TABLE OF CONTENTS

1. INTRODUCTION

Large, multidisciplinary engineering systems can require a complex design cycle. Before a design cycle begins, possible couplings among the design processes must be determined. After these couplings have been identified, a design cycle can be decomposed to identify its organization. The Design Manager's Aid for Intelligent Decomposition (DeMAID) is a knowledge-based software tool for ordering the sequence of design processes [1,2]. The DeMAID tool displays the processes in a design structure matrix (DSM) format in which an element on the diagonal is considered to be any process (or module) that requires input and generates an output [3]. Off-diagonal elements indicate a coupling between two processes. The primary advantage of the DSM format over other display tools, such as PERT or process flowcharts, is the ability to group and display the iterative subcycles that are commonly found in the design cycle. After the iterative subcycles have been determined, their processes must be ordered in such a way to produce the best design in the least time at a minimum cost. The original version of DeMAID handled this task with a knowledge base (KB). The KB, however, examines a limited number of orderings, which only provides the user a starting point to interactively search for the optimum. A genetic algorithm (GA) capability was added to the latest version of DeMAID, which is called DeMAID/GA; the GA allows a large number of orderings of processes in an iterative subcycle to be examined and optimizes the ordering based on cost, time, and iteration requirements.

2. BACKGROUND

This section provides background information. included in this information is an overview of the program and a description of two of the underlying concepts of DeMAID/GA, the DSM and the KB.

2.1 PROGRAM FUNCTION OVERVIEW

The DeMAID/GA software is written in C and is available on Mac and UNIX computers. DeMAID/GA is a menu-driven program with interfaces to a KB. Figure 1 indicates the major functions of DeMAID/GA. These functions are further described in section 4.

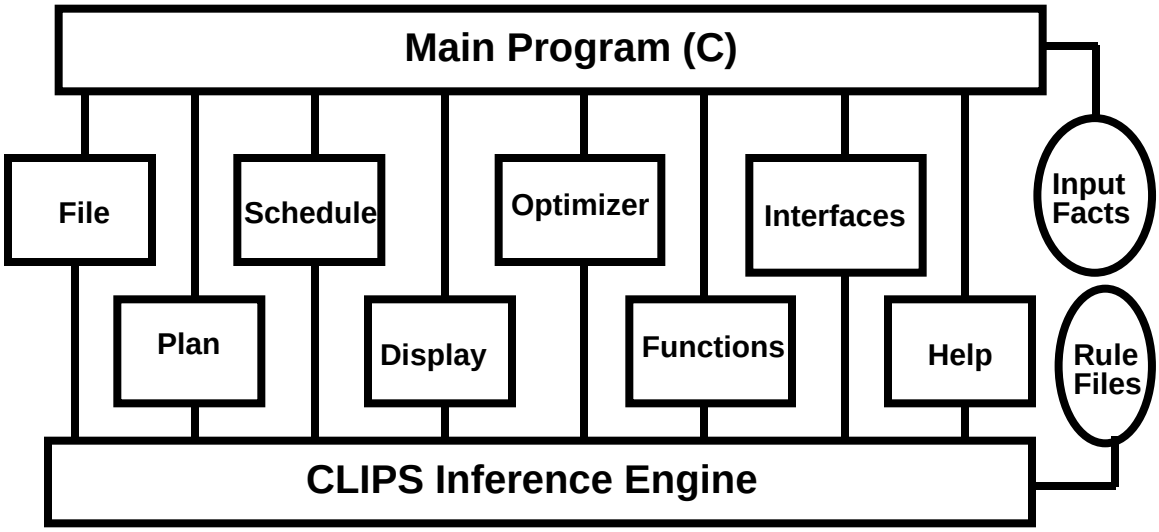

Figure 1. Diagram of DeMAID/GA.

2.1.1 File Structure

The user is only responsible for creating one file, the input file, which contains a list of all modules that might possibly contribute to the solution of the problem. The components of this file are described in section 3. Several of the DeMAID/GA functions require the user to specify a file name for input and/or output. A window opens to aid the user in making the selection. The prefix for the user-defined filenames should represent the problem, while the suffix should represent the function that creates the file. For example, if the user is designing a brake, the input file might be called brake.npt and the output files from the **Plan** and **Schedule** functions might be called brake.plan and brake.sked respectively. Because the user can name the files and their location, different directories or folders may be easily maintained for different projects. Each function in DeMAID/GA reads and writes from its own set of files described in section 4. Some of the functions write information to a log file which can be used as a reference in the decision making process.

2.1.2 Program Execution

DeMAID/GA is executed by “clicking” on the DeMAID/GA icon for Macs or typing in the name of the executable program on a UNIX computer. The executable file must be in the same folder with the rule files. The **Plan** function must always be executed first, followed by the **Schedule** function, followed by the **DSM** function. Within the context of DeMAID/GA, the term *plan* identifies the process in which modules contribute to the solution of the problem and the term *schedule* is used to describe the dividing the modules into iterative subcycles and the ordering the subcycles into a meaningful solution sequence. Once the data has been displayed as a DSM, any of the other functions in DeMAID/GA may be called. If the user has saved the data file created by the **Plan**, **Schedule** or **DSM** functions, then DeMAID/GA can be restarted from that point without having to execute previously called functions.

2.1.3 Computer Differences

DeMAID/GA functions are almost identical regardless of the computer. However, there three differences which need to be mentioned.

1. Before executing DeMAID/GA on a UNIX computer, the user must set the appropriate default values with the statement

setenv XENVIRONMENT app-defaults

2. Intermediate information is often written to a console window for the user to see. The data written to this window is also saved to a log file for future reference. On Mac computers, the console window is only displayed when information is written to it and sometimes it may be hidden behind the window displaying the DSM. On UNIX computers, the console window is always displayed.

3. The user can move the DSM around in the display window with scroll bars on a UNIX computer and with arrows on a Mac computer. Currently, there is no icon to show when the program is executing an function.

2.2 DESIGN STRUCTURE MATRIX

The DSM is used to display the sequence of processes [3]. A sample DSM is shown in figure 2.

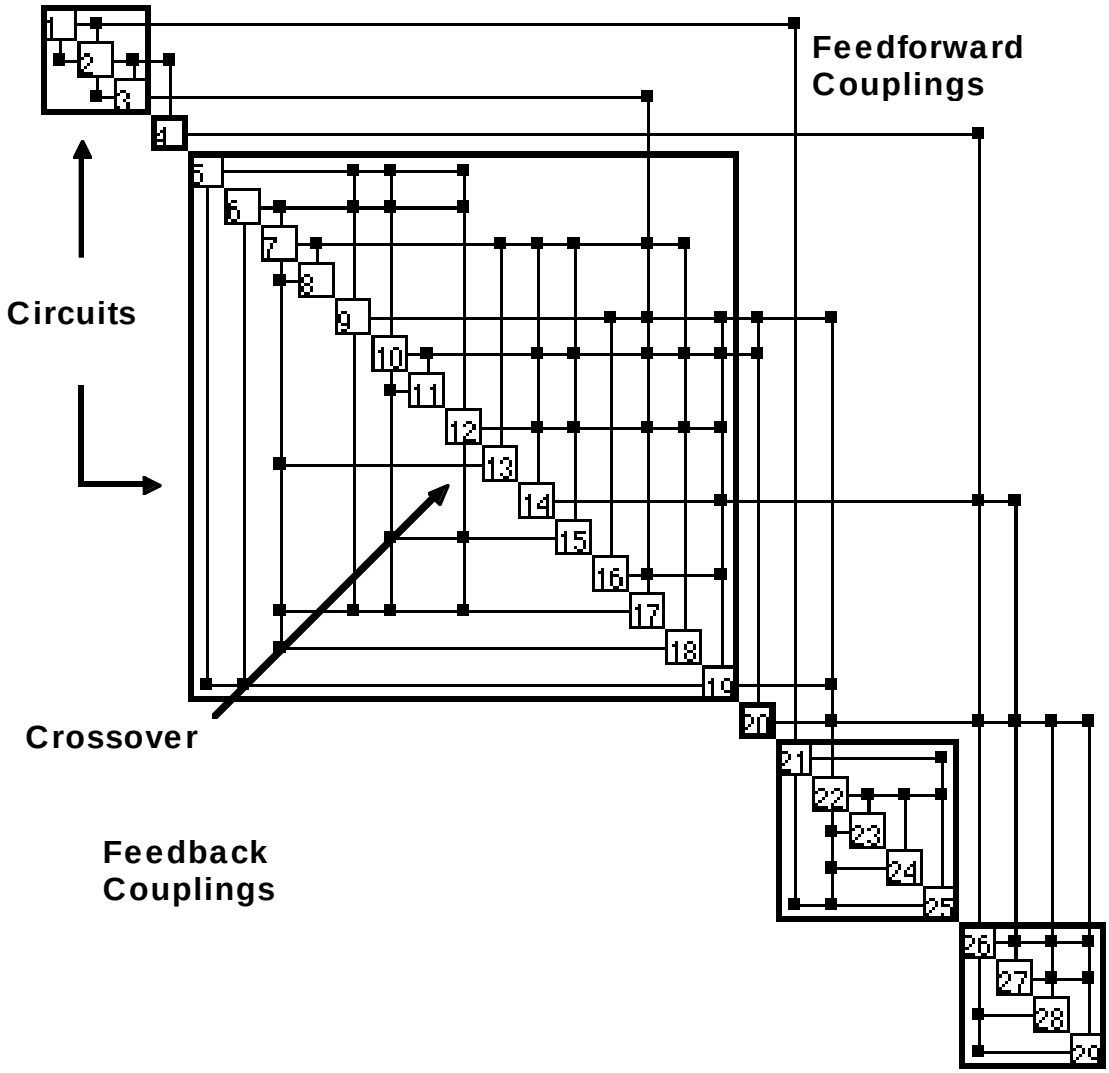

Figure 2. Design structure matrix.

In the DSM, the processes are shown as numbered boxes on the diagonal. Output from a process is shown as a horizontal line that exits a numbered box, and input to a process is shown as a vertical line that enters a box. The off-diagonal squares that connect the horizontal and vertical lines represent couplings between two processes. Squares in the upper triangle of the DSM represent *feedforward couplings*; squares in the lower triangle of the matrix represent *feedback couplings*. Feedback couplings imply iterative subcycles in which initial data estimates must be made. The KB within DeMAID/GA which is written with CLIPS [4], orders the processes in such a way that feedback couplings are eliminated. However, in many cases, not all of the feedback

couplings can be eliminated. If certain feedback couplings cannot be eliminated, DeMAID/GA groups the processes into iterative subcycles called *circuits*. In figure 2, processes 1-3, 5-19, 21-25, and 26-29 are grouped into circuits. The circuits boxes are drawn with a thicker line to distinguish them for the process boxes.

The DeMAID/GA software also identifies *crossovers*.. Crossover, in this context, occurs when feedback from one process crosses that of another process without an exchange of data through the intersection (no off-diagonal square). Crossovers are only defined in terms of feedback couplings. For example, in figure 2, a crossover occurs when the feedback coupling process 13 to process 7 crosses the feedback coupling process 17 to process 12. Crossovers should be avoided if possible because they can obscure straightforward convergence of the design process. The DSM in figure 2 contains 20 feedback couplings and 3 crossovers.

In the original version of DeMAID, a KB was used to minimize feedback couplings and identify circuits. Crossovers were identified but were not minimized. No time factor, cost factor, or iteration factors (i.e. the number of iterations required for convergence) were applied. After the circuits were identified, DeMAID attempted to minimize the feedback couplings within a circuit. In most cases, although more than one ordering could produce the minimum number of feedback couplings, only one ordering would be found.

A large circuit, such as the one in figure 2 that contains processes 5-19 can be very expensive to converge because the iterative subcycles defined by the feedback couplings are nested, which requires numerous executions of potentially expensive processes. Thus, the original version of DeMAID was enhanced to provide a GA that rapidly examines many different orderings of processes within a circuit and selects the best ordering based on cost, time, and iteration requirements. This enhanced version is called DeMAID/GA.

2.3 THE KNOWLEDGE BASE

The CLIPS [4] knowledge-based system was developed at NASA Johnson Space Center. The CLIPS system is written in C and performs forward chaining based on the Rete pattern matching algorithm. Three main components make up this knowledge-based system (fig. 1): the facts (section 2.3.1), the rules (section 2.3.2), and the inference engine (section 2.3.3).

