NASA Technical Paper 2903

1989

A Knowledge-Based Tool for Multilevel Decomposition of a Complex Design Problem

James L. Rogers
Langley Research Center
Hampton, Virginia

National Aeronautics and Space Administration
Office of Management
Scientific and Technical Information Division

Introduction

Many engineering systems are large and multidisciplinary. Before the design of such complex systems can begin, much time and money must be invested in determining the possible interactions among the participating subsystems and their parts. For designs based on existing concepts, like commercial aircraft design, the subsystems and their interactions are usually well-established. However for designs based on novel concepts, like large space platforms, the determination of the subsystems, interactions, and participating disciplines is an important task. Moreover, this task must be repeated as new information becomes available or as the design specifications change. Determining the subsystems is not an easy, straightforward process and often important interactions are overlooked. The design manager must know how to divide the design work among the design teams so that changes in one subsystem will have predictable effects on other subsystems. The resulting subsystems must be ordered into a hierarchical structure before the planning documents and milestones of the design project are set. The success of a design project often depends on the wise choice of design variables, constraints, objective functions, and the partitioning of these among the design teams. Very few tools are available to aid the design manager in determining the hierarchical structure of a design problem and assist in making these decisions.

Recently Sobieski (ref. 1) showed the value of multilevel optimization as an approach to solving complex design problems. But to use this approach, a novel design problem must be decomposed to identify its hierarchical structure. Steward (ref. 2) developed a project management tool to organize and display the interactions among tasks in an $N \times N$ matrix format using matrix manipulations. Amarel (ref. 3) recognized the value of using artificial intelligence (AI) techniques to decompose a complex system into loosely coupled subsystems, handling the subsystem interactions, and combining partial solutions. Although much work has been done in applying AI tools and techniques to problems in different engineering disciplines (refs. 4 and 5), only recently has the application of AI tools begun to spread to the decomposition of complex design problems (ref. 6). A new tool has been developed to implement a decomposition scheme suitable for multilevel optimization. It is based on AI techniques, displays the data in an $N \times N$ matrix format, and replaces the matrix manipulations with a knowledge base to provide much more flexibility.

This paper describes the approach taken by this tool to decompose a novel, complex design problem into a multilevel structure. It begins with a discussion of the design process modeled as an optimization problem. It then presents a discussion of the functions of the tool as well as its components. A sample problem showing how the tool can be applied is used throughout the paper as an example.

A Proposed Model of the Design Process

This tool incorporates only one model out of the many possible models of the design process. Before beginning a discussion of the tool, it is necessary to lay some groundwork for the basic understanding of the approach taken for the design process. To help gain that understanding, this section discusses the elements of the design process, the purpose of the $N \times N$ matrix formulation, what is viewed as a desirable structure for the design process, and the process used to attain that desirable structure.

Elements of the Design Process

The model of the design process presented in this paper has four elements which require discussion: the design variables, the behavior variables, the constraints, and the objective function. Kirsch (ref. 7) presents more details about each of these elements.

The design process is described as a set of tasks where the completion of one task may depend on the completion of other tasks. The input to (and output from) these tasks may be fixed (parameters) or variable (design or behavior variables) during the design process. The *design variables* represent a variety of properties of the system, for example the cross-sectional area of members in a structural design. Design variables may be discrete or continuous. They are independent variables in the design process because once they are chosen, the system is completely determined. The behavior of the system can be represented by a set of *behavior variables*, for example stresses or displacements. These behavior variables

are determined from the results of intermediate analysis equations and are therefore dependent upon the value of the design variables.

If a design manager desires to produce a feasible design, that is, one that is adequate in terms of function and behavior, then certain restrictions must be placed on the range of the design variables. These restrictions are called design *constraints*. There are also behavior constraints (for example, maximum stress) which limit the behavior of the design. In this paper, both design constraints and behavior constraints are referred to as constraints and both are functions dependent upon the design variables.

If a design manager desires to produce not just a feasible design, but the best feasible design out of an infinite number of possibilities, then a function is required which is based on the design and behavior variables that can be used to compare alternative solutions. This function is called the *objective function* and may represent the weight, cost, etc., of the design. One of the most important decisions in the design process is the formulation of the objective function based on the design variables. The purpose of the design process is to find a minimum for this objective function.

One method for finding the minimum objective function is to model the design process as an optimization problem. The optimization problem determines the design variables which minimize the objective function while satisfying the constraints. As suggested by Sobieski (ref. 1), to do this effectively for a large problem may require decomposing the problem into a hierarchy of much smaller subproblems. This is called multilevel decomposition. Sobieski's approach suggests that after the problem has been decomposed, then (1) the subproblems are minimized by minimizing their constraint violations, (2) the sensitivity derivatives for each subproblem are calculated, (3) a linear extrapolation for each subproblem is formed, (4) the system is optimized for its objective function and constraints, and (5) steps (1) through (4) are repeated to attain convergence. Before these steps can begin, a tool is needed for multilevel decomposition.

The $N \times N$ Matrix

This model of the design process parallels Steward's (ref. 2) model of a system. Steward defines the structure of a system as the way in which some parts of a system affect other parts of the system. These effects differentiate a system from just a collection of parts. The semantics of the system describe how and why these effects occur. The structure and the semantics together completely describe the system. The design manager needs to study and understand both the structure and the semantics of the system. To gain this understanding, more formal tools are needed, especially as the systems become large and more complex.

The directed graph is a natural tool for describing the structure of a system. One needs to understand a few of the basics of graph theory to fully comprehend how the tool works. (Steward presents more details in ref. 2.) The directed graph consists of nodes and links. In this model for design, a *node* represents a task for computing one of the four elements of the design process. A *link* represents a relationship between two nodes. Links exiting a node indicate that the node generates some output that affects another node. Links entering a node indicate that the node requires input from another node before it can function. In this tool, a link is made between two nodes when the output of one node is part of the input to another node. A *path* from node a to node b is a sequence of links. The *length* of the path is the number of links in the sequence. A *circuit* is a path of length greater than one whose first and last nodes are the same, indicating an iterative process in design.

Another tool for describing the structure of a system is the $N \times N$ matrix. The matrix representation is better than the graph representation when the number of nodes and links is large. The nodes of the graph are placed on the diagonal of the matrix. The rows and columns of the matrix are used to link the nodes on the diagonal. An X in column a row b of the matrix corresponds to a link in the graph (output from node a, input to node b). Feedback links correspond to an X in the $N \times N$ matrix below the diagonal. If the nodes and links are placed into the matrix without any regard for ordering, then very little information regarding the desirable structure of the system is available to the design manager (see fig. 1).

Desirable Structure

The desired structure of the design process is postulated and this tool provides a method for reaching that structure. Note that the method is heuristic, has not been proven to converge, and may not have a unique solution.

Feedback links increase the cost of the solution because they imply that information is required before it is really available. This, in turn, implies that guesses must be made to initiate the process and that iterations are necessary. The aim of multilevel decomposition is to order the nodes and the links in such a way that a number of smaller uncoupled optimization problems can be identified. Therefore, a new tool is needed that will group and order the nodes of the initial $N \times N$ matrix representation of the system into a structure that limits the number of feedback links and decomposes the nodes into a hierarchical set. (This is quite different from making a matrix banded.)

One may remark, parenthetically, that limiting the feedback links is not the only means for improving the design process. The design manager may also want to make changes after examining the trade-offs between limiting the feedback links and the potential gains from parallel processing. Even though the natural order of processing for some tasks may be sequential, these tasks can be ordered for parallel processing by artificially introducing feedback links and therefore iterations. This is done by assuming that certain pieces of information are available when, in reality, they have not yet been computed. The examination of these trade-offs is beyond the scope of this paper.

