Fig. 1 Flow chart of the EVOLVE

Fig. 1 Flow chart of the EVOLVE (continued)

Fig. 2 Geometry of riveted lap joint

/X/ Percent_fea	13, 15, 18
/Y/ NEW_P, OLD_P, PEN	11, 13, 15, 18, 23, 35

APPENDIX B

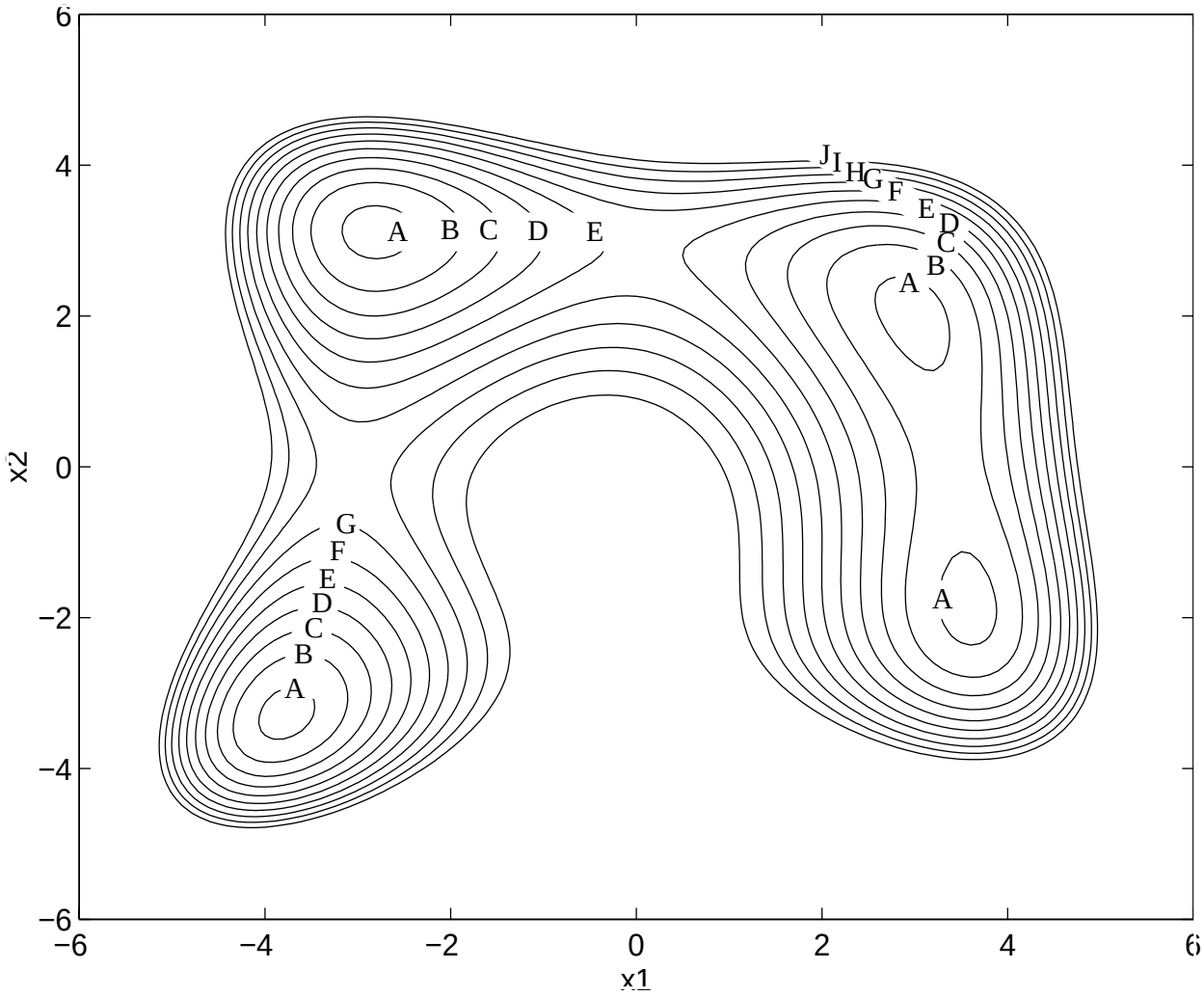
List of Common Blocks	Subroutines using the block
/A/ OldSeed	21, 31
/B/ StringLength, Popsize, Bytesize, Packsize	4, 5, 6, 7, 9, 11, 13, 15, 18, 20, 21, 23, 25, 27, 29, 30, 33, 34, 35, 36, 37
/BB/ Bias, Lost, Conv	4, 10, 18, 33
/BEST/ BestObj	18, 23
/C/ C_rate, M_rate, Mu_next	6, 7, 21, 25, 33
/CLUSTER/ Num_clusters	3, 14, 21
/DC/ Old_Curve, NDC	9, 21
/DCROSS/ Isite1, Isite2, last	6, 7, 9, 27
/DCR2/ Qual, Ch_Next, Effect_Curve	7, 9, 27
/E/ OLDfit, NEWfit, WORST, AveCurFit	9, 11, 13, 15, 18, 23, 27, 34, 35
/ERANK/ rank, num_elitist	11, 21, 23
/F/ NewGene, OldGene	4, 6, 7, 11, 13, 15, 18, 20, 25, 33, 34, 35
/H/ Doneflag, Bestflag	14, 15, 18, 21
/J/ GEN	13, 15, 18, 21, 23, 34
/K/ WorstCurFit, BestCurFit, BestDesign, Windowsize, Window	11, 18, 21, 23, 26
/KK/ Totaltrials, Trials	10, 13, 18, 21, 34
/L/ Upper_bound, Lower_bound, Accuracy	1, 5, 21
/M/ Num_Con, Num_Int, Num_Dis, Num_ele, Bit_Length, Num_Total, DIS	1, 5, 16, 18, 20, 21, 22, 24, 28, 30, 36
/MULTI/ Num_stage, ISTAGE, Num_melt_design	1, 3, 5, 13, 14, 15, 18, 19, 21, 24, 30, 32
/N/ firstflag	18, 21, 25, 34
/P/ Num_Seed	20, 21
/PM/ ADDED	13, 15, 21
/PR/ Pc_control	18, 21
/Q/ Graycodeflag	16, 21, 22
/R/ NEWobj, NEW_PEN_TOTAL, OLDobj, OLD_PEN_TOTAL	11, 13, 15, 23, 34, 35
/RELAXATION/ Relax	14, 23
/REV/ lost_bit, chance_revive	4, 21
/RR/ NeedEV	6, 7, 11, 13, 15, 20, 25, 33, 35
/S/ BestPenalty	11, 18, 23
/SHAREFLAG/ share_flag	13, 16, 21, 35
/SHARING/ alpha, sigma_share, xlow, xdiff, xxx	16, 21, 36
/T/ Num_Penalty	11, 13, 15, 18, 21, 23, 28, 35
/TL/ TITLE	18, 21
/TRACE/ I_trace	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37
/U/ MAXCAP, COE_PENALTY	13, 15, 21, 23
/V/ Savesize, Bestsize	13, 21, 30, 34
/W1/ B_Gene, B_GEN, B_Trials	30, 34
/W2/ B_Fit, B_PENALTY, B_obj	30, 34

