID system include a data management system, structural analysis modules and structural design tools. Other important components include tools

for drafting and quantity estimation, but these are not being addressed at this time in our research.

A data management system is of critical importance because it maintains the data in an organized and consistent form, and provides an interface between different CID applications. Structural analysis involves creating mathematical models and simulating the performance of the structure under applied loads. Design tools allow analysis results to be processed and design constraints to be applied to help in the selection of structural members and connections to withstand critical load conditions.

These three areas will be the subject of separate doctoral dissertations under the overall research project. The present report deals only with the structural analysis problem. The basic concept behind this part of the research is that structural modeling and analysis are logically well-defined tasks, which can be virtually fully automated. Most of the this report describes the problem of automated modeling.

In the next section, an overall picture of an automated modeling/analysis system is presented. The model creation procedure and the resulting data structure are then discussed. Finally, a brief discussion of the analysis module and results organization is provided.

OVERALL PICTURE

2. GENERAL

At one end of the analysis system is a *Central Database* (CDB), containing a physical description of the structure, and at the other end is a finite element analysis module. In between a link must exist, to interpret the physical description and create a finite element model. This link is the modeler.

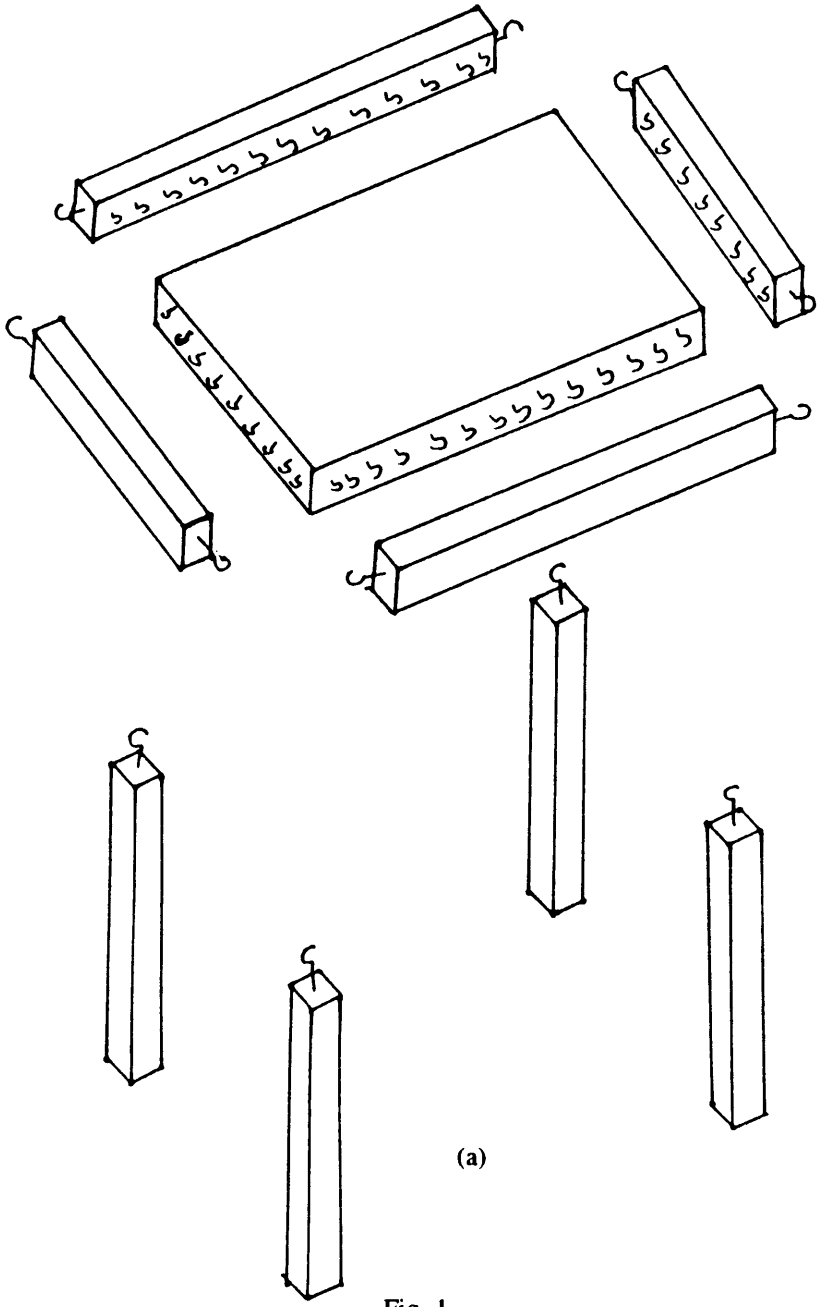
In this section, the CDB is described. The role of the modeler is then discused. Finally, the problem of retaining consistency is considered.

2.1. THE COMPONENT-CONNECTION MODEL

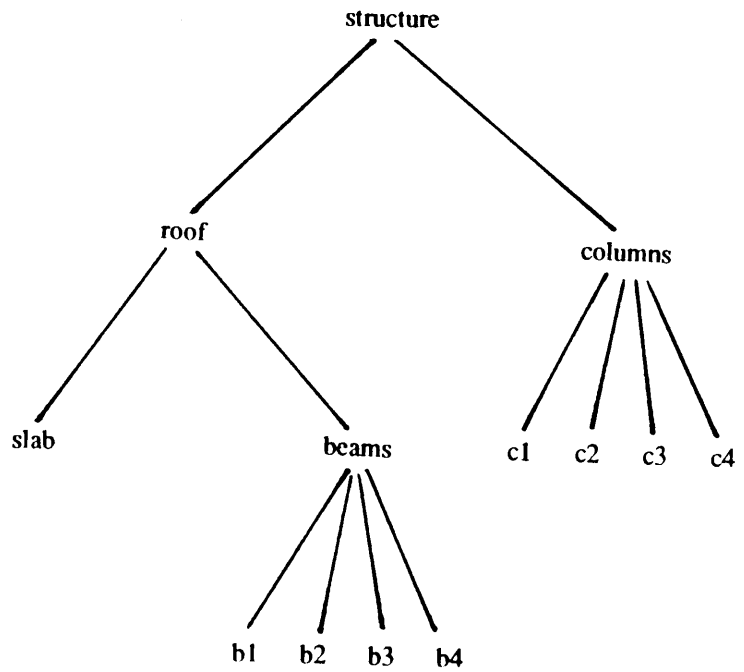
The CDB is based on a Component-Connection model. A basic *component* is a fundamental building block of the structure, such as a beam, column or slab. Two or more components may be joined together by a *connection* to form a higher level assembled component (or assembly). Assemblies may themselves be connected to other components or assemblies, to form still higher level components or assemblies. This process is repeated until the complete structure has been defined. The resulting hierarchy is the *component hierarchy*. Together with data on the components, this hierarchy describes the topology of the structure and the details of how the components connect to each other.

In the CDB, each component is a database *object*, which carries a geometric description of the component, the properties of the material composing it, and other information. Similarly, each connection is a database *object*, containing information describing the connection. Connections are linked to components by defining *hooks* within the components. A connection is created by *closing* two or more hooks (i.e. linking them together to form a connection). In any assembled component, hooks may be *open* or *closed*. Open hooks are those that have not yet participated in connections. Closed hooks have participated, and can no longer be used in connections.

Fig. 1 shows a building structure broken into components, and the resulting component hierarchy. In Fig. 1a, the building is divided into columns, beams, and slabs. Fig. 1b shows the structure hierarchy of the building, indicating how beams, and slabs are assembled into floors and how the columns and floors are assembled into the whole building.



(a)



(b)

Fig. 1

Component-connection representation of a structure. (a) Individual structure components. (b) The structure hierarchy.

2.2. THE BASIC ROLE OF THE MODELER

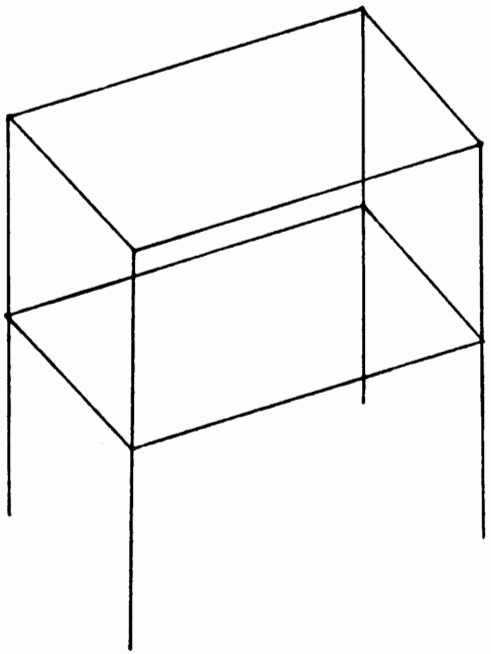
The principal analysis tasks are (1) model generation, (2) analysis of the model, and (3) linking the results back to the CDB. The CDB stores geometric and material properties of components and connections, plus information on hooks. An analysis program, however, must deal with finite element data composed of nodes, elements, degrees of freedoms (DOF's) and loads. Modeling involves transforming the CDB information into finite element program input.