This tool limits the number of feedback links by partitioning the nodes of the system into circuits. Circuits represent subsystems where each node is simultaneously dependent on all the other nodes within the same circuit. Feedback links are contained within circuits indicating that an iteration is required. Circuits are connected to each other only by feedforward links. Since there are no feedback links among circuits, there is no iteration among circuits and they can be ordered in a multilevel format. Thus a complex design process can be decomposed into a hierarchical set of tasks.

The process of attaining the desired structure is iterative and interactive. An input file is created to describe the user's perception of the relationships among the various elements of the design problem. Once the tool has been executed and a structure has been proposed, the semantics of the system as well as the structure of the system must be considered before a final decision is made on choosing the best decomposition. The design manager looks at the structure in the $N \times N$ matrix and makes changes to improve the structure to meet the known semantics and requirements. Changes may include removing or adding links, moving nodes within a circuit, or redefining the nodes which make up the system. This process continues until the design manager reaches the desired structure of the system.

A diagram of the system decomposed by Padula (ref. 8) is shown in figure 2 as an example. The reference explains the changes made by the design manager to arrive at this particular decomposition. After the system was decomposed, it was divided among design teams such as *structures* and *controls* as indicated in the figure.

Functions of the Tool

This tool performs several useful functions. These functions are planning, scheduling, displaying the $N \times N$ and dependency matrices, displaying the multilevel organization, and examining the potential time savings by exploiting parallel processing of the circuits and modules. (The term *module* is used interchangeably with the term *node* from graph theory.) Each of these functions is located within a subroutine of the main program (fig. 3). The planning function is always done first, followed by the scheduling function. Calling the other functions depends on the needs of the user. After each function is completed a file containing the current list of modules is written. This allows the user to restart the process without having to go back to the start each time.

The functions of the tool are discussed in the remainder of this section, using a generic design problem as a sample problem. All figures and tables in this paper, except figures 2 and 3, pertain to this particular problem. The problem has 45 modules. These modules perform one of the following tasks: (1) set the value of one or more design variables, (2) evaluate one or more constraint functions, (3) calculate

intermediate results and behavior variables, and (4) evaluate the objective function. The problem is defined in terms of the relationships among these various design elements in table I. The dependency of the objective and constraint functions on the design and behavior variables can be defined explicitly by mathematical equations. The same is true for defining the dependency of the behavior variables on the design variables. However, the dependency of the design variables on the functions depends on the design manager's view of the problem; therefore engineering judgment is required when determining these dependencies. The main requirement is that a design variable can only depend on a function if that function is dependent on the design variable.

Planning

The term *planning* within the context of this tool means determining which modules contribute to the solution of the problem. The user begins with a list of modules as shown in table II. This list should contain all modules that might be used in the problem. The first step in the planner is to determine whether or not a module contributes to the problem. This is done by checking the output of each module against the input requirements of the other modules. If the output of the module is contained in the input list of at least one other module then that module contributes to the solution of the problem. If a module is found not to be a contributor then it is removed from the list of modules but saved for possible use later. If two modules have duplicate output, then either one module is removed or the output is renamed. The output is renamed when more than one source of the same output is required, for example two sources with different execution times and accuracy.

In the second step, the planner examines the input lists of all the modules to determine if all input requirements are satisfied by the output of other modules. Some modules have no input requirements. These modules are used for initialization purposes by representing external inputs or have their input requirements satisfied by external inputs when they are used to retrieve the value of one or more design variables. If an input requirement is not satisfied, then the user must add a new module to the list interactively or remove the input requirement. If a new module is added, its input requirements are also checked. If one or more of its input requirements are not met, then the modules removed from the list earlier are checked first to determine if they satisfy the requirement; if not, then another module must be added. This step continues until all input requirements are satisfied.

At this point the list of modules contains only those modules contributing to the problem (see table III where the modules that have been added to the list, TASKD04 and TASKC02, are denoted by an asterisk). All extraneous modules have been removed from the list (see table II where the modules that have been removed from the list, TASKD99, TASKC98, and TASKB50, are denoted by an asterisk).

Scheduling

The scheduling function is the heart of this tool. Within the context of this tool, *scheduling* means the ordering of the modules into a meaningful solution sequence while limiting the number of feedback links among the modules. The scheduling function reorders the modules based on their links. The links of the initial data for the sample problem are very disorganized and contain a substantial number of feedback links (fig. 1). Limiting the feedback links among the modules is done by examining the links and grouping the modules into circuits. This tool also orders the modules within the circuits and orders the circuits within the design process. While Steward (ref. 2) implements the grouping into circuits with matrix manipulations, this tool follows the same steps but replaces the matrix manipulations for grouping by applying rules contained in a knowledge base. Additional rules have been added to control the ordering of the modules within circuits and the ordering of circuits within the design process.

The list of modules output from the planning function is used as input to the scheduling function. In **step 1**, the scheduling function finds all modules with no input requirements and adds them at the top of the $N \times N$ matrix. In addition, these modules are removed from further consideration.

In **step 2**, the tool determines which modules are tightly coupled. Two modules are tightly coupled when the output of module a is an input to module b and the output of module b is an input to module a. These modules are collapsed into a single module. Collapsing two or more modules into a single module

considerably speeds up the scheduling function because all the modules that have been collapsed are removed from further consideration. Only the single module containing lists of the input requirements and output values of the collapsed modules remains for further consideration. Although the collapsed modules are no longer considered, their data remain available for later use and display.

In **step 3**, if no modules remain for consideration go to **step 6**. If modules remain, then all remaining modules are examined to see if they have an immediate predecessor among the modules remaining under consideration. Module a is a predecessor of module b if there is a path from module a to module b, and module a is an immediate predecessor if there is a link directly from module a to module b. If every remaining module has an immediate predecessor, then go to **step 4**, otherwise go to **step 5**.

To begin **step 4**, one of the remaining modules is chosen as a starting point. An immediate predecessor module is chosen, then an immediate predecessor module of this module, etc. This continues until some module is encountered a second time. At this point a circuit has been found. One module in the circuit is chosen and all the other modules in the circuit are collapsed into that module, which then represents all modules within that circuit. The representative module has a link to or from another module if and only if some module in the circuit had a link to or from the other module. All modules and their links in the circuit other than the representative module are removed from further consideration. **Step 3** is repeated.

To reach **step 5**, a module exists which does not have an immediate predecessor among the remaining modules. This module along with all the links entering into it or exiting from it are removed from consideration. As modules are removed from consideration they are added to the $N \times N$ matrix, starting from the top left-hand corner moving down the diagonal, thus the modules and circuits are properly sorted. This implies that any modules within a circuit added to the $N \times N$ matrix may require input from modules above them. Modules already a part of the $N \times N$ matrix will not require the output values of any modules just added, therefore there are no feedback links among the circuits. The modules are given a new number as they are reordered into circuits and added to the $N \times N$ matrix. **Step 3** is repeated.

At **step 6** no module remains to be tested for an immediate predecessor, and this part of the scheduling is complete. All the modules have been given new numbers and a circuit number has been added to each module in the list. The only feedback links exist within circuits. There are no feedback links from one circuit to another.