APPENDIX A

Subroutine Name	Common Blocks Used
1. Actual_value:	L, M, MULTI, TRACE
2. Binary_value:	TRACE
3. Clusters:	B, CLUSTER, M, MULTI, SHARING, TRACE
4. Converge:	B, BB, F, REV, TRACE
5. Convert:	B, L, M, MULTI, TRACE
6. Cross:	B, C, DCROSS, F, RR, TRACE
7. Cross_curve:	B, E, DC, DCROSS, DCR2, TRACE
8. Degray:	TRACE
9. Directed_cross:	B, C, DCROSS, DCR2, F,RR, TRACE
10. Done:	B, BB, KK, TRACE
11. Elitist:	B, E, ERANK, F, K, R, RR, S, T, TRACE, Y
12. Error:	TRACE
13. Evaluate:	B, E, F, J, KK, MULTI, PM, R,RELAXATION, RR, SHAREFLAG, T, TRACE, U, V,X, Y
14. Evolve:	CLUSTER, H, MULTI
15. Generate:	B, DC, E, F, H, J, MULTI, PM, R, RR, T, TRACE, U, X, Y
16. Get_dv:	M, Q, SHAREFLAG, SHARING, TRACE
17. Gray:	TRACE
18. History:	B, BB, BEST, E, F, H, J, K, KK, M, MULTI, N, PR, S, T, TL, TRACE, X, Y
19. Init_data:	B, C, H, J, KK, L, M, MULTI, N, TRACE, V
20. Initialize:	B, F, M, P, RR, TRACE
21. Input:	A, B, C, CLUSTER, DC, ERANK, K, KK, M, MULTI, L, P, PR, PM, Q, RELAXATION, REV, SHAREFLAG, SHARING, T, TL, TRACE, U, V
22. Itob:	M, Q, TRACE
23. Measure:	B, BEST, E, ERANK, J, K, R, S, T, TRACE, U, Y
24. Melt:	L, M, MULTI, TRACE
25. Mutate:	B, C, F, N, RR, TRACE
26. Newworst:	K, TRACE
27. Next_chance:	B, DCROSS, DCR2, E, TRACE
28. Obj:	M, T, TRACE
29. Pack:	B, TRACE
30. Printbest:	B, M, MULTI, V, W1, W2
31. Ranint:	A
32. Re_initialize:	B, F, M, MULTI, RR, TRACE
33. Revive:	B, BB, C, F, REV, RR, TRACE
34. Savebest:	B, E, F, J, KK, N, R, TRACE, V, W1, W2
35. Select:	A, B, E, F, R, RR, SHAREFLAG, T, TRACE, Y
36. Share:	B, M, SHARING, TRACE
37. Unpack:	B, TRACE

Appendices A and B summarize the common blocks particular to the different subroutines in EVOLVE.

