Cambridge Genetic Algorithm Software Package

CamGASP

N.R. Shaw, M.B. Grieve, R.E. Ansorge and T.A Carpenter

University of Cambridge
Cavendish Laboratory

Feb 2001

Table of Contents

1. Introduction

This report aims to give a brief overview of software developed within the Low Temperature Physics Group of the Cavendish Laboratory to optimise a variety of engineering designs. The software is generic in nature and is envisaged to be applicable to a wide range of engineering problems. The software makes use of Genetic Algorithms to facilitate the optimisation of a design and the reader is assumed to be familiar with this technique (see "Genetic Algorithms in Search, Optimisation, and Machine Learning", David E. Goldberg, Addison-Wesley, 1989).

As engineering optimisation requires significant computing resources, the software has been written to run both on a single machine, and in parallel on clusters of workstations or supercomputers such as the University of Cambridge's Hitachi SR2201. The LAM Message Passing Interface standard is used for parallel execution.

In order to make the software generic, the constraint was imposed that any design could be represented by a relatively simple fixed format. This format is discussed more fully in Section 3 of this document.

To aid clarity the software has a fixed division between that which is wholely generic, and that which must be coded by a user for a specific design. A summary of the generic components of the program are given in Section 4, and of the design specific components in Section 5.

2. Code management and compilation

2.1 Software content and the CVS repository

All files relating to the software are stored within a central repository, located on the node ‘tifa’ under the directory location /cvsroot/Generic. The software is managed by the Concurrent Versions System (CVS) (see “Version Management with CVS”, Per Cederqvist et al) and consists of the following components;

```
Makefile
header.h
main.c
init_global_struct.c
comline_parse.c
display_help.c
logger_debug.c
logger_data.c
random_generators.c
ga_main.c                 Generic components
ga_selection.c
ga_mating.c
ga_mate_chrom.c
ga_mutation.c
ga_mutate_chrom.c
ga_replacement.c
ga_stagnation.c
ga_migration.c
ga_sortings.c
ga_sharing.c

design.h
design_init.c
design_fit.c              Design specific components
design_functions.c
```

Although CVS is installed as standard within Red Hat Linux, a user must be an allowed host on the node tifa and their local .bash_profile must be modified to indicate the location of the repository. The relevant alterations needed are given below;

```
CVSROOT=tifa.phy.cam.ac.uk:/cvsroot
CVSEDITOR=emacs
EXPORT CVSROOT, CVSEDITOR
```

A sample session would then take the form;

```
$ cvs co Generic
$ mkdir Working
$ cp Generic/* Working/
$ cvs release –d Generic
```

2.2 The Makefile

To aid in the compilation of the software a Makefile has been designed to allow automatic compilation for three different target platforms, viz. a stand-alone node, a Beowulf Cluster and the Hitachi SR2201 supercomputer. To select the target platform a user is required to modify the initial variables defined in the header of the Makefile as shown below. In the file all lines starting with the character # represent comment lines and are ignored. Thus in the example given below the software would be compiled to run on a stand-alone node.

```
# Set variables depending on which platform to run the program.
# CC = compiler, LEVEL = optimisation level, DIR = search directory
# for specific header files, MPI = -D_MPI to compile with MPI, blank
# to compile without, LOPT = -D_LONGOPT to compile allowing long
option
# parameters in the command line, blank for short option parameters
# Settings for Babbage
#CC    = xcc
#LEVEL = +O2
#DIR   = -I./
#MPI   = -D_MPI
#LOPT  =

# Settings for the Beowulf Cluster
#CC    = hcc
#LEVEL = -O3
#DIR   =
#MPI   = -D_MPI
#LOPT  = -D_LONGOPT

# Settings for non-parallel single node
CC     = hcc
LEVEL  = -O3
DIR    =
MPI    =
LOPT   = -D_LONGOPT
```

An example session to compile the software would then take the form;

```
$ cd Working
$ make
$ make clean
```