One of the advantages of using a knowledge-based tool over matrix manipulations is the ease with which new rules can be added. This gives the knowledge-based tool more flexibility. For **step 7**, more rules have been added to the scheduling function that were not in Steward's (ref. 2) procedure. These new rules order the modules within the circuit and were developed in conjunction with Padula's design problem (ref. 8). The ordering is done based on the weight assigned to the modules. This step reorders the modules within a circuit by moving the modules with the highest weight to the beginning of the circuit. The modules with ever-decreasing weights are moved to be below but near the top priority modules to which they are linked. Using this method, tasks can begin as soon as possible but the modules with the highest weights are given priority. (In the sample problem, the objective function module has a weight of 4, the design variable modules have a weight of 3, the behavior variable modules have a weight of 2, and the constraint modules have a weight of 1.) Once this step is completed the design manager can examine the $N \times N$ matrix display (see fig. 4) and use the graphics interface to manipulate the modules and links to meet the requirements and semantics of the problem.

Multilevel Organization

The circuits and their links can also be displayed in an $N \times N$ matrix form (fig. 5). By examining the circuits, it is apparent that there are no feedback links among the circuits, therefore there is no iteration among the circuits. The only iterations are contained within the circuits. Thus, once the circuits have been found during the scheduling function, it is simple to achieve a multilevel organization of the problem. The knowledge base scans a list of circuits to determine the multilevel hierarchy. As circuits which have their input requirements satisfied are found, they are placed on a level. A circuit is placed on the level below the lowest level containing a circuit which generates input for the circuit being placed (fig. 6).

Dependency Matrix

Another function of the tool is to build the dependency matrix of the problem. The usefulness of this matrix is described by Barthelemy (ref. 9). It is an ordered table that identifies the functional dependence between constraints and independent design variables. Behavior variables can be evaluated using design variables, therefore each behavior variable can be replaced by a list of independent design variables. Each constraint is examined to determine its dependency on design and behavior variables. Whenever a constraint depends on a behavior variable, the dependency of that behavior variable on the independent design variables is substituted. For example, if behavior variable 1 (BV01) is dependent on design variables 2 and 3 (DV02 and DV03) and constraint 4 (G004) is dependent on design variable 5 (DV05) and behavior variable 1 (BV01) then

$$\text{BV01} = f(\text{DV02}, \text{DV03})$$

$$\text{G004} = f(\text{DV05}, \text{BV01})$$

where $= f(...)$ means *is a function of* and G004 is indirectly dependent on BV01.

After substitution

$$\text{G004} = f(\text{DV05}, \text{DV02}, \text{DV03})$$

where G004 is directly dependent on DV05, DV02, and DV03. This produces a rectangular matrix with the constraints listed along the rows and the independent variables along the columns (fig. 7). An X marks the dependency. Building the dependency matrix after the planning and scheduling functions reveals dependency patterns that may prove advantageous when developing multilevel optimization algorithms. The module numbers in the figure reflect the renumbering after scheduling.

Exploiting Parallel Processing

Each module in the sample problem was assigned an arbitrary execution time requirement. If the modules were executed sequentially, 1841 units of time would be required. But suppose the design project has a time constraint placed on it, causing the design manager to examine time savings options. One option would be to execute some of the modules or circuits in parallel. This tool allows the user to see two methods of exploiting parallel processing. The first method shows the benefits of exploiting parallelism within the circuits, while the second method shows the benefits from executing the circuits in parallel.

To determine the savings from executing modules within circuits in parallel, the problem is first broken down into circuits. Then the modules within the circuits are examined to see how they might be executed in parallel. Modules can begin execution if modules that satisfy their input requirements have completed execution. A list of the modules executing in parallel is kept along with their time requirements. The maximum number of modules executing in parallel at any one time indicates the number of processors that will be required. The tool lists the amount of time that can be saved by executing certain modules in parallel and the number of processors that would be needed.

A more substantial time savings can probably be realized by executing the circuits on the same level in parallel. The maximum time required to complete execution at a level is determined by finding the circuit requiring the maximum time at that level. The level times are totaled and subtracted from the total sequential time to determine the time that could be saved by executing the circuits in parallel. The number of processors that will be required is determined by the maximum number of circuits at any one level. The tool lists the amount of time that can be saved by executing the circuits in each level in parallel and the number of processors that would be needed.

If each circuit corresponds to an optimization subproblem in a multilevel decomposition, then time estimates for each circuit depend on the number of iterations allowed by the optimizer. The number of iterations is unknown a priori. It is felt, however, that an arbitrary number of iterations may be assumed. As shown by Padula and Sobieski (ref. 10), the number of iterations may be kept low without significantly

impeding the convergence of the whole process. To demonstrate the potential time savings from using parallel execution, the number of iterations in the sample problem was assumed to be one. Executing the modules in the circuits in parallel would save 128 units of time out of 1841, and would require 4 processors, By executing the circuits in parallel, a substantial time savings of 1009 time units out of 1841 could be realized. The times for each circuit are shown in table IV with the times for each level (the maximum circuit time within that level) indicated by an asterisk. This process would also require four processors.

Components of the Tool

This section describes the workings of the components of the tool. The user begins the design of a complex system that is divisible into modules by determining the outputs that contribute to the objective function and constraint functions of the system. The user divides the system into these modules and determines the input and output of each module, creating a data file for the main program of the tool. The main program is written entirely in FORTRAN. The other components—DI-3000 (ref. 11) for the graphics and CLIPS (C Language Production System, ref. 12) for the inference engine—were added by linking existing software to the main program. The rules are contained on seven different files which are loaded into the knowledge base as needed. A diagram of the tool is shown in figure 3.

Input

The data file contains the number of modules and a list of the modules whose output values contribute to the objective function of the system. The input of the sample problem is shown in table II.

The format of a single line in the list is

module number name weight time output **unknown** input-list

where the items in **bold** print are not to be changed by the user, and the items to be changed by the user are described below:

number	the number of the module
name	the name of the module
weight	a number defining the element of the design process
time	an estimated execution time requirement for the module
output	the output value created by the module
input-list	a list of all the required input values

The Main Program

The main program controls the execution of the tool through a system of menus. Through the main menu, the user has the choice to plan, schedule, display the $N \times N$ matrix, display the multilevel organization, examine parallelism, or examine the dependency matrix. Each of these areas has been discussed in detail in the section on functions of the system. Depending upon the choice from the menu, the main program calls a subroutine which performs the desired task. Each subroutine reads a file of rules, reads the necessary data, asserts facts into the knowledge base, and executes the CLIPS inference engine. Data are returned from the knowledge base to a single subroutine, KBANS1, where they are stored in a common block for later use. All the calls to the DI-3000 graphics package are made from a single subroutine, GRFXEC, making it simple for the user to replace DI-3000 with another graphics package.

The Graphics System

The DI-3000 graphics package (ref. 11) is a device independent graphics system and is the primary graphics package used at NASA Langley Research Center. Since the graphics calls are very simple

operations such as moving the cursor, drawing lines and circles, text, and receiving data from the mouse (or arrow keys), it should pose no problem to switch to a different graphics package. The graphics window is divided into two parts. One is a dialogue area for the user to interface with the tool, and the other is a graphics display.

The data can be displayed on a Tektronix 4014 window of the DEC VaxStation or on a DEC VT240 color monitor. The main display is the $N \times N$ matrix display of the modules, their links, and the circuits. The modules are displayed as boxes down the diagonal. On the VT240 color monitor, each module is given a color according to its weight. This makes it much easier to see the relationships among the different types of modules and also helps in manipulating the modules within the circuits. The display of the links in the $N \times N$ matrix is slightly different from Steward's (ref. 2). The links are lines connected horizontally to a box to indicate an output from that module and vertically to indicate an input into that module. A circle on the links indicates the interface between two modules. Circuits are larger boxes containing the smaller boxes for modules. The user can display 25 or 50 modules at a time. A menu is used to make a selection of what data are to be displayed. The user can display the links, the circuits, or both the circuits and the links. In addition, the user can move modules around in the matrix, list the modules, or examine module data in detail. The user can also use cross-hairs guided by a mouse (or arrow keys) to display the interface data between two modules. Since the display is static, it is possible that not all modules can fit onto a single display. To allow the user the capability of seeing all the modules, the user can specify which module is to be at the top of the display. The knowledge base scans the list of modules and returns the appropriate data for display.