Himmelblau's Function:
$f(x)=(x_1^2+x_2-11)^2+(x_1+x_2^2-7)^2$



Contour ID	A	B	C	D	E	F	G	H	I	J
	5.00	20.00	35.00	50.00	65.00	80.00	95.00	110.00	125.00	140.00

2. HIMMELBLAU'S FUNCTION MINIMIZATION

OBJECTIVE: Exercise solving a multimodal problem using sharing strategy, and practice a multiple stage search with varying granularity.

PROBLEM STATEMENT: see attached figure.

PARTIAL INPUT AND OUTPUT FILES: see attached.

This variable is therefore of the discrete type. The objective of this optimization is to maximize the efficiency of the joint, defined as the ratio of the strength of joint to the strength of the plate. The strength of the joint is obtained as the minimum of the sharing failure strength P_s, tension failure strength P_t, and compression or bearing failure P_b. These can be formulated as follows:

$$P_s = \begin{cases} \pi x_3^2 x_1 x_2 \dfrac{\tau}{4} & , \text{if } x_1 < 3 \\ \pi x_3^2 x_1 x_2 \dfrac{\tau}{4[1.06 + 0.126(x_1 - 3)]} & , \text{otherwise} \end{cases}$$

$$P_t = (2000 - x_2 x_3) t \sigma_t$$

$$P_b = \begin{cases} t x_3 \sigma_b x_1 x_2 & , \text{if } x_1 < 3 \\ \dfrac{t x_3 \sigma_b x_1 x_2}{[1.06 + 0.126(x_1 - 3)]} & , \text{otherwise} \end{cases}$$

where,

τ = 80 MPa σ_t = 90 MPa σ_b = 120 MPa

Stress concentration due to the close placement of any two rivets were avoided by the imposition of the following linear constraints.

$$3x_3 x_1 + 2x_3 \leq 500$$

$$3x_3 x_2 + 2x_3 \leq 2000$$

Further, the strength of the plate was specified as 2700 kN.

Riveted Lap Joint

The first problem involves the design of a lap joint between two steel plates in which the rivet size and the number and arrangement of the rivet pattern were considered as design variables. The configuration of the plates and the rivets is shown in Fig. 2. The number of rows parallel to side AB is represented by an integer variable x_1 with permissible values between 1 and 32. The number of the rivets in each row x_2 was an integer variable and was allowed to assume values between 1 and 128. The diameter x_3 of all rivets was assumed to be the same, and is chosen from a commercially available set:

x_3 = [6,8,10,12,14,16,18,20,22,24,27,30,33,36,40,45] (mm)

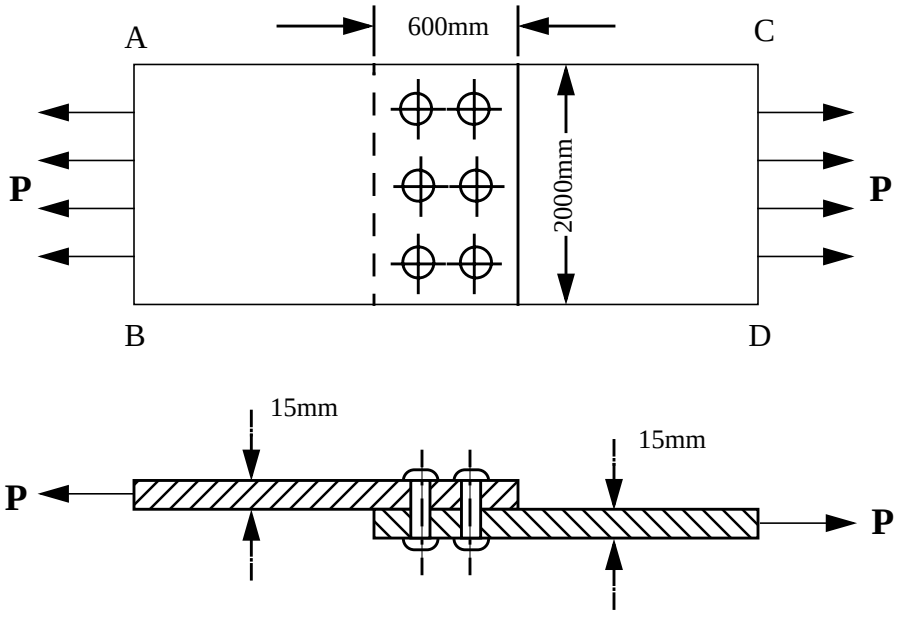


Fig. 2 Geometry of riveted lap joint

11. TEST PROBLEMS

1. RIVET JOINT EFFICIENCY MAXIMIZATION

OBJECTIVE: Exercise the use of integer and discrete variables, solve a maximization problem, and practice the exterior penalty approach for constrained optimization.

