# NASA Technical Memorandum 83191



# A PROGRAMING SYSTEM FOR RESEARCH AND APPLICATIONS IN STRUCTURAL OPTIMIZATION

JAROSLAW SOBIESZCZANSKI-SOBIESKI AND
JAMES L. ROGERS, JR.

AUGUST 1981



NASA
National Aeronautics and
Space Administration

**Langley Research Center**
Hampton, Virginia 23665

# A PROGRAMING SYSTEM FOR RESEARCH AND APPLICATIONS IN STRUCTURAL OPTIMIZATION

Jaroslaw Sobieszczanski-Sobieski* and James L. Rogers, Jr.**
NASA Langley Research Center
Hampton, VA 23665

## Abstract

The paper describes a computer programing system designed to be used for methodology research as well as applications in structural optimization. The flexibility necessary for such diverse utilizations is achieved by combining, in a modular manner, a state-of-the-art optimization program, a production level structural analysis program, and user supplied and problem dependent interface programs. Standard utility capabilities existing in modern computer operating systems are used to integrate these programs. This approach results in flexibility of the optimization procedure organization and versatility in the formulation of constraints and design variables. Features shown in numerical examples include: (1) variability of structural layout and overall shape geometry, (2) static strength and stiffness constraints, (3) local buckling failure, and (4) vibration constraints. The paper concludes with a review of the further development trends of this programing system.

## List of Symbols

| | |
|---|---|
| F | - objective function |
| f | - function in general sense, denotes F or g |
| g | - constraint function |
| K | - stiffness matrix |
| L | - load |
| m | - number of constraint functions |
| n | - number of design variables |
| P | - load factor used as a variable |
| $P_t$ | - magnitude of the target level for P |
| t | - thickness |
| u | - vector of displacements |
| v | - velocity |
| $\vec{x}$ | - vector of design variables |
| $x_i$ | - a design variable |
| $z_i$ | - vertical displacements in a box beam |
| $\Omega$ | - cumulative constraint |

Subscripts:

| | |
|---|---|
| i | - design variable |
| j | - constraint function |
| o | - original value in extrapolation |

Acronyms:

| | |
|---|---|
| A-O Processor | - program converting the analysis program output to the optimization program input |
| CONMIN | - a particular optimization program used in PROSSS |
| O-A Processor | - program converting the optimization program output to the analysis program input |
| PROSSS | - Programing System for Structural Synthesis |
| SPAR | - a particular analysis program used in PROSSS |

* Head, Multidisciplinary Analysis and Optimization Branch, LAD
** Computer Scientist

## Introduction

The purpose of this paper is to describe a structural optimization program, called a Programing System for Structural Synthesis (PROSSS), which uniquely combines the almost unlimited flexibility required of a research tool for method development, with the reliability and simplicity of use expected from an application tool.

To provide a rationale for the implementation approach presented, the paper begins with a review of the requirements posed by the intended uses of the program in both research and applications. The implementation options are examined next, leading to a programing system alternative as a logical choice.

The principal components of the system, the way they are integrated, and the execution options are examined; and numerical examples are provided to illustrate the salient features pertinent to research and application. Included in the examples is a description of a version of the system operating in a distributed manner on a mainframe and a minicomputer.

The paper concludes with a brief review of the development trends stemming from the system capabilities and the current directions of the state-of-the-art evolution.

## Structural Optimization Application and Research Requirements

In general, the function of the optimization procedure is to find a vector of design variables x that minimizes an objective function $F(\vec{x})$ while satisfying constraint equations $g(\vec{x})$. In a standard notation:

$$F(\vec{x}) \rightarrow \min \qquad (1)$$

subject to

$$g_j(\vec{x}) \leq 0 \quad 1 \leq j \leq m \qquad (2)$$

In various applications, the variables in eq. 1 and eq. 2 acquire different meanings, and solution of the equations with acceptable efficiency and accuracy may require a fairly elaborate numerical process. Therefore, building a computer program to support both the development and application of structural optimization methods poses a unique challenge of making the software flexible and adaptable, yet reliable and easy to use.

### Diversity of Applications

The need for flexibility and adaptability stems, in part, from the need to be able to use the program to optimize structures of various types, e.g., an aircraft fuselage, a large space truss, or a nuclear reactor vessel. A single program general enough to answer all analysis needs for all types of

structures of interest, and efficient enough to perform well in an optimization loop at a reasonable cost, is not available. Consequently, an open-ended library of analysis programs has to be used.

In addition to the variety of types of structures, a variety of optimization problem formulations that differ by unique definitions of the design variables, constraints, and objective function will be of interest for each type of structure. Therefore, these portions of the code in which these formulations are embedded should be easy to replace.

## Diversity of the Optimization Techniques and Procedures

A distinction between an optimization technique and an optimization procedure will be useful in this discussion. An optimization technique is a search algorithm whose function is to find a constrained minimum in a space defined by the design variables in which the objective function and constraint functions are computed by another algorithm--an analysis algorithm. A few examples of optimization techniques are: (1) a nonlinear mathematical programing using an interior or exterior penalty functions (Sequential Unconstrained Minimization Technique, SUMT, ref. 1), (2) a usable-feasible direction algorithm, ref. 2, (3) a linear programing algorithm (e.g., ref. 3), and (4) optimality criteria methods, such as the fully stressed design method. An optimization procedure is an entity of higher order and as such may command execution of several optimization techniques, analysis algorithms, and auxiliary housekeeping and user-interface algorithms. Two examples of optimization procedures are: (1) a simple arrangement in which the executions of a search algorithm (an optimization technique) and an analysis alternate until the search algorithms localize a constrained minimum, and (2) a more complex arrangement in which a linear programing algorithm is combined with the exact and approximate analyses to solve a nonlinear optimization problem as a series of linearized subproblems.

By its very nature, the optimization methodology development requires use of many existing techniques and procedures and a continual creation of new ones, hence, only a program of practically unrestricted flexibility in its organization will qualify as a test bed for such development. If the same test bed program is also to be used for solving quickly and efficiently the application problems as a routine support of ongoing design projects, then there is a need to reconcile the seemingly contradictory requirements of "researchy" flexibility on one hand, and the reliability and relative constancy expected from a production tool, on the other hand.

## Hardware Adaptability

An additional consideration in development of an optimization program for research and applications is a need to capitalize on the opportunities periodically created by improvements in computer hardware. One such recent opportunity is distributed computing which, potentially at least, should improve computational efficiency by judiciously exploiting special features of the dissimilar computers in a network, and performing calculations in parallel whenever possible. It will be shown later that software flexible enough to meet the requirements pointed out in the previous two sections can also be adapted rather easily to distributed computing.

## Considerations Leading to Programing System Approach

Regardless of the manner of implementation, the basic function of an optimization procedure is to carry out an iteration shown by a flowchart in figure 1 in which a search algorithm and an analysis algorithm are labeled Optimizer and Analyzer. The procedure can be implemented on the computer in a number of ways illustrated in figure 2 and reviewed in this section in order of increasing application flexibility.

The discussion begins with a "closed box" approach which is the least flexible, but potentially the most efficient in execution, the simplest to use, and progresses to the concept of a programing system of potentially complete generality. Between these two extremes, the concept of concentrating the search and analysis functions in separate subroutines is discussed.

## Special Purpose "Closed Box"

By definition, the inner workings of a "closed box" program (fig. 2(a)) are set up for a predetermined scope of applications but not for easy access and modification by the users. This leaves the users with the input data as the only means for controlling the program functions within a range of options prescribed by the program developers. This adaptability limitation is an inevitable consequence of the "closed box" approach.

On the other hand, developers of a "closed box" program, being free of the user access and modification considerations, can gear the program organization for maximum computational efficiency, frequently by means of dispersing and intertwining the search and analysis functions with each other. Thus, in addition to being efficient, the program is practically impossible to "tinker" with and, therefore, maintains a permanent configuration and a repeatability of results. These are important features expected from a production tool in an engineering organization that might be concerned with a large number of applications of a limited variety and driven by stringent project deadlines.

## Search and Analysis Algorithms as Subroutines

Under this approach, shown in figure 2(b), the user has available the two basic elements of an optimization procedure in the form of separate subroutines. It is up to the user to assemble these subroutines in a functional program and to include various convenience features such as stop and restart, intermediate result displays, and special termination criteria.

This implementation approach has the advantage of modularity, so that everything that depends on the physics of the program can be isolated in the analysis subroutine while the best available search algorithm can be selected and coded in the search subroutine. However, the obviously attractive option of using preexisting codes for these subroutines is restricted by the practical limitations the main program-subroutine organization imposes on the subroutine size.