2.3. ANOTHER ROLE OF THE MODELER: PRUNING

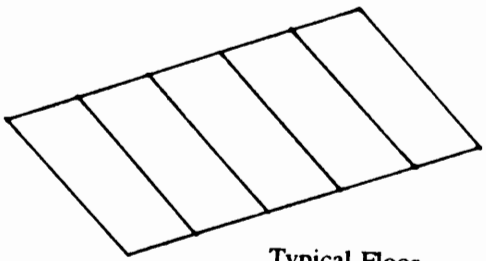
When a building is analyzed, usual practice is to divide the structure into floor and frame systems, and to analyze each system separately. *Pruning* refers to the process of selecting those components which are to be included in an *analysis model* by pruning off those branches of the hierarchy (tree) which are not to be included. Components in pruned branches are called *pruned components* while those included in the analysis model are called *retained components*. Connections between pruned components and retained components are called *pruned connections*.

Fig. 2 shows an example of the pruning concept applied to a building structure. In this example, it is desired to analyze the floors and the frame separately. Fig. 2a shows the separate floor and frame structures which are to be analyzed. Fig. 2b shows the *pruned* hierarchy representing the frame system.

Consider the building structure of Fig. 3. Let an analysis model be created by pruning the frame components from the hierarchy and retaining the floor system. A model generated for the floor system will include finite elements to model the retained components. However, it must be recognized that these components connect to the rest of the structure. This is indicated by the fact that certain connections have been pruned (in this example, the beam-girder connections). Wherever a connection is pruned, the structure being analyzed interacts in some way with the pruned part of the structure through pruned connections. This interaction must be represented in the analysis model by *boundary conditions* (BC's). In the present example, it can be assumed that the girders provide stiff vertical support and negligible torsional restraint. Hence, appropriate BC's would be simple vertical supports (i.e. displacement BC's).

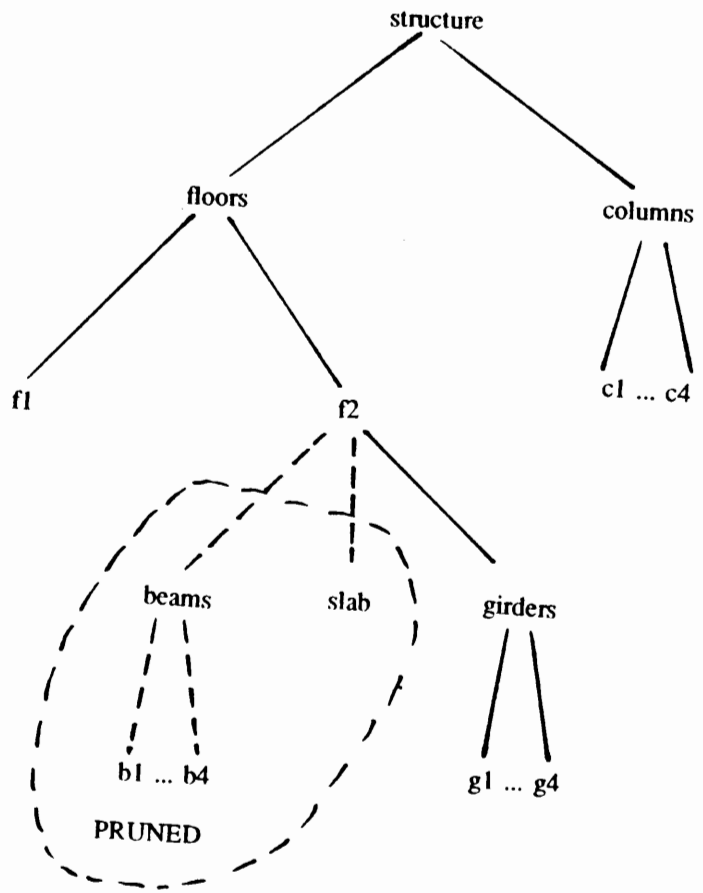


Frame Structure



Typical Floor

(a)



(b)

Fig. 2

Example of pruning. (a) The separated frame structure and a typical floor system. (b) The pruned hierarchy representing the frame.

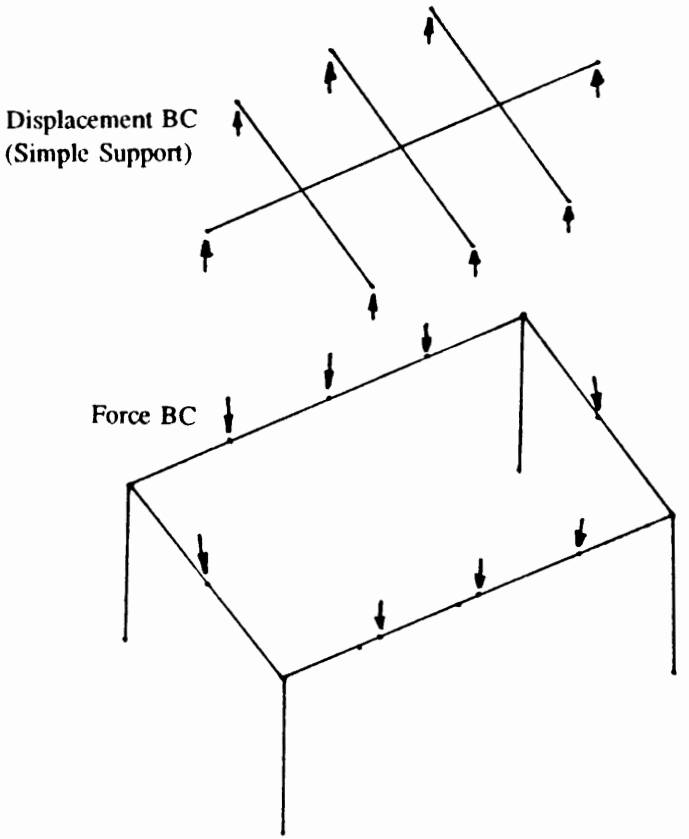
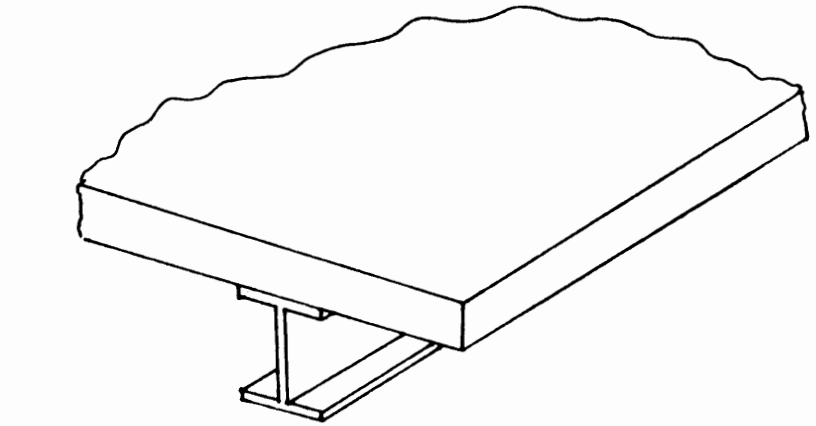


Fig. 3

Boundary conditions at pruned connections.



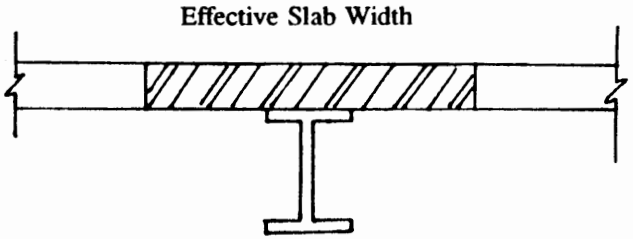


Fig. 4

Pruned slab-beam connection in a composite beam-slab.

When modeling the frame structure, different sets of retained components and pruned connections are encountered (beam-column connections, as before, plus column-footing connections). As in the preceding example, the pruned footing connections can be replaced by displacement BC's. At the beam-column connections, however, it is not displacements which are defined but forces, which are equal and opposite to the support reactions calculated in the floor analysis. That is, these pruned connections must be modeled by force BC's.

As a further example of pruned connections and corresponding BC's, consider an analysis model for a floor in which the slab is composite with the beams. A model can be created by pruning the connection between the beams and the the slab (as in Fig. 4). In this example the slab contributes flexural and shear stiffness to the beams. Hence, to account for the pruned slab component, additional stiffness must be added to the retained beams.

The above three examples illustrates three different ways in which pruned connections may need to be modeled. In general, any combination of force or displacement BC's and/or added stiffness may be necessary to account for the pruned components, depending on the type of pruning and the connection details.