PROBLEM STATEMENT: see attached.

PARTIAL INPUT AND OUTPUT FILES: see attached.

<SECTION 5> needed only where N_integer > 0

Order	the number of variable.
Lower bound	lower bound of the integer variable.
Upper bound	upper bound of the integer variable.

<SECTION 6> needed only where N_discrete > 0

Num_elements	number of allowable values for each discrete variable; an array with as many elements as N_discrete.

<SECTION 7> needed only where N_discrete > 0

DIS(I,K)	list all allowable values in an ascending order for each discrete variable.

<SECTION 8> needed only where Sh = 1 (sharing strategy)

alpha	defines the nature of the sharing function.
sigma_share	defines the radius of sharing neighborhood.

<SECTION 9> needed only where Sl > 0

Bit_length(I)	bit length of binary substring for each of all variables.

<SECTION 2>

N_continuous	number of continuous design variables.
N_integer	number of integer design variables.
N_discrete	number of discrete design variables.
N_penalty	number of constraints

*** in EVOLVE, the design variables in an optimization problem are require to be arranged in a sequence as continuous, integer and discrete ***

<SECTION 3> needed only where N_penalty>0

Pe	Starting penalty coefficient
PM	Stepwise increment on the penalty coefficient every 10 generations.
MC	Maximum cap on the undeflected penalty. Any penalty which exceeds this maximum cap will be adjusted as the sum of the maximum cap and 20% of the amount of penalty over the value of MC.

<SECTION 4> needed only where N_continuous > 0

Order	the number of variable.
Lower bound	lower bound of the continuous variable.
Upper bound	upper bound of the continuous variable.
Accuracy(I)	precision level of the continuous variable in I-th stage; this precision level is needed for every stage resulting in a total number of STAGE entries.

reside in the current generation. Elitist strategy.

Ws — Window size; The parameter WORST is selected from the largest pseudo objective value occurred in previous "Ws" generations.

<SECTION 1B>

I_trace — 1 for tracing the routines during the search; 0 for NOT.

Ip — 1 for printing information in "detail.dat" on the monitor; 0 for NOT.

Os — initial seed for random number generator; an integer.

Nc — number of clusters in which to place designs obtained in final generation of the last stage of search into "clusters.dat"; Nc = 1 will turn off the clustering scheme.

DC — number of the generation in which the directed crossover start to be active; 0 for only regular crossover.

NDC — number of previous generations on.which the crossover gain curve is based.

CR — revival rate; a directed mutation; the probability to revive each lost bit in a design.

VG — varying granularity approach; 0 for NOT and 1 for YES.

<SECTION 1C> needed only VG=1

STAGE — number of stages in a complete genetic search

Nmd — number of designs in the final population of a previous stage of search, which will be used to generate the initial population for the next stage of search using the melting strategy.

RELAX(I) — constraint relaxation percentage for the I-th stage; need as many as STAGE for this RELAX array.

10. CONTROL FILE INPUT DESCRIPTION

The control file is set up such that all variables are read in free format.

<SECTION 0> optional

Title: Fill in title of the optimization problem.

<SECTION 1A>

Pc — Probability of crossover; if Pc is 0.8, 80% of designs will form pairs to undergo the crossover operation.

Pm — Probability of mutation; if Pm is 0.01, one of every 100 bits in the binary string chain (contained in a series of strings of all designs) will have a chance to switch from 0 to 1, or vice versa.

Ps — Population size in a generation.

Em — Maximum allowed function evaluations.

Ss — Number of best designs to be saved in "bestfile.dat"

Sd — Number of seeded designs to be included in the initial population. Seeded designs are stored in "**seed.dat**". If **0** is chosen, all initial population will be randomly generated.

Sl — String length for each design; **0** for automatic creation of binary representations, **1** for the option that the user can provided string length for each variable. In the latter case, users have to provide lengths for every design variable.

Gy — Gray code binary implementation; **0** for fixed point binary representation and **1** for gray code binary representation.

PRc — Printing frequency control in "history.dat".

Sh — Sharing function implementation; **0** for NOT,and **1** for YES.

Et — Number of best designs in previous generation guaranteed to

A run of the EVOLVE code will create the following output files.