2.3.1 Facts

Facts are the basic form of data in the KB and are contained in a facts-list. A fact is composed of one or more fields, with each field separated by a space. A field can contain a number, a word, or a string. Facts can be asserted into the facts-list by an *assert* command in the calling program before the inference engine is executed. The user creates a list of facts as an input file (sections 3 and 5) to the program. An example of a fact defining a module (section 3.1.3) is

(module 7 structures 10 20 feout uk geom aero)

2.3.2 Rules

The name of the rule is declared in the *defrule* statement and must be unique. A rule states that specific actions (after the => sign) are to be taken if certain conditions (before the => sign) are met. Examples of actions include returning data to the calling program or asserting a new fact into the facts-list. A "?" before a name in a rule indicates a single variable and a "$?" before a name indicates a multi-variable list. Any line beginning with a ";" is a comment.

A rule is executed based on the existence or non-existence of facts in the facts-list. Each major function (section 4) in DeMAID/GA has its own independent rule base. Currently 21 rule files can be loaded into the KB as needed. The user cannot modify the rules. An example of a rule which finds tightly coupled modules is

```
; Find tightly coupled modules where the output of
; module "name1" is input to module "name2" and vice-versa
;---------------------------------------------------------------------------------------------
(defrule find-tightly-coupled-modules
;
;  conditions
;
  (module ?number1 ?name1 ?type1 ?time1 ?output1 uk $?inlist1)
  (module ?number2 ?name2 ?type2 ?time2 ?output2 uk $?inlist2)
  (test (member ?output2 $?inlist1))
  (test (member ?output1 $?inlist2)) =>
;
; actions
;
        (assert (tightly-coupled ?name1 ?name2)))
```

2.3.3 Inference Engine

The inference engine applies the knowledge (rules) to the data (facts) by pattern matching the facts in the facts-list against the conditions of the rule. The basic execution cycle begins by examining the KB to determine if the conditions of any rules have been met. All rules with currently met conditions are placed on the agenda, which is essentially a push down stack. After the agenda has been completed, the rule on the top of the stack is selected and its action(s) are executed. As a result of the action(s) of the rule, new rules may be placed on the agenda, and rules already on the agenda may be removed. This cycle repeats until all rules that can be executed have done so. DeMAID/GA passes control to CLIPS for execution of the inference engine, and CLIPS returns control back to DeMAID/GA after all rules have been executed.

3. INPUT DATA

The input file consists of lists where each list contains one or more items enclosed in parentheses. The input data for two sample problems ("conceptual_design" and "test") are listed in section 5. The first field in each list designates the type of list for the inference engine. Fields marked in **boldface** are not to be changed by the user. The data in the input file are case-sensitive. Fields are blank-delimited (tabs are not allowed). Some of the lists defined below are required (section 3.1), while others are optional (section 3.2). The lists can appear in the input file in any order.

3.1 REQUIRED INPUT

The following lists are required as input to DeMAID/GA**:**

3.1.1 (**title** title-name)

title-name - a name that describes the problem, no spaces are allowed in the title.

Example: (title conceptual_design)

3.1.2 (**maximum** max-number)

max-number -a number that represents the total number of modules.

(Note: maximum number of modules on a Mac is 85 and 200 on a UNIX computer.)

Example: (maximum 25)

3.1.3 (**module** number name type time output-name **uk** input-list)

(sections 5.1.1 and 5.2.1 contain sample problems)

number - a unique number for each module (beginning with 1 and numbered consecutively).

name - a unique name that describes the module.

type - this item can have two meanings. If the **(cost)** list (section 3.2.1) is part of the input file, then an estimate of how much the module will cost to complete execution can be placed in the field. Otherwise, the filed defines the type of function, where, the type is 4 for an objective function, 3 for a design variable, 2 for a behavior variable, and 1 for a constraint function. If neither cost nor type of function is desired, place a 0 (zero) in the field as a place holder.

time - an estimate of how long the module takes to complete execution.

output-name - a unique name that describes the output of a module. Regardless of whether the output is a single value or many values, this field is given a single, unique name. (Sometimes a module must be added to keep modules from being removed from the input file because the *output-name* is not a member of an *input-list*. This module will have as its *input-list* the *output-names* of all modules with an *output-name* not used in an *input-list*. The term “goal” must be in lower case and is the *output-name* for this module.)

uk - stands for unknown. This item is a status marker for the inference engine and is changed internally by the KB.

input-list - a blank-delimited list of the input requirements for a module. If a module requires no input from within the system (i.e., the input is from an external source or the module is for initialization purposes), then the term “no-input” is used. This term must be in lower case and must have the hyphen.

Examples: (module 1 init 5 25 data uk no-input)
(module 2 structures 10 20 feout uk dv bv cons data)
(module 29 final 30 10 goal uk feout matout)

3.2 OPTIONAL INPUT

Several optional lists can be added to the input file to take advantage of different capabilities available in DeMAID/GA. These lists include

3.2.1 (**cost**)

If this list is used, then the value in the type field of the **module** list is an estimate of how much the module will cost to complete execution.

3.2.2 (**strength uk** coupling-strength output-name input-name)

(section 5.1.2 contains a sample problem)

uk - stands for unknown. This filed is a status marker for the inference engine and is changed internally by the KB.

coupling-strength - defines the coupling strength of the *input-name* field to the *output-name* field. Seven choices are available for *coupling-strength*: ew - extremely weak, vw - very weak, w - weak, n - nominal, s - strong, vs - very strong, and es - extremely strong. The strength can be based on user knowledge of the problem or defined by a normalized sensitivity number. The sensitivity number is converted into a coupling strength in the **Plan** function (section 4.2).

output-name - the name of the output that is generated by a module using *input-name*.

input-name - the name of an item in the *input-list* field for the module that generates *output-name*.

Examples: (strength uk vw disp dv)
(strength uk .235 disp dv)

3.2.3 (**override** coupling-strength iterations)

This list, in conjunction with **strength** lists, aids in determining the cost and/or time in iterative subcycles. An assumed relationship exists between *coupling-strength* and the number of iterations required for the convergence of an iterative subcycle. The defaults are as follows: 2 iterations for an extremely weak coupling strength; 3 iterations for a very weak coupling strength; 4 iterations for a weak coupling strength; 5 iterations for a nominal coupling strength; 6 iterations for a strong coupling strength; 7 iterations for a very strong coupling strength; and 8 iterations for an extremely strong coupling strength. If a different

relationship is preferred, then this list can be used to override the defaults.

coupling-strength - defines the coupling strength of an *input-name* item to the *output-nam*e item. Seven choices are available for *coupling-strength* (section 3.2.2).

iterations - the number of iterations required for convergence in relation to a particular *coupling-strength*. Do not use 1 as an override value.

Example: (override vw 5)

changes the number of iterations required to converge a subcycle associated with a very weak (vw) feedback coupling from 3 iterations to 5 iterations.

3.2.4 (**output** output-name component-list)

If this list is used, then the *output-name* of a **module** list (section 3.1.3) is divided into a blank-delimited *component-list*. If the *output-name* for a module is *goal*, then the list is (**output** goal no-output). When this list is used, the **module** list terminates after the **uk** item. The DeMAID/GA software builds the *input-list* for a module list from the **output** and **requires** lists (section 3.2.5).

output -name - a unique name that describes the output of a module.

component-list - names of the various components of *output-name*.

Example: the output name, feout, that is created by the module structures in the list

(module 1 structures 10 20 feout uk dv bv cons data)

could be divided into three items, disp, stress, and buckling with the list:

(output feout disp stress buckling)

3.2.5 (**requires uk** coupling-strength outputi inputi)

This list is used in conjunction with the **output** list (section 3.2.4). If this list is used, then the **strength** lists (section 3.2.2) must be omitted. DeMAID/GA generates the **strength** lists from the **requires** lists.

uk - stands for unknown. This item is a status marker for the inference engine and is changed internally by the KB.

coupling-strength - defines the coupling strength of the *inputi* item to the *outputi* item. Seven choices are available for *coupling-strength* (section 3.2.2). If more than one **requires** list exists between two modules, then the maximum *coupling-strength* is retained in generating the **strength** list. If the *coupling-strength* is not known, then use n (nominal). If sensitivity data are available, then the coupling strength is replaced by a number that represents the normalized sensitivity. This number is converted to a coupling strength (section 4.3.3).

outputi - the name of a component of the *output-name* generated by a module list that contains the *inputi* item.

inputi - the name of a component of an item in the *input-list* field to the module list that generates *output-name*.

Examples: given two modules

(module 1 initial 5 15 dv uk no-input)
(module 2 structures 10 20 feout uk dv bv cons data)

with the output divided by the lists

(output dv dv1 dv2 dv3 dv4)
(output feout disp stress buckling)

then either of the two following lists provide the coupling strengths

(requires uk vw disp dv1)
(requires uk .235 disp dv1)

3.2.6 (**sensdata** ncoupling k1 k2)

For this list to be used, sensitivity data must have been computed external to DeMAID/GA. This list is an input when converting sensitivity data (section 4.3.3) to coupling strengths [5].

ncoupling - number of couplings in the problem.

k1, k2 - user-defined values based on experience for computing the upper and lower bounds of the local normalized sensitivity space in terms of the statistical mean value and standard deviation.

Example: (sensdata 24 26.375 4.75)

4. PROGRAM FUNCTIONS

The functions of DeMAID/GA are described in the following sections. Each of the major functions is called via a menu.

4.1 FILE MENU

The File function includes three selections: Save Facts (section 4.1.1), Print (section 4.1.2), and Quit (section 4.1.3).

4.1.1 Save Facts

The Save Facts selection allows the user to save the current facts (from the "disp.out" file) to a user-specified filename. This selection is particularly useful if the user has made changes to the DSM with the **DSM** function.

4.1.2 Print

The Print selection allows the user to print the current DSM. Before printing, the DSM must be displayed by the **DSM** function.

4.1.3 Quit

The Quit selection exits DeMAID/GA.

4.2 PLAN MENU

The **Plan** function provides three selections: First Pass (section 4.2.1), Preplan (section 4.2.2), and Sensitivity (section 4.2.3). Flowcharts of the **Plan** function are shown in figures 3a and 3b. The menu selections are shown in the rectangles. The rule files all have a ".bin" suffix. The input file (section 3) to all selections is the "file.npt" file and all selections create an output file. If sensitivities are available (section 3.2.6), that selection creates the "file.sens" file which can be input to either the Preplan or the First Pass selection. If the output has been divided (section 3.2.4), then Preplan is selected and it creates the "file.preplan" file which is input to the First Pass selection. If sensitivities are not available, then the "file.npt" file is input to Preplan. If neither the Sensitivity nor the Preplan selection is chosen, the "file.npt" file is input to the First pass selection. The Preplan selection and the First Pass selection both create log files, "preplan.log" and "plan.log" respectively.

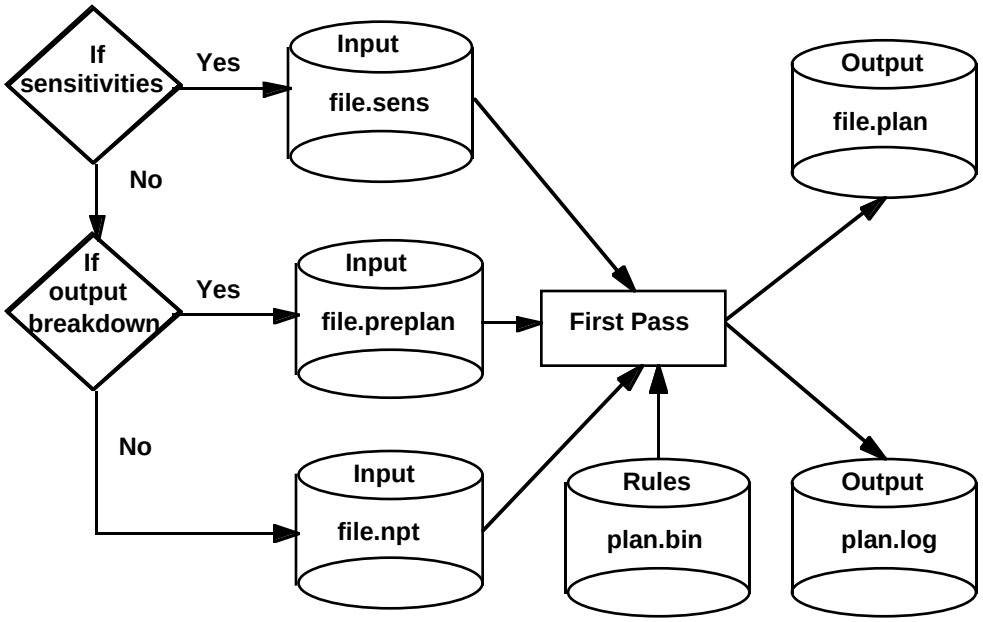

Figure 3a. Flowchart of First Pass selection of the Plan function.