In principle, a model could be produced by pruning any part of the structure. In practice, however, it will be reasonable to allow pruning of a connection only if the pruned components can be replaced by appropriate BC's. In any automated modeling system, therefore, there must be restrictions on the analysis models that may be extracted. For example, a floor model which includes both beams and girders, with pruned column-girder connections, might be inappropriate since it is not easy to provide BC's which model the rotational restraint offered by the columns. Similar restrictions are, of course, needed for conventional analysis methods (i.e. without automated modeling) for precisely the same reasons. Hence, automated modeling imposes no additional restrictions on the analysis.

2.4. MODEL DATA STRUCTURE

The modeler's role dictates that it produces a finite element model that can be analyzed by conventional structural analysis algorithms. This model is stored in a Model Data Structure (MDS), which

can then serve as input data for an analysis program. Most of this report deals with the MDS and the steps needed to generate it from the Component-Connection model.

2.5. APPLICATION DATABASE

A frequent problem faced in multiuser data management systems is how to coordinate accesses to the database by different users, in order to prevent database updates performed by one user from interfering with database retrievals and updates performed by another. Suppose, for example, that during the modeling of a structure in the CDB, another application were to update the CDB by changing the dimensions of certain components. This could result in the generation of a model that does not include the recent updates to the CDB, or even a model that includes only part of these updates. Further, after analysis of the model, the analysis results may be inconsistent with the updated CDB. Such inconsistencies are not tolerable.

It follows that if the modeler is allowed to deal directly with the CDB, updates to the CDB by other applications must be delayed until the modeling/analysis is complete (i.e. the CDB must be frozen for a period of time). Since the duration of the modeling/analysis operation may be long (from minutes to, possibly, days), this is also not tolerable. A solution is to make a *snap-shot* copy of the CDB, and store it in a private *Application Database* (ADB). This ADB is created by copying and possibly modifying the CDB by a utility called the *extractor*. Access to the ADB can then be restricted to other applications without affecting their operation since they either deal with the CDB or their own ADB's.

2.6. SUB-SYSTEM ORGANIZATION

It is possible now to construct a picture of the modeling/analysis sub-system, and to clarify the nature of the the link between the CDB and the finite element analysis module. The ADB is first extracted from the CDB. The modeler works on the data in the ADB to create the model and stores it in the MDS. This is then used as input data for the analysis module. This is shown in Fig. 5.

The remainder of this report focuses on the structure of the MDS, and the steps involved in creating it. Brief discussion is included of the analysis module and the organization of the final analysis results.

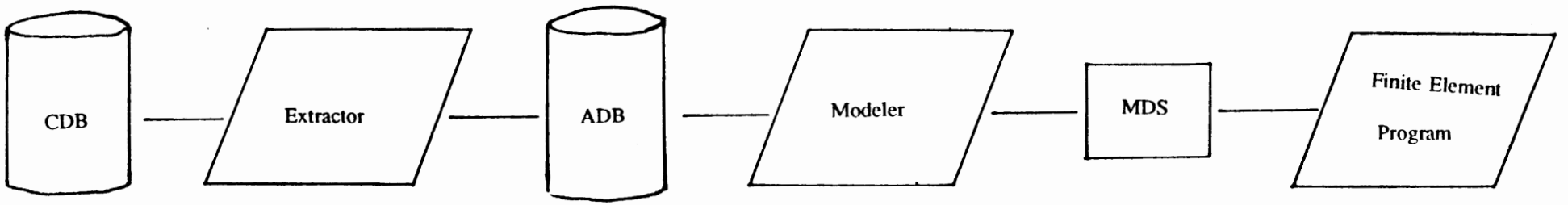


Fig. 5

Analysis sub-system organization.

MODEL DESCRIPTION

3. GENERAL

In a finite element model, a structure is idealized as a finite number of rigid body joints, or *nodes*, connected by a finite number of deformable members, or *elements*. The elements can generally be any mixture of frame elements and solid finite elements. For this discussion, only frame elements are considered.

The state of deformation of the structure model can be completely defined in terms of a finite number of displacement parameters at the nodes called the *degrees of freedom* (DOF's) of the structure. DOF's are typically the translational and rotational displacements of the nodes. However, other types of DOF may be needed (e.g. warping deformations to model torsion with warping).

In the model, *loads* may be applied to the nodes (*nodal loads*) or to the elements (*element loads*). Both act as *force boundary conditions* on the model. Element loads may include loads applied directly on the elements or as effects, such as temperature change, causing initial stresses in the elements.

The displacements at some of the nodes may be known to have certain values. For example, the displacements may be assumed to be zero at a support that is relatively rigid compared with the frame. These specified displacements are *displacement boundary conditions.*

In addition, the displacements at a node may not be independent, but may be constrained to be functions of the displacements at one ore more other nodes. For example, when analyzing a plane frame building, the floor members may be assumed to be inextensible. Hence, the horizontal displacements of all the nodes at any floor may be constrained to have the same value (Fig. 6). This type of constraint is a *slaving constraint.* The dependent displacements are referred to as *slaved* DOF's, and the independent displacements are referred to as the *master* DOF's. As will be seen, slaving constraints are an important part of the MDS.

In this section, the structure of the MDS is first discussed, and the coordinate systems used in the model are explained. Details of the model data are then presented, considering: (1) element data, (2) node, DOF, and slaving data, and (3) force and displacement boundary condition data.

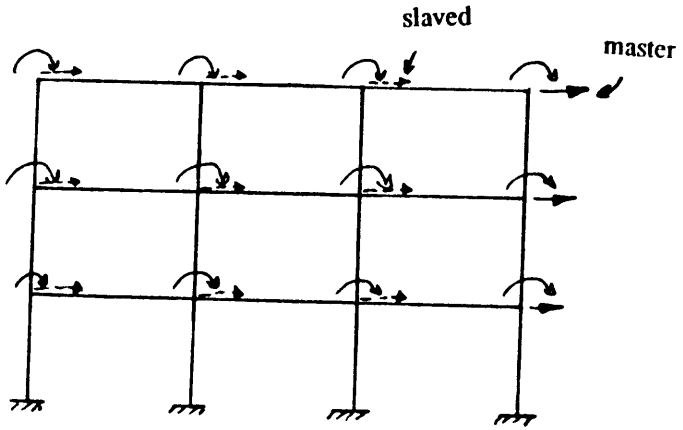


Fig. 6

An example of slaving.

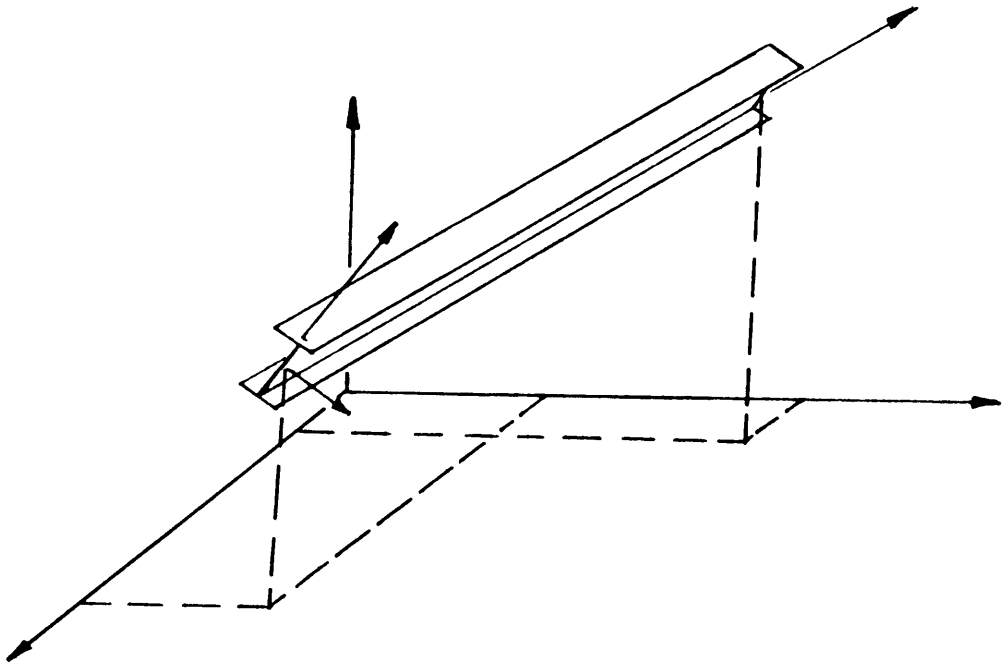


Fig. 7

Cross section orientation of a 3-D frame element.

3.1. THE STRUCTURE OF THE MDS

A database is nothing more than a computer-based record keeping system. The major components of such a system are: data, hardware, software and users. Where the data is both integrated and shared by different users. The MDS is different from a real database in that the data is accessed only by the modeler during modeling. Only after the modeling is complete, may the analysis program access the MDS. This means that no real sharing of data is allowed because only one application at a time has exclusive access to the data.

The MDS has to satisfy several requirements. As will be seen, the model grows dynamically as it is generated, so that its final size is not known until the modeling process is completed. This means that the MDS must be able to grow dynamically. It is also important that the performance of the MDS be fast, so as not to slow the modeling process. Further, it must be possible to store and recover the MDS on a permanent file. Finally, it is desirable to come up with a *simple* design.

