NASA TM-84472

NASA Technical Memorandum 84472

NASA-TM-84472 19820016970

COMBINING ANALYSIS WITH OPTIMIZATION AT LANGLEY RESEARCH CENTER--AN EVOLUTIONARY PROCESS

JAMES L. ROGERS, JR.

APRIL 1982

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

COMBINING ANALYSIS WITH OPTIMIZATION AT LANGLEY RESEARCH CENTER--

AN EVOLUTIONARY PROCESS

James L. Rogers, Jr.
NASA Langley Research Center
Hampton, Virginia

INTRODUCTION

There has been a considerable investment of resources by both government and industry in the development of automated structural design and analysis methods (e.g., refs. 1 and 2). In addition, a number of interdisciplinary design studies have been completed which indicate the benefits of these methods in design (e.g., ref. 3). Computer implementations of such methods for use in design studies typically take the general form given in figure 1. As the figure shows, each discipline begins its own intradisciplinary analysis and optimization with an initial design. However, one discipline (e.g., flutter sizing) may be dependent upon results found in the analysis and optimization process of another discipline (e.g., strength sizing). Thus, once the strength sizing has been completed, the results are used to set a minimum gauge in place of estimates previously input for the flutter sizing. This is a sequential approach to automated design and implies that iterations are to be made until an optimum design is obtained. However, because of budget and time constraints, very few (if any) iterations with interdisciplinary optimization are carried out in design study applications. What is needed, therefore, is an approach which allows concurrent multidisciplinary analysis and optimization (fig. 2). In such an approach, the software system is capable of performing the analysis for several disciplines in parallel (concurrent analysis) and then have the optimizer take into account the constraints from all the different analyses (concurrent optimization), allowing users to achieve more iterations and therefore obtain a more optimal design than of that from the sequential approach. One of the long-term goals at NASA's Langley Research Center (LaRC) is to develop the methodology for such systems.

As a part of the effort to reach this long-term goal, LaRC has been combining analysis and optmization codes since 1971. The resulting programs show a steady evolution from relatively elementary, special-purpose programs with limited capabilities to modular, flexible systems of programs with more general capabilities. This paper first traces the three evolutionary lines along which computer programs combining analysis and optimization have beendeveloped at LaRC, namely, strength sizing, concurrent strength and flutter sizing, and general optimization (fig. 3). Analytical and computational advances contributing to this evolutionary process are described. The near-term goal, a state-of-the-art software system which executes the analysis and optimization in a sequential rather than concurrent mode, is then described as a major step toward reaching the long-term goal. Finally, one of LaRC's current efforts in combining analysis and optimization codes to be incorporated into the software system satisfying the near-term goal is described. The description of this current effort is in terms of how this analysis and optimization system works, how this system is connected using a special-purpose language, how this system communicates with a data base, and how new programs can easily be added to the system. Some numerical results are also shown.

PAST COMBINATIONS OF ANALYSIS AND OPTIMIZATION AT LANGLEY RESEARCH CENTER

Beginning in 1971, there has been a steady progression of programs or systems of programs combining analysis and optimization which were developed either in-house at LaRC or under contract. This progression of programs is listed below along with the meaning of their names (where applicable), the primary date of publication, and reference(s):

1971 DAWNS (Design of Aircraft WiNg Structures, ref. 4)
1971 SWIFT (refs. 5 and 6)
1972 SAVES (Sizing Aerospace VEhicle Structures, ref. 7)
1972 FADES (Fuselage Analysis and DEsign of Structures, refs. 8 and 9)
1973 WIDOWAC (WIng Design Optimization With Aeroelastic Constraints, ref. 10)
1978 ISSYS (Integrated Synergistic SYStem, ref. 11)
1979 PARS (ProgrAm for Resizing Structures, ref. 12)
1979 PROSSS (PROgraming Structural Synthesis System, refs. 13 and 14)
1981 Distributed PROSSS (refs. 15 and 16)

Programs combining analysis and optimization at LaRC have evolved along the three lines indicated in figure 3. These lines are: (1) strength sizing, (2) concurrent strength and flutter sizing, and (3) general optimization. The evolution of programs along each of these lines was a natural consequence of advances in six areas: (1) structural application (components), (2) structural representation (mathematical model), (3) analysis, (4) optimization, (5) flexibility of use, and (6) computer implementation features. The effect of these advancements on the evolutionary process is described briefly below. A more detailed comparison of these six areas is given in Tables 1-3.

Strength sizing is the first line of evolution. The first codes in this line, DAWNS and FADES, were very limited in the size of the models they could analyze because all of the analysis and data handling was executed in core. The control of their execution was done through main programs and subroutines, or overlays. For optimization, DAWNS used the weight/strength method coupled with fully-stressed design (FSD). Techniques allowing the use of mathematical programing to solve optimization problems were incorporated into later programs beginning with FADES. SAVES was the first system to use a general-purpose finite-element program (NASTRAN, ref. 17) for analysis. The use of NASTRAN allowed much larger problems to be solved in a batch environment because not all of the data had to be kept in core. The data handling

N82-24846 #

was done using sequential data files stored on disk during execution and magnetic tapes for restarting. SAVES was controlled by a FORTRAN callable subroutine which called sequences of control cards to execute various programs. Although SAVES was capable of analyzing a complete airframe, the optimization process was adapted from DAWNS and limited to wing structures.

When SPAR (ref. 18) was developed, the user was provided with a finite element analysis program suitable for an interactive environment because of its modularity, but still applicable to large models. At about the same time, the Control Data Corporation (CDC) Network Operating System (NOS)[1] became available at LaRC. With NOS, the user was able to take advantage of permanent disk files, interactive operation, and a method of combining CDC executive control language commands (ref. 19) into procedure files. ISSYS then evolved to take advantage of the features offered by SPAR and NOS. Most of the data handling and storage is done using the SPAR data management system (DMS). The controlling network accessed procedure files using CDC executive command control language to connect more than one program into a modular system. ISSYS, using SPAR and the method of usable-feasible directions, could perform structural analysis and optimization on a complete airframe. The modularity of ISSYS allowed programs to be added which performed aerodynamic and aeroelastic analysis. However, in ISSYS, the structural and flutter optimizations were executed sequentially rather than concurrently.