The Knowledge-Based System

The CLIPS system (ref. 12) is a knowledge-based system developed at NASA Johnson Space Center. It is written in C, performs forward chaining based on the Rete pattern-matching algorithm, and has a FORTRAN interface. There are three main parts to a knowledge-based system: the facts, the rules, and the inference engine.

Facts are the basic form of data in the knowledge base and are contained in a facts list. A fact is composed of several fields with each field being separated by a space. A field can contain a number, a word, or a string. Facts are asserted into the facts list before execution by the *deffacts* construct or by an assert command in the calling program, or during execution as the action caused by executing a rule. An example of a fact about a module is

```
(module ?no ?name ?wt ?tm ?out ?stat $?inlist)
```

The knowledge base also contains *rules* which are defined by the *defrule* construct. A rule states that specific actions, the right-hand side (RHS), are to be taken if certain conditions, the left-hand side (LHS), are met. The symbol => separates the LHS from the RHS. If and only if all conditions on the LHS are satisfied, then the actions on the RHS will be performed sequentially. Each rule must contain at least one condition and one action; however, there is no upper limit on either the number of conditions or the number of actions. A rule executes based on the existence or nonexistence of facts in the facts list. Currently there are 156 rules divided among 7 files. The example below is a rule for determining that a link exists between two modules.

```
(defrule links                                          ; names the links rule
   (module ?no1 ?name1 ?wt1 ?tm1 ?out1 ?stat1 $?inlist1)   ; list for module 1
   (module ?no2 ?name2 ?wt2 ?tm2 ?out2 ?stat2 $?inlist2)   ; list for module 2
   (test (member ?out1 $?inlist2)) =>                   ; test for membership in input list
      (KBANS1 LINK ?no1 ?no2))                          ; return answer to main program
```

This is interpreted to read as follows: If there are two different modules where the output of one is an input into the other, then a link exists between those two modules. Any words following a ";" are

a comment. The parameters preceded by a "?" are single-field variables ("$?" is a multifield variable) and can take on any values for matching purposes. The action, based upon the three conditions being met, is to return to the main program via the KBANS1 parameter the fact that a link exists between module numbers ?no1 and ?no2. The LINK parameter shows where to store the numbers in the KBANS1 subroutine. If, for example, module number 7 has the output parameter DV07, and module number 13 has the input parameter list DV01 DV07 DV20, then the test would succeed and the module numbers 7 and 13 would be returned to the KBANS1 subroutine to indicate a link exists between the two modules.

The *inference engine* in CLIPS applies the knowledge (rules) to the data (facts). Pattern matching occurs on the LHS for the fixed terms and the single- and multiple-field variables. The basic execution cycle begins by examining the knowledge base to determine if the conditions of any rules have been met. All rules with currently met conditions are pushed onto the *agenda* which is essentially a push down stack. Once the agenda is complete, the top rule is selected and its RHS is executed. As a result of the action(s) of the rule execution, new rules may be placed on the agenda and rules already on the agenda may be removed. This cycle is repeated until all rules that can execute have done so. The main program passes control to CLIPS for execution of the inference engine and CLIPS passes control back to the main program after all the rules have executed.

Concluding Remarks

A tool using AI techniques has been developed for decomposing complex design problems into a suitable multilevel structure based on the multilevel optimization approach. This tool requires an investment of time to generate and refine the input for each design module. This investment may not be justified for a small, well-understood problem, but should save a significant amount of money and time in organizing a new design problem where the ordering of the modules is still unknown. The decomposition of a complex design system into subsystems requires an interaction with the judgment of the design manager. This tool can aid the design manager in making decomposition decisions early in the design cycle.

This tool provides help to the design manager by reordering and grouping the modules based on the links (interactions) among the modules. The modules are grouped into circuits (the subsystems) and displayed in an $N \times N$ matrix format. The feedback links, which indicate an iterative process, are limited and restricted to be within a circuit. Since there are no feedback links among the circuits, the circuits can be displayed in a multilevel format. Thus, a large amount of information is reduced to one or two displays. The displays are stored and can be easily retrieved and modified. The design manager and leaders of the design teams are given a visual display of the design problem and the intricate interactions among the different modules so that they can see how a change in one subsystem will effect other subsystems. It also helps reduce the possibility of overlooking important links.

The tool gives the design manager the capability of examining the potential savings in time by executing some of the modules in a circuit in parallel. A substantial time savings can be obtained if circuits on the same level of the multilevel structure are executed in parallel. The time savings as well as the number of processors that will be required are determined. In addition to decomposing the system into subsystems, the tool examines the dependencies of the problem and creates a dependency matrix. This matrix shows the relationship among the independent design variables and the dependent objective and constraint functions.

Since the tool is based on AI knowledge base techniques, it has proven to be very flexible in adding new capabilities. Given its current capabilities, this knowledge-based tool can provide the design manager with a great deal of insight in decomposing large, complex design systems into more manageable subsystems, thereby saving considerable time and money in the total design process.

NASA Langley Research Center
Hampton, VA 23665-5225
March 28, 1989

References

1. Sobieszczanski-Sobieski, Jaroslaw: *A Linear Decomposition Method for Large Optimization Problems—Blueprint for Development.* NASA TM-83248, 1982.
2. Steward, Donald V.: *Systems Analysis and Management: Structure, Strategy and Design.* Petrocelli Books, Inc., c.1981.
3. Amarel, S.: *Artificial Intelligence and Design: Opportunities, Challenges, Research Problems and Directions.* LSCR-TR-113, Dep. of Computer Science, Rutgers Univ., July 1988.
4. Sriram, D.: Bibliography—A Bibliography on Knowledge-Based Expert Systems in Engineering. ACM *SIGART Newsl.*, no. 89, July 1984, pp. 32–40.
5. Sriram, D.; and Joobbani, R., eds.: Special Issue—AI in Engineering. ACM *SIGART Newsl.*, no. 92, Apr. 1985, pp. 38–127.
6. Rogan, J. E.; and Kolb, M. A.: *Application of Decomposition Techniques to the Preliminary Design of a Transport Aircraft.* NASA CR-178239, 1987.
7. Kirsch, Uri: *Optimum Structural Design—Concepts, Methods, and Applications.* McGraw-Hill Book Co., c.1981.
8. Padula, Sharon L.; Sandridge, Chris A.; Haftka, Raphael T.; and Walsh, Joanne L.: Demonstration of Decomposition and Optimization in the Design and Experimental Space Systems. *Recent Advances in Multidisciplinary Analysis and Optimization*, Jean-François M. Barthelemy, ed., NASA CP-3031, Part 1, 1989, pp. 297–316.
9. Barthelemy, J-F M.: *Engineering Applications of Heuristic Multilevel Optimization Methods.* NASA TM-101504, 1988.
10. Padula, S. L.; and Sobieszczanski-Sobieski, J.: A Computer Simulator for Development of Engineering System Design Methodologies. *International Conference on Engineering Design—ICED 87,* W. E. Eder, ed., American Soc. of Mechanical Engineers, 1987, pp. 147–161.
11. *DI-3000 User's Guide.* Central Scientific Computing Complex Document G-5, Precision Visuals, Inc., c.1984.
12. Riley, Gary; Culbert, Chris; Savely, Robert T.; and Lopez, Frank: CLIPS: An Expert System Tool for Delivery and Training. *Third Conference on Artificial Intelligence for Space Applications—Part I,* J. S. Denton, M. S. Freeman, and M. Vereen, compilers, NASA CP-2492, 1987, pp. 53–57.