## Programing System

In comparison to the main program-subroutines arrangement, the concept of a programing system shown in figure 2(c) is the next logical step toward greater application flexibility. In a programing system (term

introduced in ref. 4), a user must furnish problem dependent code modules in addition to input data, in contrast to an ordinary program or system of programs which need only the input data to execute. A programing system allows the use of large stand-alone programs, or even systems of several large programs for the analysis and optimization functions, and isolates the definitions of the design variables, objective function, and constraints in separate problem dependent, user supplied programs executed between the Optimizer and the Analyzer. The system is controlled by an executive command language and other software utilities that constitute a connecting network, thus, in principle, any optimization procedure can be implemented, including optimization problems that in their analysis require many engineering disciplines in addition to structures. An example is a system for optimization of airframes including aerodynamic loads and aeroelastic effects, described in references 5 and 6. The concept of a programing system for optimization in general is a multifaceted subject for which literature references exist (e.g., refs. 4 and 7), therefore, the focus of the discussion which follows is on a particular programing system specialized for structural optimization.

## Features of the PROSSS System

The particular system to which the attention is now turning is called PROSSS for Programing System for Structural Synthesis. Its principal components are, in general terms: optimizer, analyzer, processors interfacing optimizer to analyzer (O-A) and vice versa (A-O), and the connecting framework. Executions of these components can be sequenced in various ways as required by a particular optimization procedure. The components and the procedure execution options are reviewed in this section.

### Analyzer

The function of the analyzer is to compute values of the behavior variables which characterize the physical object's response to the input quantities.

Overall characteristics. In PROSSS, the analyzer is the finite-element program SPAR documented in reference 8. SPAR was selected for the analyzer's function because of its computer efficiency, modularity, and data base capability. Input quantities consist of structural cross-section dimensions, material properties, element connectivity data, nodal point coordinates, and loads. Output quantities consist of displacements, internal forces, stresses, eigenvalues, and eigenmodes for vibration and buckling, etc. Another output quantity is the structural mass which is commonly used as the objective function. The library of finite elements in SPAR is adequate for analysis of skeletal and thin-walled structures.

SPAR is a collection of individual programs (processors that communicate with each other through a data base as indicated in fig. 3). The data base consists of one or more files which contain data sets output from the different processors. Each data set has a specific identifying name with which any processor can access it for input. Subroutines documented in reference 9 are available to store and retrieve the SPAR data sets by name from the SPAR data base. These subroutines can be executed by FORTRAN CALL statements and hence can be used to make the SPAR data storage accessible to non-SPAR FORTRAN programs.

SPAR executes on a processor-by-processor basis; each processor execution is commanded by a separate explicit command. A string of such commands interlaced with the input numerical data is written by the user for the problem at hand, and will be called a runstream in this paper. The data base facilitates an efficient and selective data transfer from SPAR to other programs, and the individual processor control allows the user to limit the number of processors executed repetitively in an optimization loop. For this purpose, the analyzer is divided into a nonrepeatable part executed once at the beginning of the procedure and a repeatable part executed many times in the optimization loop. Specifically, for invariant overall geometry, the nonrepeatable part generates nodal coordinates, material properties, constraint data and defines the loads. The repeatable part generates solutions of the load-deflection equations.

Computation of gradients. Most of the efficient mathematical optimization algorithms require not only the objective function and the constraint values but also their gradients to be evaluated for a given set of input values of the design variables $x_i$. The gradients can be computed by a finite-difference technique or by an analytical technique. An example of an analytical gradient is the differentiation of the matrix load-deflection equation, $Ku = L$, with respect to a design variable $x_i$. The result is a matrix equation

$$K \frac{\partial u}{\partial x_i} = \frac{\partial K}{\partial x_i} u + \frac{\partial L}{\partial x_i} \qquad (3)$$

from which $\partial u/\partial x_i$ can be obtained at a relatively small computational cost by reusing the previously decomposed stiffness matrix K, as shown in references 10 and 11. In SPAR, the analytical gradient computation is implemented by means of runstreams which are established specifically for this purpose and are a permanent part of PROSSS.

### Optimizer

The function of the optmizer is to calculate a new vector of design variables $\vec{x}$ on the basis of the values of the objective function and the constraints, and, optionally, their gradients returned by the analyzer in response to a previously defined vector $\vec{x}$.

In PROSSS, the optimizer is the program CONMIN (ref. 12), which is based on the mathematical nonlinear programing technique of usable-feasible directions. In this report, CONMIN is viewed as a "black box" and attention is focused on the type of data it requires from the rest of the system, and on its execution options, since these features influence organization of the programing system.

The following execution modes are available in CONMIN:

(a) Execution that requires current values of the objective function and constraints.

(b) Execution that requires current values of the objective function, constraints and their gradients.

(c) Execution accelerated because of the linearity of either the objective function and/or the constraints.

In PROSSS, program CONMIN is embedded as a subroutine in a program which calls it, and saves intermediate data for restart.

## Interface Processors

The optimizer provides input information to the repeatable part of the analyzer through an Optimizer-to-Analyzer (O-A) Processor and the analyzer supplies the information to the optimizer through an Analyzer-to-Optimizer (A-O) Processor. The O-A and A-O processors are user supplied and problem dependent. Capability of adding these two programs is the basis for the system's generality.

Optimizer-to-analyzer processor. The function of the O-A Processor is to convert the design variables to a set of input parameters written in a format required by the analyzer. In the case of structural optimization, these parameters are structural member sizes and nodal point coordinate data which are actual physical quantities and are seldom directly equivalent, one-to-one, to the design variables output by the optimizer. Thus, in a typical application, the conversions within the O-A Processor are not limited to formal changes only but also include such commonly used techniques as variable linking, scaling, and changing from direct to reciprocal variables (e.g., ref. 13). In PROSSS, the O-A Processor reads an output vector $\vec{x}$ from CONMIN, computes the structural parameters, and embeds them in a runstream written for the SPAR execution.

Analyzer-to-optimizer processor. The function of the A-O Processor is to compute the objective function, the constraints, and their gradients (if required) and to provide them in the format required by the optimizer. To do so, the A-O Processor extracts the pertinent behavior variables such as stresses, displacements, natural vibration frequencies, mode vectors, or buckling loads from the SPAR data base and combines them with the allowable values to form the constraint equations. Frequently, the allowable values are functions of $\vec{x}$ (instead of being constants) as, for example, in the case of local buckling constraints (e.g., ref. 14). Computation of such variable allowable values can be included among the functions of the A-O Processor. This processor may also be equipped with a logic to limit the set of constraints to those whose probability of remaining or becoming active is high, as proposed in reference 13. Organization of the A-O Processor, illustrated by the flow chart in figure 4, is problem independent, but the processor contains a section of code (box 4) which does depend on the problem at hand and must be tailored to it. In addition, the parameters in the call statements to the subroutines (box 3) (see ref. 9) that access the SPAR data base depend on the kind of data sets that need to be extracted as required by the particular constraints and objective function.

## Connecting Network

A connecting network (executive software) is required to carry out a computational process such as shown in figure 1. It is also required to enable the user to monitor progress of the optimization process, and to stop and restart without loss of information generated before the interruption.

In PROSSS, the CDC-NOS (Network Operating System)[1] documented in reference 15 serves as the connecting network using the approach described in reference 5. The CDC-NOS furnishes the user with a repertory of commands (job control language, (JCL)) for executing programs in sequences, including if-test branching and transferring to a labeled statement, and for manipulation of permanent and temporary files. These capabilities are common in most current operating systems, consequently such systems as IBM's MVS or UNIVAC's Exec 8 could function as a connecting network instead of CDC-NOS.

## Execution Flow Options

A variety of execution flow options can be set up using the components described previously. Organization of each flow option depends on how the optimizer is used, and on whether gradients are required as input to the optimizer and, if so, whether these gradients are generated analytically or by finite differences. The flow options currently available in PROSSS are the five shown in Table I.

Basic flow options. The two optimization procedures in Table I are: nonlinear mathematical programing (NLP) and piecewise linear approximations (PLA). Under the conventional NLP approach, the objective function and constraints are treated as nonlinear functions of the design variables. In the PLA procedure, which has been successfully used in a number of applications (e.g., refs. 13, 16, 17) the nonlinear optimization progresses as a sequence of linear optimization subproblems (stages). A linear approximation based on the Taylor series expansion, $f = f_o + \nabla f_o^T \Delta\vec{x}$, is used to compute the objective function and the constraint functions within each subproblem (stage). Side constraints on $\vec{x}$ control the linearization error.