Concurrent strength and flutter sizing is the second line of evolution. SWIFT and WIDOWAC, like DAWNS and FADES, were very limited in the size of the models they could analyze because all of the analysis and data handling was done in core. Both programs used the sequential unconstrained minimization technique (SUMT, ref. 20) for optimization. SWIFT succeeded in performing concurrent strength and flutter sizing, but only for simple plate wings. WIDOWAC evolved from SWIFT to take advantage of the finite-element method (FEM) for structural representation. PARS was developed to take advantage of the features in SPAR. For example, PARS used the SPAR special-purpose language (runstreams) to control the flow of execution. Programs were added to SPAR to perform optimization, and aerodynamic and aeroelastic analysis making PARS a modular system within a single program. These programs, when incorporated into SPAR, are called SPAR processors. PARS became the first system to allow the user to perform strength and flutter sizing on a general structure. Although WIDOWAC and PARS were both originally designed to perform concurrent strength and flutter sizing, upon implementation it was found that this was not economically feasible within the state-of-the-art of analysis as it then existed. Thus, both WIDOWAC and PARS were reset to execute in a sequential mode. Additional work is required to obtain approximation techniques that would reduce the cost of such analyses to some economically feasible level (refs. 21, 22).

General optimization is the third line of evolution. CONMIN (ref. 23) is a general-purpose optimizer based on the method of usable-feasible directions and gives the user a optimizer that can easily be interfaced with analysis codes. The Approximation Concepts Code for Efficient Structural Synthesis (ACCESS, ref. 24) provided a methodology for using reciprocal design variables, design variable linking, and linear approximations. PROSSS was developed to take advantage of the features in SPAR, CONMIN, NOS, and some of the methodology provided by ACCESS. It should be noted that before PROSSS all of the analysis/optimization codes at LaRC relied on a preset definition of the design variables, constraints, and objective function, as well as a preset optimization procedure. With PROSSS, the user gained a large degree of flexibility because of the way in which the programs within the system pass data and are connected by executive control language commands. This connecting network allows the user to substitute problem dependent programs and formulations of the design variables, constraints, and the objective function at execution time thus making the system adaptable to a wide spectrum of structural optimization problems. Although SPAR and CONMIN are used for the analysis and optimization in PROSSS, other analysis and/or optimization programs could be substituted. Distributed PROSSS evolved from PROSSS after a PRIME minicomputer was made available to LaRC researchers. Distributed PROSSS takes advantage of the best features of both the mainframe (e.g., faster CPU) and the minicomputer (e.g., virtual storage and faster interaction) by distributing the structural analysis and optimization process between the two computers. Distributed PROSSS also reduced the complexity of the system by placing a majority of the control (previously handled through executive command control language) within a FORTRAN program on the minicomputer. The ability to examine and plot intermediate results from Distributed PROSSS significantly reduced the total amount of wall-clock time required for the design process.

As the above synopsis indicates, during the period 1971-1981 there has been a steady progressive development of computer programs combining analysis and optimization at LaRC toward the long-term goal of developing the methodology to perform concurrent analysis with concurrent optimization. This long-term goal requires that the software system have the capability of performing the analysis for several disciplines in parallel (concurrent analysis) and then have the optimizer take into account the constraints from all the different analyses (concurrent optimization). One way this can be accomplished is to combine the methodology from Distributed PROSSS with a multilevel optimization system for decomposing a large optimization problem into a hierarchy of much smaller problems (ref. 25). Engineers can then work the smaller problems using a distributed network of state-of-the-art micro- and minicomputers connected to a common data base. The development of analysis/optimization programs is continuing and a near-term goal has been established. The near-term goal for combining analysis and optimization will execute in a sequential rather than concurrent mode. Hence, the near-term goal does not meet the the requirements of the long-term goal, but it is a major step in that dirction. This near-term goal is discussed in the next section.

THE NEAR-TERM GOAL FOR COMBINING ANALYSIS AND OPTIMIZATION AT LARC

As presently defined, the near-term goal for combining analysis and optimization codes at LaRC is to develop a modular software system which combines general-purpose, state-of-the-art, production-level analysis computer programs for structures, aerodynamics, and aeroelasticity with a state-of-the-art optimization program featuring an FSD capability as well as either the method of usable-feasible directions or SUMT. This system is to be applied to general structures using (1) finite element models, (2)

[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.

general, user-defined design variables, constraints, and objective function, (3) a user-formulated optimization procedure, and (4) a DMS for storing and retrieving data. PARS, ISSYS, PROSSS, and Distributed PROSSS do not satisfy all of these criteria. For example, PARS and ISSYS only allow a preset definition of design variables, constraints, objective function and optimization procedure. In addition, ISSYS is not applicable to a general structure. PROSSS and Distributed PROSSS lack aerodynamic and aeroelastic analysis capabilities. However, the modularity of PARS, ISSYS, and PROSSS and the availability of a new finite element analysis computer program called Engineering Analysis Language (EAL, ref. 26) have provided the necessary ingredients to develop a software system which satisfies all of the above criteria. Because of their modularity, each program can be incorporated as a processor within EAL. By incorporating each program into a single program, the system temporarily loses some of the benefits derived from Distributed PROSSS. However, it is anticipated that these benefits will be recovered in the long-term as advances are made in distributing hardware and software.

EAL, which evolved from SPAR, provides the required finite element structural analysis capability for general structures as well as the DMS. CONMIN, which is contained in PROSSS, satisfies the requirements for an optimization program. The modularity of EAL and the capability to add new programs provide the user with much flexibility in defining the design variables, the constraints, and the objective function as well as in formulating an optimization procedure. There are efforts currently underway to incorporate procedures, processors, and programs from PARS, ISSYS, and PROSSS into EAL. Additional analysis modules which are closely related to structural optimization (such as new, advanced aerodynamic analyses needed for aerodynamic loads and aeroelasticity) will be added to EAL to meet the requirements of multidisciplinary analysis and optimization which deal with the airframe directly. The resulting system will be designated EAL/ISSYS (fig. 3) and will satisfy the requirements set down for the software system meeting the near-term goal. Users of this system will have much greater flexibility to solve a wide variety of engineering problems as well as evaluate new techniques in both analysis and optimization. All modules will communicate through the EAL DMS. Because all of the software will be either computer-independent FORTRAN or EAL runstreams, this system should be easily portable to any other government agency or any private company.