Table I. Relationships Among the Design Elements of the Sample Problem

[The notation $=f(...)$ means *is a function of*]

Design variables

1. DV01=f(G011, G012, G013)
2. DV02=f(G011, G012, G013)
3. DV03=f(G011, G012, G013)
4. DV04=f(G003)
5. DV05=f(G003)
6. DV06=f(G014, G015)
7. DV07=f(G014, G015)
8. DV08=f(G014, G015)
9. DV09=f(G004, G005)
10. DV10=f(G004, G005)
11. DV11=f(G004, G005)
12. DV12=f(G016, G017)
13. DV13=f(G016, G017)
14. DV14=f(G016, G017)
15. DV15=f(G016, G017)
16. DV16=f(G006, G007, G008)
17. DV17=f(G006, G007, G008)
18. DV18=f(G006, G007, G008)
19. DV19=f(G009, G010)
20. DV20=f(G009, G010)
21. DV21=f(G001, G002)
22. DV22=f(G001, G002)
23. DV23=f(OB01)
24. DV99=f(G001, G003, G030)

Behavior variables

1. BV01=f(DV12, DV13, DV14, DV15)
2. BV02=f(DV21, DV22)
3. BV03=f(DV01, DV02, DV03)
4. BV04=f(DV09, DV10, DV11)
5. BV50=f(DV01, DV02, DV03)

Constraint functions

1. G001=f(DV16, DV17, BV02)
2. G002=f(DV18, BV02)
3. G003=f(DV01, DV02, DV03, DV04, DV05)
4. G004=f(DV06, DV10, DV11, BV04)
5. G005=f(DV07, DV08, DV10, DV11, BV04)
6. G006=f(DV12, DV13, DV16, DV17, DV18)
7. G007=f(DV12, DV13, DV16, DV17, DV18)
8. G008=f(DV12, DV13, DV16, DV17, DV18)
9. G009=f(DV14, DV19, DV20)
10. G010=f(DV15, DV19, DV20)
11. G011=f(DV01, DV02, DV03)
12. G012=f(DV23, BV03)
13. G013=f(DV23, BV03)
14. G014=f(DV06, DV07, DV08, DV23)
15. G015=f(DV06, DV07, DV08, DV23)
16. G016=f(DV23, BV01)
17. G017=f(DV23, BV01)
18. G098=f(DV18, DV26, DV32)

Objective function

1. OB01=f(DV23)

Table II. Original Input Data

[Italicized column heads are not part of the input file]

46	*No.*	*Name*	*Wt.*	*Tm.*	*Out.*	*Status*	*Input*
module	1	TASKC10	1	27	G010	unknown	DV15 DV19 DV20
module	2	TASKD07	3	12	DV07	unknown	G014 G015
module	3	TASKD17	3	12	DV17	unknown	G006 G007 G008
module	4	TASKD23	3	81	DV23	unknown	OB01
module	5	TASKD20	3	17	DV20	unknown	G009 G010
module	6	TASKD15	3	23	DV15	unknown	G016 G017
module	7	TASKB03	2	53	BV03	unknown	DV01 DV02 DV03
module	8	TASKC14	1	18	G014	unknown	DV06 DV07 DV08 DV23
module	9	TASKC07	1	18	G007	unknown	DV12 DV13 DV16 DV17 DV18
module	10	TASKC15	1	62	G015	unknown	DV06 DV07 DV08 DV23
module	11	TASKD21	3	12	DV21	unknown	G001 G002
module	12	TASKC04	1	15	G004	unknown	DV06 DV10 DV11 BV04
module	13	TASKC17	1	35	G017	unknown	DV23 BV01
module	14	TASKC06	1	23	G006	unknown	DV12 DV13 DV16 DV17 DV18
module	15	TASKC03	1	24	G003	unknown	DV01 DV02 DV03 DV04 DV05
module	16	TASKC13	1	17	G013	unknown	DV23 BV03
module	17	TASKB04	2	70	BV04	unknown	DV09 DV10 DV11
module*	18	TASKD99	3	84	DV99	unknown	G001 G003 G030
module	19	TASKD11	3	22	DV11	unknown	G004 G005
module	20	TASKD02	3	12	DV02	unknown	G011 G012 G013
module	21	TASKC01	1	35	G001	unknown	DV16 DV17 BV02
module*	22	TASKC98	1	10	G098	unknown	DV18 DV26 DV32
module	23	TASKC16	1	55	G016	unknown	DV23 BV01
module	24	TASKD13	3	15	DV13	unknown	G016 G017
module	25	TASKD05	3	53	DV05	unknown	G003
module	26	TASKD14	3	74	DV14	unknown	G016 G017
module	27	TASKC08	1	53	G008	unknown	DV12 DV13 DV16 DV17 DV18
module	28	TASKB02	2	17	BV02	unknown	DV21 DV22
module	29	TASKD10	3	42	DV10	unknown	G004 G005
module	30	TASKC09	1	98	G009	unknown	DV14 DV19 DV20
module	31	TASKC11	1	14	G011	unknown	DV01 DV02 DV03
module	32	TASKD16	3	67	DV16	unknown	G006 G007 G008
module	33	TASKD06	3	68	DV06	unknown	G014 G015
module	34	TASKD19	3	74	DV19	unknown	G009 G010
module	35	TASKD03	3	46	DV03	unknown	G011 G012 G013
module	36	TASKD09	3	57	DV09	unknown	G004 G005
module	37	TASKD12	3	25	DV12	unknown	G016 G017
module	38	TASKC12	1	19	G012	unknown	DV23 BV03
module	39	TASKD22	3	57	DV22	unknown	G001 G002
module	40	TASKD18	3	84	DV18	unknown	G006 G007 G008
module	41	TASKD01	3	40	DV01	unknown	G011 G012 G013
module	42	TASKD08	3	93	DV08	unknown	G014 G015
module	43	TASKF01	4	44	OB01	unknown	DV23
module	44	TASKB01	2	62	BV01	unknown	DV12 DV13 DV14 DV15
module	45	TASKC05	1	12	G005	unknown	DV07 DV08 DV10 DV11 BV04
module*	46	TASKB50	2	39	BV50	unknown	DV01 DV02 DV03

*Indicates modules not contributing to solution which are removed during planning function.