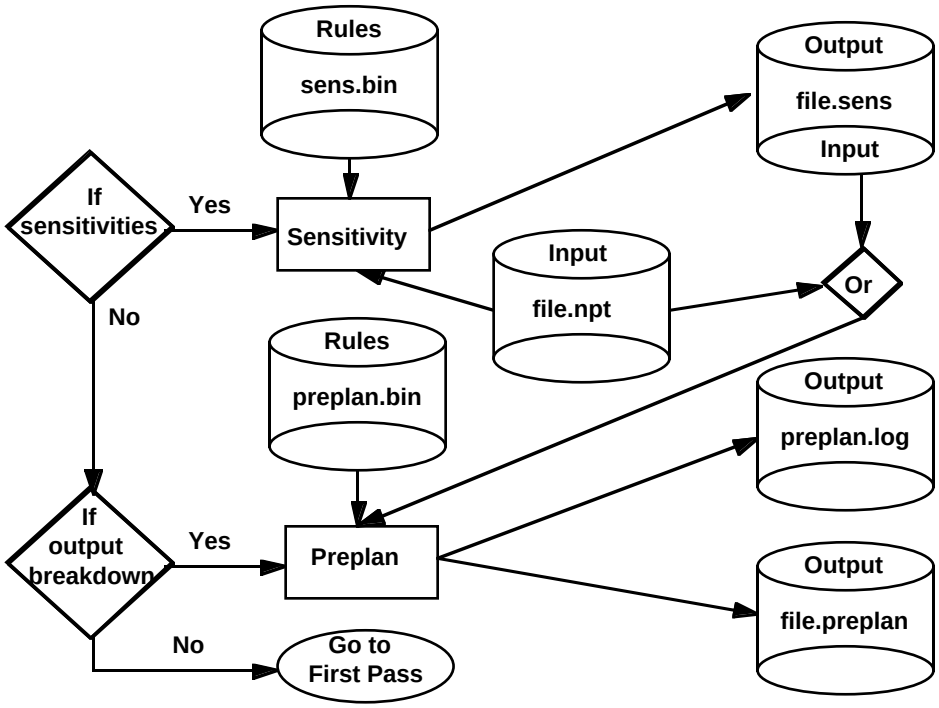

Figure 3b. Flowchart of Sensitivity and Preplan selections of the Plan function.

4.2.1 First Pass

Input file - user-defined filename, created by user
Output file - user-defined filename, input to the **Schedule** function
Rule file - plan.bin
Log file - plan.log

In the First Pass selection, the rule file, plan.bin, is loaded into the KB. Input data are read from a user-defined filename (section 4.2) and asserted as facts into the KB. The inference engine is executed.

The first function in the First Pass selection is to determine those modules that contribute to the solution of the problem. This function is accomplished by checking the output-name of each module in the modules list (section 3.1.3) against the input-list items of the other modules. If the output-name of the module is an item in the input-list of at least one other module, then that module contributes to the solution of the problem and is retained in the" file.plan" file. If a module is not a contributor, then it is removed from the list of modules in the "file.plan" file and a message is printed to notify the user.

The second function is the determination whether any two modules have the same output-name. If two modules do have the same output-name, then a message is printed notifying the user to correct the error and DeMAID/GA stops. The user must either remove one of the modules or change the output-name for

one of the modules. If the output-name is changed, then the effected input-lists of other modules must reflect that change.

Next, the input-lists for all modules are examined to determine whether all input items of the list are satisfied by the output-name from other modules. Modules with “no-input” as an input-list usually represent data initialization or are satisfied by external inputs; therefore these modules are not checked. If an item in an input-list to a module is not satisfied, then a message is printed notifying the user to correct the error and DeMAID/GA stops. The user must add a new module to the input file to satisfy the input requirement or remove the item from the input-list of the module.

Finally, if coupling strength lists (section 3.2.2) are part of the input file, these lists are checked in a manner similar to that for the module lists. Each coupling strength list is checked against the module lists; and if no match exists (either input or output), then that particular coupling strength list is removed and a message is printed to notify the user.

The “file.plan” created by the **Plan** function is slightly different from the original input file. Two new fields (both containing “null”) have been added between the status filed and the input-list fields for use in the **Schedule** function. The new format is

(module 10 TASKC15 1 0 G015 uk null null DV06 DV07 DV08 DV23)

In addition, if one or more modules are removed by the **Plan** function, the modules are renumbered to maintain a continuous numbering sequence. The maximum number of modules in the maximum list (section 3.1.2) is modified to reflect the change.

4.2.2 Preplan

Input file - user-defined filename, created by user
Output file - user-defined filename, input to First Pass selection of the **Plan** function
Rule file - preplan.bin
Log file - preplan.log

Preplan is selected when the user has broken down the output name (section 3.2.4) in a module list (section 3.1.3) with an output list (section 3.2.4). The format for a module list when Preplan is chosen is

(module number name type time output-name uk)

Notice that the input-list following the “uk” filed is missing. The input-list is constructed from the Preplan rules in the “preplan.bin” file.

This selection also needs the requires lists (section 3.2.5). As an option, the user can also select coupling strengths with the requires lists. If no coupling strengths (section 3.2.2) are available, then a nominal strength should be used.

The Preplan rules determine whether the input-name of each requires list appears as an output-name in another requires list. Both the input and output of all requires lists are checked to determine if they are members of the output item list. If any list fails the test, the user is notified and DeMAID/GA stops.

The requires and output lists are used to build the input-list of a module. The requires lists are also used to build the strength lists. Because the output-name has been broken down in the output lists, more than one coupling strength may exist between the same two modules. The Preplan rules select the strongest coupling strength for the strength list.

4.3.3 Sensitivity

Input file - user-defined filename, created by user
Output file - user-defined filename, input to First Pass or Preplan selection of the **Plan** function
Rule file - sens.bin
Log file - none

The Sensitivity selection [5] allows the user to compute coupling strengths from normalized sensitivity data. A linear distribution between upper and lower bounds of the local sensitivity space is used to quantify the seven levels of coupling strengths (section 3.2.2). In this approach the mean and standard deviation are calculated where the mean value of the local normalized sensitivity derivatives (s_i) can be determined from

$$\bar{s} = \frac{1}{N}\sum_{i=1}^{N} s_i$$

where N is the number of couplings. The associated standard deviation can now be determined from the relation,

$$\sigma(s) = \left[\frac{1}{N-1}\sum_{i=1}^{N}\left(s_i - \bar{s}\right)^2\right]^{1/2}$$

The upper and lower bounds of the local normalized sensitivity space are defined in terms of the mean value and standard deviation as,

$$s^u = \bar{s} + k_1\sigma(s)$$
$$s^l = \bar{s} - k_2\sigma(s)$$

where k_1 and k_2 are user-prescribed values based on experience and heuristics. Anything outside these bounds is either extremely weak (ew) or extremely strong (es). The sensitivity rules read the requires (section 3.2.5) or strength lists (section 3.2.2), which contain sensitivity data in the strength field. A statistical analysis is performed on the sensitivity data to determine the ranges for the seven levels of coupling strengths. Based on where in the range their values fall, the sensitivity data are replaced by a coupling strengths.

4.3 SCHEDULE MENU

The **Schedule** function contains three selections: I/O Schedule (section 4.3.1), Parallel Schedule (section 4.3.2), and Skip Schedule (section 4.3.3). A flowchart of the **Schedule** function is shown in figure 4. The menu selections are in the rectangles. The rule files all have the ".bin" suffix. The input file to all selections is the "file.plan" file and all selections create a "file.sked" file as output. The I/O schedule selection also creates a "sked.log" file.

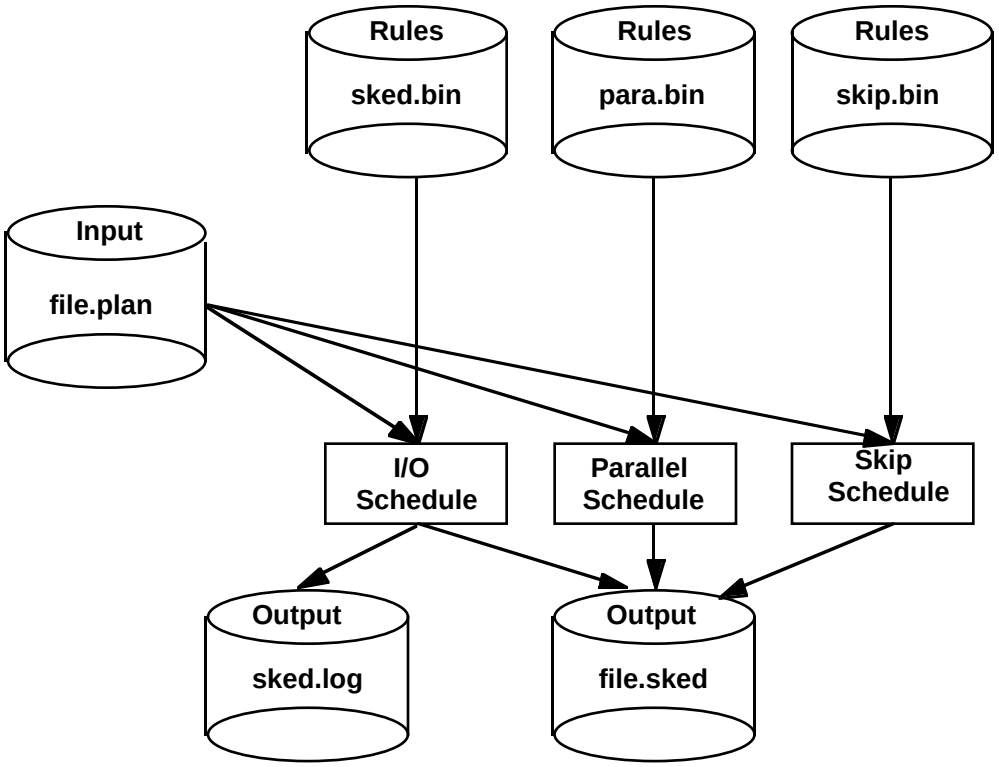

Figure 4. Flowchart of Schedule function.

After a selection has been made for the **Schedule** function, two new fields are created in the module lists (section 3.1.3) in the "file.sked" file that are not found in the original data or the :file.plan" file from the **Plan** function. One field (field 3) contains the original number of the module. Field 2 now contains the current module number found by reordering the modules in the **Schedule** function. The second field (field 9) contains a number that indicates the circuit in which the module is contained. This circuit number is the number of the first module contained in that circuit.

Example:

After scheduling, module 10 becomes module 7 located in circuit 3.

Before: (module 10 TASKC15 1 0 G015 uk DV06 DV07 DV08 DV23)

After: (module 7 10 TASKC15 1 0 G015 uk 3 DV06 DV07 DV08 DV23)

4.3.1 I/O Schedule

Input file - created by First Pass selection in the **Plan** function
Output file - user-defined filename, input to the **DSM** function
Rule file - sked.bin
Log file - sked.log

If the user selects the I/O Schedule, then the **Schedule** function reorders the modules based on their couplings. If the modules and their couplings are placed into the DSM without regard to the ordering, then little information regarding the desirable structure of the design process is available to the design manager because the modules are most likely disorganized and contain a substantial number of feedback couplings. The feedback couplings among the modules are eliminated by examining the couplings and moving modules along the diagonal to convert feedback couplings into feedforward couplings. If any feedback couplings remain, the coupled modules are partitioned into iterative subcycles called circuits. The I/O schedule selection no longer orders the modules within the circuits but continues to order the circuits within the design process. Steward [3] implements the partitioning into circuits with matrix manipulations; DeMAID/GA follows the same steps as Steward but replaces the matrix manipulations for partitioning by applying rules from the “sked.bin” file.

The DSM’s for the “conceptual_design” problem and the “test” problem defined in section 5 are shown in figures 5 and 6 respectively. These DSM’s are the results of I/O scheduling.