It is anticipated that in addition to EAL/ISSYS, which will be focused on airframes, there will be a need for a system supporting much broader applications such as general aerospace vehicles. In these applications, there will be many other major analyses involved, each represented by a program or system of programs too large and too complex to be added as processors in EAL. Integration of these programs with EAL/ISSYS can be accomplished with the aid of a state-of-the-art DMS such as RIM (Relational Information Management, ref. 27). A proposed system is diagramed in figure 4. In this system, each major analysis program has its own data base for storing data used only by that program. RIM is used to store data (such as design variables and constraints) that is to be passed between the major analysis program and procedure files for controlling the flow of execution.

One of the current efforts at LaRC that will aid in reaching the near-term goal (EAL/ISSYS) is the incorporation of PROSSS into EAL to form EAL/PROSSS (fig. 3). The remainder of this paper describes the salient features of PROSSS and the process for incorporating PROSSS into EAL. Since this process is representative of the manner in which any code can be incorporated into EAL, the reader should gain some understanding as to how EAL/ISSYS is being developed.

STRUCTURAL ANALYSIS AND OPTIMIZATION USING PROSSS

The flowchart of the PROSSS structural optimization software system is shown in figure 5. This flowchart is the same regardless of which of the three versions of PROSSS is being used. The five major components in the system are: (1) initialization; (2) analysis (EAL or SPAR); (3) optimization (CONMIN); (4) interface processing; and (5) termination.

During initialization, certain problem-dependent variables, files, and sometimes file names are initialized to the desired values. An example of this is the input data required for CONMIN. The initialization is not repeated during the analysis/optimization process.

There are two parts to the analysis portion of the process. One part, the nonrepeatable analysis, is executed outside of the optimization loop and is executed only once unless the user changes the structural model. The nonrepeatable part of PROSSS generates tables of material constants, section properties, joint locations, and element connectivities for the initial design variables. If analytical gradients are required, then the derivatives of the mass and stiffness matrices with respect to each design variable are also computed. These data are then stored in the data base on a temporary disk file and are referenced by other programs in the optimization loop. These data can also be saved on a permanent file so that it is not necessary to execute the nonrepeatable analysis for each subsequent execution of PROSSS. This is a very desirable feature when the user has a fixed model and is evaluating various new optimization and analysis techniques. The second part of the analysis portion of the process is the nonrepeatable analysis which is executed iteratively in the optimization loop. During this portion of the analysis, the changed design variables are used to calculate the new behavior variables for the structure. If analytical gradients are required, th the derivatives of the behavior variables are also computed. These data are also stored in the data base.

CONMIN, the optimizer, computes a new set of design variables based on the values of the objective function, constraints, and optionally their gradients. In PROSSS, CONMIN is treated as a "black box". A user-supplied, problem-dependent driver program is written to input the data created by the end processor, call the optimization subroutines, and output data using either files or the data base, depending upon the version of PROSSS being used.

The interface processors are also user-supplied, problem-dependent programs. They are used to communicate between the analysis and optimization programs. The front processor receives the updated design variables from CONMIN and converts the data into a format suitable for input into SPAR or EAL. The end processor receives the behavior variables, and optionally their derivatives for analytical gradient calculation, and converts the data into constraint data formatted for input into CONMIN. The capability of adding these two processors and the CONMIN driver program to EAL contributes to the flexibility of the system.

Certain termination criteria (such as the objective function not changing greater than a given tolerance within three successive passes through the system) are determined within the CONMIN driver program. If these criteria are met, the CONMIN driver program creates a termination file causing execution to terminate. Execution may also terminate if a predefined number of optimization loops is exceeded. The last criterion prevents the user from spending too much computer time on a poorly defined problem.

INCORPORATION OF PROSSS INTO EAL

Incorporating PROSSS into EAL to form EAL/PROSSS is one of LaRC's current efforts in the evolutionary process of combining analysis and optimization codes. A system with enhanced portability and flexibility is achieved by taking advantage of the EAL special-purpose language commands and data base. Use of the EAL data base simplifies the restarting of an abnormally terminated EAL execution. The ease in which new processors can be added to EAL makes the system readily adaptable to a wide spectrum of structural optimization problems. The remainder of this paper describes these features in detail. In addition, the model used for validating each of the PROSSS systems is described and the key results are shown.

Connecting Processors in EAL with the EAL Special-Purpose Language

The connecting network ties all of the system's components together. For PROSSS, an executive control language network connects procedures, programs, and data. For Distributed PROSSS, FORTRAN programs using a PRIME feature of FORTRAN callable procedure files replaced the complex procedure files used in PROSSS. The procedure files in Distributed PROSSS are very simple, because the FORTRAN driver programs contain all the looping, branching and testing logic required for executing the programs in PROSSS.

EAL (Engineering Analysis Language) is, as the name implies, a special purpose language contained within a state-of-the-art finite element computer program. This language gives the capability of doing most of the operations normally done in FORTRAN. The operations include testing, branching, looping and arithmetic calculations. There are also commands that simplify retrieval of data from the data base. The commands are simple and easy to use. For example, to branch in EAL requires:

```
*JUMP 100
   .
   .
   .
*LABEL 100
```

Thus, the controlling network can now be written in computer-independent EAL runstreams.

In the two earlier versions of PROSSS, SPAR was used for the analysis and two FORTRAN programs were written to create SPAR runstreams to aid in calculating analytical graidents. FORTRAN programs were used because the runstreams had to be general and satisfy any number of load cases and any number and type of design variables. SPAR, although quite similar to EAL in many respects, lacks the EAL commands for looping, testing, and branching. Thus, another advantage to incorporating PROSSS into EAL is that the two FORTRAN programs can be replaced by two general EAL runstreams that take advantage of EAL's looping and branching commands. This increases portability while decreasing complexity. A portion of the listing containing the EAL runstream for computing the derivatives of the stiffness and mass matrices with respect to the design variables is shown in table 4 as an example of the commands used in EAL.

Communicating Between New Processors and the EAL Data Base

Incorporating PROSSS into EAL takes advantage of the EAL data base. The data base is written so that it can easily be accessed by any processor using FORTRAN callable utility subroutines. These utility subroutines are stored in the main overlay which remains in core at all times; thus these subroutines are available not only for the original EAL processors, but also for any processors the user may wish to add.