1) "history.dat"

contains basic statistics of the genetic search, which includes the average fitness of a population, the objective value of the best design in the population, its penalties due to constraint violations, percentage of feasible designs in the population, number of bits which are identically zeros or ones (lost), number of bits which are 95% zero or ones (conv), the degree of bias for zero bits or one bits (bias) which is the average of percentages for each bit.

2) "detail.dat"

contains detailed information for the best design in each generation.That includes the design variables, objective value and constraint values.

3) "bestfile.dat"

contains a number of best designs explored during the genetic search. users can decide the number of best design to be stored.

4) "restart.dat"

contains design variables of ranked designs in the last generation; this can be used later as the initial population for a new genetic search.

5) "clusters.dat"

contains designs obtained at the end of the last generation of the final stage of genetic search in a form of clusters. Each cluster can contain designs with its metric distance close to some leading design. User can select the number of clusters.

7. RUNNING THE PROGRAM

Before the user can run the program, the file "**obj.f**" needs to be completed. All genetic algorithm parameters and information of design variable space is defined in "**ga_control.dat**". All information is input by answering a number of structured questions. The user is suggested to keep a copy of "**ga_control.dat**" in case that some format is accidentally altered. After generating these two files, the user can type "**make**" command to create an executable file "ga". Type "**ga**" and hit return to initiate the genetic search.

8. TERMINATION OF SEARCH

The genetic search will be terminated either when the maximum permissible function evaluations are reached, or when a convergence criterion is satisfied. Users can choose the maximum number of function evaluations allowed for a genetic search. A convergence criterion is currently set as a point when 90% of bits in binary strings are identically zeros or ones for all designs. Users can change the percentage in "**done.f**".

9. INPUT AND OUTPUT FILES

Three input files for EVOLVE are defined as follows:

1) "ga_control.dat"

   a required control file to run EVOLVE; user provides all necessary search parameters and chooses desired options.

2) "obj.f"

   a fortran routine where user defines the objective function and constraints.

3) "seed.dat"

   User can seed the initial population by providing seeded designs in this file.

Re_initialize	This routine creates initial population for different stage of search, except the first stage, in a genetic search with varying granularity
Savebest	This routines saves [Bestsize] best designs.
Select	This routine performs the selection of parents for next generation; The chance for each design to be selected as one parent depends on the ratio between its fitness and the average fitness of the population.
Share	This function computes the sharing penalty for a design. The magnitude of penalty is related to the presence of designs in its neighborhood.
Unpack	This routine unpacks a binary string [where one integer stores 15 bits] into a loose binary string [where one integer stores only one bit]

	minimization, also constraints to form a pseudo-objective function.
Evolve	The main program of EVOLVE which is a genetic search optimization program capable of solving minimization problems with any combination of discrete, integer and continuous design variables.
Generate	This routine entails (1) forming a new generation, (2) evaluating the population, and (3) gathering performance statistics, for each generation.
Get_dv	This routine computes actual design variable values from a representative binary string of this design.
Gray	This routine transforms floating-point binary segment into gray-coded binary segment.
History	This routine obtains the statistics of each generation and make a number of reports on that information; this includes (1) history.dat (2) detail.dat (3) restart.dat.
Init_data	This routine restores initial parameters and computes length of binary string for the design in the upcoming stage of search with a new granularity.
Initialize	This routine sets up the initial population.
Itob	This routine determines the binary string for a design from its decimal representation.
Measure	This routine computes the performance statistics of the genetic search in the current generation.
Melt	This routine distributes a design randomly in a region which covered by a design located in a previous stage of search. Used only in a multistage search scheme.MutateThis routine performs random bitwise mutation in the current population according to the probability of mutation.
Newworst	number between two given integer lower and upper bounds.

6. DESCRIPTION OF ROUTINES

routine	Purpose
Actual_value	This function converts the decimal representation of a binary string into the actual design variable value.
Binary_value	This function determines the decimal representation of a binary string.
Clusters	This routine determines and prints designs of final generation of a last stage search in the form of clusters.
Converge	This routine computes the convergence status of the genetic search after each generation.
Convert	This routine converts design variables of a seeded design to a corresponding binary string.
Cross	This routine performs two-point crossover on paired designs in the population with assigned probability of crossover.
Degray	This routine transforms a gray-coded binary segment into a float-point binary segment.
Directed_cross	This routine performs two-point crossovers where crossover sites partially depend on the bitwise crossover gain in previous generations. Used only in directed crossover strategy.
Done	This routine checks the termination criteria of genetic search. A stage of a genetic search would be stopped if the maximum number of function evaluations is reached OR convergence criterion is satisfied.
Elitist	This routine serves to keep a selected number of best designs of previous generation in the current generation.
Error	This routine prints the error message and stops the program.
Evaluate	This routine computes the objective function, and in constrained