Label	Time	Cost
DYNMODL	30	30
STDMOCH	40	20
STRMODL	10	50
HANDQUL	10	50
STRMODE	10	50
GEOMDEV	50	10
AROSRVO	40	20
STRDYNA	50	10
CSYSANL	20	40
FAEROCH	20	40
INITDAT	40	20
RYSEDAT	30	30
MISPERF	30	30
VEHPERF	20	40
RAEROCH	30	30
AEROANL	20	40
PRESDEF	30	30
STRANAL	40	20
STRCTWT	50	10
WIANAL	40	20
AEROMDL	20	40
FINLDAT	20	40

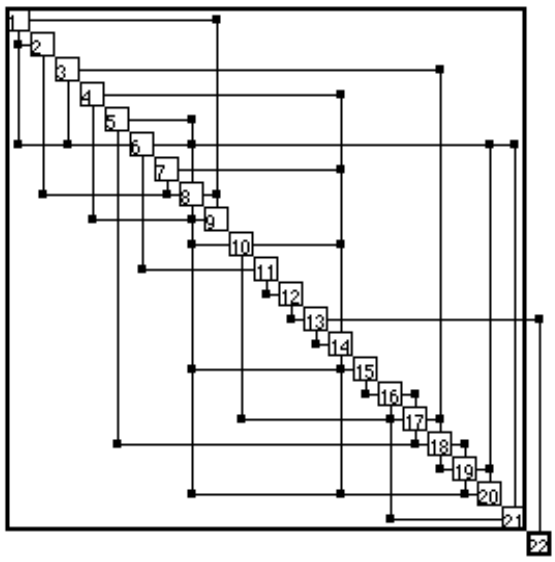

Figure 5. “conceptual_design” problem DSM.

The “conceptual_design” problem (section 5.1) has one circuit. Each module has an associated cost and time.

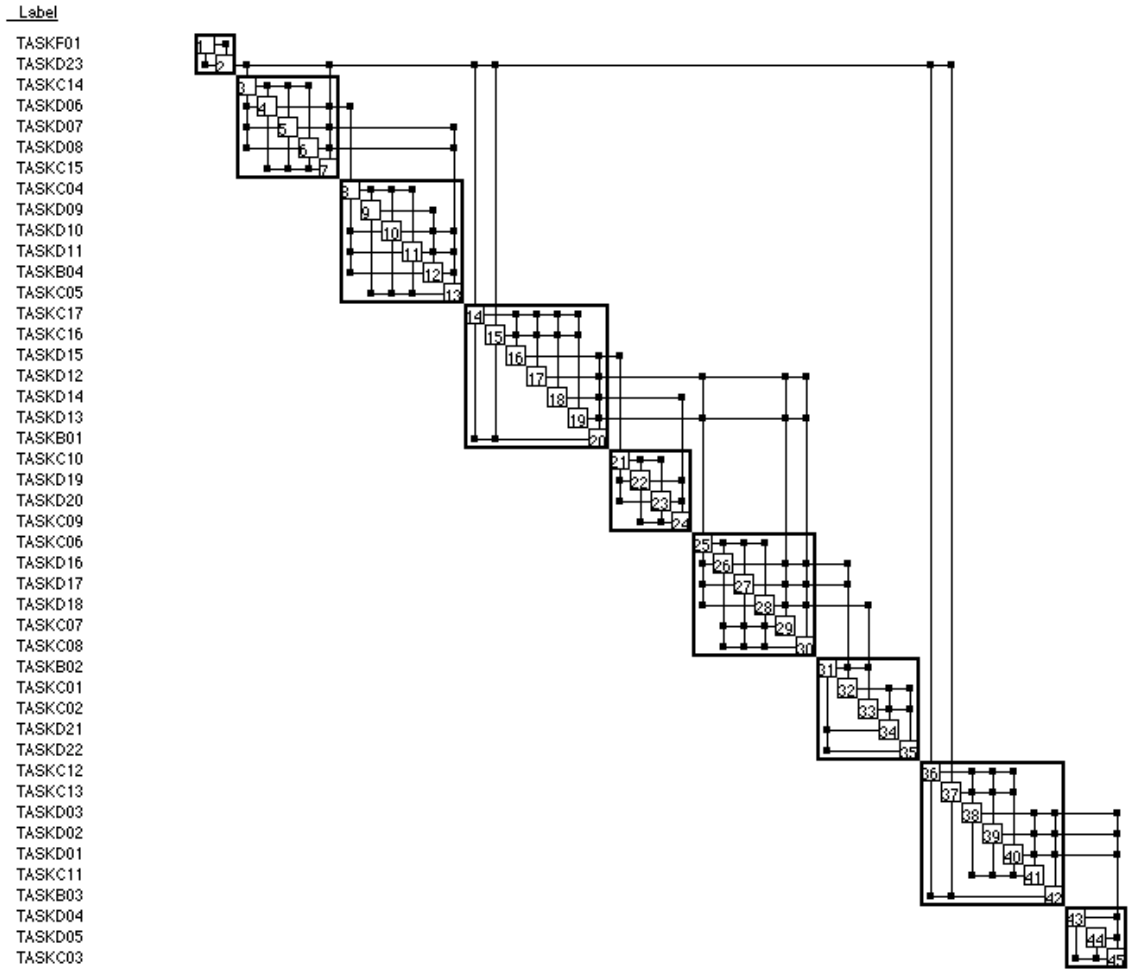

Figure 6. “test” problem DSM.

The “test” problem (section 5.2) has nine circuits. No costs or times are associated with this problem. The type field in the module list (section 3.1.3) defines the design elements: constraints; behavior variables; design variables; and objective function.

4.3.2 Parallel Schedule

Input file - created by First Pass in the **Plan** function
Output file - user-defined filename, input to the **DSM** function
Rule file - para.bin
Log file - none

Parallel Schedule is selected if the user wishes to examine the potential gains from parallel processing. When scheduling by parallel requirements, the user must provide the number of available processors and each module list (section 3.1.3) in the input file must have a time associated with it. A window appears to request input for the number of processors. For parallel scheduling, the circuit boxes represent the processors and contain the modules to be executed on that processor. The rules in the "para.bin" file compute the time required to execute all modules in sequence and divides that time by the number of processors available, which yields an average time per processor.

Ideally, all processors would complete processing at the same time; for this reason, each processor is assigned the average time as a starting point. DeMAID/GA begins assigning modules to the processors; those modules that require the most time are assigned first. The module times are then subtracted from the time available on a given processor. The remaining time slots are assigned by determining the module with the maximum time that is less than the remaining time available on a given processor. This process continues until all modules have been placed or until no time slot is available on any processor.

If no time slots are available and a module has not yet been assigned to a processor, then the module time is divided by the number of processors available and added to the average time of the processors. Then the placement process is repeated. After all modules have been placed, the parallel scheduling function is complete. DeMAID/GA lists the number of iterations with parallel processing that can be completed before the time is equal to the sequential time. This information can be used to examine the trade-offs between sequential and parallel processing but is of no use unless the module times are known.

4.3.3 Skip Schedule

Input file - created by First Pass in the **Plan** function
Output file - user-defined filename, input to the **DSM** function
Rule file - skip.bin
Log file - none

The Skip Schedule selection is chosen when the user wishes to skip the **Schedule** function and go directly to the **DSM** function. The rules in the "skip.bin" file convert the file output from the **Plan** function into the new format (section 4.3) for input to the **DSM** function.

4.4 DSM MENU

Input file - first call uses a file created by a selection in the **Schedule** function, and subsequent calls read file disp.out
Output file - disp.out
Rule file - disp.bin
Log file - none

A flowchart of the **DSM** function is shown in figure 7. There are several options available in this function: Display DSM (section 4.4.1); Labels & Titles (section 4.4.2); Scale (section 4.4.3); Delete a Coupling (section 4.4.4); List Module Info (section 4.4.5); List Coupling Info (section 4.4.6); Edit order (section 4.4.7); Reverse Display (section 4.4.8); and Exit DSM (section 4.4.9).

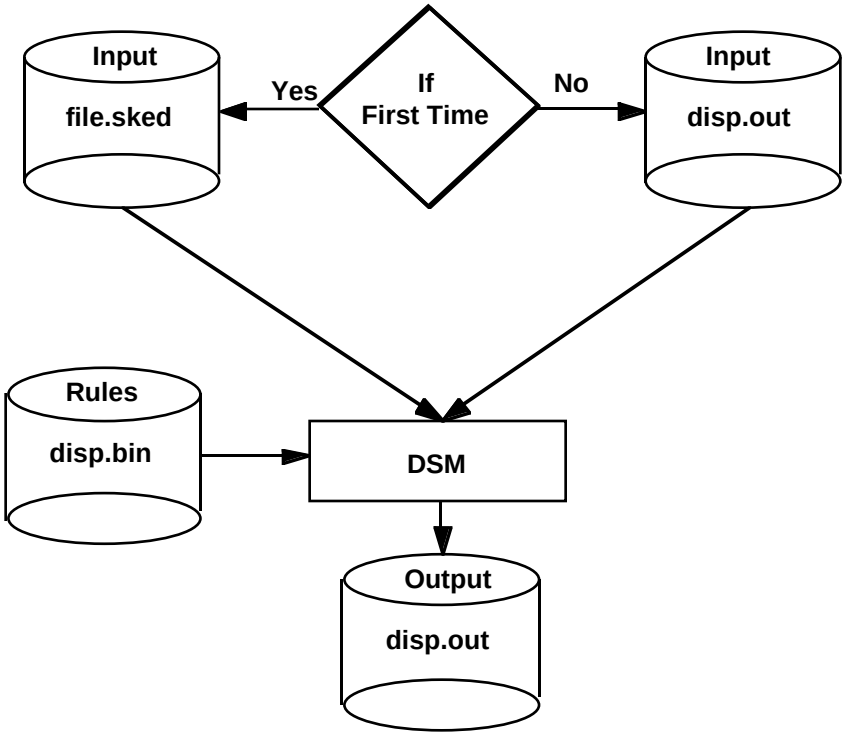

Figure 7. Flowchart of DSM function.

4.4.1 Display DSM

The user must select Display DSM before selecting any other options in this menu to load the rules in the "disp.bin" file. If the **DSM** function is reentered after Exit DSM (section 4.4.9) has been selected, then Display DSM must be selected again. Display DSM displays the DSM. Some DSM's may not fit in the display window. If necessary, the user can enlarge the window by dragging the bottom right corner outward, or scrolling through the window using the arrow keys on a Mac or the scroll bars on a UNIX computer.

Important: The user **must** select Exit DSM (section 4.4.9) before proceeding to other functions outside the **DSM** function.

4.4.2 Labels & Titles

Labels & Titles opens a window shown in figure 8 which allows the user to select the font (default is Helvetica) and size (default is 24) for the title, and the font (default is Helvetica) for the labels.

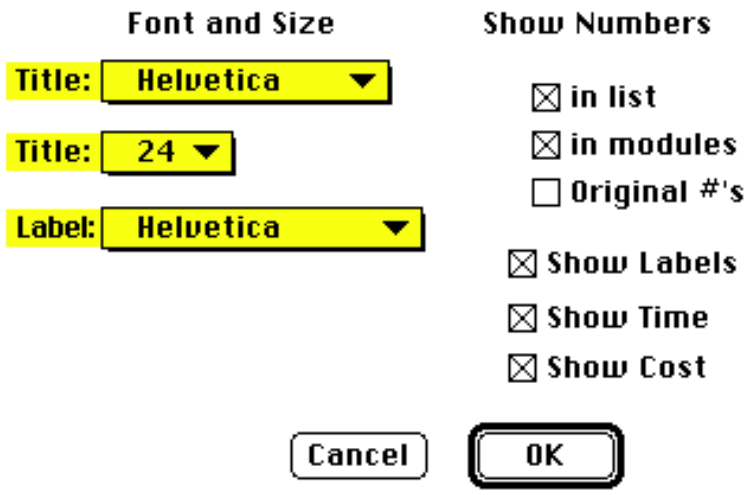

Figure 8. Labels & Titles window.

The user can select the module information to be displayed with the DSM by "clicking" on selected boxes. An "**X**" means the selection has been activated. These options allow the user to display the numbers (both in the list down the side and in the modules on the diagonal), the labels, the time, and the cost. The Original #'s box can be selected to display the original module numbers (section 4.3) in the list down the side. This box is not selected initially.

4.4.3 Scale

Scale allows the user to choose one of four different sizes from a submenu to display the DSM. The four choices are tiny, small, normal (default), and large. If "tiny" is selected, much of the display information is not available for display.