Table III. Modules After Planning

[Italicized column heads are not part of the input file]

45	*No.*	*Name*	*Wt.*	*Tm.*	*Out.*	*Status*	*Input*
module	25	TASKD05	3	53	DV05	ok	G003
module*	45	TASKD04	3	44	DV04	ok	G003
module	35	TASKD03	3	46	DV03	ok	G011 G012 G013
module	33	TASKD06	3	68	DV06	ok	G014 G015
module	34	TASKD19	3	74	DV19	ok	G009 G010
module	17	TASKB04	2	70	BV04	ok	DV09 DV10 DV11
module	6	TASKD15	3	23	DV15	ok	G016 G017
module	37	TASKD12	3	25	DV12	ok	G016 G017
module	21	TASKC01	1	35	G001	ok	DV16 DV17 BV02
module	38	TASKC12	1	19	G012	ok	DV23 BV03
module	36	TASKD09	3	57	DV09	ok	G004 G005
module	32	TASKD16	3	67	DV16	ok	G006 G007 G008
module	31	TASKC11	1	14	G011	ok	DV01 DV02 DV03
module	30	TASKC09	1	98	G009	ok	DV14 DV19 DV20
module	29	TASKD10	3	42	DV10	ok	G004 G005
module	28	TASKB02	2	17	BV02	ok	DV21 DV22
module	27	TASKC08	1	53	G008	ok	DV12 DV13 DV16 DV17 DV18
module	24	TASKD13	3	15	DV13	ok	G016 G017
module	23	TASKC16	1	55	G016	ok	DV23 BV01
module	20	TASKD02	3	12	DV02	ok	G011 G012 G013
module	19	TASKD11	3	22	DV11	ok	G004 G005
module	15	TASKC03	1	24	G003	ok	DV01 DV02 DV03 DV04 DV05
module	14	TASKC06	1	23	G006	ok	DV12 DV13 DV16 DV17 DV18
module	13	TASKC17	1	35	G017	ok	DV23 BV01
module	10	TASKC15	1	62	G015	ok	DV06 DV07 DV08 DV23
module	9	TASKC07	1	18	G007	ok	DV12 DV13 DV16 DV17 DV18
module	8	TASKC14	1	18	G014	ok	DV06 DV07 DV08 DV23
module	7	TASKB03	2	53	BV03	ok	DV01 DV02 DV03
module	5	TASKD20	3	17	DV20	ok	G009 G010
module	4	TASKD23	3	81	DV23	ok	OB01
module	2	TASKD07	3	12	DV07	ok	G014 G015
module	26	TASKD14	3	74	DV14	ok	G016 G017
module	16	TASKC13	1	17	G013	ok	DV23 BV03
module	12	TASKC04	1	15	G004	ok	DV06 DV10 DV11 BV04
module	1	TASKC10	1	27	G010	ok	DV15 DV19 DV20
module	11	TASKD21	3	12	DV21	ok	G001 G002
module	3	TASKD17	3	12	DV17	ok	G006 G007 G008
module	22	TASKC05	1	12	G005	ok	DV07 DV08 DV10 DV11 BV04
module	18	TASKB01	2	62	BV01	ok	DV12 DV13 DV14 DV15
module	39	TASKD22	3	57	DV22	ok	G001 G002
module	40	TASKD18	3	84	DV18	ok	G006 G007 G008
module	41	TASKD01	3	40	DV01	ok	G011 G012 G013
module	42	TASKD08	3	93	DV08	ok	G014 G015
module	43	TASKF01	4	44	OB01	ok	DV23
module*	44	TASKC02	1	40	G002	ok	DV18 BV02

*Indicates modules added during planning function.

Table IV. Circuit and Level Times for Parallel Execution

Circuit	Level	Time
1	1	*125
2	2	201
3	3	121
4	2	253
5	3	218
6	2	*289
7	3	*257
8	4	*161
9	3	216

*Indicates the maximum time for each level.

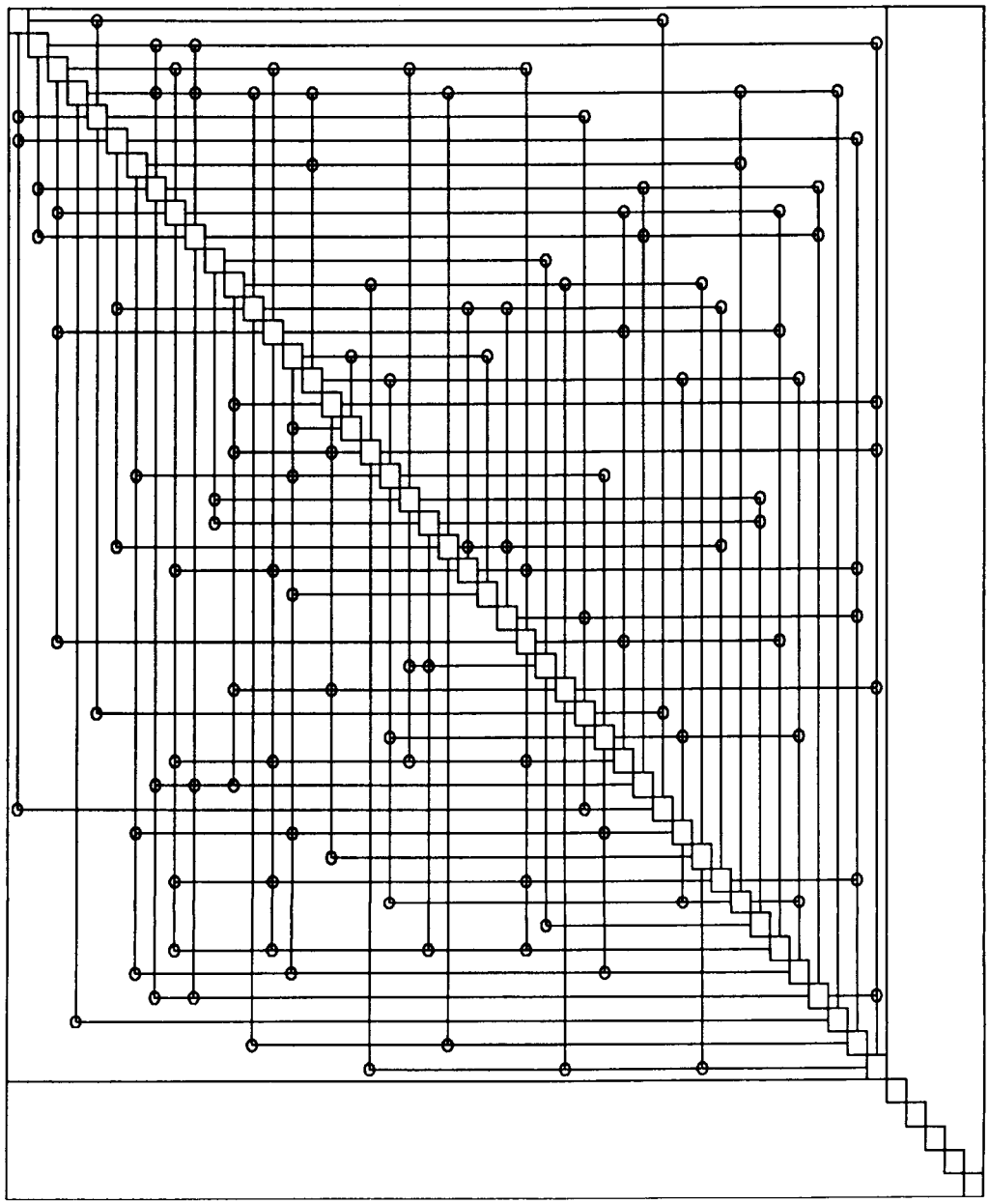

Figure 1. Unorganized data from original input.