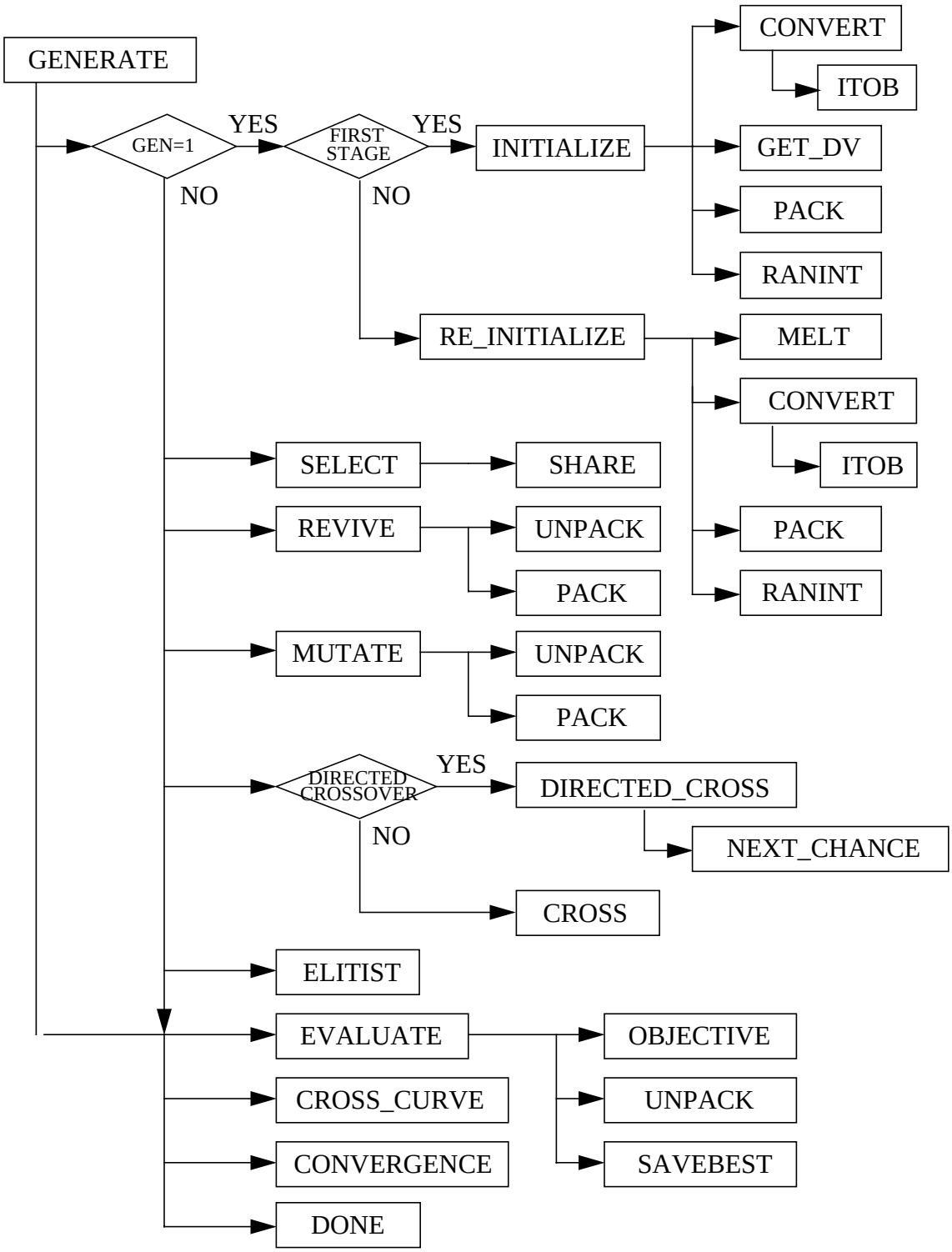


Fig. 1 Flow chart of the EVOLVE (continued)