To use these subroutines, the user must be familiar with their functions and calling parameters (ref. 28) as well as how the data are stored in the data base (ref. 29). Data are stored in the data base in two-dimensional tables or matrices called data sets. The data sets are referenced by a four-word name such as STRS E23 i j--where STRS means stress, E23 is the element type, i is the load set number, and j is the load case number within the load set. The data sets are stored on disk in libraries within the data base (fig. 6). When two data sets containing data for the same item and having the same data set name exist on the same library in the data base, only the latest stored item can be accessed by other programs. There are 30 libraries available for the user to store data sets, however, library 30 is generally reserved for system usage. Data sets can be transferred from one library to another using standard EAL runstream commands. The typical user only stores data in library 1, which is the default.

One of the powerful features of EAL is the capability of extracting data from the data base using runstream commands. This information is very useful in setting up general-purpose runstreams. Using this capability such information as the number of design variables, number of load cases, number of joints, and element names can be extracted from the data base and stored in runstream variables called registers. These registers are then used for loops, branches and calls to other data sets within the general runstream. This means that some runstreams, such as the runstream that computes the derivatives of the stiffness and mass matrices with respect to the design variables, do not have to be coded differently for different problems or different users.

It may also be necessary for users to create their own data sets for use in the processors they are coding. The user is then responsible for the naming of the data set and the format in which the data is stored within the data set. An example of this type of data set in EAL/PROSSS is the data set containing the design variables. These data sets are not used by the standard EAL processors, but are used to pass data between the front and end processors, and the CONMIN driver, all of which are coded by the user.

The ability to communicate between a processor in central memory and the data base on disk requires that certain EAL utility subroutines be used. The data being used in EAL processors are usually stored in the central memory area reserved for blank common (fig. 6). These routines are used for moving data from blank common into the data base and moving data from the data base into blank common. The call to the utility subroutine first specifies the data set which is to be addressed within the library. A starting address in central memory is also passed in the subroutine call. The starting address, usually an address within blank common, specifies the address in central memory where the data being retrieved from the data base is to be placed or an address in central memory that is to be stored in the data base. An operation code, also passed through the subroutine call, specifies whether the data areto be stored or retrieved. Once the data have been placed in the data base, it can be accessed by any other processor or runstream. Once the data have been successfully transferred from the data base to central memory, it can be used in the accessing processor just as any data might be used.

In the two earlier versions of PROSSS, only data created by the SPAR analysis was stored in the data base. Data for initialization and data created by the front and end processors and CONMIN were all kept on separate disk files. Managing all of these files as well as the flow of the system is very cumbersome in an executive control language. Because all of the data are stored in the EAL data base in EAL/PROSSS, the use of files is minimized which greatly reduces the complexity of the connecting network.

Restarting an execution in EAL is simple because of the data base. A disk file (or files) with libraries containing the most recently computed data sets is saved at the end of an EAL execution. This is true even if the execution terminated abnormally. If the execution does terminate abnormally, changes can be made either to a processor or to a runstream; then the execution can be restarted at the beginning of the last pass through the optimization loop.

A capability that allows the user to examine intermediate results is currently available in Distributed PROSSS and is to be added to EAL/PROSSS. With this feature the user will be able to examine plots and/or listings of the intermediate results and determine if the analysis/optimization process is executing as anticipated. If a problem is discovered, the user can stop the execution, make the necessary changes, and then restart execution. For example, the user, upon examining his intermediate results, may find that one of the design variables is lodged against a lower-bound constraint which was input to CONMIN. At this point, the user can stop execution, relax the constraint condition, and then restart execution. Because of this restart feature, loss of previous (and sometimes expensive) calculations can be prevented.

Adding Processors to EAL

Just as FORTRAN programs call subroutines to perform a specific task, EAL calls processors. The processors are incorporated into EAL as overlays. The main overlay, which is always in core, contains the input/output routines and commonly used utility programs. The next (primary) level of overlay contains the processors. The main overlay can call a primary overlay into core (fig. 6). In EAL, this is done with runstream commands such as:

```
*XQT processor name
```

To add PROSSS to EAL requires the addition of new processors external to EAL, such as the front processor, the end processor and the CONMIN driver. Since EAL is a proprietary program, its source code is not available at LaRC. To add a new processor requires using an external overlay as shown in figure 7. When an *XQT command is encountered in EAL, a test is made to see if a legitimate EAL processor is being called. If it is not a legitimate EAL processor and an *XQT EXTERNAL command has already been encountered, then EAL branches to an external overlay called EXT to execute the additional processors. The division of the processors between the two overlays is shown in figure 8.

The primary overlay in the external overlay contains a driver program to check and determine if the called processor is a legitimate processor in the external overlay. If it is not a legitimate processor, an error message is issued. If it is a legitimate processor, a secondary overlay is called to execute the processor.

As mentioned previously, one of the key features of PROSSS is its flexibility. To maintain this flexibility when incorporating PROSSS into EAL, dummy drivers were added to the external overlay. The dummy driver program performs only one task, the calling of a subroutine with a specific name. Thus, once the relocatable overlay structured file is created it does not have to be recreated for different users or different problems. The unsatisfied subroutines called by the dummy driver programs in the external overlay program are satisfied at load time with user-supplied, problem-dependent subroutines. The only requirement placed on the user is that the subroutine names used in the dummy driver programs must also be used in the subroutines. The file containing the subroutine can have any name.

Another key feature available when adding new processors to EAL is the use of reset. EAL provides a utility routine for resetting certain variables used in the processors. This is similar to the passing of parameters to subroutines. When a user writes a problem-dependent processor, variables can be initialized with default values in data statements. Should the user decide to change the defaults at execution time, there is no need for him to change the source code, recompile, and reload the absolute overlay. A reset command with the variable name and its new value following the command that executes the processor can be added. For example, suppose the external processor EPRC required the number of beam elements (NE21) used in the model as input. Suppose the user initially defaults the NE21 parameter to 75, but now wishes to change the number of beam elements to 100, then the following commands are required:

```
*XQT EPRC
RESET NE21=100
```

The EAL utility routine is then called by the processor to reset the variable. The only thing required of the user is to plan ahead and design code to handle all variables that might be reset.