3. Data Structures

3.1 The structure of a gene

The structure of a gene is entirely design specific and must be coded by the user. A gene represents a single sub unit of the design, where the design is composed of a variety of similar sub units (ie an MRI magnet consists of several axially symmetric solenoid windings). The elements of a gene can be floating point or integer values, although they must be arranged with the floating point values first. The structure of a gene is given below where the values 1, 2, and 3 represent the design specific elements;

```
typedef struct {
  double value1
  double value2;
  int value3;
} gene_type;
```

3.2 The structure of a chromosome

The structure of a chromosome must consist of its fitness value, two integer values indicating its parents and an array of genes. The gene array size is set to a defined value, which may be bigger than the actual number of genes envisaged to adequately model the design. It is possible to add design specific elements into the structure of the chromosome without affecting the correct operation of the generic components of the program. The chromosome structure and its diagrammatical representation are given below;

```
typedef struct {
  double fitness;
  int parent1;
  int parent2;
  double value4;
  int value5;
  gene_type gene[MAX_GENES];
```

} chromosome_type;

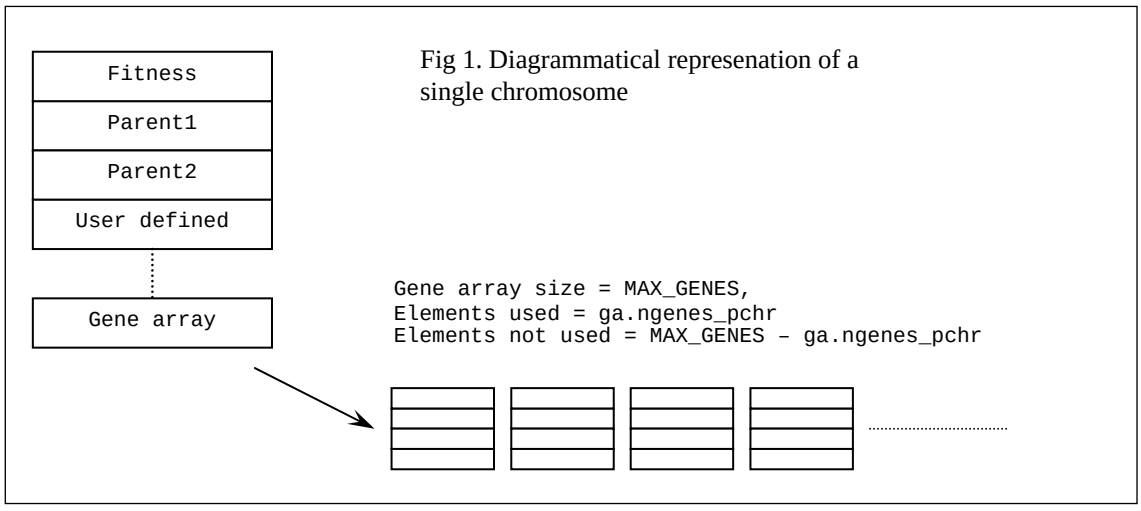

Fig 1. Diagrammatical represenation of a single chromosome

4. Summary of the generic components

4.1 File name: **header.h**

Description:
The header file for all generic components of the software. The file declares all type and structure definitions and prototypes all functions to be externally visible.

Type and structure definitions;

select_type; /* Types used for selection eg migration type*/

```
print_type;          /* The level of verbose logging */
run_param_type;      /* Holds all info about program execution status */
niche_type;          /* Define the geometry of the niche map */
genetic_algorithm_type;  /* Defines the genetic algorithm parameters */
input_output_type;       /* Defines the io parameters */
```