For simplicity, the data is arranged in tables. In which, the table rows are also called records and the table columns are also called attributes. Attributes may be (a) numerical values, (b) character strings or (c) pointers to rows in the same or in any other table. These tables must be implemented using a dynamic data structure (e.g. linked lists). To achieve fast performance, the tables are kept in main memory during modeling, and saved on permanent storage (disk) only after the modeling is complete. Further, to increase performance, pointers are used to refer to table rows whenever a need arises for such a reference. The alternative solution of storing a record identifier and searching the tables for a match may be inefficient unless more complicated structures than linked lists are used to implement the tables. This solution still provides lower performance. These pointers may even be kept in the ADB to eliminate the need for searching when the analysis results are extracted from the MDS.

As an example of a table, consider a table (table name = Element) that holds information on all of the elements in a finite element model. Also, let the model be substructured, so that both elements and super-elements are included in the table, so that any element may be composed of several sub-elements. The Element table header, which specifies the attributes of this table, is as follows:

Element

element_id	no_sub_elements	sub_elements	other_data
(str)	(int)	(Element *)	(...)

Each table record represents one element in the model. For each element, the first attribute is the element_id, which is of character string type. The second attribute is the number of sub-elements from which the current element is assembled, which is of integer type. The third attribute is a pointer to another row in the Element table. (Here '*' defines the attribute to be of pointer type, and Element indicates that it is a pointer to a row in the Element table.) The row pointed to contains data on the first sub-element of the current element. If the sub-elements of each element are grouped successively in the table, then the number of sub-elements plus the pointer to the first sub-element provide sufficient information for accessing all of the sub-elements.

3.2. COORDINATE SYSTEMS

In the process of setting up an analysis model, the modeler is faced with several requirements. First, the model must include the location in space of each node in the structure model. Usually, the nodal locations are used to determine the orientations of elements. However, for a frame element, the nodal locations of its two nodes are not enough to determine the orientation of the element in space. In particular, the orientation of principal axes of the cross-section remain ambiguous as shown in Fig. 7. Further, the directions of DOF's at each node must be specified because they may be more convenient in certain directions to describe the stiffness of the element the nodes connects to than others. There is also a problem when it comes to describing slaving constraints. Specifying a slipping edge boundary is an example where the directions of DOF's are critical as shown in Fig. 8. For this case, the DOF directions must be parallel and perpendicular to the slipping edge.

The above requirements all suggest a model with multiple coordinate systems (CS). This adds the need to transform from one DOF CS to another when assembling substructures. The latter problem can be solved by having one CS serving as a Reference Coordinate System (RCS) and specifying all other CS's as rotations from the RCS. The RCS may also serve as the basis for describing the location in

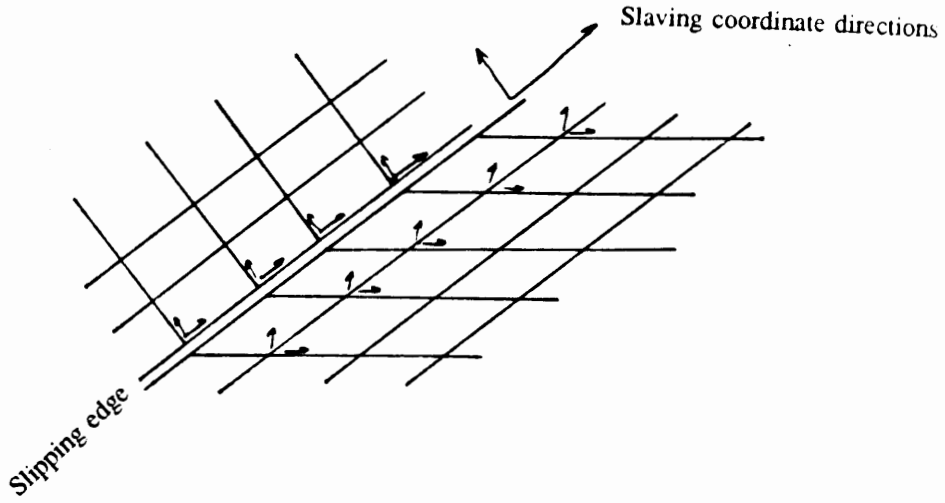


Fig. 8

Slipping edge boundary to be imposed by slaving.

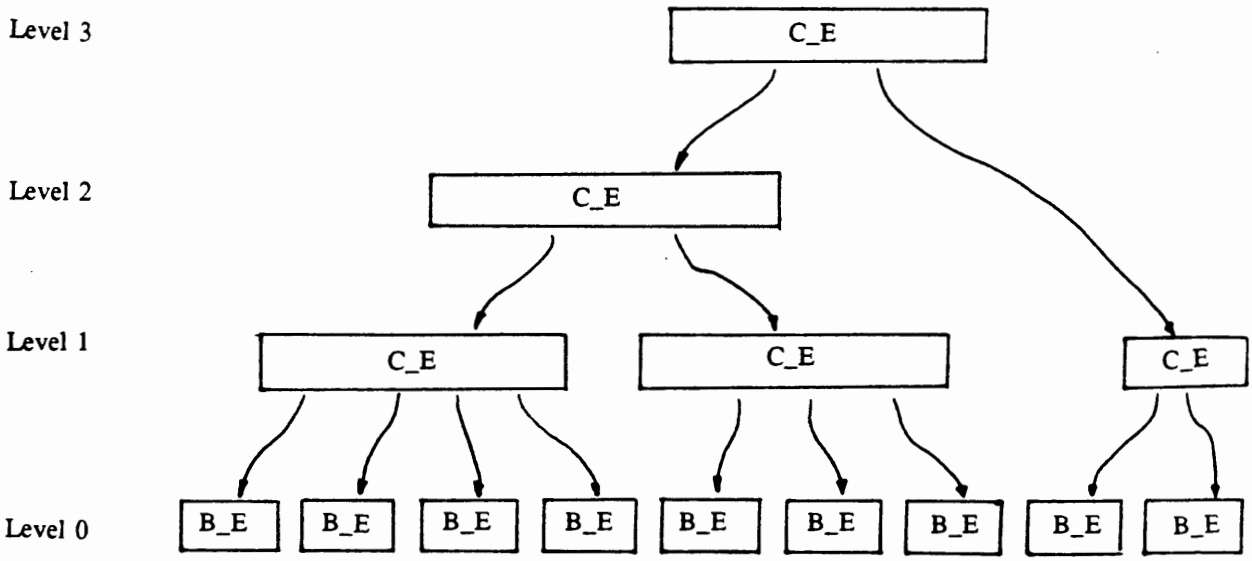


Fig. 9

Substructuring topology.

B_E = Basic_Element

C_E = Complex_Element

space of the nodes and the other CS's define directions only and need not have an origin translated from the RCS origin.

Each frame element may, then, have its own cross sectional orientation described as rotation between the RCS and the orientation of the principal axes of the section. Each node in the model data structure can have the orientation of its DOF's by specifying a CS. Also, whenever a slaving constraint is imposed, a slaving CS (SCS) is specified along directions selected to simplify the description of the slaving constraint. Finally, load and displacement boundary conditions must be specified in the DOF CS of the node they act on.

In the MDS, table DOF_CS stores the raw data needed to transform any CS to the RCS:

DOF_CS

coor_sys
(9 double)

3.3. ELEMENT REPRESENTATION

Now the attention is focused on the model data. In this sub-section, the element representation is described. Because the use of substructuring in structural analysis offers flexibility in structural description and has a potential to reduce computational effort, it is desirable to have a multi-level substructuring topology. Using the standard assembly algorithms, substructures can be assembled from *basic* finite elements (Basic_Elements). Then the static condensation process is used to reduce substructures into *super-elements* (Complex_Elements). In this process, the substructure's stiffness and loads are expressed in terms of the DOF's at the nodes that *connect* the substructure to adjacent substructures (*external/donated nodes*) and eliminate the DOF's of the *internal nodes*.

In the proposed model, elements at the lowest level of the substructuring topology (level 0) are Basic_Elements. (See Fig. 9 and Basic_Element table header.) Each such element must identify its *finite element* type. Depending on this type, the number of nodes connected to this element and the data needed to describe the elements geometric and material properties are determined. For example, frame elements have two end nodes and their geometric properties are the length of the element, the geometric

properties of its cross-section and the orientation of this cross-section. Further, elements may be built up from several segments of different materials. Attribute count determines the number of records needed to define the properties of the element. Also, the super_element pointer identifies the Complex_Element that a Basic_Element gets assembled into. Finally, a Basic_Element points to the element loads applied at itself using the no_load count and the element_load pointer.

Elements in higher levels of the substructuring topology (levels 1 and higher) are Complex_Elements. Each such element is assembled from lower level elements and has a substructuring level number that exceeds the level of its highest level sub-element by one. (See Fig. 9.) These elements must carry typical substructuring information such as its sub-elements, super_element and the donated nodes along with nodal loads.