Test Case

A 337 degree-of-freedom finite element model of an idealized segment of a fuselage with a cutout (fig. 9) was used to test each version of PROSSS. The model consists of 80 joints connected to form 76 rod elements, 58 beam elements and 56 membrane elements. The cross-sectional areas of the rods and beams and the thicknesses of the membranes were used as design variables. For this test case, all rod areas are equal, as are the areas of the beams, and the thicknesses of the membranes.

The final objective function (mass) for EAL/PROSSS was 6343 Kg which agrees with the 6344 Kg found with PROSSS. The final cross-sectional area of the rods and the thicknesses of the membranes were in exact agreement at 2.0390 cm^2 and 0.1207 cm, respectively. There was only a slight difference in the final cross-sectional area of the beams between the two systems. EAL/PROSSS resulted in a cross-sectional area of 1.6025 cm^2 while the final area for PROSSS was 1.6002 cm^2.

CONCLUDING REMARKS

The evolutionary process of combining analysis and optimization codes at Langley Research Center is described with a view toward providing insight into the long-term goal of developing the methodology for an integrated, multidisciplinary software system for the design of aerospace structures. A current effort in this evolutionary process is discussed, particularly as it relates to the near-term goal of combining state-of-the-art, general-purpose, production-level analysis computer programs with a state-of-the-art optimization program. This effort, a software system designated EAL/PROSSS, is described in terms of its special-purpose language and data base features, and the process used for adding new programs. Some numerical results showing the accuracy of EAL/PROSSS are given.

REFERENCES

1. Markowitz, J., and Isakson, G., "FASTOP-3: A Strength, Deflection and Flutter Optimization Program for Metallic and Composite Structures," AFFDL-TR-78-50, Vol. 1, May 1978.

2. Dreisbach, R. L., and Giles, G. L., "The ATLAS Integrated Structural Analysis and Design Software System," NASA CP 2059, pp. 1-14, November 1978.

3. Silwa, S. M., "Sensitivity of the Optimal Design Process to Design Constraints and Performance Index for a Transport Airplane," AIAA Paper No. 80-1895, AIAA Aircraft Systems and Technology Meeting, Anaheim, CA, August 4-6, 1980.

4. Giles, G. L., "Procedure for Automating Aircraft Wing Structural Design," ASCE Journal of the Structural Division, Vol. 97, No. ST1, January 1971, pp. 99-113.

5. Stroud, W. J., Dexter, C. B., and Stein, M., "Automated Preliminary Design of Simplified Wing Structures to Satisfy Strength and Flutter Requirements," NASA TN D-6534, December 1971.

6. Stroud, W. J., "Automated Structural Design with Aeroelastic Constraints: A Review and Assessment of the State of the Art," Presented at the ASME Winter Annual Meeting, New York, NY, November 17-21, 1974. Proceedings entitled Structural Optimization Symposium, ASME, New York, AMD, Vol. 7, 1974, pp. 77-118.

7. Giles, G. L., Blackburn, C. B., and Dixon, S. C., "Automated Procedures for Sizing Aerospace Vehicle Structures (SAVES)," AIAA Journal of Aircraft, Vol. 19, No. 12, December 1972, pp. 812-819.

8. Sobieszczanski, J., and Loendorf, D. D, "A Mixed Optimization Method for Automated Design of Fuselage Structures," AIAA/ASME/SAE 13th SDM Conference, San Antonio, Texas, April 1972. AIAA Paper No. 72-330.

9. Sobieszczanski, J., "Sizing of Complex Structure by the Integration of Several Different Optimal Design Algorithms," AGARD-LS-70, Oct. 10-11, 1974, Hampton, Virginia.

10. Haftka, R. T., "Automated Procedure for Design of Wing Structures to Satisfy Strength and Flutter Requirements," NASA TN D-7264, July 1973.

11. Dovi, A. R., "ISSYS - An Integrated Synergistic System," NASA CR-159221, February 1980.

12. Haftka, R. T., Prasad, B., and Tsach, U., "PARS - Programs for Analysis and Resizing of Structures User's Manual," NASA CR 159007, April 1979.

13. Sobieszczanski-Sobieski, J.; and Bhat, R. B., "Adaptable Structural Synthesis Using Advanced Analysis and Optimization Coupled by a Computer Operating System." A Collection of Technical Papers on Structures - AIAA/ASME/ASCE/AHS 20th SDM Conference, April 1979, pp. 20-71, AIAA Paper No. 79-0723.

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

15. Rogers, J. L., Jr., Dovi, A. R., and Riley, K. M., "Distributing Structural Optimization Software Between a Mainframe and a Minicomputer," Presented at the Second International Conference and Exhibition on Engineering Software, London, England, March 24-26, 1981. Proceedings entitled "Engineering Software II," Hobbs The Printers, Southampton, England, 1981, pp. 400-415.

16. Rogers, J. L., Jr., "An Implementation of the Distributed Programing Structural Synthesis System (PROSSS)," NASA TM 83253, December 1981.

17. "The NASTRAN Theoretical Manual," NASA SP-221(04), 1978.

18. Whetstone, W. D., "SPAR Structural Analysis System Reference Manual, System Level 13A," Vol. 1. NASA CR 158970-1, December 1978.

19. Control Data Corporation, "FORTRAN Extended Version 4 User's Guide," Publication No. 60499700, 1979.

20. Fiacco, A. V.; and McCormick, G. P., Nonlinear Programing: Sequential Unconstrained Minimization Techniques. John Wiley and Sons, Inc., 1968.

21. Ricketts, Rodney H., and Sobieszczanski, J., "Simplified and Refined Structural Modeling for Economical Flutter Analysis and Design," Presented at AIAA/ASME/SAE 18th Structures, Structural Dynamics and Materials Conference, San Diego, CA, March 21-23, 1977. AIAA Paper No. 77-421.

22. Sobieszczanski-Sobieski, J., and Bhat, R. B., "Adaptable Structural Synthesis Using Advanced Analysis and Optimization Coupled by a Computer Operating System," Presented at AIAA/ASME/ASCE/AHS 20th Structures, Structural Dynamics, Materials Conference, St. Louis, MO, April 4-6, 1979. AIAA Paper No. 79-0723.