Efficiency of PLA stems from replacing the full analysis of the physical problem with approximate analysis by linear extrapolation, which in structural applications requires a computer time of at least an order of magnitude smaller than the full analysis time. Additional time savings result at each stage because the optimizer executes faster when the problem is defined as linear (mode c in section "Optimizer"). The number of consecutive linear stages required for overall convergence depends on the degree of the problem nonlinearity.

The analysis capabilities with respect to calculation of gradients in Table I are: (1) computation of the behavior variables without gradients; (2) inclusion of gradients computed by finite differences; and (3) inclusion of gradients computed analytically.

Each of the five resulting options shown in Table I requires its own organization of the procedure flow. The organizationally simplest and most complex execution options, 1.1 and 2.3, respectively, are illustrated by flow charts in figures 5 and 6. The other options are described in detail in reference 18 and documented in reference 19. The flow chart in figure 5 is self-explanatory. In figure 6, the boxes 2, 3, and 4 may be regarded as functions of a single main program which calls the optimizer represented by box 1.

Auxiliary Option for Fully Stressed Design (FSD). If strength constraints are present in the problem, then convergence of all the foregoing optimization procedures can be improved by using a limited number (e.g. 3 to 5) of FSD iterations to generate initial cross-sectional dimensions of the structural members. Allowable stresses used in the FSD procedure can include material allowables (e.g., yield stress) and local buckling stresses which are

[1] Use of commercial products and names of manufacturers in this report does not constitute an official endorsement of such products or manufacturers, either expressed or implied, by the National Aeronautics and Space Administration.

functions of the cross-sectional dimensions as described in reference 20.

Two Basic Forms of the System

PROSSS exists in two basic forms: Skeleton Form and Specialized Form. The Skeleton Form consists of the following:

(1) The problem independent component programs such as SPAR, CONMIN, and the programs controlling execution of the linear stage optimization (figure 6) and FSD procedure.

(2) The SPAR runstream files for analytical gradients.

(3) The procedure files.

(4) The sets of JCL statements for each option.

To be used in a specific application, the Skeleton Form has to be turned into a Specialized Form. Problem dependent Interface Processors and the input data, including the SPAR runstreams, must be created and stored as files. In addition, standard names in the JCL statement file corresponding to the option chosen must be replaced with names selected for the problem dependent files.

Once the Specialized Form has been set up for a particular application, it can be protected from unauthorized alterations by using "software locks" (passwords) on all its files except the input data files. Several such Specialized Forms can be created from the common Skeleton Form for a variety of applications as illustrated in figure 7. Each such "frozen" Specialized Form can be used as a "black box" for a given class of problems that differ only by their input data. It is important to realize that although a Specialized Form is intended for use as a "black box" whose user is concerned only with the input and output data, it never becomes a previously defined "closed box" because it is always modular and accessible for modification.

In industrial organizations, preparation of the Specialized Forms would fall naturally into the domain of the staff specialist, while their application would be the task of the production oriented engineers. In research applications, the system modularity permits its major components SPAR and CONMIN to be replaced with other equivalent programs, and execution flows different than those described previously can also be constructed. Thus, PROSSS can be used as a test bed for development of new optimization procedures as well as an application tool.

## Numerical Examples

Examples presented in this section illustrate PROSSS as an application tool and as a research test bed. The application examples have been selected from a larger sample (given in ref. 18) to show the variety of design variable formulations, types of constraints, and some of the execution options. The purpose of the research examples is to illustrate usefulness of PROSSS in trying out improvements in the ways of conducting the optimization, including the case of the system distributed between a mainframe and a minicomputer.

### Application Examples

Example 1: Stiffened cylindrical shell. Several variants of a circular cylindrical shell reinforced by frames and longerons were studied. Finite-element model of the computationally largest variant (referred to as variant 1) is shown in figure 8. This variant is built up of membrane panels to represent skin, and beam elements (axial, bending and torsional stiffnesses) simulating transverse frames and longerons. Each frame and longeron may be regarded as a lumped representation (ref. 14) of several real frames and longerons. One end of the shell is clamped around the circumference, the other end is loaded by concentrated loads simulating distributed forces equivalent to a transverse force and torque. This variant has a large cut-out and a floor, and represents a simplified model of a transport aircraft fuselage segment.

This structure was expected to constitute a demanding test case for the following two reasons. First, the model contains 798 degrees of freedom, so it is a computationally large problem as far as optimization by mathematical programing is concerned. Secondly, the overall bending state of stress in a shell of this type depends on the in-plane stiffness of the frames; therefore, the design variables that govern the member cross-sectional dimensions become strongly coupled (e.g., ref. 14) and the optimization process is more difficult to converge.

Variant 1 was optimized by PLA using finite-difference gradients (Option 2.2) and the 10 design variables shown in figure 8. As indicated in the figure, many structural parameters are linked to a single design variable, so that 10 design variables govern the cross-sectional dimensions of all 356 elements in the finite-element model. Variables $x_1$ through $x_6$ govern the cross-sectional areas of the beam elements which have a channel cross-section whose proportions remain constant as its area changes. Thus, the cross-sectional area becomes a single variable that governs all the beam stiffness parameters. Variables $x_7$ through $x_{10}$ govern the membrane panel thickness. The optimization constraints were on the beam element stresses and equivalent stresses (Huber-von Mises stress) in the panel elements. The iteration history of the design variables is shown in figure 9. Convergence is quite good considering the problem size and use of the piecewise linear approximations. As expected, the elements flanking the cutout have "grown" in the optimization process, as illustrated in figure 10.

To further demonstrate the adaptability of the procedure, a simplified variant 2 of the shell structure was formed by eliminating the floor and two end bays and substituting rods for longerons. Initially, the structure was optimized with stress constraints only. Subsequently, the resultant structure was optimized with an additional overall shell buckling constraint which required a 21 percent increase of the buckling load over and above the buckling load computed for the structure optimized with stress constraints only. Both optimizations were carried out by Option 1.1. A comparison of these two results showed that the structural mass increased by 9.6 percent because of the additional buckling constraint. Additional optimization of this variant (with two loading cases) was carried out with stress constraints using only analytical gradients (Option 2.3). Use of analytical gradients was found to reduce the execution time to approximately one-sixth of that required for Option 1.1. Three design variables, one for longerons, one for transverse-frames, and one for the skin were used in this case.

Locations of the node points in the finite-element model were considered as design variables in a further simplified variant 3 of the

shell structure. This variant has the cut-out eliminated, is subject to only one loading case (transverse force), and has the longerons restored to the beam form. Previously defined cross-sectional variables were retained. The three geometrical variables governed locations of the three intermediate frames. Optimization using Option 1.1 with stress and overall shell buckling constraints resulted in translation of the frames toward the loaded and unsupported end, as seen in figure 11.

Example 2: Portal framework shape optimization. A framework shown in figure 12 has been optimized with geometrical variables only to demonstrate the structural shape optimization. The variables defined in figure 12 are intended to allow the frame to transform into a truss. The constraints were imposed on stress and horizontal displacement as indicated in figure 12. Optimization, carried out by means of Option 1.1, has indeed produced the expected transformation of shape to an almost triangular truss. A side constraint on the length of the top horizontal member, necessary to preserve that member's nonzero length to avoid a matrix singularity, has kept the top of the frame from shrinking to a point.

Example 3: Torsion box vibration. This example demonstrates a "tuning" of the stiffness and mass distributions to achieve a prescribed change in an original structural vibration mode shapes. The structure is a torsion box shown in figure 13(a), built up of membrane panels, with thicknesses as the design variables indicated in the figure. Because concentrated nonstructural masses are affixed to one side of the box as shown in figure 13(a), the vibration modes for a structure that was initialized uniformly to a minimum gage exhibit a distinct torsion-bending coupling in the first four modes. This is illustrated in figure 13(b) by displacement of segment AB seen in view C. To reduce that coupling, the constraints of $(z_2 - z_1)/z_1 \leq 2$ are imposed on the free end vertical displacements $z_1$ (point A) and $z_2$ (point B) in modes 1 through 4. The result is a set of new modes shown in figure 13(b) which comply with the constraints. The thickness changes required to meet the constraints are indicated in figure 13(c). There are increases of thicknesses $t_3$ and $t_5$ in the SPAR beam directly supporting the concentrated masses, and $t_4$ in a panel that forms a counterbalance to the fixed masses.