As will be seen later, each Complex_Element in the MDS models a component in the ADB. The table header of Complex_Elements shows that each element has the id of the component it models (c_id). It also shows the substructuring data, namely, the super_element pointer, the sub_elem pointer and count and the donated nodes pointer and count. The latter pairs use *indirect pointers*, that is the objects they point to are pointers themselves (Donated_Nodes and Super_Sub records). The nodal loads and displacement boundary condition on the donated nodes of the Complex_Element are accessed through two pairs of counts and pointers (no_load, load_bc, no_disp, and disp_bc). Finally, the SubStr_Header table groups the elements in each level of the substructuring topology in a list.

Following is a list of the corresponding table headers in the MDS:

SubStr_Header

level	no_elements	first_element
(int)	(int)	(Basic/Complex_Element *)

Basic_Element

type	nodes	count	element_prop	super_element	no_load	element_load
(byte)	(Element_Nodes *)	(int)	(e.g. Frame_segment *)	(Complex_Element *)	(int)	(Element_Load *)

Complex_Element

c_id	super_element	no_sub_elem	sub_elem	no_donated_nodes	donated_nodes	no_load	load_bc	no_disp	disp_bc
(str)	(Complex_Element *)	(int)	(Super_Sub *)	(int)	(Donated_Nodes *)	(int)	(Load_BC *)	(int)	(Disp_BC *)

Donated_Nodes

old_node	new_node
(Node *)	(Node *)

Super_Sub

sub_element	level
(Basic/Complex_Element *)	(int/byte)

Element_Nodes

node
(Node *)

Frame_Segment

mat_prop	Xsection_prop	section_orientation	curvature_prop
(double)	(n double)	(9 double)	(n double)

Finally, in the ADB an attribute is present with every component that points to its Complex_Element model in the MDS. This fact is shown here as the Comp_Model table header:

Comp_Model

C_ID	CE_ID	Complex_Element
(str)	(int)	(Complex_Element *)

3.4. NODE, DOF AND SLAVING REPRESENTATION

To effectively describe the model and provide enough freedom in setting it, what is needed is a structure that gives nodes an unlimited number of DOF's. These DOF's may have a type that is not restricted to the traditional 3-D frame DOF types, namely, the three translational and three rotational

DOF's. Further, each node may have its own CS to define the directions of its DOF's.

In selecting a structure to describe the slaving constraints, it is important that this structure allows the slaving of individual DOF's at a certain node to DOF's at more than one node. Further, it is necessary that the slaving relation has its own CS to describe the directions of involved DOF's.

In the MDS, a Node record contains the location of the node in space, a pointer to a DOF CS for its DOF's, a count of its DOF's and a pointer to the first in a series of these DOF records. (See the Node table header.) A DOF record has a pointer back to its Node record and an attribute to specify its type and direction in the DOF CS. The assembly_type byte specifies if the DOF is external (value 0) , internal (negative value) or slaved (positive value). This information is important during the assembly of substructures. If it is slaved, the value of assembly_type represents the number of master DOF's and the masters pointer is set to the first in a series of Master_DOF records.

Specifying the slaving constraints on the DOF may provide the required flexibility. Each Master_DOF record represents the slaving of a DOF to a master DOF. It has a pointer to the master DOF and two other attributes which specify the nature of the slaving. The lever_arm specifies if the slaving relation has a lever arm and the direction of the arm in the slaving CS. The interpolation_type is used to specify whether to interpolate (typically when slaving to more than one node's DOF's) and the interpolation function to be used.

Following is a list of the table headers that represent DOF's in the MDS:

Node

location	dof_cs	no_dof	dof
(3 double)	(DOF_CS *)	(int/byte)	(DOF *)

DOF

node	type	assembly_type	masters
(Node *)	(byte)	(byte)	(Master_DOF *)

Master_DOF

master	lever_arm	interpolation_type
(DOF *)	(byte)	(byte)

As will be seen, a hook in the ADB is modeled by by one or several nodes. Each hook in the ADB keeps pointer(s) to the node(s) modeling it. This is shown here by the Hook_Node table header:

Hook_Node

H_ID	Node_ID	Node
(str)	(int)	(Node *)

3.5. BOUNDARY CONDITION REPRESENTATION

The finite element model boundary condition data consists of load and displacement boundary conditions. The former are composed of nodal loads and element loads. Nodal loads, as will be seen later, arise during the modeling of pruned connections and are usually applied to external nodes of Complex_Elements. Meanwhile, element loads arise due to component loads and are associated with the Basic_Elements modeling these components. On the other hand, displacement boundary conditions arise at pruned connections and are similarly associated with the external nodes of Complex_Elements.

In design, several loading conditions and combinations must be considered. For this reason, the proposed model must be able to support several loading conditions. Also, design codes specify different critical design sections to be considered. The logical process of determining which critical sections to be considered is part of the overall logical design process and is best dealt with within design tools. For this reason, the analysis module does not have to pick the critical design sections. This limits the required data for element loads to their fixed end action.

In the proposed model (see the table headers in this sub-section and in the Element Representation sub-section), nodal loads and specified displacements are represented by Load_BC and Disp_BC tables. Both tables are accessed through the Complex_Elements on which they are applied. In both cases the value of the boundary condition is specified along with a pointer to the DOF it is specified on. Simi-

larly, Element_Loads are accessed through the Basic_Elements they act on. Each such record carry the value of the load and a pointer to the DOF it acts on. Note that each boundary condition record has a pointer to a Load_Condition description record.

Following is a list of the boundary condition tables:

Load_Cond

load_cond
(str)

Load_BC

load_cond	dof	node_load
(Load_Cond *)	(DOF *)	(double)

Disp_BC

load_cond	dof	disp
(Load_Cond *)	(DOF *)	(double)

Element_Load

load_cond	DOF	initial and equivalent forces
(Load_Cond *)	(DOF *)	(double)

MODEL CREATION

4. GENERAL ALGORITHM

A second look at the component-connection model of the database is needed to identify the the steps required to model the structure. The structure is hierarchically composed of *individual* components *assembled* by connections to produce assembled components which, themselves, are assembled to others. This very same hierarchy may be used to divide the modeling of the whole structure into the modeling of the individual components and the assembled components.

The proposed strategy involves following the structure hierarchy from bottom to top. Start by the leaf components and produce models for each of them. Then assemble these models into super-elements modeling the assembled components. This step involves modeling the connections in each assembly. Repeat this process until the whole structure is assembled.

So far components and connections have been modeled. Nothing has been done to model the loads. Unlike components and connections, loads are not objects in the database. Instead they are derived from attributes of other objects, extracted from previous analysis results and/or specified by design codes. This requires special attention.

In this section, the modeling steps are presented. This includes: (1) modeling components, (2) modeling assemblies, and (3) modeling loads.

4.1. MODELING COMPONENTS

Modeling individual components involves generating a finite element model for each component. These models must satisfy several requirements. To illustrate these requirements, consider the girder example of Fig. 10. The modeler must select a finite element type that behaves structurally similar to girders. Choosing frame elements is appropriate in this example. Further, the shown girder has two end hooks and three other hooks along its length. This imposes a requirement on the produced model to be able to connect at five nodes corresponding to the girder's five hooks. This requires four frame elements and five nodes to model the girder. Further subdivisions may be necessary in other cases to increase the accuracy of the finite element model. An example of that is shown in Fig. 11 where the

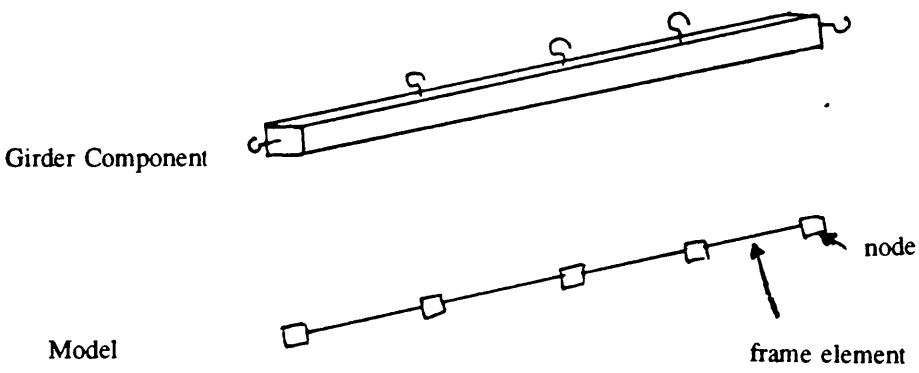


Fig. 10
Modeling a girder.

