Acknowledgments

The authors gratefully acknowledge support provided by NASA under Grant NGT–50224 and support provided by the National Science Foundation under Grant CTS–8451610. We also thank Hillol Kargupta for donating his tournament selection implementation.

References

Booker, L. B. (1982). Intelligent behavior as an adaptation to the task environment (Doctoral dissertation, Technical Report No. 243. Ann Arbor: University of Michigan, Logic of Computers Group). *Dissertations Abstracts International, 43(2)*, 469B. (University Microfilms No. 8214966)

Brindle, A. (1981). *Genetic algorithms for function optimization.* Unpublished doctoral dissertation, University of Alberta, Edmonton, Canada.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison–Wesley.

Knuth, D. E. (1981). *The Art of Computer Programming* (2nd, vol. 2 ed.). Reading, MA: Addison–Wesley.

- The chromosome length (`int`).
- Print the chromosome strings each generation (`y/n`)?
- The maximum number of generations for the run (`int`).
- The probability of crossover (`float`).
- The probability of mutation (`float`).
- Application-specific input, if any.
- The seed for the random number generator (`float`).

3.2 Chromosome Representation and Memory Utilization

SGA-C uses a machine level representation of bit strings to increase efficiency. This allows crossover and mutation to be implemented as binary masking operations (see `operators.c`). Every chromosome (as well as the population arrays and some auxiliary memory space) are allocated dynamically at run time. The dynamic memory allocation scheme allocates a sufficient number of unsigned integers for each population member to store bits for the user-specified chromosome length. Because of this feature, it is extremely important that `BITS_PER_BYTE` be properly set (in `sga.h` and `external.h`) for your machine's hardware and C compiler.

4 Implementing Application Specific Routines

To implement a specific application, you should only have to change the file `app.c`. Section 2 describes the routines in `app.c` in detail. If you use additional variables for your specific problem, the easiest method of making them available to other program units is to declare them in `sga.h` and `external.h`. However, take care that you do not redeclare existing variables.

Two example applications files are included in the SGA-C distribution. The file `app1.c` performs the simple example problem included with the Pascal version; finding the maximum of x^{10}, where x is an integer interpretation of a chromosome. A slightly more complex application is include in `app2.c`. This application illustrates two features that have been added to SGA-C. The first of these is the **ithruj2int** function, which converts bits i through j in a chromosome to an integer. The second new feature is the **utility** pointer that is associated with each population member. The example application interprets each chromosome as a set of concatenated integers in binary form. The lengths of these integer fields is determined by the user-specified value of **field_size**, which is read in by the function `app_data()`. The field size must be less than the smallest of the chromosome length and the length of an unsigned integer. An integer array for storing the interpreted form of each chromosome is dynamically allocated and assigned to the chromosome's **utility** pointer in `app_malloc()`. The `ithruj2int` routine (see `utility.c`) is used to translate each chromosome into its associated vector. The fitness for each chromosome is simply the sum of the squares of these integers. This example application will function for any chromosome length.

5 Final Comments

SGA-C is intended to be a simple program for first-time GA experimentation. It is not intended to be definitive in terms of its efficiency or the grace of its implementation. The authors are interested in the comments, criticisms, and bug reports from SGA-C users, so that the code can be refined for easier use in subsequent versions. Please email your comments to **rob@comec4.mh.ua.edu**, or write to TCGA:

The Clearinghouse for Genetic Algorithms
The University of Alabama
Department of Engineering Mechanics
P.O. Box 870278
Tuscaloosa, Alabama 35487

For modularity, each selection method is made available as a compile time option. Edit `Makefile` to choose a selection method. Each of the three selection files contains the routines `select_memory` and `select_free` (called by `initmalloc` and `freeall`, respectively), which perform any necessary auxiliary memory handling, and the routines `preselect()` and `select()`, which implement the particular selection method.

`stats.c` contains the routine `statistics()`, which calculates populations statistics for each generation.

`utility.c` contains various utility routines. Of particular interest is the routine `ithruj2int()`, which returns bits i through j of a chromosome interpreted as an `int`.

`app.c` contains application dependent routines. Unless you need to change the basic operation of the GA itself, you should only have to alter this file Further instructions for altering the SGA application are included in section 4.

`application()` should contain any application-specific computations needed before each GA cycle. It is called by `main()`.

`app_data()` should ask for and read in any application-specific information. This routine is called by `init_data()`.

`app_malloc()` should perform any application-specific calls to `malloc()` to dynamically allocate memory. This routine is called by `initmalloc()`.

`app_free()` should perform any application-specific calls to `free()`, for release of dynamically allocated memory. This routine is called by `freeall()`.

`app_init()` should perform any application-specific initialization needed. It is called by `initialize()`.

`app_initreport()` should print out an application-specific initial report before the start of generation cycles. This routine is called by `initialize()`.

`app_report()` should print out any application-specific output after each GA cycle. It is called by `report()`.

`app_stats()` should perform any application-specific statistical calculations. It is called by `statistics()`.

`objfunc(critter)` The objective function for the specific application. The variable `critter` is a pointer to an `individual` (a GA population member), to which this routine must assign a fitness. This routine is called by `generation()`.

`Makefile` is a UNIX makefile for SGA-C.

3 New Features of SGA-C

SGA-C has several features that differ from those of the Pascal version. One is the ability to name the input and output files on the command line, i.e. `sga my.input my.output`. If either of these files is not named on the command line, SGA-C assumes `stdin` and `stdout`, respectively. Another new feature of SGA-C is its method of representing chromosomes in memory. SGA-C stores its chromosomes in bit strings at the machine level. Input-output and chromosome storage in SGA-C are discussed in the following sections.

