A FORTRAN 90 GENETIC ALGORITHM MODULE FOR DESIGN OF COMPOSITE LAMINATE STRUCTURES

M. T. McMahon and L. T. Watson
Departments of Computer Science and Mathematics
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106 USA

G. A. Soremekun and Z. Gürdal
Department of Engineering Science and Mechanics
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0219 USA

R. T. Haftka
Department of Aerospace Engineering, Mechanics, and Engineering Science
University of Florida
Gainesville, FL 32611-6250 USA

http://csgrad.cs.vt.edu/˜mcmahon/gamod.ps

The goal.

To develop a genetic algorithm tool for composite laminate structural design optimization that can:

- be used for a wide variety of structures,
- be integrated with analysis codes easily,
- facilitate development and testing of new genetic algorithms.

The target problem.

A stiffened panel made from an assemblage of laminated composite plates, where each plate is composed of several plies made of one or more fiber reinforced plastic materials.

Discrete variables: material and orientation of the fibers.

Continuous design variables: geometric dimensions of the plates.

The test problem.

- 36 in. long by 30 in. wide, simply supported panel.
- Loaded under any combination of axial and shear loads.
- Two material types: graphite-epoxy and Kevlar-epoxy.
- Ply orientations from -75 to 90 degrees, in increments of 15 degrees.
- A fitness function which is a measure of the weight, penalized by small or violated safety margins.

The structure of a population.

- Population
  - Subpopulation
    - Individuals
      - Laminate Chromosomes
        - Ply Genes
          - orientation
          - material
      - Geometry Chromosomes
        - Geometry Genes
          - real number

Why Fortran 90? Abstract data types.

Data representations reflect underlying concepts better.

Inheritance allows a data type definition to use other data types.

e.g., the data types for an individual are built with:

```
! Gene data types:
  type ply_gene
    integer (KIND=small)  ::orientation
    integer (KIND=small)  ::material
  end type ply_gene

  type geometry_gene
    real (KIND=R8)  ::digit
  end type geometry_gene
```

Why Fortran 90? Abstract data types.

```
! Chromosome data types:
type laminate_chromosome
   type (ply_gene), pointer, dimension(:):: ply_array
end type laminate_chromosome
!
type geometry_chromosome
   type (geometry_gene), pointer, dimension(:):: geometry_gene_array
end type geometry_chromosome
!
! Individual (structure) data type:
type individual
   type (laminate_chromosome),pointer,dimension(:):: laminate_array
   type (geometry_chromosome),pointer,dimension(:):: geometry_array
end type individual
```

Why Fortran 90? Dynamic Memory Allocation.

Array sizes can be specified at program run time, which supports structure designs and populations of variable size.

e.g., a child individual's laminate array is sized with:

```
!Allocate laminates:
type (individual) :: child
allocate (child%laminate_array(individual_size_lam))
```

and deallocated with:

```
!Deallocate laminates:
deallocate (child%laminate_array)
```

Array dimensions, and thus memory usage, depend on user input.

Why Fortran 90? High Level Array Operations.

Operations which were previously confined to scalar data elements can be performed on arrays:

```
! Declare ply array:
integer, dimension (3,10) :: ply_array
:
! Initialize ply_array to zero:
ply_array = 0
```

Portions of an array may be accessed as a single object. Array programming becomes more efficient and code more readable. e.g., single-point crossover can be implemented with:

```
type (laminate_chromosome) :: child, parent(2)
:
!Crossover from parent 1:
child%ply_array(1:cross_point) = parent(1)%ply_array(1:cross_point)
!Crossover from parent 2:
child%ply_array(cross_point+1:) = parent(2)%ply_array(cross_point+1:)
```

Why Fortran 90? Modularity.

Encapsulation—data type declarations and related subprograms are kept in a separate syntactic unit.

Code maintainability and reusability are enhanced—the GA implementation is separate from the specific optimization problem.