Prototypes of all externally visible functions;

```
int comline_parse(int argc, char *argv[]);
int init_global_struct(void);
int display_help(void);
int debug_file_open(int stamp);
int debug_file_reopen(int stamp);
int debug_file_close(int stamp);
int debug_log_params(print_type level);
int debug_log_command_line(print_type level, int argc, char *argv[]);
int debug_log_fitness_funct(print_type level);
int debug_log_chrom(print_type level, chromosome_type chromosome);
int debug_log_line(print_type level, char *strptr, ...);
int data_file_open(char *str);
int data_file_close(void);
int data_log_info(void);
int data_log_chrom(chromosome_type chromosome);
int data_read_chrom(chromosome_type *chromosome);
double gasran(long *idum);
double ran1(long *idum);
int ga_main(chromosome_type *nichepool, chromosome_type *parentpool,
           chromosome_type *offspringpool, int generation, niche_type
           niche, int node_rank, int nodes_avail);
int ga_selection(chromosome_type *nichepool, chromosome_type *parentpool);
int ga_mating(chromosome_type *parentpool, chromosome_type *offspringpool);
int ga_mate_chrom(chromosome_type par1, chromosome_type par2,
               chromosome_type *child1, chromosome_type *child2);
int ga_mutation(chromosome_type *offspringpool);
int ga_mutate_chrom(chromosome_type *chromosom, int ngene, int ncomp,
                 int mut_type);
int ga_replacement(chromosome_type *offspringpool,
                chromosome_type *nichepool);
int ga_sharing(int niche_size, chromosome_type *nichepool);
int ga_stagnation(chromosome_type *nichepool, int generation,
               niche_type niche);
int ga_migration(chromosome_type *population, int generation,
               niche_type *niche, int node_rank);
int sort_by_fitness(unsigned long n, chromosome_type *ra);
```

4.2 File name: **main.c**

Description:
The main control program managing the command line parsing, initialisation of global variables, initialisation of array and data structures, opening of log files, execution of the genetic algorithm and tidying up after program execution. The software has been designed to operate on niches within a population - to this aim a niche map is used to guide the program through its execution. For parallel running two scenarios are possible viz, a given node operates on a constant number of niches, or a constant number of nodes operate on a given niche. Each scenario assumes a homogeneous system, ie the number of niches per node, or nodes per niche is constant and of integer value.

Contains;

```
int main(int argc, char *argv[])
```

Uses the service of;

```
int init_global_struct(void)
int comline_parse(int argc, char *argv[]);
int display_help(void);
int debug_file_open(int stamp);
int debug_file_reopen(int stamp);
int debug_file_close(int stamp);
int debug_log_params(print_type level);
int debug_log_command_line(print_type level, int argc, char *argv[]);
int debug_log_fitness_funct(print_type level);
int debug_log_chrom(print_type level, chromosome_type chromosome);
int debug_log_line(print_type level, char *strptr, ...);
int data_file_open(char *str);
int data_file_close(void);
int data_log_info(void);
int data_log_chrom(chromosome_type chromosome);
int data_read_chrom(chromosome_type *chromosome);
int init_chrom_random(chromosome_type *chromosome);
int sort_by_fitness(unsigned long n, chromosome_type *ra);
int ga_main(chromosome_type *nichepool, chromosome_type *parentpool,
          chromosome_type *offspringpool, int generation,
          niche_type niche, int node_rank, int nodes_avail);
int ga_stagnation(chromosome_type *nichepool, int generation,
               niche_type niche);
int ga_migration(chromosome_type *population, int generation,
              niche_type *niche, int node_rank);
```

Declares the external types;

```
genetic_algorithm_type ga;
run_param_type rp;
input_output_type io;
design_type des;
```

4.3 File name: **init_global_struct.c**

Description;
Used to initialise the generic global structures, `ga` (parameters relating to the execution of the genetic algorithm, eg niche size), `rp` (parameters relating to the execution of the program, eg time left available), and `io` (parameters relating to input and output of data eg level of verbose logging). The function uses the services of `init_chrom_struct` to initialise the design specific global variable structure des.