23. Vanderplaats, G. N., "CONMIN - A FORTRAN Program for Constrained Function Minimization User's Manual," NASA TM X-62282, August 1973.

24. Schmit, L. A., and Miura, H., "Approximation Concepts for Efficient Structural Synthesis," NASA CR-2552, March 1976.

25. Sobieszczanski-Sobieski, J., "A Linear Decomposition Method for Large Optimization Problems-Blueprint for Development," NASA TM 83248, February 1982.

26. Whetstone, W. D., "EISI-EAL: Engineering Analysis Language," Proceedings of the Second Conference on Computing in Civil Engineering, ASCE, 1980, pp. 276-285.

27. Erickson, W. J., "User Guide: Relational Information Management (RIM)," Report No. D6-IPAD-70023-M, Boeing Commercial Airplane Company, Seattle, Washington, 1981.

28. Giles, G. L., and Haftka, R. T., "SPAR Data Handling Utilities," NASA TM 78701, September 1978.

29. Cunningham, S. W., "SPAR Data Set Contents," NASA TM 83181, October 1981.

TABLE I.- COMPARISON OF PROGRAMS WITH RESPECT TO STRUCTURAL APPLICATION AND REPRESENTATION

		71 DAWNS	71 SWIFT	72 SAVES	72 FADES	73 WIDOWAC	78 ISSYS	79 PARS	79 PROSSS	81 DIST. PROSSS
STRUCTURAL APPLICATION	WINGS	×	×	×		×	×	×	×	×
	FUSELAGE				×		×	×	×	×
	COMPLETE AIRFRAME						×	×	×	×
	GENERAL STRUCTURE							×	×	×
STRUCTURAL REPRESENTATION	FINITE ELEMENT MODEL (DISCRETE)	×		×	×	×	×	×	×	×
	PLATE (CONTINUOUS)		×							

TABLE II.- COMPARISON OF PROGRAMS WITH RESPECT TO ANALYSIS AND OPTIMIZATION

		71 DAWNS	71 SWIFT	72 SAVES	72 FADES	73 WIDOWAC	78 ISSYS	79 PARS	79 PROSSS	81 DIST. PROSSS
ANALYSIS	STRUCTURAL (STATIC)	×	×	×	×	×	×	×	×	×
	STRUCTURAL (DYNAMIC)		×	×		×	×	×	×	×
	AERODYNAMIC	×	×	×		×	×	×		
	AEROELASTICITY (STATIC)					×	×	×		
	AEROELASTICITY (DYNAMIC)					×	×	×		
OPTIMIZATION	WEIGHT/STRENGTH	×		×						
	USABLE-FEASIBLE DIRECTIONS				×		×		×	×
	SUMT		×		×	×		×		
	FSD	×		×			×	×	×	×

TABLE III.- COMPARISON OF PROGRAMS WITH RESPECT TO FLEXIBILITY AND COMPUTER IMPLEMENTATION FEATURES

		71 DAWNS	71 SWIFT	72 SAVES	72 FADES	73 WIDOWAC	78 ISSYS	79 PARS	79 PROSSS	81 DIST. PROSSS
FLEXIBILITY	PRESET DEFINITIONS OF DESIGN VARIABLES, CONSTRAINTS, AND OBJECTIVE FUNCTION	×	×	×	×	×	×	×		
	GENERAL, USER DEFINED DEFINITIONS OF DESIGN VARIABLES, CONSTRAINTS, AND OBJECTIVE FUNCTION								×	×
	PRESET OPTIMIZATION PROCEDURE	×	×	×	×	×	×	×	×	×
	USER FORMULATED OPTIMIZATION PROCEDURE								×	×
COMPUTER IMPLEMENTATION FEATURES	FORTRAN	×	×	×	×	×	×	×	×	×
	EXECUTIVE CONTROL LANGUAGE			×			×		×	×
	SPECIAL PURPOSE LANGUAGE						×	×	×	×
	DATA MANAGEMENT SYSTEM						×	×	×	×
	DISTRIBUTED PROCESSING									×
	SINGLE PROGRAM		×			×				
	MODULAR SYSTEM	×		×	×		×		×	×
	MODULAR SYSTEM WITHIN A SINGLE PROGRAM							×		

TABLE IV.- SAMPLE EAL RUNSTREAM

```
*XQT AUS
!LCOS=DS,1,4,1(1,INFO,LOAD,0,0)
!NJNT=DS,1,1,1(1,JDF1,BTAB,1,8)
$
$  COMPUTE UNIT VECTOR
$
SYSVEC
UNIT VEC
I=1
J=1,"NJNT"
1.0
DEFINE UN=UNIT VEC
!NELT=TOC,NWDS(1,EL,NAME,0,0)
!ICNT=0
$
$  COMPUTE OBJECTIVE FUNCTIONS
$
*LABEL 100
!ICNT=ICNT+1
DEFINE W=DMDV DIAG "ICNT" 1
OBJF G "ICNT" 1=XTY(UN,W)
*JGZ,-1(NELT,100)
!NOLC=DS,1,1,1(1,INFO,LOAD,0,0)
!ICNT=0
!TOLC=LCOS-1
OUTLIB=3
```

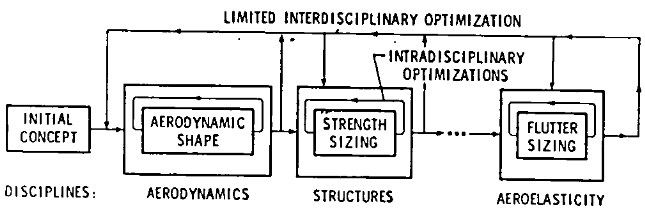

Fig. 1 A sequential approach to the design process.

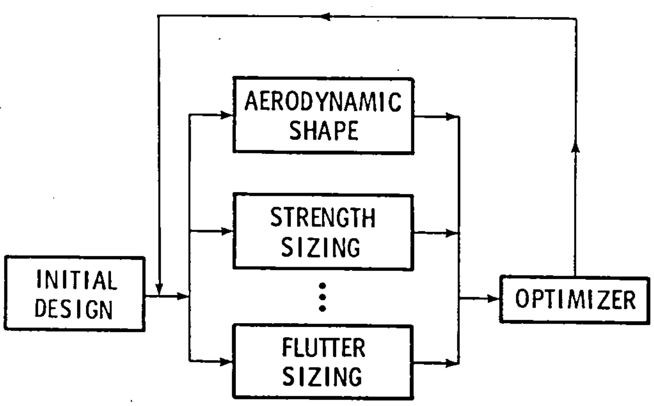

Fig. 2 A concurrent approach to the design process.