## Research Examples

Cumulative constraint. A direct search method such as usable-feasible directions method in the current PROSSS optimizer has a drawback--namely a large computer memory is required to keep track of each individual constraint in application to structures with numerous stress constraints, such as the stiffened cylindrical shell in the previous group of examples. To overcome this drawback, a cumulative constraint (ref. 7) was tried. The constraint is, in essence, the same as the well-known exterior penalty function and is formulated as:

$$\Omega = \sum_j \left(\langle g_j \rangle\right)^2 \qquad (4)$$

where

$$\langle g_j \rangle = \begin{cases} g_j, & \text{if } g_j > 0 \\ 0., & \text{if } g_j \leq 0 \end{cases}$$

The cumulative constraint $\Omega$ is a single measure of many constraint violations and its zero boundary is interpreted for purposes of usable-feasible directions algorithm as a hypersurface which is continuous through the first derivatives, providing the $g_i$ functions are also continuous.

The stiffened cylindrical shell, variant 3 of example 1 above, optimized for minimum mass subject to individual strength constraints was taken as a reference case. The single cumulative constraint was introduced to replace 190 constraints by modifying the A-O processor. Relative to the reference case, the results indicated a slight (3 percent) reduction of the objective function, an increase of the total number of iterations from 6 to 8, and the optimizer memory requirement reduced by 99.7 percent.

Influence of the move limits. It is expected that the results of a piecewise linear optimization procedure such as Option 2.3 in PROSSS depend to some extent on the move limits allowed in each linear stage, but the extent of that dependence is not known. To shed some light on the dependence, the same stiffened cylindrical shell was optimized using PROSSS Option 2.3 by systematically changing the relative move limits. The result is shown in figure 14 as a plot of the objective function versus the relative move limit value imposed on all design variables and maintained constant from one linear stage to the next. The plot indicates that a wide interval of the move limit values exists where the optimal objective function is practically independent of these values, while the dependence is strong outside of the interval.

A leading variable technique. The same stiffened shell used in the two preceding examples was optimized for minimum mass subject to individual strength constraints using a somewhat unusual technique.

The two basic elements of the technique are: (1) adding the load magnitude P as another variable to the vector of design variables whose initial values were set large for the structural design variables but very small for the load variable, (2) restricting the load variable P by constraints, $g = 1 - P/P_t$ and $0 \leq P \leq P_t$, in order to make P grow to, and remain at, the desired level of fully-developed load $P_t$. Under this formulation, the load variable becomes a "leading" variable which grows to its target level "pulling" the entire design toward its final state.

The technique is of interest because of its implications for those cases where conventionally formulated optimization fails to find a feasible design. (The design feasibility per se was not an issue in the example case itself.) In such cases, it is usually easy to identify a physical quantity which is not a natural design variable but whose reduction in magnitude renders the initial design feasible. Such physical quantity may then be converted to a leading variable of a suitably low initial value, and therefore remove the difficulty of finding a feasible design.

A good example of this would be an optimization of a strength-sized wing structure for a required flutter speed and a minimum of a flutter structural mass penalty. In this case, the velocity v would be a candidate for a leading variable, analogous to P, and the required flutter speed would be analogous of $P_t$. A natural starting value for v would be the flutter velocity of the strength-sized structure.

Implementation of the technique required changes only to the O-A and A-O processors and produced a result illustrated in figure 15 by a plot of the objective function versus the consecutive iterations.

The final result is practically the same as in the reference case, and examination of the stress constraints as they were changing over the iterations illustrated in figure 16 for the constraints active at optimum, shows that they were never significantly violated. The only constraint that was ever strongly violated was the computationally trivial one imposed on the load variable (leading variable).

Distributing the system between a mainframe and a minicomputer. To test the system adaptability to different hardware configurations, a version of PROSSS was constructed placing the analyzer and the A-O processor on the CDC mainframe computer and the remainder of the system on the PRIME minicomputer as shown in figure 17. It was found that the modular organization of PROSSS was essential for expeditious development of the distributed version. The distributed version of PROSSS was verified (ref. 21) for correctness of its results as compared to the mainframe-only version and was used to explore systematically the relative efficiency of the five PROSSS options. Results of the efficiency results are plotted in figure 18 and show that Option 2.3, and PLA with analytical gradients as, by far, the most efficient one.

This distributed implementation, documented in detail in reference 21, has advantage of the optimal use of the best features of each type of computer--namely, the mainframe computer capability to perform a massive numerical analysis and the minicomputer flexibility and fast interactive response helping in the preparation of the problem, judgmental control of the execution, and review of the results. The main resulting benefit is improved productivity of the "man-machine system" manifested by a very significant reduction of the calendar time needed to complete optimization tasks. Various factors leading to that reduction are examined in reference 21.

### Summary of the Examples

Summarizing the application examples, the following observations are noted. Transforming the system from one optimization option to another was simple to accomplish by changing the sequence in which components of the system were called for execution. Adaption from one variable and constraint combination to another was carried out by changes in the O-A and A-O processor codes. These adaptations as well as changes from one structure to another did not require any changes to the Connecting Network nor to the Analyzer and Optimizer.

Similarly, the research examples demonstrated adaptability of a programing system to the algorithm procedural changes that reached deep into the problem formulation and yet required only minor and very localized modifications to the system modules.

It was a routine matter to monitor the status of the optimization process by means of displaying the intermediate data files. Stopping and restarting were facilitated by storing intermediate data.

This monitoring and interaction with the process was particularly easy and efficient in the distributed version owing to the quick response of the minicomputer in the interactive mode.

## PROSSS Development Trends

Because of its test bed nature, PROSSS undergoes continual development; some possible future changes are summarized in this section.

### Optimization Algorithms

Several improved optimization algorithms became available in recent years. Particularly, promising among these are: The augmented Lagrangian technique and the primal-dual methods (e.g., ref. 22). Programs based on those algorithms are logical candidates to convert the current single optimizer in PROSSS into a library of optimizers.

### Optimum Sensitivity

It was shown in reference 23, that information about sensitivity of the optimum solution with respect to problem parameters can be generated at a relatively minor cost. An example of such sensitivity information might be a set of derivatives of the optimum cross-sectional areas and the structural mass with respect to the allowable stress value. It was also demonstrated in reference 23, that accuracy of extrapolation based on such derivatives is quite good for a fairly wide interval of the parameters. The modular organization of PROSSS should make insertion of the sensitivity analysis and the associated extrapolation capability a relatively straightforward task.

### Multilevel Optimization

It is now widely recognized that a multilevel optimization scheme which breaks one large problem into a hierarchy of separately solved but coupled subproblems has a potential of making truly large structural optimization applications practical. One such scheme is proposed in reference 24. There is also a possibility to build a multilevel scheme on the basis of the optimum subproblem sensitivities to the parameters which themselves are the master problem design variables.

The fairly complex organization of the computational sequences and the associated data flow required by the multilevel schemes should be well supported by the flexible organization of PROSSS.

### Distributed Computing

Starting from the two-computer version of PROSSS referred to in the research example section, a more ambitious undertaking may be initiated of a multicomputer network in which advantage would be taken of distributed processing. This concept fits well in a multilevel optimization scheme, because many subproblem optimizations could be performed simultaneously on many computers acting in parallel.

### Shape Optimization

In comparison to the wealth of experience with cross-sectional optimization to structures, the experience with the optimization of the overall shape is very limited. The PROSSS capability to work with various types of design variables including those of overall geometry encourages exploration of the shape optimization. A development already initiated in this direction involved optimization of trusses using an analytical gradient technique in which the derivatives of the stiffness matrix with respect to the shape design variables were computed by means of finite difference.

### Connecting Network and High-Level Language

The operating system and its command language (JCL) are an ultimate in flexibility and continue to support the PROSSS development. Their drawbacks are vulnerability to the operating system changes and the overhead penalty of the system operations. One means

available now for improvement in this regard is an Engineering Analysis Language (EAL) (ref. 25). The EAL is a system of programs and a data base which is a successor to SPAR program and is operational on many different types of computers. It is enhanced by a command language that possesses a FORTRAN-like capability of loop, branch and jump. In addition to the standard structural analysis processors (the same ones as in SPAR), the EAL library of programs can be routinely augmented by user supplied codes. Thus, implementation of PROSSS in EAL may be done by adding to EAL the optimizer, the O-A and A-O processors for each application, and by translating the JCL procedures for the PROSSS options to the EAL command language (runstreams).