Contains;

```
Externally visible functions;
int init_global_struct(void)
```

Uses the services of;

```
int init_chrom_struct(void);
```

Uses external types;

```
extern genetic_algorithm_type ga;
extern run_param_type rp;
extern input_output_type io;
```

4.4 File name: **comline_parse.c**

Description;
Parses the command line arguments supplied from the main program and writes the arguments directly into the global variable structures. Either short options (`getopt`) or short options plus long options (`getopt_long_only`) are used, depending on the compiler directive `_LONGOPT`. This has been included for portability of the code, where long options are not permitted on some systems (eg Babbage).

Contains;

```
Externally visible functions;
int comline_parse(int argc, char **argv);

Local functions;
int convert_num(double *numval,char *numptr);
```

Uses the services of;

```
No services used
```

Uses external types;

```
extern genetic_algorithm_type ga;
extern run_param_type rp;
extern input_output_type io;
```

4.5 File name: **display_help.c**

Description;
Contains a single function to display the help file to screen on request.

Contains;

```
Externally visible functions;
int display_help(void);
```

Uses the service of;

```
No services used
```

Uses external types;

```
extern genetic_algorithm_type ga;
extern run_param_type rp;
extern input_output_type io;
```

4.6 File name: **logger_debug.c**

Description;
A package of functions to control all logging to file during program execution. Writing to the log file only be carried out using the services offered within the package, eg the file pointer is not visible outside of the package. Three levels of debug logging can be set at run time using the global variable `io.verbosity`, these are minimum, modest and maximum. Modest verbosity is recommended for normal operation, with maximum verbosity recommended for debugging purposes only.

Contains;

```
Externally visible functions;
int debug_file_open(int stamp);
int debug_file_reopen(int stamp);
int debug_file_close(int stamp);
int debug_log_params(print_type level);
int debug_log_command_line(print_type level, int argc, char *argv[]);
int debug_log_fitness_funct(print_type level);
int debug_log_chrom(print_type level, chromosome_type chromosome);
int debug_log_line(print_type level, char *strptr, ...);
```

Uses the service of;

```
No services used
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
extern input_output_type io;
```

4.7 File name: **logger_data.c**

Description;

A package of functions used to control all logging to data file during program execution. Writing to the data file can again only be carried out using the services offered within the package. The primary use of the data file is to record in binary format all chromosomes in the population so that subsequent runs of the program can start from an already predefined generation number.

Contains;

```
Externally visible functions;
int data_file_open(char *str);
int data_file_close(void);
int data_log_info(void);
int data_log_chrom(chromosome_type chromosome);
int data_read_chrom(chromosome_type *chromosome);
```

Uses the service of;

```
No services used
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
extern input_output_type io;
```

4.8 File name: **random_generators.c**

Description;

A package containing two random number generators, a uniform deviate produced by the function ran1, and a normally distributed deviated produced by the function gasran. The functions were taken from Numerical Recipes in C, Second Edition, pgs 280 and 289 respectively

Contains;

```
Externally visible functions;
double gasran(long *idum)
double ran1(long *idum)
```

Uses the service of;

```
No services used
```

Uses external types;

```
None used
```

4.9 File name: **ga_main.c**

Description:
Controls the implementation of the genetic algorithm to produce the next generation of chromosomes for a given niche. For parallel execution the evolution of a niche is carried out by either a single node, or by several nodes. Where several nodes are used a node is assigned to be the control node (node rank = 0) or a slave node (node rank > 0). In this situation the control node handles the selection, mating, mutation and replacement operations and the slave nodes are used only to share in the fitness evaluation.