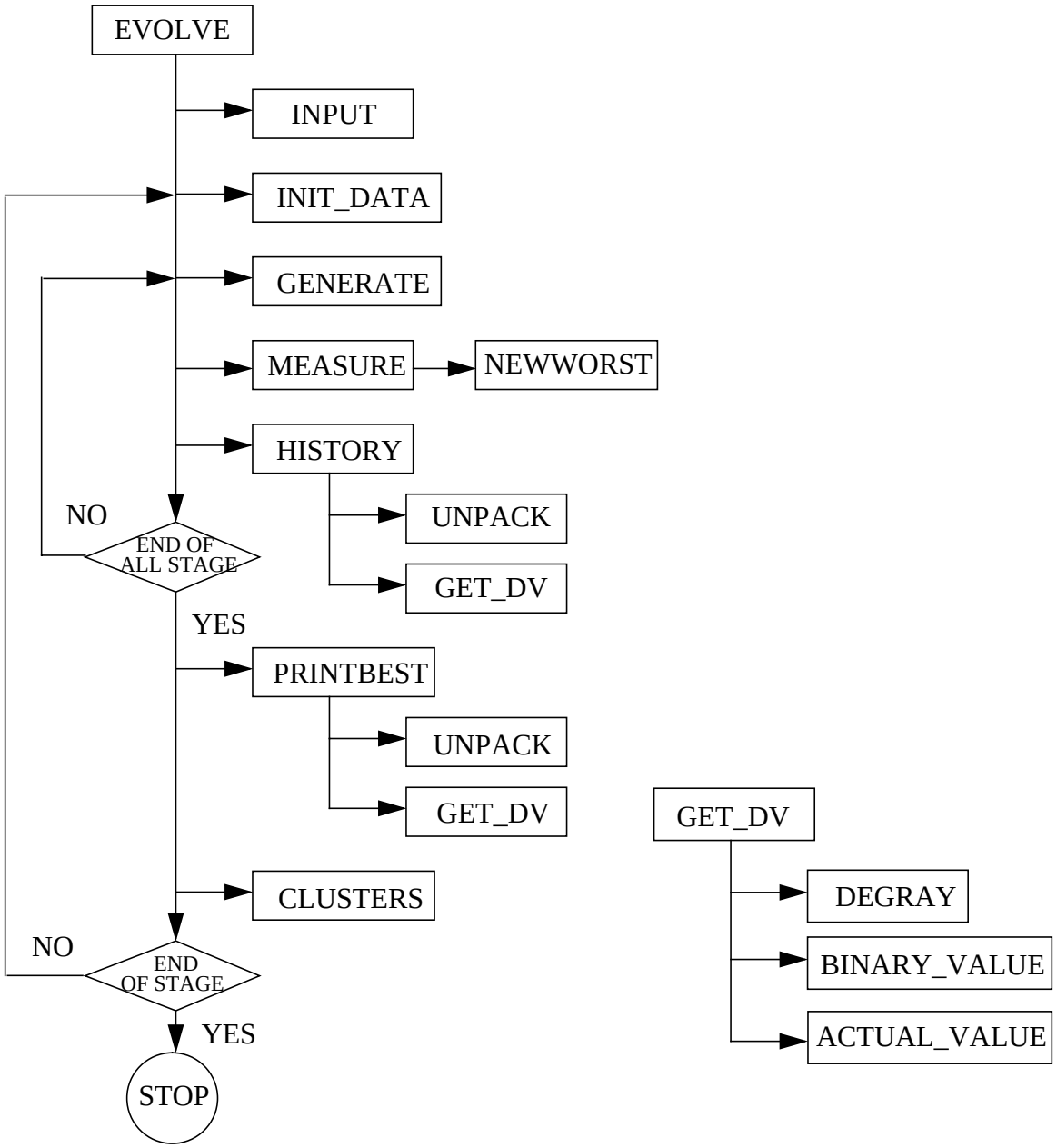


Fig. 1 Flow chart of the EVOLVE

5. DESCRIPTION OF MODULES

A flow chart of the entire code is given in Figure 1

4) CROSSOVER

In the crossover process, two new designs may be created by swapping a section of the binary string from two parent designs. As an example, once the underlined section (crossover site) is defined, two offspring designs are obtained from the parent designs, by exchanging the underlined binary bits as follows:

dad	101<u>01</u>0101
mom	010<u>10</u>1010
kid1	101<u>10</u>1101
kid2	010<u>01</u>0010

The sites of crossover are randomly generated.

5) MUTATION

To minimize the possibility of the loss of a design trait characterized by the presence of all 0's or all 1's at a particular site in a given population, the process of mutation is introduced in genetic search. The mutation operator locates a bit site at random, and switches the bit from 0 to 1 or vice versa, with some low prescribed probability.

6) EVALUATION

This process consists of evaluating the objective function and constraints. A user friendly subroutine "**obj.f**" is the place a user defines the objective function and constraints. Objective function value is returned as **OBJ**, and m constraint values are returned as **PENALTY**(1), **PENALTY**(2), ..., **PENALTY**(m). If an external code such as a finite element analysis package is needed for objective function value and/or constraint values, the user can exit from "obj.f", and come back to the same point in "obj.f" after information is obtained.

The integer variable space is [1, 2, 3, 4] and the continuous variable space is [0.10, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17]. One approach is to use binary substrings 00, 01, 10 and 11 to represent 1, 2, 3 and 4 for the integer variable, and 000, 001, 010, 011, 100, 101, 110, and 111 to represent 0.10, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, and 0.17 for the continuous variable. Then, a design such as (2, 0.13) would be mapped into a binary string of the form 01011, where the first two digits represent the first variable, and the last three digits represent the second variable.

EVOLVE provides an option (users are encouraged to use this option) with which an appropriate binary substring for each design variable of any type would be automatically generated.