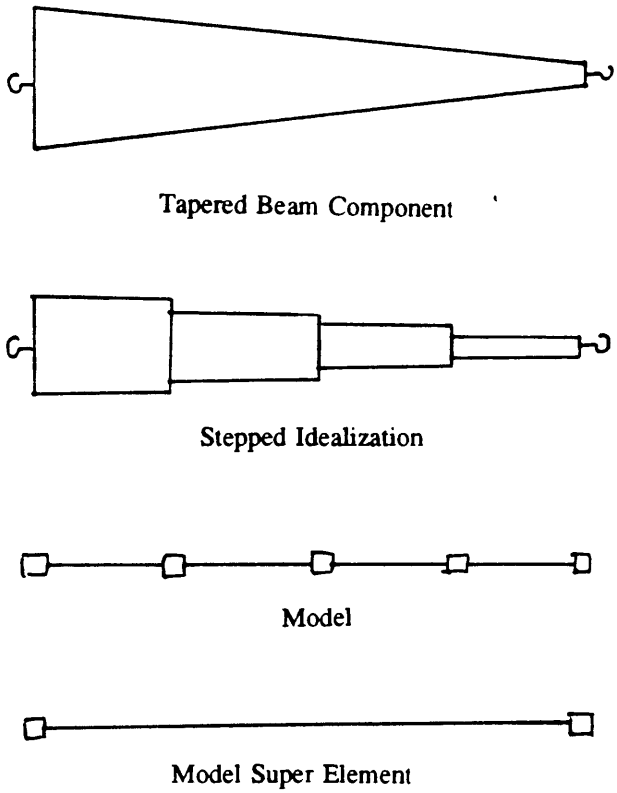


Fig. 11
Modeling a tapered beam.

tapered beam is modeled as a stepped beam.

First, the modeler has to select an element type to model the component. This depends on the nature of the component (e.g. whether it is a beam or a slab) and the type of the model (e.g. static or dynamic, linear or non-linear, and approximate or detailed). Depending on this decision the required structural properties of the component must be retrieved from the ADB or computed, and the DOF types needed at the nodes are determined. Then nodes must be created to model the hooks. Depending on the required degree of discretization, the span between the hooks may be subdivided. Next, Basic_Elements are generated to span the nodes and their properties are set. Finally, the Basic_Elements are assembled into a substructure (Complex_Element). This Complex_Element is the component's model.

Creating a node involves creating a Node record and the required DOF records and selecting the DOF directions. Similarly, generating a Basic_Element requires the creation of a Basic_Element record, setting its finite element type, pointers to its nodes and structural properties. Assembling the Basic_Elements requires creating a Complex_Element record, setting the pointers between the substructure and its sub-elements, and donating the external nodes. A detailed algorithm is proposed in Appendix A.

4.2. MODELING ASSEMBLIES (PROCESSING CONNECTIONS)

After modeling leaf components, the MDS has a collection of unconnected Level 1 Complex_Elements. The next step is to use these models to produce models of Level 2 or higher for the assembled components in the ADB. *Assembling* such models involves determining the sub-elements required, processing the connections between them and setting the substructuring data.

The structure hierarchy can be used to determine the sub-components of any assembly in the ADB. Then each sub-components identifies its Complex_Element model in the MDS.

Modeling connections involves determining whether a connection is pruned or not. If not, the modeler has to identify the nodes modeling the hooks involved in the connection. Then slaving constraints are imposed on their DOF's to force a behavior similar to that of the connection involved. Fig. 12 helps illustrate several connection types. For fixed connections (Fig. 12a), all the translational and

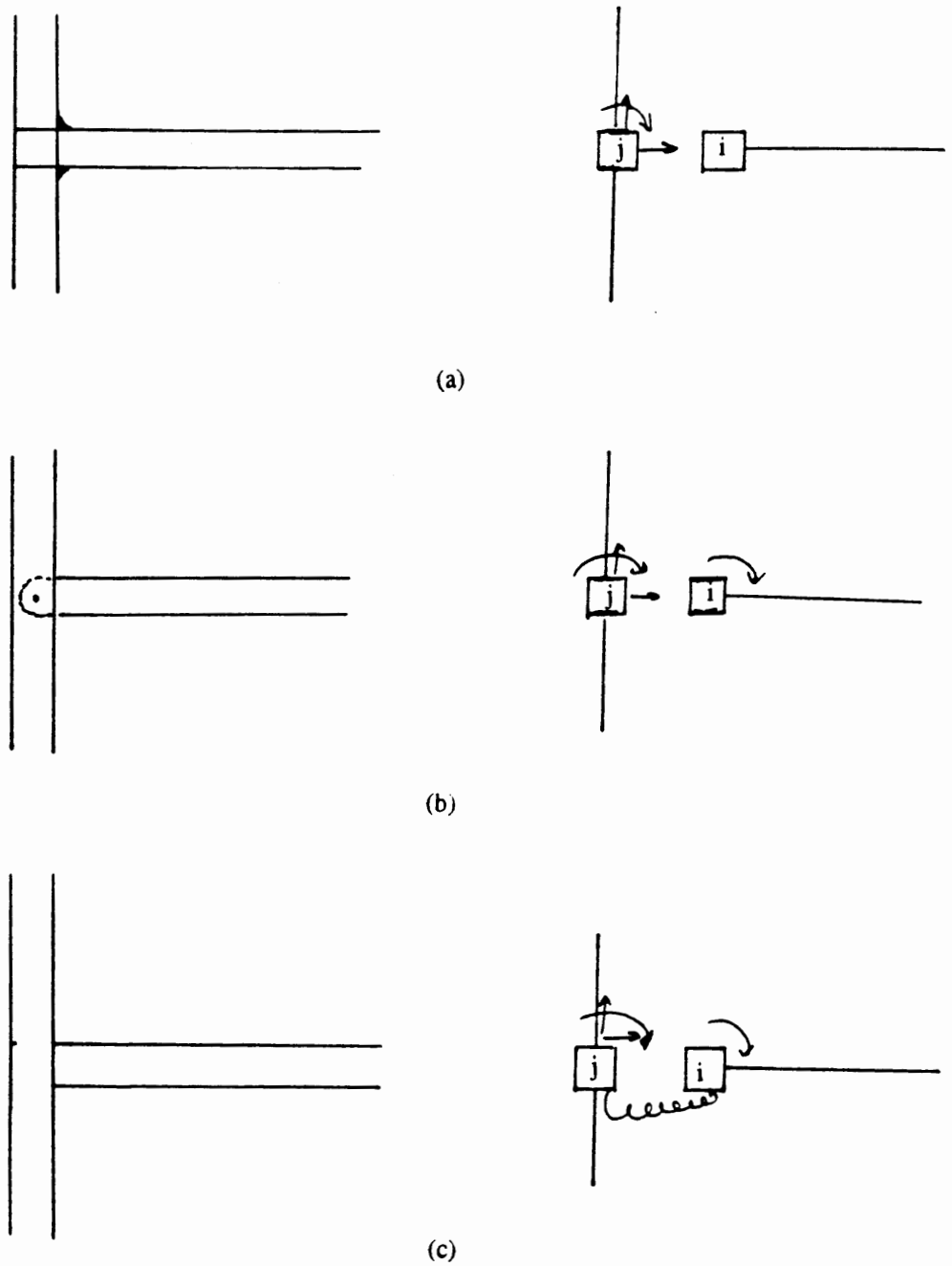


Fig. 12

Modeling connections of different types. (a) Rigid connection. (b) Pin connection. (c) Flexible connection.

rotational DOF's at nodes i are slaved to corresponding DOF's at node j. Fig. 12b shows that for a pin connection only the translational DOF's are slaved. If the connection is flexible as in Fig. 12c, special spring connection elements must be used to model the connection.

For pruned connections, the modeler must determine the load or displacement boundary conditions and/or stiffness correction needed to model the connection. In addition, slaving constraints on the retained component models must be applied if necessary.

The order of modeling assemblies becomes important. Each time an assembled model is created, each of its sub-components must have already been processed otherwise their models may not have been created yet. One way of guaranteeing this is by processing from the bottom up the hierarchy. The proposed modeler has to decide on a connection processing order then follow it.

The proposed detailed algorithms for modeling assemblies is given in Appendix B.

4.3. MODELING LOADS

Specifying the loads that act on a structure is one of the most difficult and yet important steps in the design process. The accuracy of the analysis results depends directly on the accuracy of the applied loads. Several aspects interact to complicate the load modeling.

Loads come in different types. There are concentrated loads, line loads, and surface loads. Loads that act on a structure result from the weight of the structure components, building floors, winds, snow, earthquakes, earth pressure, traffic and impacts. Thermal gradients and support displacements can also be dealt with as loads. This diversity of load types complicates the modeling process.

Typically, different load types are referred to as either dead loads or live loads. This concept is important for design. Design codes may set standard load values and/or formulas to be used to predict values for each load type. They may also require their values to be magnified by load factors. The determination of the acting loads and design factors required may best be solved by a rule based technique.

Another complication for the modeler is the load boundary conditions that must be applied at pruned connections. These loads must be brought from the results of previous analyses of the pruned

components.

For design purposes, it is difficult to determine which are the critical sections in a design unit before attempting to design it. This problem must be dealt with in the design modules. However the modeling/analysis modules must provide the design modules with enough data to calculate internal forces at any point in the design unit. This includes the raw loads on the components and the inter-element forces.

Design codes require the components to survive load combinations. For linear analysis this problem can be solved by super-position of analysis results of different load conditions. However, the principle of super-position is not valid for non-linear analysis. This makes it essential for the modeler/analysis modules to be able to combine several load conditions and analyze the structure under their action.