Still, the EAL is not the English-like language many users would like to have available to command a computer. Such language can, in principle at least, be provided so that an engineer could issue, for example, a command: "OPTIMIZE TRUSS FOR MINIMUM MASS AND STRENGTH CONSTRAINTS." For a command such as this to have the intended effect, a translator program standing between the user and the EAL would have to generate a corresponding sequence of the EAL instructions. However, the exact meaning of all the words used in the command would have to be coded first into the translator program, hence, the loss of flexibility.

A reasonable option appears to be to equip the specialized versions (see the previous discussion of the Skeleton and Specialized versions of PROSSS) with an English-like language for a production oriented user, while allowing a researcher to use the EAL command language directly.

## Conclusions

A computer programing system is described which combines an optimization program, a structural analysis program, and user supplied problem dependent interface programs, for use in the structural optimization method development and applications. Standard utility capabilities existing in modern computer operating systems are used to integrate these programs. This approach results in flexibility of the optimization procedure organization and versatility of the formulation of constraints and design variables. Features of the programing system are illustrated by numerical examples, which include design variables of cross-sectional dimensions and overall shape and constraints on static and dynamic behavior. Included in the examples is a version of the system distributed between a mainframe and a minicomputer.

Five options are described for organizing the optimization procedures. The options comprise various combinations of nonlinear mathematical programing and piecewise linear approximations with analytical and finite-difference gradient techniques. Because of the system's inherent modularity, other software components could be substituted for the particular ones used herein to achieve a similar capability.

The system can be used in the following two basic ways:

(1) As a research test bed for development of optimization techniques and analysis oriented towards optimization applications. In this role, the system offers flexibility of execution and sequencing including restart and monitoring capabilities.

(2) As an application tool that can be adapted by a specialist to a very wide scope in types of problems and then used as a "black box" by production oriented engineers.

The system development trends are reviewed in the areas of the optimization algorithms, including multilevel schemes, optimum sensitivity analysis, distributed computing, overall structural shape optimization, and use of an improved connecting network and higher level command languages.

## References


(1) Fiacco, A. V.; and McCormick, G. P.: Nonlinear Programing: Sequential Unconstrained Minimization Techniques. John Wiley and Sons, New York, 1968, Section 2.4.

(2) Zoutendijk, G.: Methods of Feasible Directions. Elsevier, Amsterdam, 1960.

(3) Garvin, W. W.: Introduction of Linear Programing. McGraw-Hill, New York, 1960.

(4) Schrem, E.: From Program Systems to Programing Systems for Finite-Element Analysis. Paper presented at U.S.-Germany Symposium: Formulations and Computational Methods in Finite-Element Analysis. MIT, Boston, MA, August 1976.

(5) Sobieszczanski, J.: Building a Computer Aided Design Capability Using a Standard Time Share Operation System. Proceedings of the ASME Winter Annual Meeting, Integrated Design and Analysis of Aerospace Structures, Houston, TX, November 30-December 5, 1975, pp. 93-112.

(6) Dovi, A. R.: ISSYS - An Integrated Synergistic Synthesis System. NASA Contractor Report 159221, Kentron International, Inc., Hampton Technical Center, Hampton, VA., February 1980.

(7) Sobieszczanski-Sobieski, Jaroslaw: From a "Black Box" to a Programing System: Remarks on Implementation and Application of Optimization Methods. Proceedings of a NATO Advanced Study Institute Session on Structural Optimization, University of Liege, Sart-Tilman, Belgium, August 4-15, 1980.

(8) Whetstone, W. D.: SPAR Structural Analysis System Reference Manual, System Level II, Volume I. NASA CR-145098-1, February 1977.

(9) Giles, G. L.; and Haftka, R. T.: SPAR Data Handling Utilities. NASA TM-78701, September 1978.

(10) Fox, Richard L.: Optimization Methods for Engineering Design. Addison-Wesley Publ. Co., Reading, Mass., 1971.

(11) Storaasli, O. O.; and Sobieszczanski, J.: On the Accuracy of the Taylor Approximation for Structure Resizing. AIAA J., Vol. 12, No. 2, February 1974, pp. 231-233.

(12) Vanderplaats, Garret N.: The Computer for Design and Optimization. Computing in Applied Mechanics, R. F. Hartung, ed., AMD - Vol. 18, American Soc. Mech. Eng., c. 1976, pp. 25-48.

(13) Schmit, L. A.; and Miura, H.: Approximation Concepts for Efficient Structural Synthesis. NASA CR-2552, March 1976.

(14) Sobieszczanski, Jaroslaw: Sizing of Complex Structures by the Integration of Several Different Optimal Design Algorithms. AGARD Lecture Series No. 70 on Structural Optimization, AGARD-LS-70, September 1974.

(15) Control Data Corporation; NOS Version 1 Reference Manual, NOS 1.3. CDC No. 60435400, September, 1979.

(16) Starnes, J. H.; and Haftka, R. T.: Preliminary Design of Composite Wings for Buckling, Strength and Displacement Constraints. A Collection of Technical Papers, AIAA/ASME 19th Structures, Structural Dynamics and Materials Conference, Bethesda, Md., April 3-5, 1978. AIAA Paper No. 78-466.

(17) Anderson, M. S.; and Stroud, W. J.: A General Panel Sizing Computer Code and Its Application to Composite Structural Panels. AIAA J., Vol. 17, No. 8, August 1979, pp 892, 897.

(18) Sobieszczanski-Sobieski, J.; and Bhat, R. B.: Adaptable Structural Synthesis Using Advanced Analysis and Optimization Coupled by a Computer Operating System. J. of Aircraft, Vol. 18, No. 2, February 1981, pp. 142-149.

(19) Rogers, J. L., Jr.; Sobieszczanski-Sobieski, J.; and Bhat, R. B.: An Implementation of the Programing Structural Synthesis System (PROSSS). NASA TM 83180, 1981.

(20) Giles, G. L.: Procedure for Automating Aircraft Wing Structural Design. J. of Str. Div. ASCE, Vol. 97, No. ST1, 1971, pp. 99-113.

(21) Rogers, J. L., Jr.; Dovi, A. R.; and Riley, K. M.: Distributing Structural Optimization Software Between a Mainframe and a Minicomputer. Proceedings of Engineering Software Second International Conference and Exhibition, London, England, March 24-26, 1981, pp. 400-415, editor, R. A. Adey, CML Publications.

(22) Fleury, Claude; and Schmit, Lucien A., Jr.: Dual Methods and Approximation Concepts in Structural Synthesis. NASA Contractor Report 3226, December 1980.

(23) Sobieszczanski-Sobieski, J.; Barthelemy, Jean-Francois; and Riley, Kathleen M.: Sensitivity of Optimum Solutions to Problem Parameters. Presented at AIAA/ASME/ASCE/AHS 22nd Structures, Structural Dynamics and Materials Conference, Atlanta, Ga., April 6-8, 1981. AIAA Paper No. 81-0548.

(24) Schmit, L. A.; and Ramanathan, R. K.: Multilevel Approach to Minimum Weight Design Including Buckling Constraints. AIAA Journal, Vol. 16, No. 2, February 1978, pp. 97-104.

(25) Whetstone, W. D.: EISI-EAL: Engineering Analysis Language. Presented at ASCE Conference on Computing and Civil Engineering, Baltimore, Md., June 1980.

Table I Optimization flow options

| Procedure | No Gradients Supplied to Optimizer | Gradients Supplied to Optimizer | |
|---|---|---|---|
| | | Finite Difference | Analytical |
| NLP | 1.1 | 1.2 | 1.3 |
| PLA | Not Applicable | 2.2 | 2.3 |

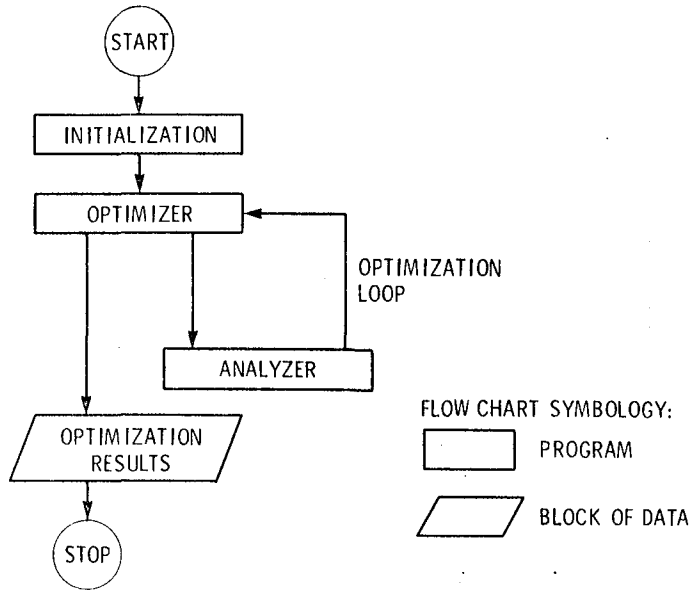



Fig. 1.- Generic components and a basic flow organization of an optimization procedure.