4.4.4 Delete a Coupling

Delete a Coupling allows the user to delete a specific coupling between two modules. Windows open to allow the user to enter the module numbers of the "from" and "to" modules. The coupling is deleted from the facts list, and the DSM is redrawn without it. A listing that describes the deleted coupling is displayed in the console window.

4.4.5 List Module Info

List Module Info allows the user to list information about a specific module in the console window. A window opens so that the user can enter the module number. The name, time, cost, output name, and couplings (both to and from the module) are shown in the console window.

4.4.6 List Coupling Info

List Coupling Info allows the user to display information about a specific coupling between two modules in the console window. Windows open so that the user can enter the number of the "from" and "to" modules. The name of the coupling and the two modules are shown in the console window.

4.4.7 Edit Order

Edit Order opens the window shown in figure 9 to allow the user to interactively move modules around in the DSM.

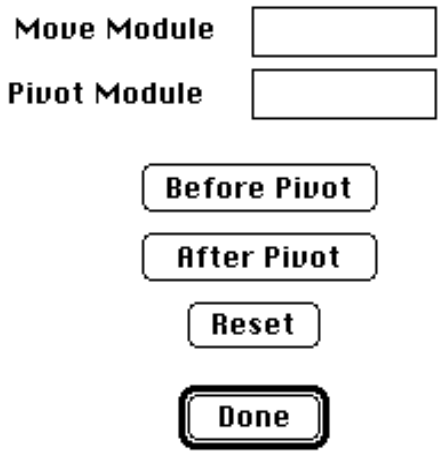

Figure 9. Edit Order window.

The user selects a module to move and a pivot module by typing the module numbers in the appropriate boxes. The user can place the "move" module before or after the "pivot" module by "clicking" the appropriate button. The "Reset" button restores all modules to their original position in the DSM. When the user is finished moving modules, the "Done" button is selected. The DSM is redrawn with the modules in their new positions.

The module numbers in the display do not change to reflect the new sequence. Upon exiting the **DSM** function, module numbers written to the “disp.out” file reflect all changes made within the function. Therefore, if the **DSM** function is reentered, the process boxes along the diagonal will be in the same order as when the function was exited, but the module numbering sequence will have changed.

4.4.8 Reverse Display

Reverse Display changes the DSM display so that the feedback couplings are in the upper triangle of the DSM and the feedforward couplings are in the lower triangle of the DSM. There is a “check” mark by Reverse Display in the menu to indicate when the DSM is displayed in the reverse format. This display format is used by Steward [3]. Selecting this option a second time reverses the display back to the original format. The reverse display for the “conceptual_design” problem DSM (figure 5) is shown in figure 10.

Label	Time	Cost
DYNMODL	30	30
STDMOCH	40	20
STRMODL	10	50
HANDQUL	10	50
STRMODE	10	50
GEOMDEV	50	10
AROSRVO	40	20
STRDYNA	50	10
CSYSANL	20	40
FAEROCH	20	40
INITDAT	40	20
RYSEDAT	30	30
MISPERF	30	30
VEHPERF	20	40
RAEROCH	30	30
AEROANL	20	40
PRESDEF	30	30
STRANAL	40	20
STRCTWT	50	10
WIANAL	40	20
AEROMDL	20	40
FINLDAT	20	40

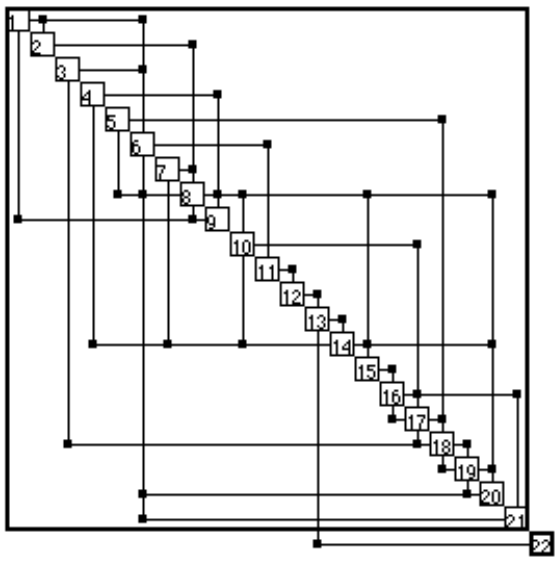

Figure 10. Reverse DSM for the “conceptual_design” problem.

4.4.9 Exit DSM

The user **must** select Exit DSM to close the “disp.bin” file, clear the KB, and save the modified fact-list on the “disp.out” file before proceeding to other functions.

4.5 OPTIMIZE MENU

Input file - disp.out created in the **DSM** function
Output file - modified disp.out file
Rule file - ga.bin
Log file - none

The **Optimize** function is selected to optimize the ordering of the modules within the circuits with a GA. The user should become familiar with GA operations and terminology, such as population, selection, crossover, and mutation [7,8,9]. A flowchart of the **Optimize** function is shown in figure 11. The menu selection is in the rectangular block.

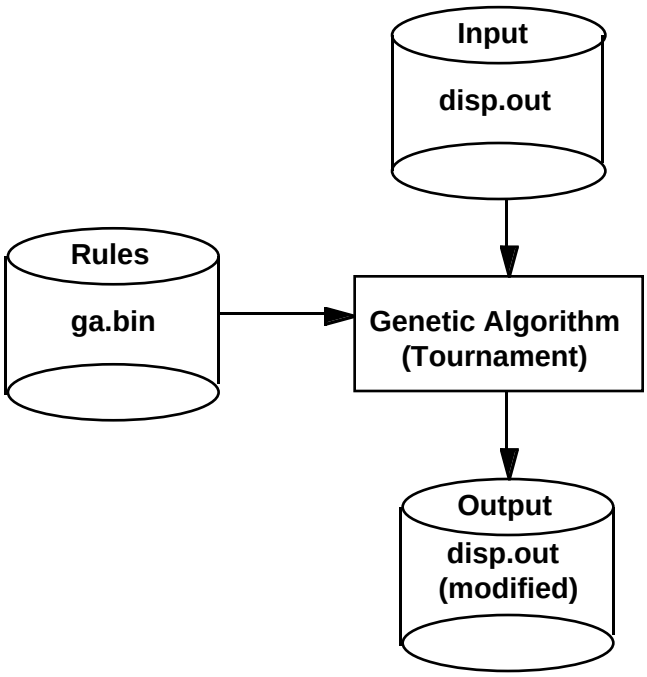

Figure 11. Flowchart of the Optimize function.

4.5.1 Tournament

Currently, only one GA method, tournament, is available in DeMAID/GA. Others may be added at a later date. Each circuit is passed to the GA to optimize individually. A window (figure 12) is displayed for each circuit. The default values for the GA are in parentheses in the parameter descriptions below. The user can change the default parameters by typing new numbers in the appropriate boxes. “Click” the OK box when the parameters have been selected.

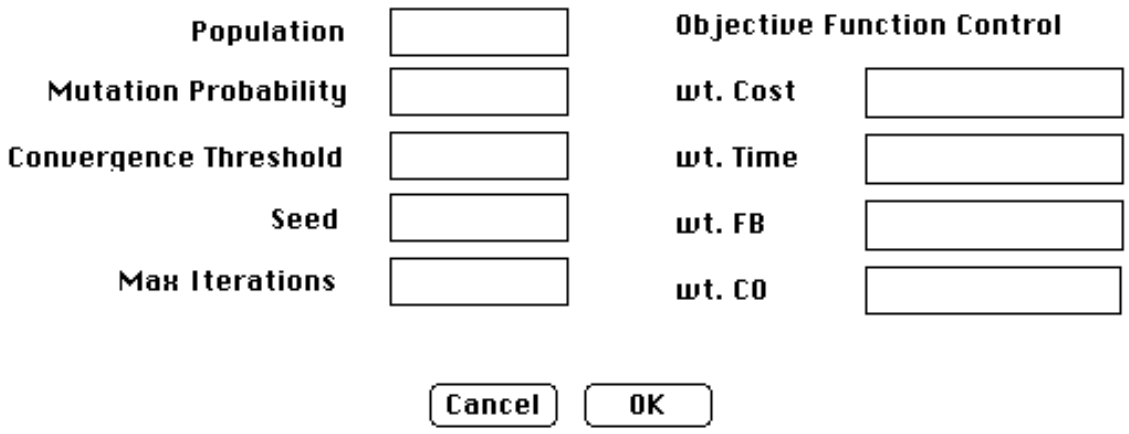

Figure 12. Window for setting GA parameters.

- Population (100) - population size
- Mutation Probability (1.0) - mutation probability in percent, default is 1%
- Convergence Threshold (0.9) - a converged population is one for which the average fitness is at least the “Convergence Threshold” of the best fitness seen so far (default is 90%)
- Seed (3818969) - seed for random number generator
- Max Iterations (500) - maximum number of iterations to determine the best sequence
- wt. Cost (1.0) - cost weight
- wt. Time (1.0) - time weight
- wt. FB (1.0) - feedback weight
- wt. CO (1.0) - crossover weight

The population consists of members which are different combinations of ordering sequences of the modules. The GA begins with a randomly generated initial population of a size determined by the user and proceeds from generation to generation by applying the three main GA operations - selection, mutation, and crossover. For the tournament method, two members are randomly chosen from the current population. The fitness levels of each of the selected members are compared and the member with the best fitness level is

added to the mating pool. In addition to minimizing the number of feedback couplings and crossovers, the fitness function for the GA in DeMAID/GA determines the minimum cost and time required for the convergence of each circuit. The GA sums the time and cost of each process contained in a feedback loop and multiplies those sums by the iteration factor (section 3.2.3) for the feedback to obtain the total cost and time to converge a circuit. The user-definable weights determine the relative importance of each major component of the fitness function. The fitness function in DeMAID/GA is

$$fitness=1.0/((wf*f+wc*c+wtime*time+wcost*cost)^{**}4)$$

where f is the number of feedback couplings, c is the number of crossovers, time is the total time required to converge the circuit, cost is the total cost to converge the circuit; and wf, wc, wtime, and wcost are user-definable weights. For the simple tournament selection, the relative scale of this fitness function is unimportant; Only the relation of the values (i.e. whether one fitness function is larger than the other) is important.

Convergence is achieved when the average fitness of a population rises above the user-defined percentage (convergence threshold) of the best fitness for that population. At that point, the member of the population with the best fitness is considered as the optimal. That sequence is written to the “disp.out” file. After the GA has completed reordering all the circuits, the **DSM** function can be reentered to display a DSM with the optimal ordering. The optimized DSM for the “conceptual_design” problem (figure 5) is shown in figure 13.

Label	Time	Cost
INITDAT	40	20
GEOMDEV	50	10
STRMODL	10	50
AEROMDL	20	40
AEROANL	20	40
PRESDEF	30	30
STRANAL	40	20
STRCTWT	50	10
WIANAL	40	20
STRMODE	10	50
RYSEDAT	30	30
RAEROCH	30	30
FAEROCH	20	40
STRDYNA	50	10
STDMOCH	40	20
DYNMODL	30	30
CSYSANL	20	40
HANDQUL	10	50
AROSRYO	40	20
VEHPERF	20	40
MISPERF	30	30
FINLDAT	20	40

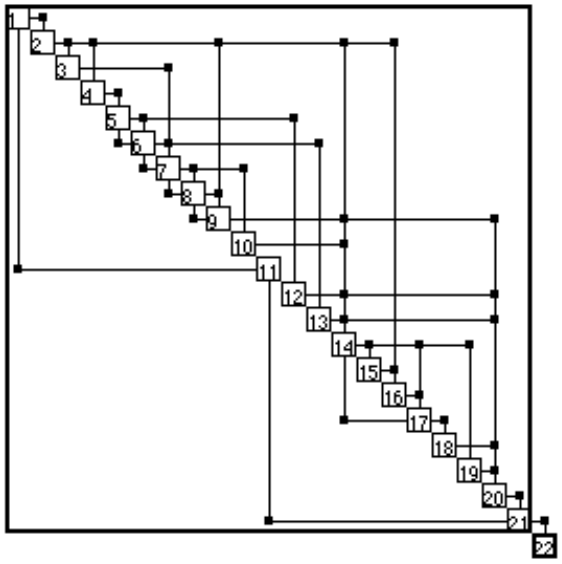

Figure 13. “conceptual_design” problem DSM after GA reordering.