With the exception of load boundary conditions at pruned connections, all other loads can be dealt with as component loads (loads acting directly at the components). Loads acting on a given component may be *distributed* on the Basic_Elements modeling it. These are the element loads. The case of load boundary conditions is different in that these loads arise at the pruned connections of the assemblies and hence must be applied at the external nodes of the assembly hooks that participate in the pruned connections.

ANALYSIS AND RESULTS

5. GENERAL

Before this report can be completed, a brief discussion of the analysis module and analysis results organization must be given. First, the analysis module features are discussed. Next the data structure and algorithm of the analysis module are described. Finally, the organization of the analysis results is described.

5.1. ANALYSIS MODULE FEATURES

Several features of the analysis module can be identified by taking a second look at its input data (the MDS). The analysis module must be capable of supporting the multi-level substructuring topology of the MDS. It must support the multipile CS used in setting the model. Also, it must be able to process slaving constraints efficiently. This is important because of the extensive use of slaving constraints in the model. Finally, the analysis module must be able to support multiple load conditions.

Most of these features are supported by *off-the-shelf* analysis programs. What is needed is one program that support *all* of them. This suggests the development of a specialized analysis module for the CID system.

5.2. DATA STRUCTURE

The analysis data is different from the model data in that it includes stiffness matrices and load and displacement vectors to represent the behavior of the structures. These matrices and vectors are generated during the analysis of the model. What is needed is a data structure that can accommodate the MDS data and the analysis matrices and vectors. Extending the MDS tables provides the required analysis data structure.

The Complex_Element table must be extended to include pointers to total and reduced stiffness matrices (k_t and k_r), and load and displacement vectors (an indirect pointer to an array of pointers (Load_Dips_Vec) which has pointers to total and reduced load and displacement vectors for each load case). (See the extension of the Complex_Element table and the Load_Disp_Vec table headers). Also,

the order of the DOF's included in total and reduced stiffnesses must be added (idof_t and idof_r) and the corresponding DOF records must also be extended to include the equation number in the stiffness matrices they participate it.

Basic_Elements table must also be extended to include pointers to inter-element action corresponding to each load case (Element_Node_Reaction).

From an implementational point of view, the sizes of the tables are known before the analysis starts. This enables their implementation as arrays of records. However, dynamic storage allocation is still necessary to create storage for the different matrices and vectors. Following, the additions to the MDS tables are shown along with the Load_Disp_Vec and Element_Node_Reaction tables:

Complex_Element

load_vectors	k_t	idof_t	k_r	idof_r
(int + Load_Disp_Vec *)	(?)	(DOF *)	(?)	(DOF *)

DOF

eqn_no
(int)

Load_Disp_Vec

load_case	lv_t	dv_t	lv_r	dv_r
(Load_Case *)	(double *)	(double *)	(double *)	(double *)

Basic_Element

node_reaction
(Element_Node_Reaction *)

Element_Node_Reaction

load_case	value	dof
(Load_Case *)	(double *)	(DOF *)

5.3. ALGORITHMS

The general algorithm of the analysis module does not differ much from that of any other analysis module with multi-level substructuring capabilities. The difference is in the ability to process the slaving. Following is a listing of this module's algorithm:

+ Input data and build the data structure.
+ Assemble stiffnesses.
+ Set load vectors.
+ Solve for global displacements.
+ Back substitute displacements.
+ Process inter-element element forces.

The analysis module starts by reading the MDS and building its data structure. Then the stiffnesses of Complex_Elements are created and stored. The algorithm of forming the stiffnesses is a typical multi-level substructuring stiffness assembly algorithm with the exception of processing slaving constraints. These are dealt with in a manner similar to DOF rotation. For more details, see the proposed algorithm in Appendix C.

Similarly, the total and reduced load vectors are formed for each complex-element. A detailed algorithm is proposed in Appendix D.

Next, the simultaneous equations in the external displacements of the highest level super-element (global displacements) are solved. This may use a typical skyline or profile solution algorithm.

The back substitution of displacements algorithm is a typical multi-level substructuring back substitution algorithm except for dealing with slaving constraints. A detailed algorithm is proposed in Appendix F.

Finally, the inter-element forces calculation is the typical stiffness by displacement multiplication algorithm.

It is important to note that at no time full matrices need to be formed or stored. Neither a triple matrix product need to be performed. Instead, all transformations are proposed to be done on DOF by

DOF basis. Note also that it is assumed that a virtual memory environment with paging (UNIX like) exists. This means that the module does not have to do memory management.

5.4. RESULTS ORGANIZATION

The analysis results must be organized in a way that enables further processing by design tools and/or the modeler. Design tools need enough data to enable them determine internal action in each component at any point in the component. This is possible if the applied loads distribution along the components and the external inter-element action (internal action) at the nodal points of the model are known. Statics may then be used to determine internal action at any point within the component. These loads and results data must be indexed by the corresponding (unassembled) components.

Reactions resulting at pruned connections which were subjected to displacement boundary conditions must be part of the result data. It is an important part of the data because it is needed by the modeler as load boundary conditions to be applied at the pruned components of the structure. These reactions must be indexed by the assemblies with the pruned connections.

CONCLUSION AND FUTURE WORK

6.

The preceding sections presented the need to integrate structural engineering functions to develop a CID system. The nature of utilities needed to integrate an analysis utility with such a system through a central database was described. Ideas for automated finite element model generation and storage were discussed. The implementations will be largely guided by the discussed concepts.

In implementing the proposed modeler, particular attention will be paid to techniques that will facilitate future extensions. It is important to implement a system that is modular in nature which make adding new functions an easy task, Also, it is important to separate the knowledge of the system from the driving routines as much as possible. This expert system technique simplifies both development and expansion of the system.

This work is to be expanded into a doctoral dissertation in the CID area. Still more work need to be devoted to develop ideas for dealing with load modeling before implementation starts and much to be learned before the CID area of structural engineering reaches maturity.

REFERENCES

- Bhateja R., "A Database System for Computer-Aided Structural Engineering", College of Engineering, University of California, Berkeley, 1986.
- Date C. J., "An Introduction to Database Systems", 3rd ed., Addison-Wesley, Reading, Massachusetts, 1981.
- Kanaan A. E. and Powell G. H., "Drain-2D: A General Purpose Computer Program for Dynamic Analysis of Inelastic Structures", Report EERC 73-6, College of Engineering, University of California, Berkeley, April 1973.
- Kardestuncer H., "Elementary Matrix Analysis of Structures", McGraw-Hill Book Co., New York, 1974.
- Katz R. H., "Information Management for Engineering Design", Springer-Verlag, Berlin, 1985.
- Row D. G. and Powell G. H., "A Substructure Technique for Nonlinear Static and Dynamic Analysis", Report EERC 78-15, College of Engineering, University of California, Berkeley, August 1978.
- West H. H., "Analysis of Structures", John Wiley & Sons, Inc., New York, 1980.
- Wilson E. L., "SAP-80 - Structural Analysis Programs for Small or Large Computer Systems", Proceedings, CEPA Fall Conference, Newport Beach, California, 1980.

APPENDIX A

THE COMPONENT MODELING ALGORITHM:

1- Select a component and prepare for modeling:

+ While traversing the defined structure tree, select the first leaf component that has not been modeled yet.

+ Read its C_ID, and primitive (template) type.

+ Based on the template type and the type of analysis to be performed, select an element model.

+ Create temporary storage for processing the current element type.

+ Request from the DB the component attributes required to model the component. This includes dimensions, material properties and hook data.

+ Create a new Complex_Element record. Initialize its attributes, point to C_ID, append to level 1 list of complex elements. Append the attribute Comp_Model with the component to refer to its model.

+ If the local coordinate system (LCS) of a previously processed component is the same, set a pointer to it, otherwise, create a DOF_CS record, set its values and set a pointer to it. (Search only previously processed components under the same parent.)

2- Node creation:

+ Node creation algorithm:

For each "framed" hook do:

a- Create a node record and append to the Node table.

b- Set location, LCS pointer, and the number of DOF's to 6(for frame elements.)

c- Create the 6 DOF records, set node pointers to current node record. Append to DOF table. Set node's first DOF pointer to the first DOF record.

d- In each DOF, set type, initialize assembly_type to external and the master attributes to free.

+ Enter tuples in the Hook_Node table to point to these hooks and nodes then donate.(The node donation algorithm is to be listed.)

+ Now, create/delete nodes depending on the elements to be created (layout selection):

  1- Determine an estimate of the number of elements (NOE), NOE = number of nodes - 1,

  2- Select an end node and call 1st,

  3- Calculate the distance of each node from 1st,

  4- Sort the nodes on increasing distance from 1st.

  5- Let a pointer CURRENT point to 1st and

  Repeat:

    a- Let NEXT be a pointer to the next node after CURRENT.

    b- Calculate distance between CURRENT and NEXT.

    c- Check with design rules if length is too long. If true, create an internal node in the middle, insert between CURRENT and NEXT, increment NOE and start next loop.

    d- If the distance is too short, eliminate the NEXT node and its DOF records, decrement NOE, and start next loop.

    e- For non-prismatic elements, divide span between CURRENT and NEXT to a number sub-members, create the needed nodes, and update NOE.

    f- Let Current point to NEXT.