3.1 Input-Output

SGA-C allows for multiple GA runs. When the program is executed, the user is first prompted for the number of GA runs to be performed. After this, the quantity of input needed depends on the selection routine chosen at compile-time, and any application-specific information required. When compiled with roulette wheel selection, the input requested from the user is as follows:

- The number of GA runs to be performed (`int`).
- The population size (`int`).

`initialize()` is the central initialization routine called by `main()`.

`initdata()` is a routine to prompt the user for SGA parameters.

`initpop()` is a routine that generates a random population. Currently, SGA-C includes no facility for using seeded populations.

`initreport()` is a routine that prints a report after initialization and before the first GA cycle.

`memory.c` contains routines for dynamic memory management.

`initmalloc()` is a routine that dynamically allocates space for the GA population and other necessary data structures.

`freeall()` frees all memory allocated by `initmalloc()`.

`nomemory()` prints out a warning statement when a call to `malloc()` fails.

`operators.c` contains the routines for genetic operators.

`crossover()` performs single-point crossover on two mates, producing two children.

`mutation()` performs a point mutation.

`random.c` contains random number utility programs, including:

`randomperc()` returns a single, uniformly-distributed, real, pseudo-random number between 0 and 1. This routine uses the subtractive method specified by Knuth (1981).

`rnd(low,high)` returns an uniformly-distributed integer between `low` and `high`.

`rndreal(low,high)` returns an uniformly-distributed floating point number between `low` and `high`.

`flip(p)` flips a biased coin, returning 1 with probability `p`, and 0 with probability `1-p`.

`advance_random()` generates a new batch of 55 random numbers.

`randomize()` asks the user for a random number seed.

`warmup_random()` primes the random number generator.

`noise(mu, sigma)` generates a normal random variable with mean mu and standard deviation sigma. This routine is not currently used in SGA-C, and is only included as a general utility.

`randomnormaldeviate()` is a utility routine used by noise. It computes a standard normal random variable.

`initrandomnormaldeviate()` initialization routine for `randomnormaldeviate()`.

`report.c` contains routines used to print a report from each cycle of SGA-C's operation.

`report()` controls overall reporting.

`writepop()` writes out the population at every generation.

`writechrom()` writes out the chromosome as a string of ones and zeroes. In the current implementation, the most significant bit is the *rightmost* bit.

Three selection routines are included with the SGA-C distribution:

`rselect.c` contains routines for roulette-wheel selection.

`srselect.c` contains the routines for stochastic-remainder selection (Booker, 1982).

`tselect.c` contains the routines for tournament selection (Brindle, 1981). Tournaments of any size up to the population size can be held with this implementation[2].

[2]The tournament selection routine included with the distribution was written by Hillol Kargupta, of the University of Alabama.

SGA-C: A C-language Implementation of a Simple Genetic Algorithm

Robert E. Smith
The University of Alabama
Department of Engineering Mechanics
Tuscaloosa, Alabama 35405

David E. Goldberg
The University of Illinois
Department of General Engineering
Urbana, Illinois 61801

Jeff A. Earickson
Alabama Supercomputer Network
The Boeing Company
Huntsville, Alabama 35806

March 2, 1994

1 Introduction

SGA-C[1] is a C-language translation and extension of the original Pascal SGA code presented by Goldberg (1989). It has some additional features, but its operation is essentially the same as that of the original, Pascal version. This report is included as a concise introduction to the SGA-C distribution. It is presented with the assumptions that the reader has a general understanding of Goldberg's original Pascal SGA code, and a good working knowledge of the C programming language. The report begins with an outline of the files included in the SGA-C distribution, and the routines they contain. The outline is followed by a discussion of significant features of SGA-C that differ from those of the Pascal version. The report concludes with a discussion of routines that must be altered to implement one's own application in SGA-C.

2 Files Distributed with SGA-C

The following is an outline of the files distributed with SGA-C, the routines contained in those files, and the `include` structure of the SGA-C distribution.

`sga.h` contains declarations of global variables and structures for SGA-C. This file is included by `main()`. Both `sga.h` and `external.h` have two `defines` set at the top of the files; `LINELENGTH`, which determines the column width of printed output, and `BITS_PER_BYTE`, which specifies the number of bits per byte on the machine hardware. `LINELENGTH` can be set to any desired positive value, but `BITS_PER_BYTE` must be set to the correct value for your hardware.

`external.h` contains external declarations for inclusion in all source code files except `main()`. The `extern` declarations in `external.h` should match the declarations in `sga.h`.

`main.c` contains the main SGA program loop, `main()`.

`generate.c` contains `generation()`, a routine which generates and evaluates a new GA population.

`initial.c` contains routines that are called at the beginning of a GA run.

[1]**Disclaimer:** SGA-C is distributed under the terms described in the file `NOWARRANTY`. These terms are taken from the GNU General Public License. This means that SGA-C has no warranty implied or given, and that the authors assume no liability for damage resulting from its use or misuse.

SGA-C: A C-language Implementation of a Simple Genetic Algorithm

by

Robert E. Smith
The University of Alabama

David E. Goldberg
The University of Illinois
and
Jeff A. Earickson
Alabama Supercomputer Network

TCGA Report No. 91002
March 2, 1994

The Clearinghouse for Genetic Algorithms
The University of Alabama
Department of Engineering Mechanics
Tuscaloosa, AL 35487