4.6 FUNCTIONS MENU

Functions allows the user to select from among nine different functions to obtain more detailed information about the design project. The “disp.out” file must be available for these functions to execute. These functions include: Decomposition (section 4.6.1); Dependency Matrix (section 4.6.2); Trace Range (section 4.6.3.1); Trace Forward (section 4.6.3.2); Trace Back (section 4.6.3.3); Decompose Circuit (section 4.6.4); Coupling Strengths (section 4.6.5); Cost and Time (section 4.6.6); and User Function (section 4.6.7).

A flowchart for three of the functions is shown in figure 14. The menu selections are in the rectangular blocks.

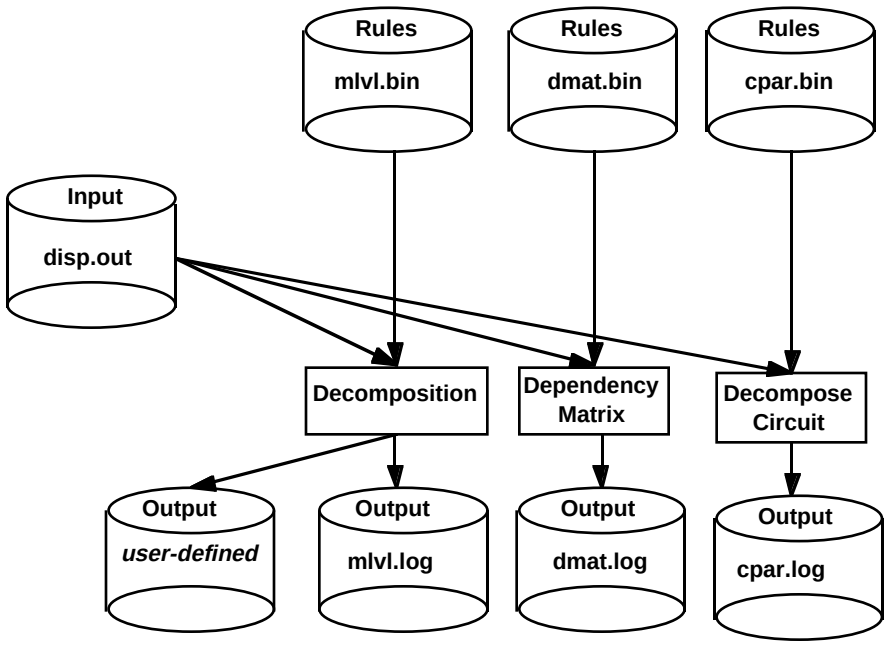

Figure 14. Flowchart of the Decomposition, Dependency Matrix, and Decompose Circuit selections.

4.6.1 Decomposition

Input file - disp.out
Output file - user-defined filename, list of circuits for input to DeMAID/GA
Rule file - mlvl.bin
Log file - mlvl.log

Decomposition allows the user to examine the circuits to determine where parallel processing is possible. Once a circuit has been found it can be treated as a single process. No feedback couplings exist among the circuits; therefore, no iteration occurs among the circuits. All iterative subcycles are contained within the circuits. Thus, after the circuits have been identified by the **Schedule** function, a multilevel hierarchical organization of the problem can easily be achieved. As circuits with satisfied input requirements are identified, they are placed on a given level of the hierarchy. Each circuit is placed on the level below the lowest level that contains a circuit that generates input for the placed circuit. All circuits on the same level can be executed in parallel. The DSM for the “test” problem shown in figure 6 would have the multilevel display shown in figure 15a. This display is saved in the “mlvl.log” file along with other descriptive data.

Multilevel Display of Circuits

Level 1	**1**
Level 2	**2 8 4**
Level 3	**3 9 5 6**
Level 4	**7**

Figure 15a. Display of the multilevel structure of circuits.

The output file contains the circuits from the DSM of the “test” problem in figure 6 listed as modules (figure 15b). This file can be input to the Plan function (section 4.2.1) to eventually display the circuits in the DSM format.

```
(title test)
(maximum 10)
(module 1 crkt1 0 125 out1 uk no-input)
(module 2 crkt2 0 253 out2 uk out1)
(module 3 crkt3 0 218 out3 uk out2)
(module 4 crkt4 0 289 out4 uk out1)
(module 5 crkt5 0 216 out5 uk out4)
(module 6 crkt6 0 257 out6 uk out4)
(module 7 crkt7 0 161 out7 uk out6)
(module 8 crkt8 0 201 out8 uk out1)
(module 9 crkt9 0 121 out9 uk out8)
(module 10 crkt10 0 0 goal uk out3 out5 out7 out9)
```

Figure 15b. Circuits listed as modules in output file.

4.6.2 Dependency Matrix

Input file - disp.out
Output file - none
Rule file - dmat.bin
Log file - dmat.log

Dependency Matrix builds an ordered matrix that identifies the functional dependence between the constraints and the independent design variables (See Section 3.1.3 to chose the correct type field for this module input.). Behavior variables can be evaluated by using design variables; therefore, each behavior variable can be replaced by a list of independent design variables. Each constraint is examined to determine its dependency on design and behavior variables. Whenever a constraint depends on a behavior variable, the dependency of that behavior variable on the independent design variables is substituted. This produces a rectangular matrix with constraint functions listed row-wise and the independent design variables listed column-wise. A partial dependency matrix for the “test” problem shown in figure 6 is shown in figure 16. An ”**X**” indicates the dependency. The row and column numbers are the module numbers.

		Design Variables						
	Module	2	3	4	10	11	13	14
	7	X	X	X				
	8	X	X	X				
	9		X	X				
Constraints	12	X	X	X	X			
	16	X					X	X
	17	X			X		X	X

Figure 16. Partial dependency matrix for the “test” problem.

For example, in the “test” problem TASKC17 is a constraint function that generates G017 as output. G017 requires (is dependent upon) DV23, a design variable, and BV01, a behavior variable, as input. The behavior variable, BV01, requires (is dependent upon) design variables DV12, DV13, DV14, and DV15. Thus the module generating BV01 would be replaced by the modules generating DV12, DV13, DV14, and DV15 in the dependency matrix and have an “**X**” in the appropriate column. An “**X**” would also appear in the column for the module generating DV23.

Building the dependency matrix reveals dependency patterns that may prove advantageous in the development of multilevel optimization algorithms. The dependency matrix is saved on the “mlvl.log” file.

4.6.3 Trace Changes

The following three menu selections: Trace Range (section 4.6.3.1); Trace Forward (section 4.6.3.2); and Trace Backward (section 4.6.3.3) allow the user to examine the effects of changes made in the design process. Simply because a change is made to one module in the design process, does not mean that all modules must be reexecuted. To start the trace, a window is opened to allow the user to enter the output name of the module involved in the trace. Two modules are involved in the Trace Range option. After the trace is completed, the **DSM** function is reentered and DeMAID/GA highlights (in yellow) the modules that need to be reexecuted. If any module in a circuit is effected by the change, then all modules in that circuit are highlighted. If the user is unsure of the output name of the module to be traced, the List Module Info option (section 4.4.5) in the **DSM** function can be used to identify the module output name.

A flowchart of the Trace Changes selections is shown in figure 17. The menu selections are in the rectangular blocks.

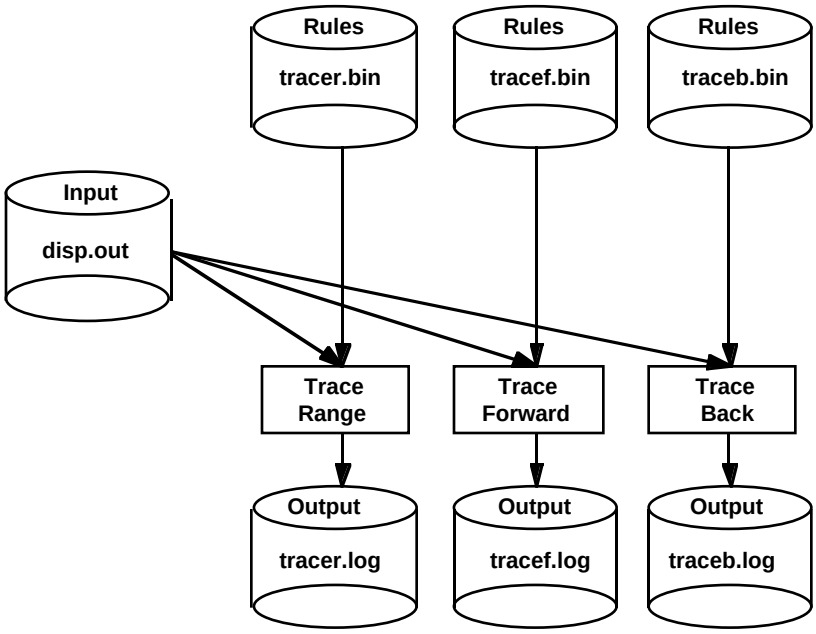

Figure 17. Flowchart of the Trace Changes selections.

4.6.3.1 Trace Range

Input file - disp.out
Output file - none
Rule file - tracer.bin
Log file - tracer.log

T race Range allows the user to trace the effects of a change made to one module on a second module. The user specifies a range of interest providing the output names of the modules that define the range. An example is shown in figure 18. In this figure, a change is made to module 5. The user wants to see the effects on module 29. The affected modules are indicated by the high-lighted boxes. Modules 6-16, and 18 (because they are not affected by the change to module 5); and modules 23, 24, and 26 (because they do not affect module 29) do not need to be reexecuted.

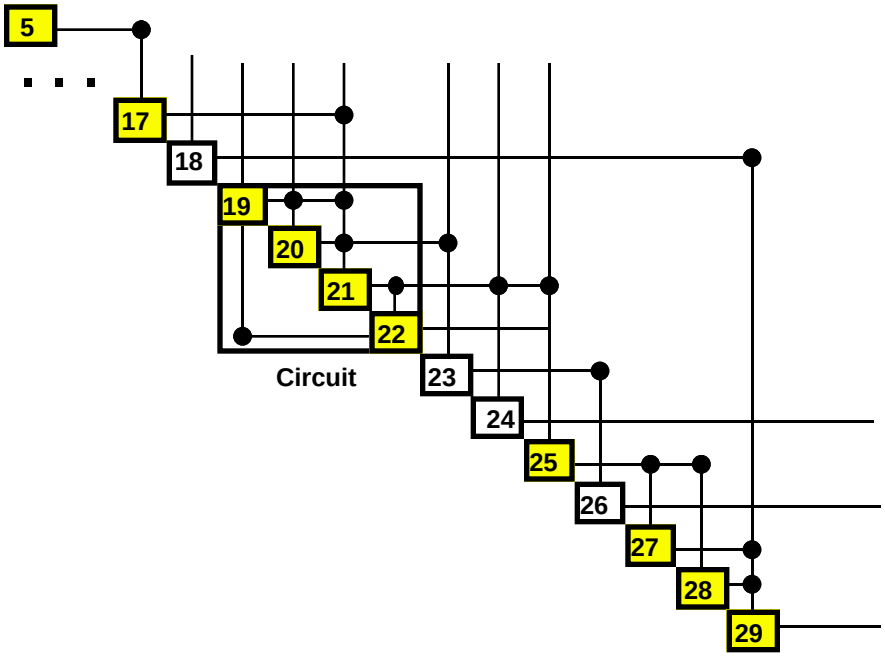

Figure 18. Display of effects of change to module 5 on module 29.

4.6.3.2 Trace Forward

Input file - disp.out
Output file - none
Rule file - tracef.bin
Log file - tracef.log

Trace Forward allows the user to trace the effects of a change made to one module on the modules executed after it. The affected modules are highlighted.

4.6.3.3 Trace Back

Input file - disp.out
Output file - none
Rule file - traceb.bin
Log file - traceb.log

Trace Back allows the user to trace backwards to indicate those modules that affect the given module. The affecting modules are highlighted.

4.6.4 Decompose Circuit

Input file - disp.out
Output file - none
Rule file - cpar.bin
Log file - cpar.log

Many circuits are large, and significant time savings can be achieved by executing some of the modules in parallel. Decompose Circuit assumes that all feedback data are available as estimates. By using this assumption, the rules that apply to circuit decomposition (section 4.6.1) can also be applied here. For example, prior to reordering, the modules in the circuit of the "conceptual_design" DSM in figure 5 decompose into the levels shown in figure 19.

Level 1	**1 2 3 4 5 6 7 10 11 12 13 15 16**
Level 2	**8 14 17 21**
Level 3	**9 18**
Level 4	**19**
Level 5	**20**

Figure 19. Display of levels of modules in the "conceptual_design" problem.