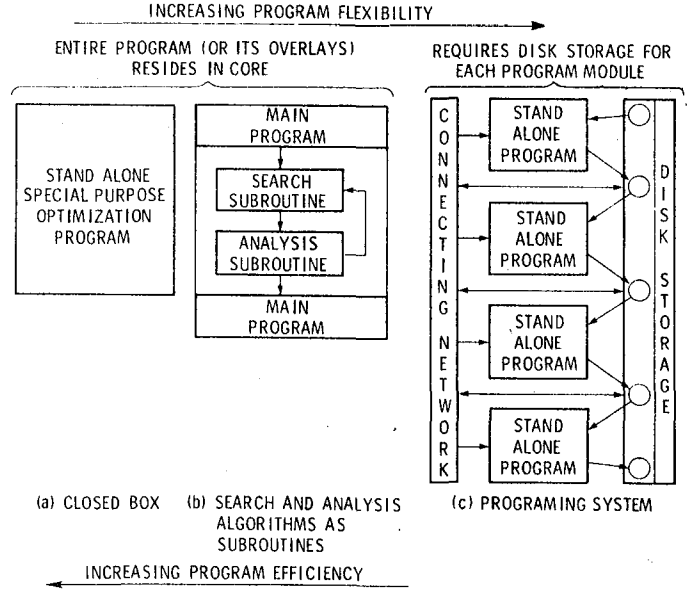



Fig. 2.- Approaches for implementing optimization methods.

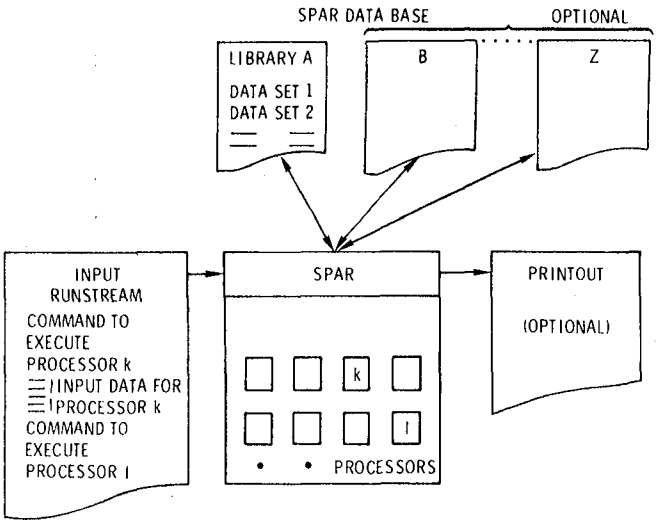



Fig. 3.- Finite-element program SPAR.

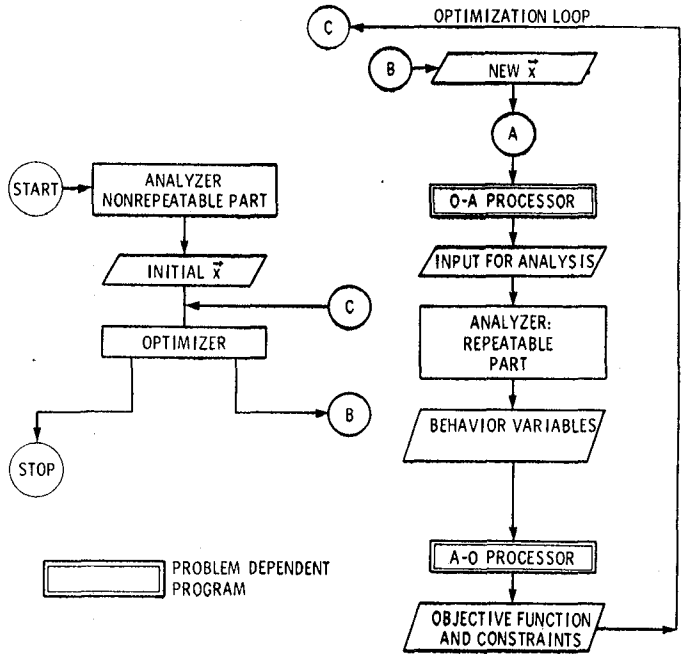



Fig. 5.- Flow chart for Option 1.1; the analyzer split into nonrepeatable and repeatable parts.

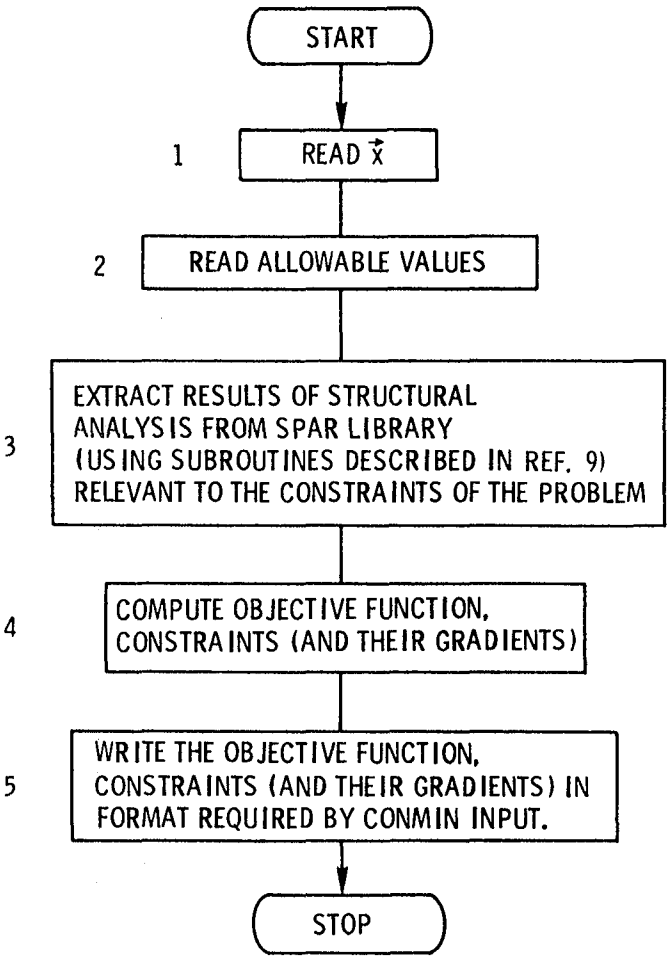



Fig. 4.- A-O Processor flow chart.

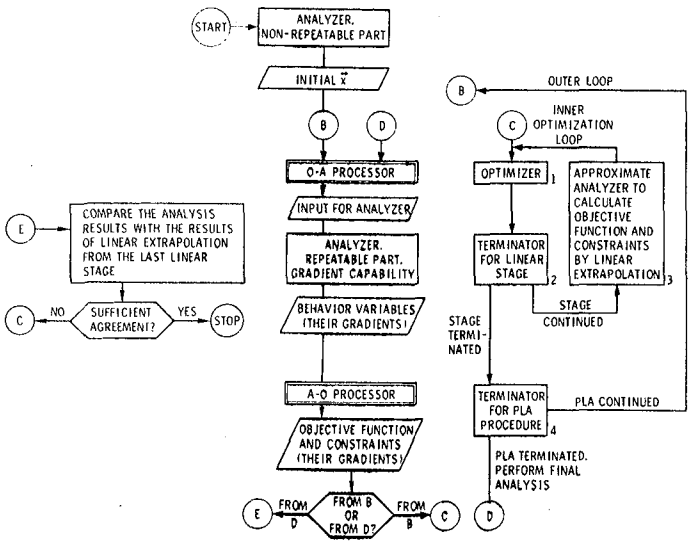



Fig. 6.- Flow chart for Option 2.3.