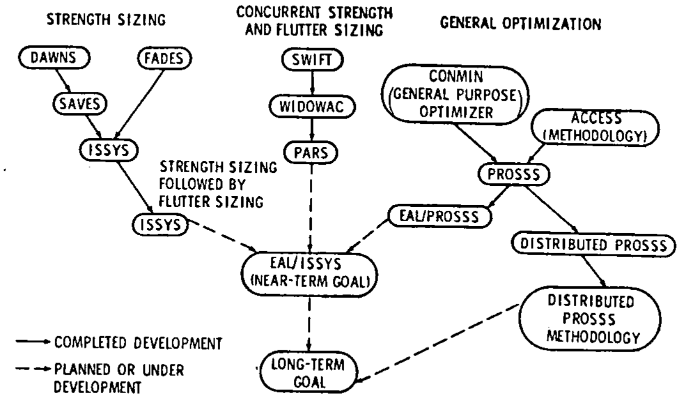

Fig. 3 Evolutionary lines for combining analysis and optimization at LaRC.

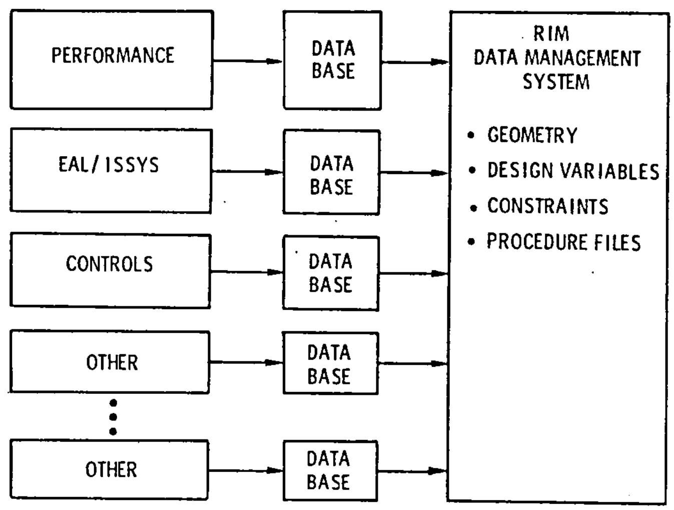

Fig. 4 Proposed multidisciplinary analysis and optimization system for general aerospace vehicles.

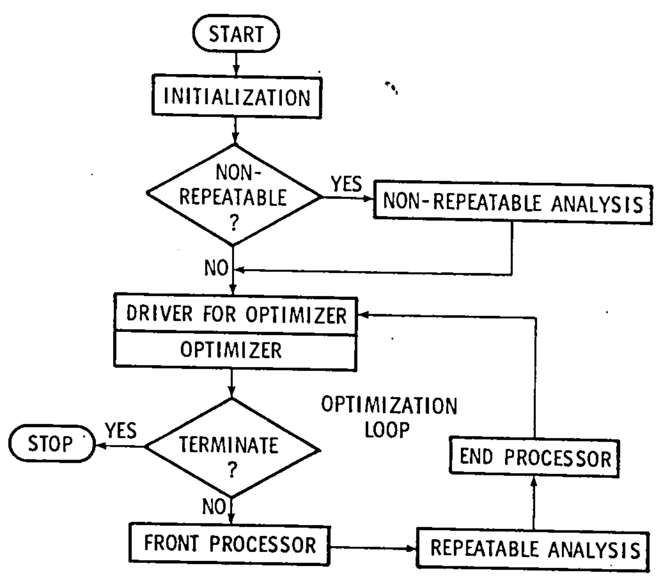

Fig. 5 Flowchart of the PROGraming Structural Synthesis System (PROSSS).

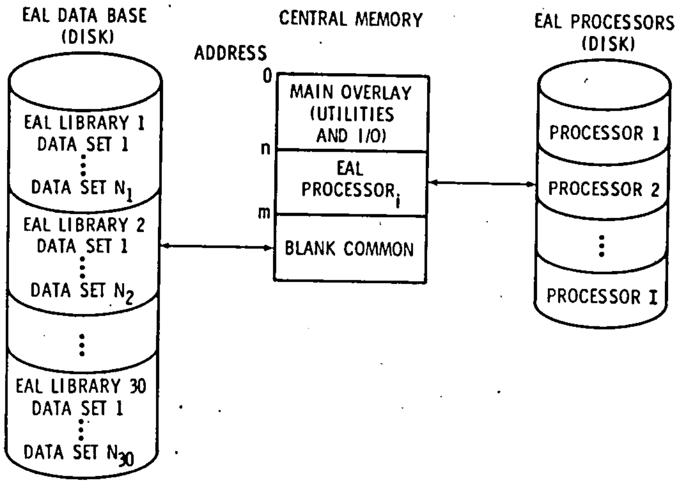

Fig. 6 Data communications in EAL.

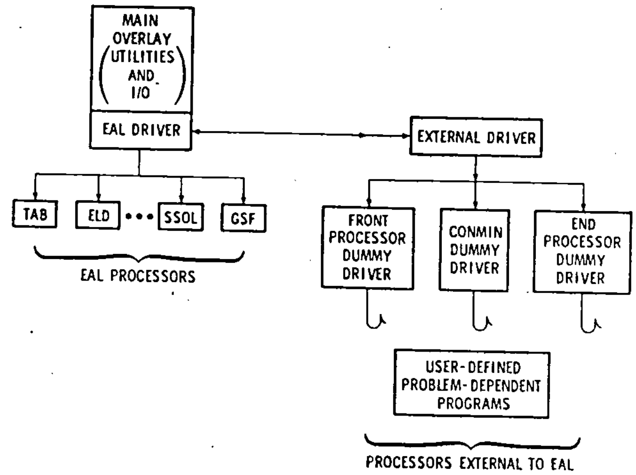

Fig. 7 EAL program structure with PROSSS dummy driver programs incorporated into an external overlay.