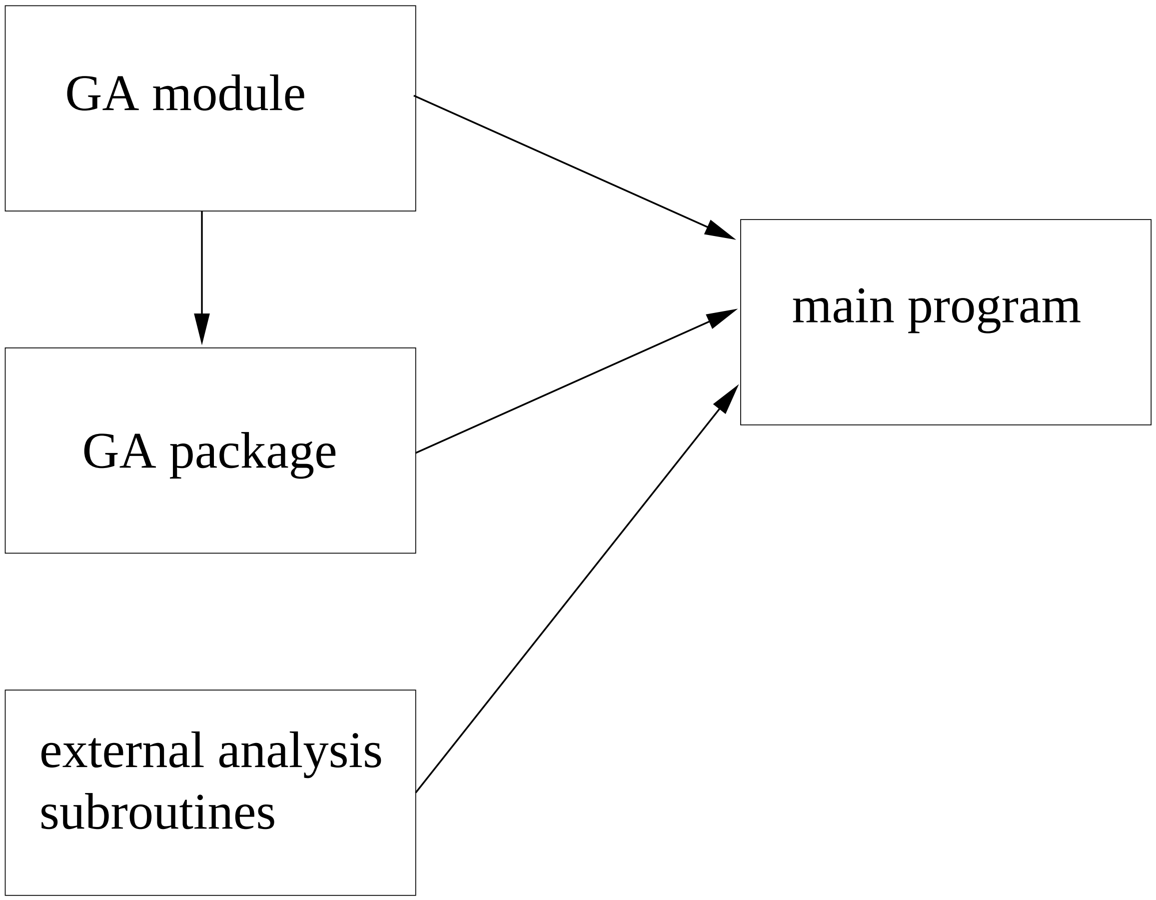
e.g., the GENERIC_GA module is introduced into a user program with:

```
PROGRAM optimize
USE GENERIC_GA
⋮
END PROGRAM optimize
```

or specific portions of the module are introduced with:

```
PROGRAM optimize
USE GENERIC_GA, ONLY : population, initialize_population
⋮
END PROGRAM optimize
```

Fortran 90 program units.



The GA package of operators.

The GA package written for use with the GENERIC_GA module contains the following genetic operators:

- crossover—one point and two point,
- mutation,
- ply addition,
- ply deletion,
- ply permutation.

By using the GA module, the package operators are guaranteed to work with any composite laminate structure.

Test runs used an elitist selection scheme and operator probabilities of crossover = 1.00, ply mutation = 0.05, orientation mutation = 0.05, addition = 0.05, deletion = 0.10, and permutation = 0.25.

The GA package continuous variable crossover.

This operator allows optimization with continuous variables not represented by discrete genes. Given parents p_1 and p_2, the geometry gene crossover first creates a randomly distributed $N(\mu, \sigma)$ real number r, where

$$\mu = \frac{p_1 + p_2}{2}, \qquad \sigma = \frac{|p_1 - p_2|}{2},$$

and then enforces physical limits by taking the child value as

$$c = \min\{\max\{r, L\}, U\},$$

where L, U are lower, upper limits on the geometry variable. Note that with probability 0.68 (the probability that a normal variate lies within one standard deviation of the mean μ), c lies between the parent values p_1 and p_2, but that c can also be well outside the segment $[p_1, p_2]$. This is essentially how evolutionary algorithms (Bäck, 1996) work, where the value of σ itself can be adapted as the evolution proceeds.

Conclusions.

The design effort described has yielded:

- a flexible, general, reusable GA optimization tool,
- Fortran 90 compliance along with compatibility with existing FORTRAN 77 analysis codes,
- optimal designs which match those from custom designed, problem specific GA implementations.

Future work.

- expansion of GA package—additional selection schemes, subpopulation interaction (migration).
- distributed memory parallel implementation.