The "Robby the Robot" Genetic Algorithm In C

Melanie Mitchell

Computer Science Department
Portland State University
and
Santa Fe Institute

This file describes the "Robby the Robot" Genetic Algorithm in C package, written by Melanie Mitchell, with contributions (via earlier code) from Jonathan Amsterdam, Peter Hraber, and Terry Jones. Each of the files in the directory is described here. Note that this code was written for demonstration purposes rather than for efficiency, generality, or good programming style. It is not guaranteed to be bug free, nor will it be maintained by any of the people listed above. This document assumes that the user is familiar with the basic vocabulary and structure of genetic algorithms. Note: This code is known to run on Linux platforms, but may require changes to the random number generation code to run on other platforms.

To run the program (on a Linux machine):

1. Compile the program by simply typing "make". This will produce an executable binary called "robby".

3. Edit the file "params" to set the parameters for the program.

The parameters you must change are

RUN_NUM_DIR <path to directory containing the "run_num" file>
OUTPUT_DIR <path to directory where you want the output files to be written>

4. To run the program, simply type "robby". To run it with a given random number seed, type "ga-c -r <seed>". Each ".header" file contains the random number seed used to produce that run. Running the program will also create a ".long" and a ".short" file, giving the long and short output for the run. You can turn off printing of these output files by setting the appropriate flags in the params file to FALSE.

Here are descriptions of the source code files:

definitions.h: This file defines constants and constant flags for the GA. It also declares as EXTERN the main parameters for the GA, which are defined in the file "params".

globals.h: This file defines the global data structures and variables for the GA. The structure type "ga_rec_type" is the data structure encoding individuals in the GA population.

prototypes.h: This file declares the functions used throughout the GA.

params.h: This file supports storing parameters in a file (here "params") and reading parameters from that file.

params.c: This file implements storing parameters in a file (here "params") and reading parameters from that file.

params: This file contains the parameters for the GA. Each parameter is declared as EXTERN in the file "definitions.h" and is declared by type in the file "main.c". This file lists the parameter names and values, one name and value per line, to be specified by the user. Comments in this file are delimited by the symbol "#".

The possible SELECTION_METHODs are "fitness proportionate", "pure rank", "linear rank", "sigma scaling", and "elite". Elite selection ranks the individuals in order of fitness, copies the top NUM_ELITE individuals directly to the new population, and fills in the rest of the new population with offspring from parents chosen at random from the top NUM_ELITE.

At present, the only type of crossover done is single-point.

For I/O, each run is given a unique run number. You need to create a file called "run_num" with the initial run number. This number will be incremented by 10 each time a run is done. The RUN_NUM_DIR gives the complete pathname for the directory containing run_num.

The program will output four kinds of files: ".header"., ".short", ".long", and ".best" files. E.g., if the run number is 1234, the output files will be "1234.header", "1234.short", "1234.long", and "1234.best".

The ".header" file contains the values of the parameters that were used in this run.

The ".short" file contains, for each generation, the average fitness, the standard deviation of the average fitness, and the best fitness in the population at that generation. This file is output if SHORT_PRINT is TRUE.

The ".long" file contains an entry for every member of the population at each generation. The entry consists of the generation number, the individual's id number (e.g., individual 12.32 was born in generation 12 and was the 32nd individual to be born in that generation; each individual in the entire run has a unique id number of that form), its fitness, and its chromosome (a bit string). This file is output if LONG_PRINT is TRUE. **Warning: This will result in very large output files!**

The ".best" file is similar to the ".long" file but contains entries only for the highest-fitness individual in each generation.

fitness.c: This file defines the fitness function being used. The function "init_fitness_function" initializes variables associated with the fitness function, and the function "calc_fitness" takes a chromosome as an input and returns a double float, which is the numerical value of fitness. Note that chromosomes here are represented as arrays of chars; if they are bit strings the entries can be the chars '1' and '0', defined as ONE and ZERO in the file "definitions.h".

ga.c: This file contains functions implementing the genetic operators and other operations needed to run the GA.

runga.c: This file contains functions to run the GA, using the operators defined in the file "ga.c". This is the implementation of the basic GA algorithm. An initial population is generated, and the GA is run until MAX_NUM_FUNCTION_EVALS fitness function evaluations have been carried out. At each generation, the fitness of each individual in the population is calculated, using the function calc_fitness (defined in "fitness.c"). The population is then sorted by fitness, and selection of parents is carried out according to whatever selection method has been specified. Pairs of parents are chosen, are crossed over according to CROSSOVER_RATE, and the offspring are mutated according to MUTATION_RATE. This creates a new population, and the process is repeated.

main.c: This file implements the control loop for carrying out runs of the GA. It also contains functions to declare the parameters and their types, and to parse the command line.

files.c: This file contains functions for I/O.

random_number_generation.c: This file contains functions that implement the random number generator. This code uses Knuth's subtractive-method portable random number generator, described in the first edition of Numerical Recipes in C, p. 212.

misc.c: This file contains some miscellaneous utility functions for memory management.