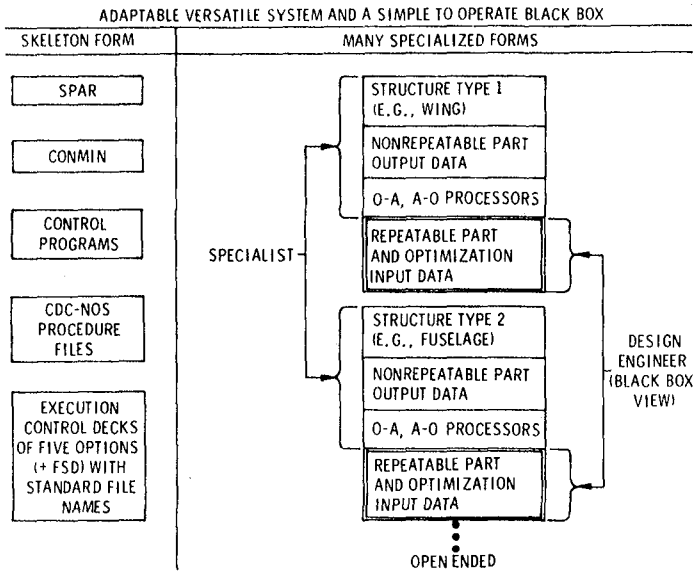



Fig. 7.- Skeleton and specialized forms of PROSSS.

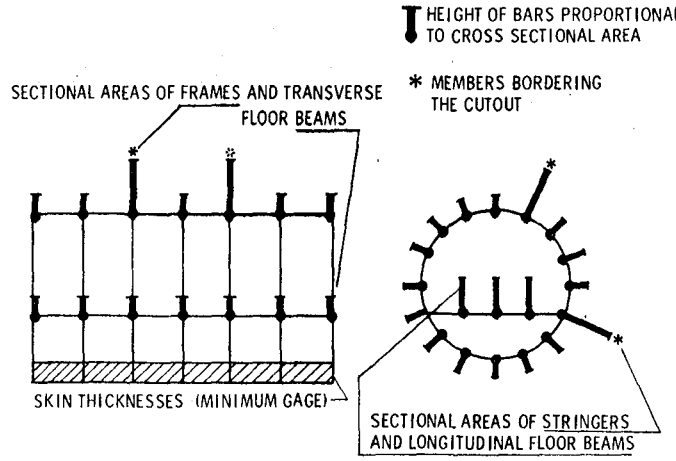



Fig. 10.- Relative member sizes obtained by optimization for Example 1, variant 1.

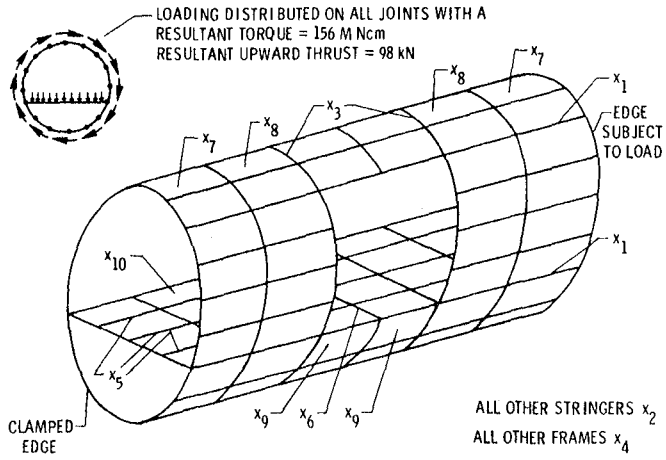



Fig. 8.- Example 1, stiffened cylindrical shell, variant 1.

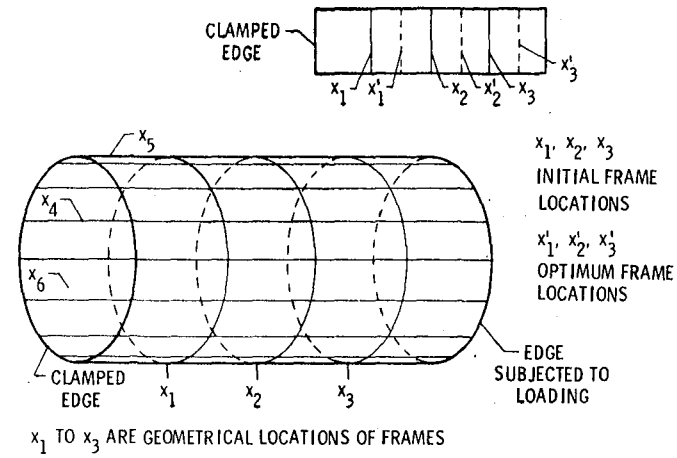



Fig. 11.- Initial and optimized positions of transverse frames in Example 1, variant 3.

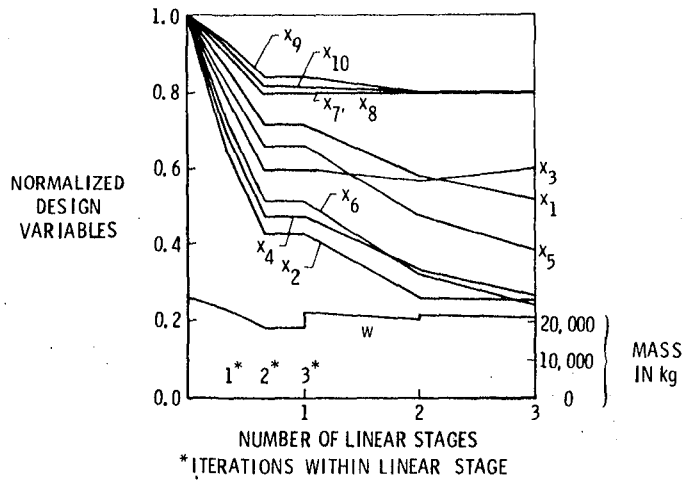



Fig. 9.- History of iterations for Example 1. Variant 1 using Option 2.2.

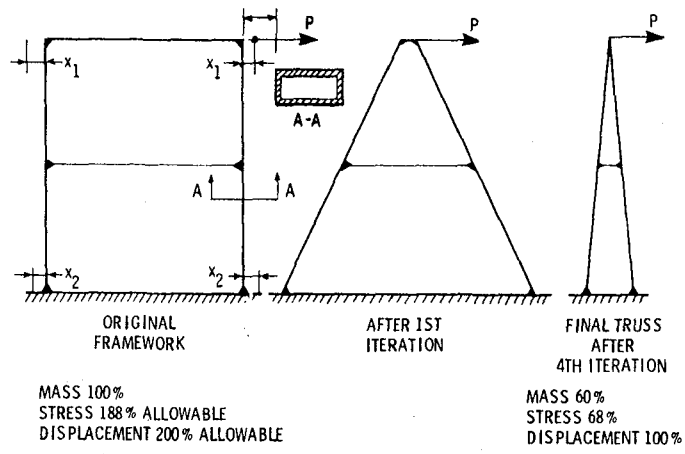
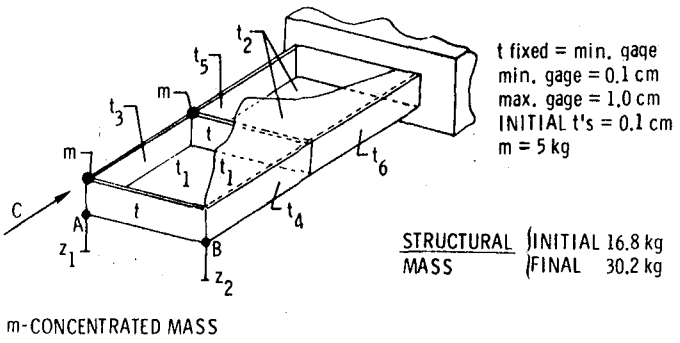
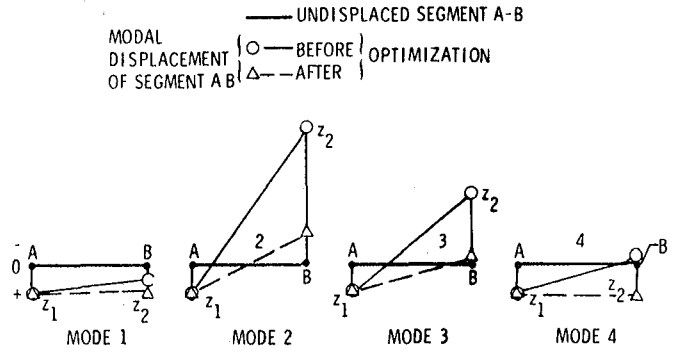


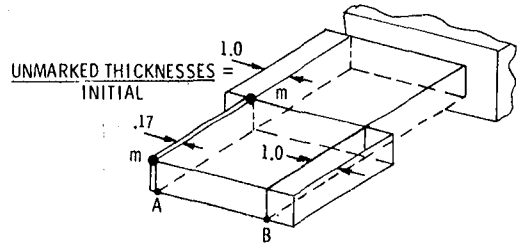

Fig. 12.- Transformation of a framework (Example 2) to a truss by optimization with geometrical variables.