  Until NEXT is the last node.

3- Element creation:

+ Set the no_of_subelements in level 1 record to NOE.

+ Create NOE Super_Sub records, set pointer in level 1 record to first one.

+ Loop over NOE starting by setting CE to first Super_Sub, CURRENT to 1st node, and NEXT to node following CURRENT:

1- Create a Basic_Element record.

2- Append 2 records to Element_Nodes table pointing to CURRENT and NEXT.

3- Set nodes pointer to the first of the Element_Nodes records.

4- Set the super_element pointer in Basic_Element to the level 1 record.

5- Let CE record point to Basic_Element record, and advance the CE pointer.

6- Set Basic_Element type.

7- Determine the no_of_segments.

8- For prismatic components, set segment pointer to first segment for the group (similar to LCS treatment), otherwise create new segment record and set attributes.

9- For non-prismatic components, create segments and interpolate attributes.

APPENDIX B

THE ASSEMBLY MODELING ALGORITHM:

1- Set Connection Processing Order:

+ Scan the whole structure tree, assign depth values to each connection level encountered. (Create a list composed of a level depth and a pointer to a connection level.)

+ Sort this list on decreasing level depth.

2- Set CURRENT_LEVEL pointer to deepest connection level.

3- While CURRENT_LEVEL != NULL DO:

A- Set new substructure level. For each closed connection DO:

1- Create a Complex_Element record (CE rec).

2- Scan closed connections to determine involved components. (Request from the DB to trace connection links down to find the first encounter of a connected or leaf component.) Store in a list ignoring pruned components. Call the list SubComp list.

3- Search the Comp-Model table for the components in SubComp list and get their models. (The corresponding Complex_Element's.)

4- Scan the elements in SubComp list to find maximum level number (MLN).

5- Append new CE rec to level MLN+1, set c_id to "connection _id". Set no_sub_elem and create Super_Sub list of pointers to subelements. Set sub_elem pointer in CE rec. Set parent_element to null in CE rec and to current CE rec in involved subelements. Append a tuple to Comp_Model table.

B- Now loop over connections in current level. For each closed connection DO:

1- Scan connection links. For each unpruned leaf component, determine the involved hook. Search Hook_Node table for corresponding node and Comp_Model table for level 1 elements.

2- Follow donated node links to the highest level element corresponding to node. [algorithm:

+ set ELEM = level 1 element, NODE = lowest level node

+ repeat

  . search ELEM's donated_nodes for old_node = node

  . set node = new_node

  . set ELEM = parent_element

  until ELEM = current super element.

+ add to a table of (involved connection links, lowest level node, highest level node)]

3- Determine if framed connection has no flexibility, if true do:

a- Select a master node. (A node corresponding to a non-end hook.) Mark its DOF's internal.

b- Determine connection coordinate system (CCS) and which DOF's to be slaved. (For rigid connection, each translational DOF has one translational and two rotational master DOF's and each rotational DOF has one rotational master DOF. For a pin connection, slave translational DOF's only to translational DOF's.) Use rules to find out.

c- Transform DOF's to CCS.

d- Impose the slaving by creating the necessary Master_DOF records and setting the assembly_type to slaved (number of master DOF's) and the slaving_code and pointers to the master DOF.

e- Impose rigid end zones on involved basic elements of the master node, if desired. Consult rules.

4- If framed connection has flexibility, then:

a- Select a master node in the same way, except it will not act as a master, instead it will be divided into 2 nodes. To do that go to corresponding lowest node. Create an image new node (similar location and DOF's.) Append to Node table, connect one basic element to it and donate it on all levels till current.

b- Create a flexible joint element depending on the number of connected basic elements. Append to Basic_Element table. Connect to the level 1 element with master/split node. Create necessary Node records and connect.

c- Determine CCS and which DOF's to be slaves similar to 3.b

d- Impose the slaving similar to 3.c and 3.d.

5- For pruned connections links, check if slab connection, if yes, add stiffness to involved basic elements:

a- Follow links to leaf components, find corresponding level 1 elements and follow them to Basic_Elements.

b- Apply rules to determine effective flange width from slab.

c- Adjust basic element properties by either adding a new segment or adjusting parameters of existing segments.

6- If framed connection, check modeling decisions whether load or displacement boundary condition.

+ For displacement b.c., find the master node and append Disp_BC records pointing to its DOF's.

+ For load b.c., do nothing at this point.

7- Now it is time to donate external nodes, these are the nodes which are donated to previous levels and are not marked internal. So:

Scan all involved Sub_Elem's

For each Sub_Elem, scan its donated nodes

If all its DOF's are external, donate it.

C- Set CURRENT_LEVEL pointer to the next in the Connection Processing Order.

D- End While.

Following is the Node Donation algorithm:

1- Create an image Node record. Copy location ,dof_cs, no_dof. Create image DOF's and connect to Node.

2- Set DOF's external.

3- Create a Donated_Nodes record and set its old_node and new_node pointers.

4- Insert in the Donated_Nodes table in a location consistent with its ordered nature.

5- Update Complex_Element's donated node count (and the pointer if necessary.)

APPENDIX C

THE ASSEMBLE ALGORITHM:

+ Loop over Complex_Element levels (from 1 to max):

  For each level need to assemble and reduce stiffness matrices of each Complex_Element.

  + Loop over Complex_Elements in current level:

    Need to find all subelements, get their reduced stiffnesses and idof's, make the required CS transformation and DOF slavings, construct the idof of current element, reorder and assemble the total and reduced stiffnesses.

    Scan sub-elements:

    1- Get reduced K and idof.

       If a Basic_Element, calculate and store them. Otherwise use the pointers of the previously assembled elements.

    2- Scan donated nodes. (Skip if Basic_Element):

       + If new_node.dof_cs != old_node.dof_cs then:

         a- Build $T_{on} = T_{go}^T T_{gn}$ for each node's dof's.

         b- Do $K_n = T_{on}^T K_o T_{on}$ per node without a triple product.

       + Scan dof's of donated nodes, if slaved then:

         a- Form the slaving transformation vector T_s in terms of master dof's.

         b- Replace the slaved dof's with their master dof's in idof.

         c- Do the slaving transformation which may involve expanding old K. Do $K_m = T_s^T K_s T_s$ per dof without a triple product.

         d- Check if a new master dof in idof is repeated. If so, reduce K (and idof) by adding the rows/columns corresponding to duplicate dof's (assemble.)

3- Form the idof of the current super element by:

   a- Scan idof's of involved sub-elements.

   b- If equation number = 0, assign a sequential number and enter in idof, otherwise skip.

4- Renumber equations to reduce the band width.

5- Assemble total stiffness.

6- Extract reduced stiffness matrix.

7- Delete K and idof of Basic_Element's since they are easily generated.

APPENDIX D

SETTING THE LOAD VECTORS ALGORITHM:

+ Loop over Complex_Element levels (from 1 to max):

For each level need to assemble and reduce load vectors on each Complex_Element.

+ Loop over Complex_Elements in current level:

Need to find all subelements, get their reduced load vectors, make the required CS transformation and DOF slavings, and assemble the total and reduced load vectors.

Scan sub-elements:

1- Get reduced load vectors.

If a Basic_Element, construct the total load vectors (sum of nodal and equivalent load vectors) for each load case. Otherwise use the pointers of the previously assembled elements.

2- Scan donated nodes. (Skip if Basic_Element):

+ If new_node.dof_cs != old_node.dof_cs then:

a- Build $T_{on} = T_{go}^T T_{gn}$ for each node's dof's.

b- Do $L_n = T_{on}^T L_o$ per node without a triple product.

+ Scan dof's of donated nodes, if slaved then:

a- Form the slaving transformation vector T_s in terms of master dof's.

b- Do the slaving transformation which may involve expanding old L. Do $L_m = T_s^T L_s$ per dof without a triple product.

c- Check if a new master dof in idof is repeated. If so, reduce L by adding the values corresponding to duplicate dof's (assemble.)

3- Assemble total load vectors.

4- Extract reduced load vectors.

APPENDIX E

THE BACK SUBSTITUTION ALGORITHM:

+ Loop over Complex_Element levels (from max to 1):

  Scan Complex_Element's in current level, for each, the total displacement vectors are available, need reduced total displacement vectors of sub-elements.

  + Loop over sub-elements:

    1- Extract super-element displacements from higher levels:

    Scan donated nodes (work is done for each load case displacements):

    a- Scan DOF's of each new_node and if it is not master, then extract the displacements from higher level using DOF's eqn_no, otherwise construct T_s and extract

    d_m of master DOF's and calculate $d_s = T_s d_m$. Enter into lv_r using old_node's DOF's eqn_no.

    b- If old_node.dof_cs $\neq$ new_node.dof_cs then construct T_{on} and transform by $d_o = T_{on} d_n$ over lv_r's displacements.

    2- Back substitute in reduced stiffness to get internal displacements and construct lv_t.