This decomposition shows that the circuit has a significant amount of modules that can be executed in parallel, particularly on the first level. However, this DSM contains many feedback couplings and iterative subcycles. After the DSM has been reordered with the GA, the modules in the circuit of the "conceptual_design" DSM in figure 13 decomposes into the levels shown in figure 20.

Level 1	**1 11**	**Level 9**	**14**
Level 2	**2**	**Level 10**	**15 19**
Level 3	**3 4**	**Level 11**	**16**
Level 4	**5**	**Level 12**	**17**
Level 5	**6 12**	**Level 13**	**18**
Level 6	**7 13**	**Level 14**	**20**
Level 7	**8 10**	**Level 15**	**21**
Level 8	**9**		

Figure 20. Display of levels of reordered modules in the "conceptual_design" problem.

This ordering significantly reduces the number of feedback couplings and iterative subcycles but at the expense of parallel processing. This can be seen from the many levels with few modules on a level in figure 20. The display of levels is saved on the "cpar.log" file.

A flowchart for the two remaining functions is shown in figure 21. The menu selections are in the rectangular blocks.

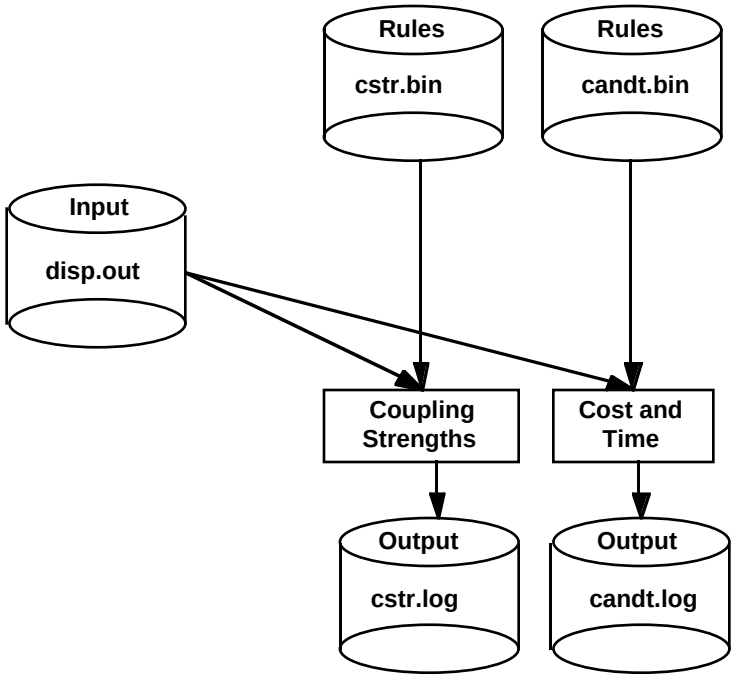

Figure 21. Flowchart of Coupling Strengths, and Cost and Time selections.

4.6.5 Coupling Strengths

Input file - disp.out
Output file - none
Rule file - cstr.bin
Log file - cstr.log

Coupling Strengths provides the user with recommendations on those modules and couplings that might be removed (or temporarily suspended) from the problem without a loss of solution accuracy [6]. There are seven levels of coupling strengths (section 3.2.2). All modules with at least one coupling of nominal or greater strength are retained. Modules with only extremely weak couplings are suggested for removal. All other recommendations depend on the relationship among the coupled modules. When reentering the **DSM** function, DeMAID/GA changes the color of the couplings to represent their strength: extremely weak - red; very weak - pink; weak - yellow; nominal - green; strong - light blue; very strong - blue; and extremely strong - black. The recommendations such as:

Suggest removing module CSYSANL because all interfaces are extremely weak"

from the "conceptual_design" problem are displayed on the console window and are saved in the "cstr.log" file.

4.6.6 Cost and Time

Input file - disp.out
Output file - none
Rule file - candt.bin
Log file - candt.log

Cost and Time provides the user with the total amounts of time and/or cost for each circuit, as well as for the entire project. The information in figure 22 is saved in the “candt.log” file.

Time required for circuit 5 is 1630

Time required for 2 iterations between modules 8 and 9 is 180
Time required for 6 iterations between modules 7 and 8 is 540
Time required for 8 iterations between modules 6 and 7 is 560
Time required for 7 iterations between modules 5 and 6 is 350

Cost required for circuit 5 is 1130

Cost required for 7 iterations between modules 5 and 6 is 490
Cost required for 8 iterations between modules 6 and 7 is 400
Cost required for 6 iterations between modules 7 and 8 is 180
Cost required for 2 iterations between modules 8 and 9 is 60

Total time required is 1910

Total cost required is 1330

Figure 22. Time and cost information from the Cost and Time function.

A relation exists between the coupling strengths (section 3.2.2) and the number of iterations for a particular feedback coupling (section 3.2.3). As in the GA, the time and cost of each module related through a feedback is summed; then the sum is multiplied by an iterative factor that has been determined from the coupling strength (section 4.5.1). The user can modify the default relation with the override list (section 3.2.3). If no coupling strengths exist, an iterative factor of 1 is used.

PROGRAM FUNCTIONS

Table 1 displays the feedback couplings for the “conceptual_design” DSM shown in figure 5; the table displays the number of iterations for the feedback coupling and the total time and cost to converge each feedback. The totals for this DSM are 21,340 units for time and 19,640 units for cost.

Table 1. Time and Cost for the Unordered Design Process.

To Module	From Module	Iterations	Time	Cost
1	2	8	560	400
1	6	4	600	840
2	8	8	1680	1680
3	6	2	160	320
4	9	7	1260	1260
5	18	6	2580	2460
6	11	8	1760	1120
7	8	6	540	180
8	9	2	140	100
8	10	8	720	720
8	15	4	960	960
8	20	7	2940	2520
10	17	8	1760	2080
11	12	5	350	250
12	13	3	180	180
13	14	6	300	420
14	15	8	400	560
14	20	4	920	760
15	16	6	300	420
16	17	7	350	490
16	21	8	1600	1280
17	18	8	560	400
18	19	6	540	180
19	20	2	180	60
Totals			**21,340**	**19,640**

Table 2 displays the same information for the reordered “conceptual_design” DSM shown in figure 13.

Table 2. Time and Cost for the Ordered Design Process.

To Module	From Module	Iterations	Time	Cost
1	11	5	1700	1600
5	6	7	350	490
6	7	8	560	400
7	8	6	540	180
8	9	2	180	400
11	21	3	960	1020
14	17	2	280	200
Totals			**4,570**	**3,950**

The number of processes contained in the iterative loops has been reduced by reordering the sequence with the GA. This reordering also reduced the total cost from 19,640 to 3,950 units and the total time from 21,340 to 4,570 units. Thus, the total cost and time in a design process is very much dependent on the ordering the sequence of the design processes.

4.6.7 User Function

Input file - user-defined filename
Output file - user-defined filename
Rule file -user-define filename
Log file - user-defined filename

User Function allows the user to define a set of rules to be called by DeMAID/GA. A window allows the user to type in the name of the file containing the rules. Typically, the input file should be the “disp.out” file. The output file and log file are optional and can be named by the user in the rule file.

A flowchart of the User Function is shown in figure 23. The menu selection is in the rectangular block. The files are all named by the user.

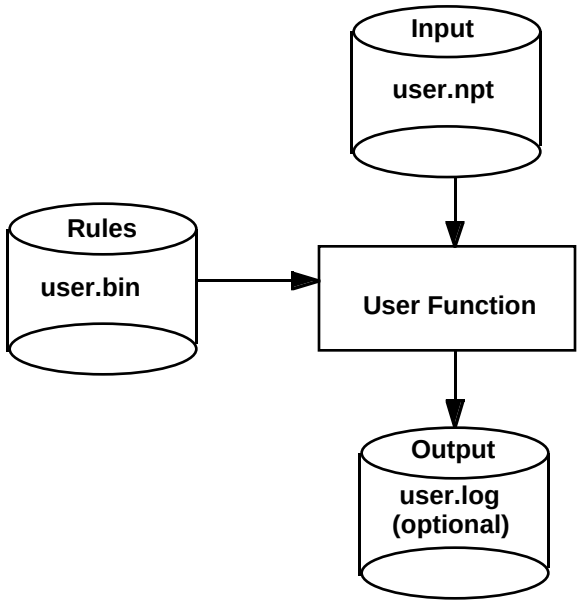

Figure 23. Flowchart of the User Function option.

4.7 INTERFACE MENU

The **Interface** function allows the user to save a file that can be input to another program. These programs include: Microsoft EXCEL© (section 4.7.1); Microsoft Project© (section 4.7.2); and Steward's DSS program© (section 4.7.3).

A flowchart of the **Interface** function is shown in figure 24. The menu selections are in the rectangular blocks. The user defines the output file for each selection.

In each section, an example is given of the way a module is written in the respective output files. The examples are taken from the WIANAL module in the "conceptual_design" problem shown in figure 13. WIANAL is the ninth module in the DSM. It requires 40 units of time. The output from WIANAL is an input to modules 8 (extremely weak), 14 (very strong), and 21 (weak). WIANAL receives input from modules 2 and 8.

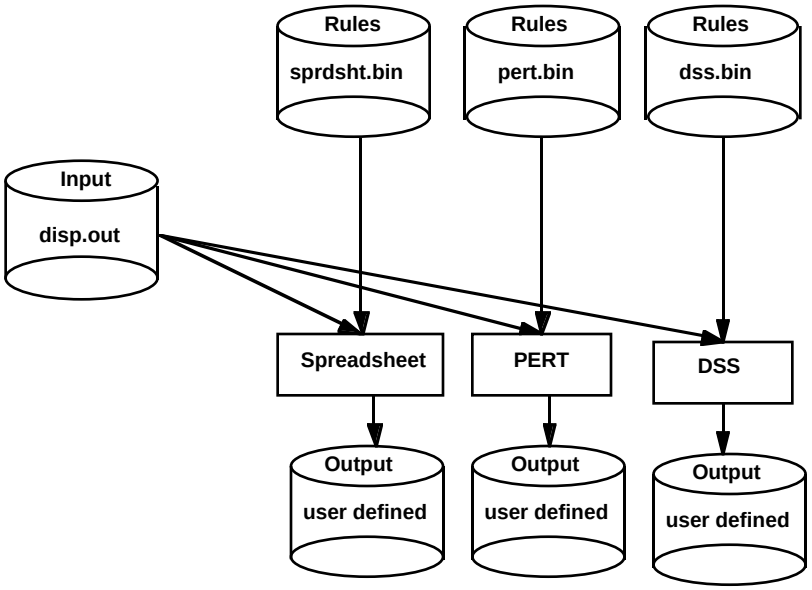

Figure 24. Flowchart of the Interface function selections.

© Microsoft Excel copyrighted by Microsoft Corporation
© Microsoft Project copyrighted by Microsoft Corporation
© DSS program copyrighted by Blitzkrieg Software

4.7.1 Spreadsheet

Input file - disp.out
Output file - user-defined filename
Rule file - sprdsht.bin
Log file - none

Spreadsheet allows the user to create a tab-delimited file for input into an EXCEL spreadsheet. The module names are written on the diagonal; and If available, the coupling strengths are written in the off-diagonal elements, otherwise, an "**X**" is placed in the coupling slot. A sample module from the spreadsheet output file is

ew WIANAL vs w

where the name WIANAL, the ninth module in the list, would appear in row 9 column 9 of the spreadsheet and the coupling strengths would be tabbed to columns 2, 14, and 21. The "ew" coupling is a feedback coupling and appears in the lower triangle of the spreadsheet. The time is not output to the spreadsheet at this time. The couplings representing input from other modules would appear in column 9, rows 2 and 8.

4.7.2 PERT

Input file - disp.out
Output file - user-defined filename
Rule file - pert.bin
Log file - none

PERT allows the user to create a comma-delimited file for input into Microsoft Project. The times are assumed to be in hours. A sample module from the PERT output file is

9,WIANAL,40h,,,"2,8",

where WIANAL is the ninth module in the list, requires 40 hours (hours is the default for the PERT output) to complete execution, and requires modules 2 and 8 to finish executing before it can begin. If there is more than one predecessor module, the modules in the predecessor field are enclosed in quotes. No coupling strengths are written to this file. The data from the feedback coupling is assumed to be available, thus the feedback coupling is not written as a predecessor module in the output file.