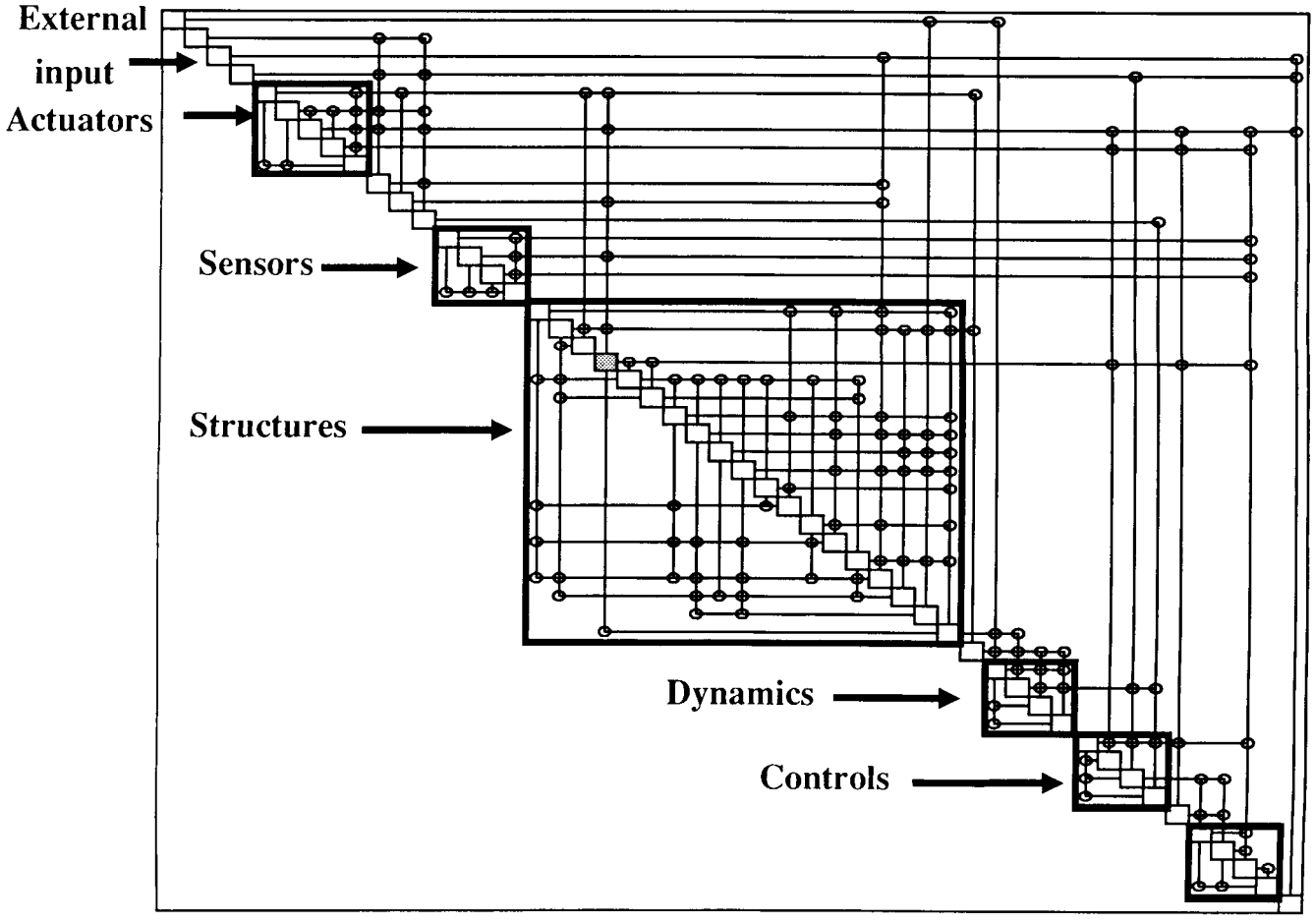

Figure 2. Desired structure of circuits and links. (Padula problem, ref. 8.)

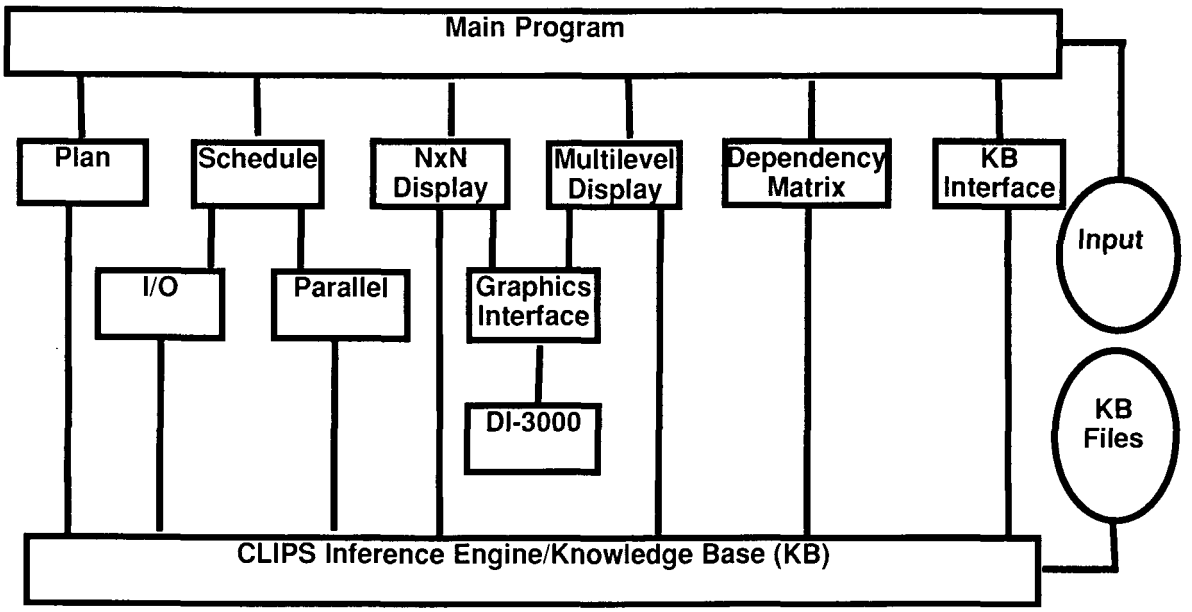

Figure 3. Diagram of the design tool.