Contains;

```
Externally visible functions;
int ga_main(chromosome_type *nichepool, chromosome_type *parentpool,
          chromosome_type *offspringpool, int generation,
          niche_type niche, int node_rank, int nodes_avail)
```

Uses the services of;

```
int debug_log_line(print_type level, char *strptr, ...);
int debug_log_chrom(print_type level, chromosome_type chromosome);
int sort_by_fitness(unsigned long n, chromosome_type *ra);
int fitness_eval(chromosome_type *chromosome, int check);
int ga_selection(chromosome_type *nichepool, chromosome_type *parentpool);
int ga_mating(chromosome_type *parentpool, chromosome_type *offspringpool);
int ga_mutation(chromosome_type *offspringpool);
int ga_replacement(chromosome_type *offspringpool,
                chromosome_type *nichepool);
```

Uses the external types;

```
extern genetic_algorithm_type ga;
extern run_param_type rp;
```

4.10 File name: **ga_selection.c**

Description:
Used to select chromosomes from the nichepool and place them into the parentpool ready to be mated. Selection is made using either random selection, a weighted roulette wheel approach or tournament selection with 2, 3, or 4 chromosomes taking part in each tournament. The selection strategy used is determined at run time and stored in the global variable `ga.select`.

Contains;

```
Externally visible functions;
int ga_selection(chromosome_type *nichepool, chromosome_type *parentpool);

Local functions;
int ga_select_randomly(chromosome_type *nichepool,
                   chromosome_type *parentpool);
int ga_select_roulette(chromosome_type *nichepool,
                   chromosome_type *parentpool);
int ga_select_tournament(chromosome_type *nichepool,
                    chromosome_type *parentpool);
```

Uses the services of;

```
No services used
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
```

4.11 File name: **ga_mating.c**

Description:
Used to manage the creation of the offspringpool by mating chromosomes in the parentpool. Parents are chosen to be mated at random from the parentpool and can produce either one or two children during the mating procedure. Should it prove impossible to create an allowed child after 100 mating attempts, a parent is chosen to take the place of the child in the offspringpool.

Contains;

```
Externally visible functions;
int ga_mating(chromosome_type *parentpool, chromosome_type *offspringpool);

Local functions;
int ga_mate_randomly(chromosome_type *parentpool,
                  chromosome_type *offspringpool);
```

Uses the services of;

```
int debug_log_line(print_type level, char *strptr, ...);
int debug_log_chrom(print_type level, chromosome_type chromosome);
int ga_mate_chrom(chromosome_type par1, chromosome_type par2,
               chromosome_type *child1, chromosome_type *child2 );
int chrom_check(chromosome_type *chromosome);
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
extern input_output_type io;
```

4.12 File name: **ga_mate_chrom.c**

Description:

Used to mate two parent chromosomes in order to produce child offspring. Mating occurs by randomly choosing one or two cross over points within the genetic structure of the parent chromosomes. The number of cross over points is determined at run time by the global variable `ga.ncrossing`. The child offspring are created by exchanging segments of the parents genetic structure in the following manner-

```
single cross over -    Parents -   A A A A A A|A A , B B B B B B|B B
                       Offspring - B B B B B B|A A,  A A A A A A|B B
double cross over -    Parents -   A A A|A A A|A A , B B B|B B B|B B
                       Offspring - A A A|B B B|A A , B B B|A A A|B B
```

To be generic in nature, pointer arithmetic and the format specifier `ga.gene_struct` are used to locate the cross over points, and exchange the chromosome segments.

Contains;

```
Externally visible functions;
int ga_mate_chrom(chromosome_type par1, chromosome_type par2,
               chromosome_type *child1, chromosome_type *child2 )

Local functions;
void int_to_double(void);
void double_to_int(void);
```

Uses the services of;

```
int debug_log_line(print_type level, char *strptr, ...);
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
```

4.13 File name: **ga_mutation.c**

Description:
Used to control the mutation of components within a chromosome. The function loops through each component of the chromosome's genes and decides whether to mutate or flip the value of the component depending on if it is a mutatable (as specified by the global variable `ga.mut`) or flipable (as specified by the global variable `ga.mut_flip`) component. For mutatable components there is a finite probability of it undergoing 3 types of mutation, viz. spread, jump, or kick mutations where the value of the possible mutation increases in magnitude respectively.