(a) Thicknesses used as design variables.




(b) Modal displacements of segment AB (Fig. 13(a)), seen in view C.




(c) Dimensional arrows mark the spar beam web thicknesses increased in the optimization process; other thicknesses remained at the initial values.

Fig. 13.- Optimization of a torsion box (Example 3) to reduce torsion bending coupling in modes 1, 2, 3, and 4.

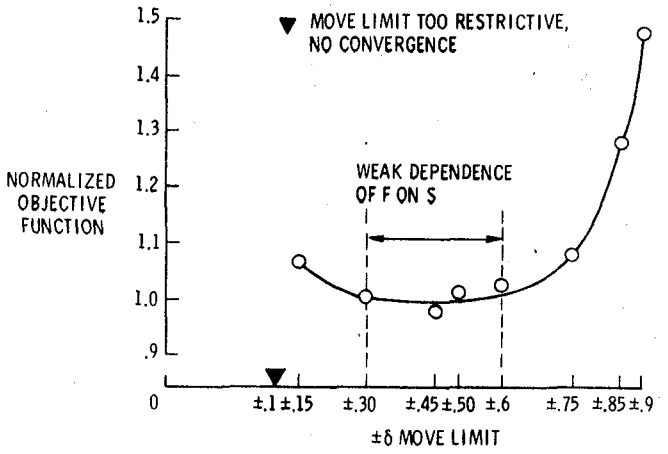



Fig. 14.- Normalized objective function (mass) vs. relative move limit.

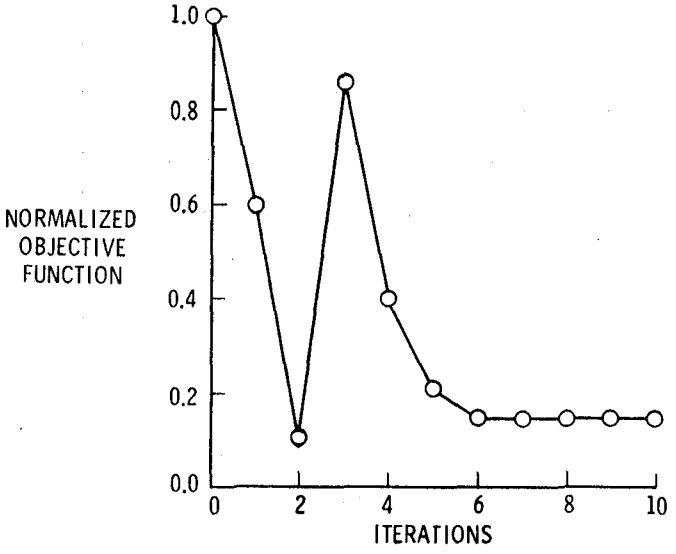



Fig. 15.- Objective function vs. the iteration number for a leading variable technique.

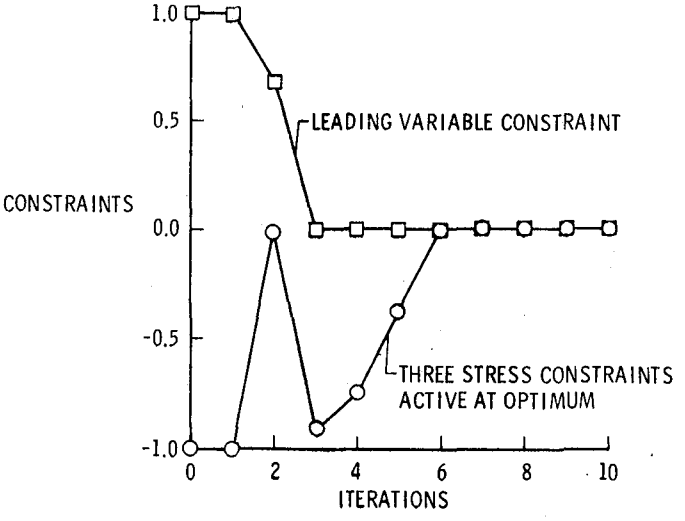



Fig. 16.- Constraints vs. the iteration number for a leading variable technique.

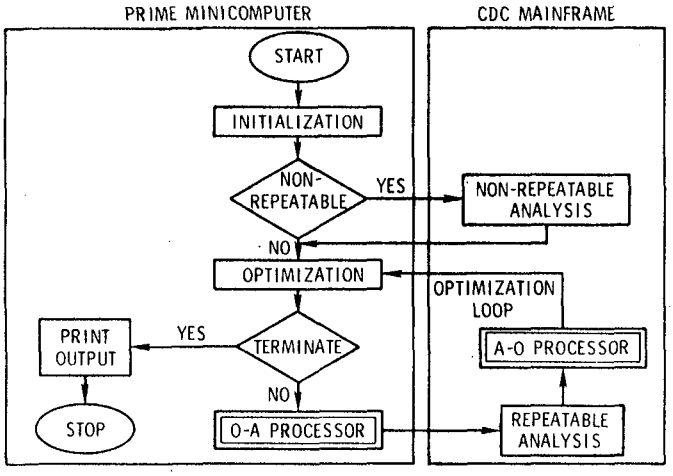



Fig. 17.- Flow chart of a distributed structural optimization software system.

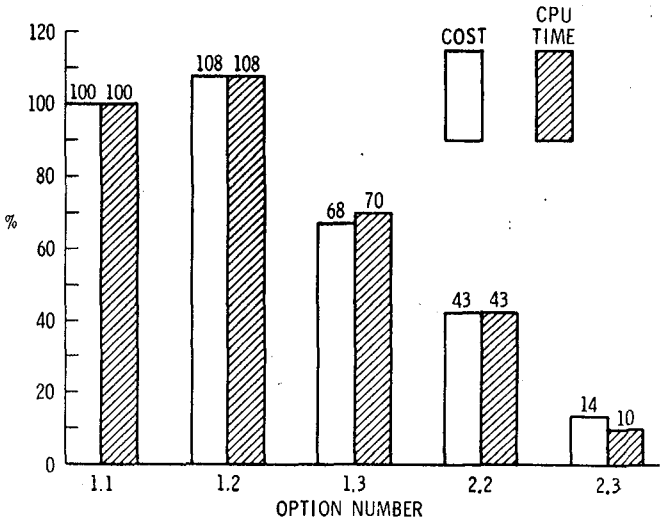



Fig. 18.- Option cost and time comparison.

| 1. Report No. | 2. Government Accession No. | 3. Recipient's Catalog No. |
|---|---|---|
| NASA TM-83191 | | |

| 4. Title and Subtitle | 5. Report Date |
|---|---|
| A Programing System for Research and Applications in Structural Optimization | August 1981 |
| | 6. Performing Organization Code 505-33-63-02 |
| 7. Author(s) Jaroslaw Sobieszczanski-Sobieski and James L. Rogers, Jr. | 8. Performing Organization Report No. |
| | 10. Work Unit No. |
| 9. Performing Organization Name and Address NASA Langley Research Center Hampton, Virginia 23665 | 11. Contract or Grant No. |
| | 13. Type of Report and Period Covered Technical Memorandum |
| 12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546 | 14. Sponsoring Agency Code |

15. Supplementary Notes

Presented at International Symposium on Optimum Structural Design, The 11th Naval Structural Mechanics Symposium, Tucson, Arizona, October 1981.

16. Abstract

The paper describes a computer programing system designed to be used for methodology research as well as applications in structural optimization. The flexibility necessary for such diverse utilizations is achieved by combining, in a modular manner, a state-of-the-art optimization program, a production level structural analysis program, and user supplied and problem dependent interface programs. Standard utility capabilities existing in modern computer operating systems are used to integrate these programs. This approach results in flexibility of the optimization procedure organization and versatility in the formulation of constraints and design variables. Features shown in numerical examples include: (1) variability of structural layout and overall shape geometry, (2) static strength and stiffness constraints, (3) local buckling failure, and (4) vibration constraints. The paper concludes with a review of the further development trends of this programing system.

| 17. Key Words (Suggested by Author(s)) | 18. Distribution Statement |
|---|---|
| Computer programing<br>Methodology research<br>Applications in structural optimization<br>Constraints and design variables | Unclassified - Unlimited<br><br>Subject Category 05 |

| 19. Security Classif. (of this report) | 20. Security Classif. (of this page) | 21. No. of Pages | 22. Price |
|---|---|---|---|
| Unclassified | Unclassified | 14 | A02 |



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