4.7.3 DSS

Input file - disp.out
Output file - user-defined filename
Rule file - dss.bin
Log file - none

DSS allows the user to create a file for input into the DSS program. If available, coupling strengths are converted to numbers recognizable by the DSS program. The conversion is: extremely weak = 9; very weak = 8; weak = 7; nominal = 5; strong = 2; very strong = 1; and extremely strong = 0. If no coupling strengths exist, a 0 (zero) indicates a coupling between two modules; and a -1 (minus one) indicates no coupling between two modules. A sample module from the DSS output file is

9: WIANAL
-1 -1 -1 -1 -1 -1 -1 9 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 7 -1 -1 .

where WIANAL is the ninth module in the list as shown in the first line. The second line contains the coupling data with -1's to indicate no coupling; and the 9, 1, and 7 respectively indicate the extremely weak, very strong, and weak coupling strengths. No time value is written to this file.

4.8 HELP MENU

Input file - none
Output file - none
Rule file - none
Log file - none

The **Help** function provides the user with on-line documentation about each of the major functions.

5. SAMPLE PROBLEMS

The following two sample problems are used throughout the text to describe the workings of the program.

5.1 "conceptual_design" Problem

This problem contains 22 processes (modules) found in a typical conceptual design activity. This input data are derived from the process flowchart shown in figure 25. The main problems with this type of chart are that it is difficult to determine where to begin the design activity and which processes are iterative.

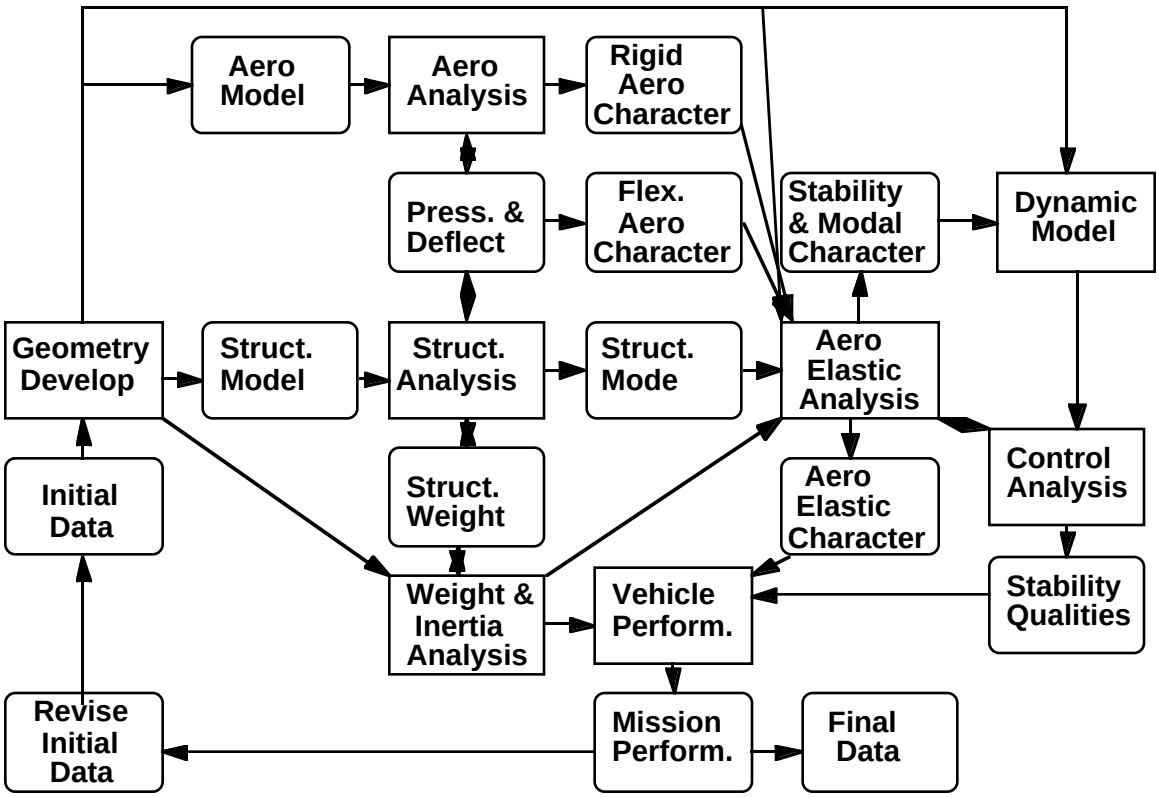

Figure 25. Process flowchart of analyses in a conceptual design activity.

The following two subsections contain the input data for each module (section 5.1.1) and data to quantify the coupling strengths (section 5.1.2). The time and cost of each module and the coupling strengths were arbitrarily selected for test purposes and not based on a real problem.

5.1.1 Module Input

```
(title conceptual_design)
(maximum 22)
(cost)
(module 1  GEOMDEV     10    50    G1   uk I2)
(module 2  INITDAT     20    40    I2    uk R3)
(module 3  RVSEDAT     30    30    R3   uk M15)
(module 4  AEROMDL     40    20    A4   uk G1)
(module 5  STRMODL     50    10    S5   uk G1)
(module 6  AEROANL     40    20    A6   uk A4 P7)
(module 7  PRESDEF     30    30    P7   uk A6 S8)
(module 8  STRANAL     20    40    S8   uk P7 S9 S5)
(module 9  STRCTWT     10    50    S9   uk S8 W10)
(module 10 WIANAL      20    40    W10 uk G1 S9)
(module 11 RAEROCH     30    30    R11 uk A6)
(module 12 FAEROCH     40    20    F12 uk P7)
(module 13 STRMODE     50    10    S13 uk S8)
(module 14 VEHPERF     40    20    V14 uk W10 R11 F12 A18 H21)
(module 15 MISPERF     30    30    M15 uk V14)
(module 16 STDMOCH     20    40    S16 uk S17)
(module 17 STRDYNA     10    50    S17 uk G1 R11 F12 S13 W10 C20)
(module 18 AROSRVO     20    40    A18 uk S17)
(module 19 DYNMODL     30    30    D19 uk G1 S16)
(module 20 CSYSANL     40    20    C20 uk D19 S17)
(module 21 HANDQUL     50    10    H21 uk C20)
(module 22 FINLDAT     40    20    goal uk M15)
```

5.1.2 Coupling Strength Data

(strength uk es G1 I2)
(strength uk n I2 R3)
(strength uk vw R3 M15)
(strength uk es A4 G1)
(strength uk ew S5 G1)
(strength uk es A6 A4)
(strength uk vs A6 P7)
(strength uk s P7 A6)
(strength uk es P7 S8)
(strength uk vs S8 P7)
(strength uk s S8 S9)
(strength uk w S8 S5)
(strength uk es S9 S8)
(strength uk ew S9 W10)
(strength uk w W10 G1)
(strength uk es W10 S9)
(strength uk s R11 A6)
(strength uk es F12 P7)
(strength uk s S13 S8)
(strength uk w V14 W10)
(strength uk es V14 R11)
(strength uk ew V14 F12)
(strength uk vs V14 A18)
(strength uk vw V14 H21)
(strength uk s M15 V14)
(strength uk es S16 S17)
(strength uk s S17 G1)
(strength uk w S17 R11)
(strength uk es S17 F12)
(strength uk ew S17 S13)
(strength uk vs S17 W10)
(strength uk ew S17 C20)
(strength uk s A18 S17)
(strength uk w D19 G1)
(strength uk es D19 S16)
(strength uk es C20 D19)
(strength uk s C20 S17)
(strength uk vs H21 C20)
(strength uk n goal M15)

5.2 "test" Problem

This problem contains 45 processes (modules). This problem is the original test case for DeMAID and provides an excellent example for decomposing a problem into a multilevel hierarchy. The following subsection contains the input data for each module section 5.2.1). Field 4 of each module distinguishes between a constraint (1), a behavior variable (2), a design variable (3), and the objective function (4).

5.2.1 Module Input

```
(title test)
(maximum 45)
(module  1 TASKC10 1 0 G010 uk DV15 DV19 DV20)
(module  2 TASKD07 3 0 DV07 uk G014 G015)
(module  3 TASKD17 3 0 DV17 uk G006 G007 G008)
(module  4 TASKD23 3 0 DV23 uk OB01)
(module  5 TASKD20 3 0 DV20 uk G009 G010)
(module  6 TASKD15 3 0 DV15 uk G016 G017)
(module  7 TASKB03 2 0 BV03 uk DV01 DV02 DV03)
(module  8 TASKC14 1 0 G014 uk DV06 DV07 DV08 DV23)
(module  9 TASKC07 1 0 G007 uk DV12 DV13 DV16 DV17 DV18)
(module 10 TASKC15 1 0 G015 uk DV06 DV07 DV08 DV23)
(module 11 TASKD21 3 0 DV21 uk G001 G002)
(module 12 TASKC04 1 0 G004 uk DV06 DV10 DV11 BV04)
(module 13 TASKC17 1 0 G017 uk DV23 BV01)
(module 14 TASKC06 1 0 G006 uk DV12 DV13 DV16 DV17 DV18)
(module 15 TASKC03 1 0 G003 uk DV01 DV02 DV03 DV04 DV05)
(module 16 TASKC13 1 0 G013 uk DV23 BV03)
(module 17 TASKB04 2 0 BV04 uk DV09 DV10 DV11)
(module 18 TASKD04 3 0 DV04 uk G003)
(module 19 TASKD11 3 0 DV11 uk G004 G005)
(module 20 TASKD02 3 0 DV02 uk G011 G012 G013)
(module 21 TASKC01 1 0 G001 uk DV16 DV17 BV02)
(module 22 TASKC02 1 0 G002 uk DV18 BV02)
(module 23 TASKC16 1 0 G016 uk DV23 BV01)
(module 24 TASKD13 3 0 DV13 uk G016 G017)
(module 25 TASKD05 3 0 DV05 uk G003)
(module 26 TASKD14 3 0 DV14 uk G016 G017)
(module 27 TASKC08 1 0 G008 uk DV12 DV13 DV16 DV17 DV18)
(module 28 TASKB02 2 0 BV02 uk DV21 DV22)
(module 29 TASKD10 3 0 DV10 uk G004 G005)
(module 30 TASKC09 1 0 G009 uk DV14 DV19 DV20)
(module 31 TASKC11 1 0 G011 uk DV01 DV02 DV03)
(module 32 TASKD16 3 0 DV16 uk G006 G007 G008)
(module 33 TASKD06 3 0 DV06 uk G014 G015)
(module 34 TASKD19 3 0 DV19 uk G009 G010)
(module 35 TASKD03 3 0 DV03 uk G011 G012 G013)
(module 36 TASKD09 3 0 DV09 uk G004 G005)
(module 37 TASKD12 3 0 DV12 uk G016 G017)
(module 38 TASKC12 1 0 G012 uk DV23 BV03)
```

(module 39 TASKD22 3 0 DV22 uk G001 G002)
(module 40 TASKD18 3 0 DV18 uk G006 G007 G008)
(module 41 TASKD01 3 0 DV01 uk G011 G012 G013)
(module 42 TASKD08 3 0 DV08 uk G014 G015)
(module 43 TASKF01 4 0 OB01 uk DV23)
(module 44 TASKB01 2 0 BV01 uk DV12 DV13 DV14 DV15)
(module 45 TASKC05 1 0 G005 uk DV07 DV08 DV10 DV11 BV04)

REFERENCES

1. Rogers, J. L.: *A Knowledge -Based Tool for Multilevel Decomposition of a Complex Design Problem*. NASA TP-2903, 1989.

2. Rogers, J. L.: DeMAID - *A Design Manager's Aid for Intelligent Decomposition User's Guide*. NASA TM-101575, 1989.

3. Steward, D. V.: *Systems Analysis and Management: Structure, Strategy and Design.* Petrocelli Books Inc, c.1981.

4. Giarratano J. and Riley, G.: *Expert Systems Principles and Programming,* PWS - Kent Publishing Company, Boston, 1989.

5. Bloebaum, C. L.: *An Intelligent Decomposition Approach for Coupled Engineering Systems*. Proceedings of the Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September 1992.

6. Rogers, J. L. and Bloebaum, C. L.: *Ordering Design Tasks Based on Coupling Strengths*. AIAA Paper No. 94-4326, 1994.

7. Goldberg, D.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Co. New York, 1989.

8. Syswerda, G.: *Schedule Optimization Using Genetic Algorithms*. *Handbook of Genetic Algorithms*, Van Nostran Reinhold, New York, 1990.

9. McCulley, C. M.; and Bloebaum, C. L.: *A Genetic Tool for Optimal Sequencing in Complex Engineering Systems*. Structural Optimization. 1996.