Contains;

```
Externally visible functions;
int ga_mutation(chromosome_type *offspringpool);

Local functions;
int ga_mutate_randomly(chromosome_type *offspringpool);
```

Uses the services of;

```
int debug_log_line(print_type level, char *strptr, ...);
int ga_mutate_chrom(chromosome_type *chromosome, int ngene, int ncomp,
                int mut_type)
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
```

4.14 File name: **ga_mutate_chrom.c**

Description:
Used to mutate the value of a single component within a chromosome gene. The function handles all four possible types of mutation, namely spread, jump, kick and flip mutations. For spread and jump mutations, the amount of the mutation is governed by the global structure `ga.mut_spread` and `ga.mut_jump`. For kick mutation the limits of the mutation are governed by global structures `ga.kick_upper` and `ga.kick_lower`. In all of these cases, if it is not possible to create an allowed chromosome after 100 mutation attempts, the chromosome is left unchanged. To be generic in nature, pointer arithmetic and the format specifier `ga.gene_struct` are used to locate the component to be mutated, and the mutation amount/limits.

Contains;

```
Externally visible functions;
int ga_mutate_chrom(chromosome_type *chromosome, int ngene, int ncomp,
                 int mut_type)

Local functions;
void double_to_integer(void);
void integer_to_double(void);
```

Uses the external services of;

```
int chrom_check(chromosome_type *chromosome);
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
```

4.15 File name: **ga_replacement.c**

Description:
Used to replace chromosomes in a nichepool with chromosomes from the offspring pool after the mating and mutation procedures are completed. Replacement is either by replacement of the fittest (so called top down), replacement of the weakest (so called bottom up), random replacement or replacement of the weakest parent. The exact replacement strategy used is determined at run time and stored in the global variable `ga.replace`. Where elitism is selected, all elitist chromosomes in the nichepool remain after the replacement procedure.

Contains;

```
Externally visible functions;
int ga_replacement(chromosome_type *offspringpool,
                chromosome_type *nichepool)

Local functions;
int ga_replace_fittest(chromosome_type *offspringpool,
                  chromosome_type *nichepool);
int ga_replace_weakest(chromosome_type *offspringpool,
                  chromosome_type *nichepool);
int ga_replace_randomly(chromosome_type *offspringpool,
                  chromosome_type *nichepool);
int ga_replace_parent(chromosome_type *offspringpool,
                  chromosome_type *nichepool);
```

Uses the external services of;
None used.

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
```

4.16 File name: **ga_stagnation.c**

Description:
Used to re-initialise all chromosomes within a niche after the niche has stagnated (there has been no improvement in the fittest member achieved in the niche in `ga.sgens` generations).

Contains;

```
Externally visible functions;
int ga_stagnation(chromosome_type *nichepool, int generation,
               niche_type niche)
```

Uses the external services of;

```
int init_chrom_random(chromosome_type *chromosome);
int sort_by_fitness(unsigned long n, chromosome_type *ra);
int debug_log_line(print_type level, char *strptr, ...);
int debug_log_chrom(print_type level, chromosome_type chromosome);
```

Uses external types;

```
extern genetic_algorithm_type ga;
```

4.17 File name: **ga_migration.c**

Description:
Used to perform migration between all niches of the population. The number of migrations to be carried out is set at run time and stored in the global variable `ga.nmigrants`. Niches are chosen at random from within the population and migration is performed either using a random approach (a random chromosome from the export niche takes the place of a random chromosome in the import niche) or a next fit approach (a chromosome from the export niche is chosen such that it will be as close to, but not better than, the fittest chromosome in the import niche). Where elitism is selected, all elitist chromosomes in a niche cannot be replaced by a migrant. For parallel execution all control nodes broadcast their chromosome data to node 0 which performs the migration operations. After completion node 0 broadcasts the new chromosome information back to the control nodes.

Contains;

```
Externally visible functions;
int ga_migration(chromosome_type *population, int generation,
              niche_type *niche, int node_rank)

Local functions;
int ga_migrate_nextfit(chromosome_type *population, niche_type *niche);
int ga_migrate_randomly(chromosome_type *population, niche_type *niche);
```

Uses the external services of;