4. MAJOR PROCEDURES

1) INITIALIZATION

Once the size of population is specified, the "initialize.f" will set up an initial population via a random number generator. If users choose to seed some initial designs, they can be stored in "seed.dat". The rest of designs needed to fill the initial population will be created randomly. EVOLVE can also initiate a new genetic search with a starting population that was obtained in the final generation of a previous genetic search. In this case, only thing that the user needs to do, is to choose the RESTART option. With this option, the code will automatically retrieve the final population from "restart.dat" which is a default product of the genetic search.

2) GENERATION

A genetic search generation consists of selection (reproduction), crossover, mutation, and evaluation.

3) SELECTION

The selection process is to choose designs from the current generation and form parent designs of the next generation, which may subsequently create offspring designs via genetic exchange processes referred to as crossover and mutation. The probability for a design to be chosen as a parent design is roughly proportional to the ratio between its fitness and average population fitness.

the reproduction plan. A scaled increase in this penalty term over several generalities of genetic search, ie. implemented in the present version of EVOLVE.

2. DESIGN SPACE

The design variables of an optimization problem can have continuously varying values, integer values, discrete values, or any combination of the three. For example, a composite laminate consists of an integer number of layers. If the number of layers is chosen as one design variable, it will be an integer design variable. A possible element set for a design variable representing the area of a truss member is usually defined by available cross section areas provided by manufacturer. This design variable is a discrete variable. The method of defining the design variable space is as follows:

1) Continuous variable,

define its lower bound, upper bound and step size. If -5.12 and 5.11 are selected as lower and upper bounds, respectively, and a step size is set as 0.01, then the design space for the continuous variable would be [-5.12, -5.11, -5.10, .., 5.10, 5.11]. In general, however, the step size of a continuous variable is determined by the precision of numerical computation on a specific machine.

2) Integer variable,

define its lower bound and upper bound. An integer variable is treated as a continuous variable with a stepsize of 1. If 1 and 20 are selected as lower and upper bound, the variable space is then [1, 2, 3, ..., 19, 20]

3) Discrete variable,

define the number of elements in the set, and list all elements in an ascending order as follows: [0.25, 0.47, 1.10, ..., 18.35, 20.84]

3. BINARY REPRESENTATION

Each design variable is coded into a binary substring. A design consisting of N design variables would be coded into a binary string which is a simple head-to-tail connection of N substrings, each of which represents one design variable. Consider a very simple design problem, with one integer variable and one continuous variable.

1. INTRODUCTION

EVOLVE genetic search code is written in Fortran, and run under the UNIX operating system. Very few amendments would be needed to run EVOLVE code in VMS or other operating systems. This code can be used to solve a function minimization (direct) or maximization (indirect) problem. A typical optimization problem that can be solved through the use of the EVOLVE code can be stated as follows:

$$\begin{array}{ll} \text{Min or Max} & F(X) \\ \text{Subject to} & g(X) \leq 0 \end{array}$$

$$X_i^l \leq X_i \leq X_i^u$$

F(X) is the objective function, g(X) are inequality constraint functions, and X is the vector of design variables where each element is bounded by a lower and an upper bound. If there are no constraints involved in the optimization, it is an unconstrained optimization problem.

In the present implementation, EVOLVE solves minimization problems more directly than maximization problems. With minimization problems, the objective function can be written directly in its original form. However, for maximization problems, the objective function needs to be redefined so that the goal to maximize the original objective function is transformed into minimizing an adjusted objective function. A simple approach to define the new objective function is to subtract the original objective function from a large number, so that a maximum value of the objective function will become the minimum value of the new function.

The constrained optimization problem is transformed into unconstrained optimization problem with the use of the exterior penalty function approach. In a minimization problem, an infeasible design will have the following objective function value:

$$F''(X) = F(X) + p * \Sigma\, g(X)$$

where
- F''(X): a pseudo-objective function
- F(X): original objective function
- p: penalty coefficient
- g(X): constraint values, g(X)=0 if g(X)≤0

The magnitude of the penalty term cannot be arbitrarily large, as it would bias

EVOLVE

A GENETIC SEARCH OPTIMIZATION CODE

USER'S MANUAL

C.-Y. LIN AND P. HAJELA

DEPARTMENT OF MECHANICAL ENGINEERING, AEROSPACE ENGINEERING AND MECHANICS
RENSSELAER POLYTECHNIC INSTITUTE

JANUARY 17, 1993

ABSTRACT

This document describes the EVOLVE system for function optimization based on genetic search techniques. This system can be used on optimization problems with a mix of continuous, integer and discrete design variables. The binary representation for each design variable may be automatically determined or defined by users as an option. Sharing function implementation is supported and may be chosen to enhance the ability of locating global optimum. Automatic execution of consecutive genetic search with decreasing granularity in design space is also implemented.