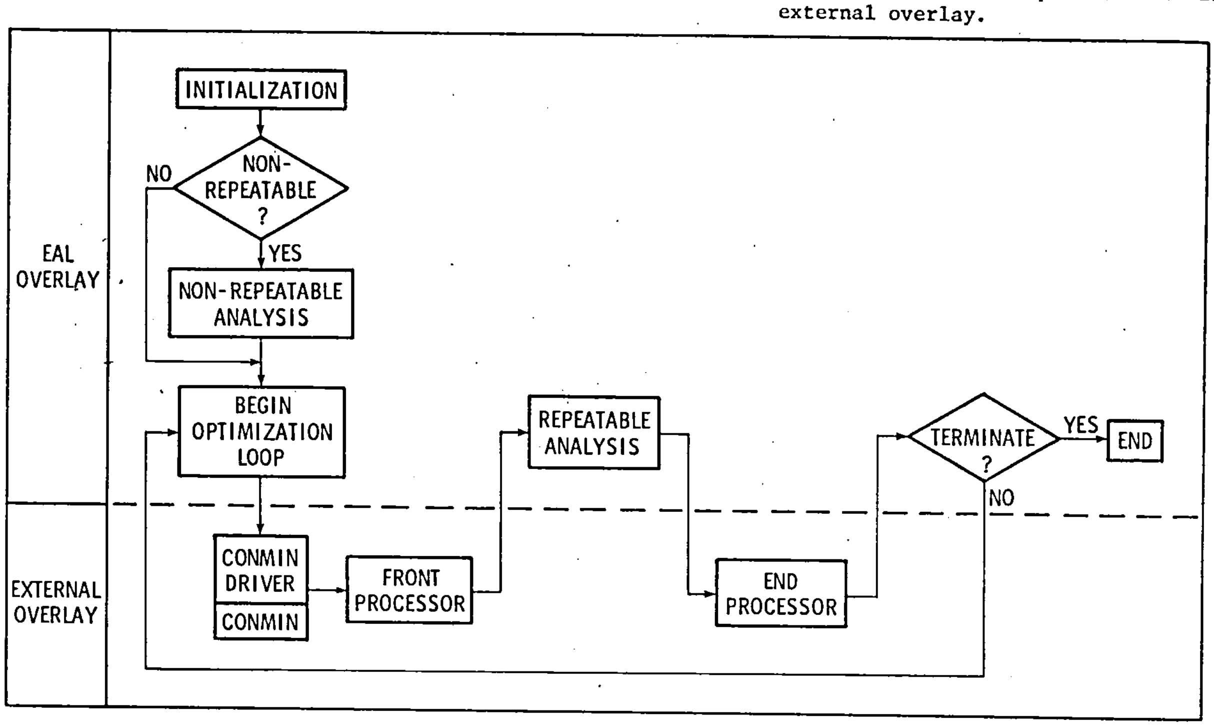

Fig. 8 Divisions of EAL/PROSSS processors between overlays.

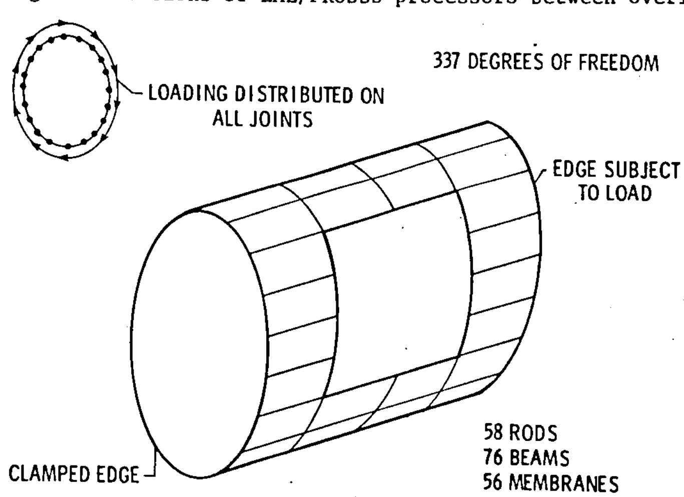

Fig. 9 337 DOF fuselage model.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM 84472		

4. Title and Subtitle

COMBINING ANALYSIS WITH OPTIMIZATION AT LANGLEY RESEARCH CENTER--AN EVOLUTIONARY PROCESS

5. Report Date

April 1982

6. Performing Organization Code

505-33-63-02

7. Author(s)

James L. Rogers, Jr.

8. Performing Organization Report No.

9. Performing Organization Name and Address

NASA Langley Research Center
Hampton, VA 23665

10. Work Unit No.

11. Contract or Grant No.

12. Sponsoring Agency Name and Address

National Aeronautics and Space Administration
Washington, DC 20546

13. Type of Report and Period Covered

Technical Memorandum

14. Army Project No.

15. Supplementary Notes

Presented at Second International Computer Engineering Conference and Show, August 15-19, 1982, San Diego, California.

16. Abstract

The evolutionary process of combining analysis and optimization codes at Langley Research Center (LaRC) is traced with a view toward providing insight into the long-term goal of developing the methodology for an integrated, multidisciplinary software system for the concurrent analysis and optimization of aerospace structures. The evolutionary process is traced along the lines of strength sizing, concurrent strength and flutter sizing, and general optimization. Research progress has led to the definition of a near-term goal for combining analysis and optimization codes at LaRC. This goal requires the development of a modular software system which combines general-purpose, state-of-the-art, production-level analysis computer programs for structures, aerodynamics, and aeroelasticity with a state-of-the-art optimization program. Because the software system satisfying the near-term goal will execute sequentially rather than concurrently, it does not meet the requirements of the long-term goal, but is a major step in that direction. One of the current efforts at LaRC that will aid in reaching the near-term goal is the incorporation of a modular and flexible structural optimization software system into a state-of-the-art finite-element analysis computer program. This effort results in a software system that is controlled with a special-purpose language, communicates with a data management system, and is easily modified for adding new programs and capabilities. A 337 degree-of-freedom finite-element model is used in verifying the accuracy of this system.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Structural Analysis, Optimization, Flutter Analysis Software System, Data Management Multidisciplinary	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	10	A02

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