```
int debug_log_line(print_type level, char *strptr, ...);
int debug_log_chrom(print_type level, chromosome_type chromosome);
```

Uses external types;

```
extern run_param_type rp;
extern genetic_algorithm_type ga;
```

4.18 File name: **ga_sortings.c**

Description:
Used to sort the chromosomes in a niche into descending order with respect to their fitness values. The package contains a single function based on the Heapsort routine as given in Numerical Recipes in C, Second Edition, page 336.

Contains;

```
Externally visible functions;
int sort_by_fitness(unsigned long n, chromosome_type *ra)
```

Uses the external services of;
None Used

Uses external types;
None used

4.19 File name: **ga_sharing.c**

Description:
Used to recalculate the fitness values of all chromosomes in a niche based on their separation, or the number of chromosomes in close proximity. Although given here the function has been used only in an experimental manner – it is called from within the `ga_main` function, although at present this code has been commented out.

Contains;

```
Externally visible functions;
int ga_sharing(int niche_size, chromosome_type *nichepool)
```

Uses the external services of;

```
double metric(chromosome_type chr1, chromosome_type chr2)
```

Uses external types;

```
extern genetic_algorithm_type ga;
```

5. Summary of the design specific components

5.1 File name: **design.h**

Description:
The header file for all design specific components of the software. The file declares all structure definitions and prototypes all functions to be externally visible.

Type and structure definitions;

```
gene_type;              /* Defines the structure of a single gene */
chromosome_type;        /* Defines the structure of a single chromosome */
design_type;            /* Defines all design specific parameters */
```

Prototypes of all externally visible functions;

```
int init_chrom_struct(void);
int init_chrom_random(chromosome_type *chromosome);
int chrom_check(chromosome_type *chromosome);
double metric(chromosome_type chr1, chromosome_type chr2)
int design_log_params(FILE *debug_fp);
int design_log_chrom_params(FILE *debug_fp, chromosome_type chromosome);
int fitness_eval(chromosome_type *individual, int check);
int fitness_funct_string(char **strptr);
```

5.2 File name: **design_init.c**

Description:
Used to initialise the design specific global variables, to randomly initialise a chromosome, to check that a chromosome is allowed (ie it does not violate any boundary conditions), to determine how similar two chromosomes are (their metric value), and to log the design specific parameters to log file on request. These functions must be included and written specifically for all designs requiring optimisation.

Contains;

```
Externally visible functions;
int init_chrom_struct(void);
int init_chrom_random(chromosome_type *chromosome);
int chrom_check(chromosome_type *chromosome);
double metric(chromosome_type chr1, chromosome_type chr2)
int design_log_params(FILE *debug_fp);
int design_log_chrom_params(FILE *debug_fp, chromosome_type chromosome);

Local functions;
gene_type make_gene(double val1, double val2, int val3);
```

Uses the external services of;

```
int fitness_eval(chromosome_type *individual, int check);
```

Uses external types;

```
extern design_type des;
extern genetic_algorithm_type ga;
extern run_param_type rp;
```

5.3 File name: **design_fit.c**

Description:
Used to determine the fitness of a single chromosome. The function can be used to either determine the fitness value and assign the value into the structure of the chromosome (check = 0), or to check that the fitness in the structure of the chromosome corresponds to the chromosome (check =1, used for debugging and validation).

Contains;

```
Externally visible functions;
int fitness_eval(chromosome_type *chromosome, int check)
```

Uses the external services of;

```
Any functions included in the file design_functions.c
```

Uses external types;

```
extern design_type des;
extern genetic_algorithm_type ga;
```

5.4 File name: **design_functions.c**

Description:
This file has been included as a specific location where all functions relevant to the determination of a chromosomes fitness value can be placed.

Contains;

```
Externally visible functions;
Dependent on design.
```

Uses the external services of;

```
Dependent on design
```

Uses external types;

```
extern design_type des;
```