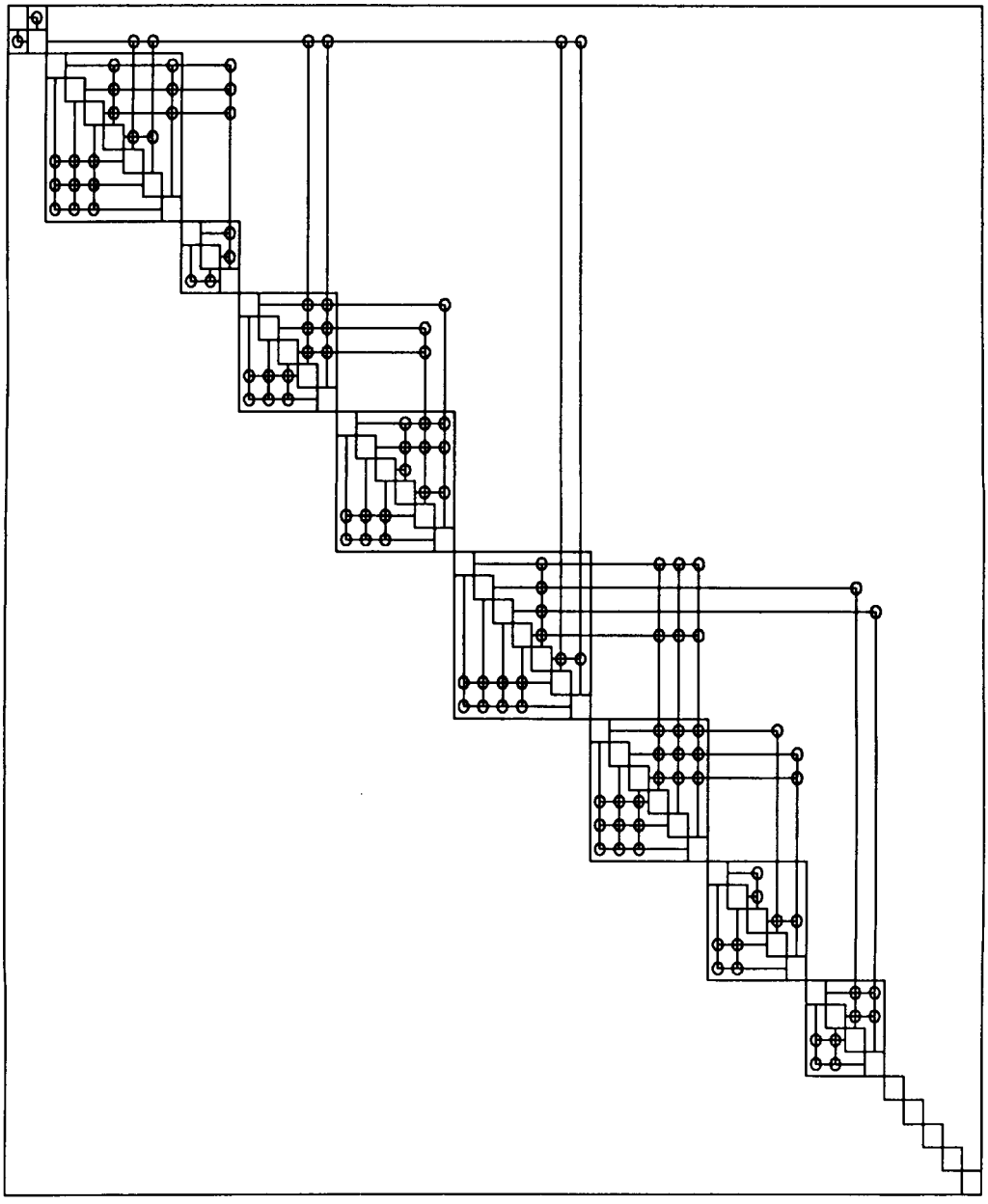

Figure 4. Modules and circuits after scheduling.

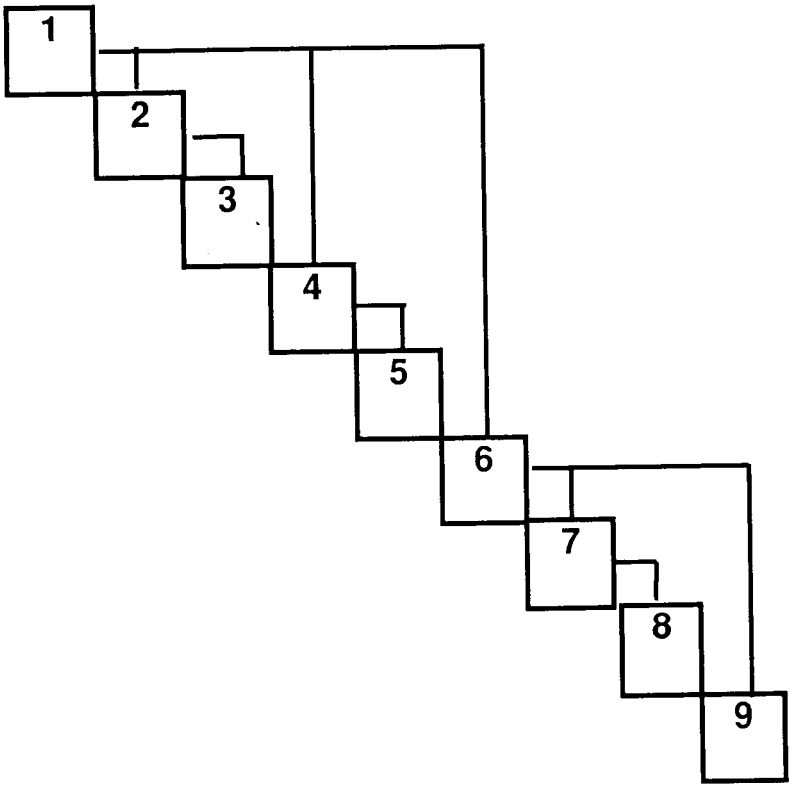

Figure 5. $N \times N$ display of circuits.

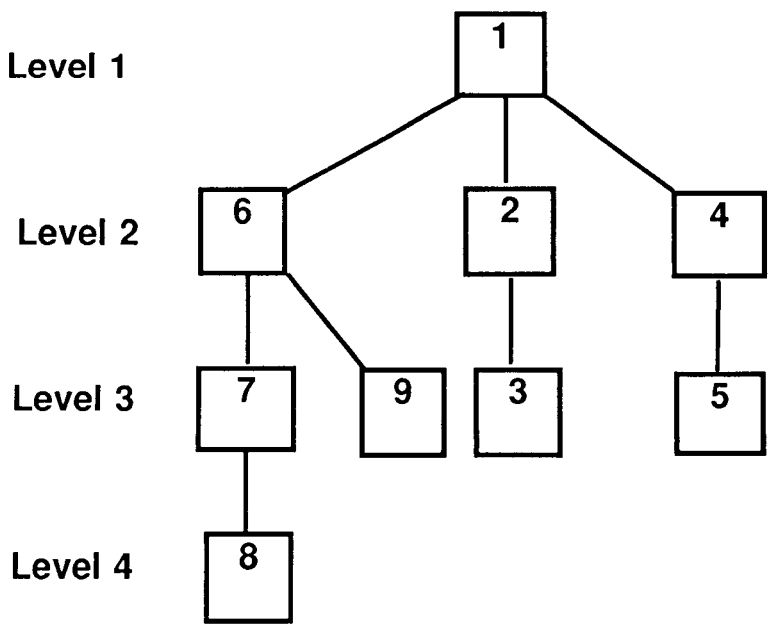

Figure 6. Multilevel display of circuits.

Module	2	3	4	5	10	11	13	14	15	18	19	20	24	25	26	27	31	32	33	37	38	42	43
7	X	X	X	X																			
8	X	X	X	X																			
9		X	X	X																			
12		X	X	X	X	X																	
16	X						X	X	X														
17	X						X	X	X														
22								X	X	X	X	X											
23							X			X	X	X											
29	X												X	X	X	X							
30	X												X	X	X	X							
34													X			X	X	X	X				
35													X			X	X	X	X				
36													X			X	X	X	X				
40																	X			X	X		
41																		X	X	X	X		
44														X								X	X
45															X							X	X

Figure 7. Dependency matrix.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TP-2903		

4. Title and Subtitle	5. Report Date
A Knowledge-Based Tool for Multilevel Decomposition of a Complex Design Problem	May 1989
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
James L. Rogers	L-16557
9. Performing Organization Name and Address	10. Work Unit No.
NASA Langley Research Center Hampton, VA 23665-5225	506-43-41-01
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, DC 20546-0001	Technical Paper
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Many engineering systems are large and multidisciplinary. For designs based on novel concepts, like large space platforms, the determination of the subsystems, interactions, and participating disciplines is an important task. Determining the subsystems is not an easy, straightforward process and often important interactions are overlooked. The design manager must know how to divide the design work among the design teams so that changes in one subsystem will have predictable effects on other subsystems. The resulting subsystems must be ordered into a hierarchical structure before the planning documents and milestones of the design project are set. Very few tools are available to aid the design manager in determining the hierarchical structure of a design problem and assist in making these decisions. Although much work has been done in applying artificial intelligence (AI) tools and techniques to problems in different engineering disciplines, only recently has the application of these tools begun to spread to the decomposition of complex design problems. A new tool based on AI techniques has been developed to implement a decomposition scheme suitable for multilevel optimization and to display the data in an $N \times N$ matrix format.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Knowledge-based system Multilevel decomposition Planning and scheduling Design process	Unclassified—Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	21	A03

NASA-Langley, 1989

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171