GAGS 0.91 : User's Manual[1]

J. J. Merelo, jmerelo@kal-el.ugr.es
Electronics & Computer Technology Dept.
Granada University.
18071 Granada (Spain)
Phone: +34-58-243162
Fax: +34-58-243230
14 June 1994

1. Description: what you got in your hands

GAGS (Genetic Algorithms from Granada, Spain) is a library and companion programs written and designed to take the heat out of designing a genetic algorithm. It features a class library for genetic algorithm programming (described in a separate document, *GAGS 0.9 Programmer´s manual)*, but, from the user point of view, is a genetic algorithm application generator. Just write the function you want to optimize and GAGS surrounds it with enough code to have a genetic algorithm up and running, compiles it, and runs it.

There are three levels in GAGS use:

1. *User*, who runs the application generator.
2. *Programmer*, that uses the classes in the class library in her own programs.
3. *Wizard*, who extends GAGS class library.

GAGS availability

GAGS is (for the time being) available at `ftp://kal-el.ugr.es/pub`, under the name `GAGS-0.91.tar.gz`. An uu-

[1] Document version 0.9.2

encoded version is also available, gags.uu, so that you can request it from your favorite ftp-mail server. If you don´t know of any, try `kal-el` server: send a e-mail message to `ftpmail@kal-el.ugr.es`, with the following contents

```
open
get gags.uu
close
```

and it will send you back a ~150K message with the uu-encoded version of GAGS.

Uncompressing the archive file

The `.gz` extension means that it is compressed with GNU's `gzip`; you will need to have it handy in order to decompress it. In order to do that, you must issue at the UNIX prompt the orders

```
unix_prompt% gunzip GAGS-<version>.tar.gz|tar xvf -
```

or else, with older versions

```
unix_prompt% mv GAGS-<version>.tar.gz GAGS-<version>.tar.z
unix_prompt% gzip -d GAGS-<version>.tar.z
unix_prompt% tar xvf GAGS-<version>.tar
```

In any case, a GAGS directory will be created, with all the files and sources and whatnots.

2. Startin'

The first rule of an UNIX (wannabe) programmer is to use everything you have at hand, in order to make your life (and other's) easier. GAGS makes use of several UNIX utilities whose use is widespread, and you should make sure you have them in your `path` before even starting to install it.

Summing up:
To configure, check for programs and the type of machine
`unix_prompt% configure`
To compile the library
`unix_prompt% make`
To copy library and header files to chosen directories
`unix_prompt% make install`
To test if everything is OK with the supplied demos:
`unix_prompt% make demo`
To issue a bug report:
`unix_prompt% make bug-report`

One of these utilities is `g++` version 2.5.7. GAGS is written in C++, so that there's no other way out than having it. GAGS will probably work with some lesser versions, since I don't use any fancy things, but you better get the last version. Other equivalent AT&T C++

3.0-compliant compiler[2] will also be OK. PERL is used in the application generator; you will not be able to use it if PERL is not installed (thanks, Larry Wall, for it![3]), and `gnuplot` is called for interactive graphical presentations. UNIX `cshell` must also be installed in `/bin/csh`, since the installation scripts have been written for this shell. The installation procedure checks this, so that it will probably complain if there is something missing.

So, firts of all, you must run the script `configure`, in this way

```
unix_prompt% configure
```

that checks for `gnuplot`, and `perl`, and tries to ascertain the type of machine. Only two kind of machines are checked, Sun or SGI. I have not the equipment to try GAGS in other machines, so that by default, if the machine is not an SGI, it guesses it is a Sun.

After checking that, you will want to edit the file `Makefile` and change the library, binary and include directories to the proper values for your system. If they all hang from the same base dir, you only need to change the `PREFIX` variable. The default values are OK, but you will probably need superuser privileges to write in those directories. Also probably, you will not trust these public domain programs found in some godforsaken place, and will want to try them first on some `tmp` directory. That's fine, but you will need to edit that file to change the default directories. These directories can be any, provided you have privvies to write in them.

Once the `Makefile` has been edited at will, run

```
unix_prompt% make
```

I have tested the program in Sun and SGI machines, and it runs OK, the only difference being that the Silicon Graphics does not need `ranlib`; there is also some difference in the way `install` works. So, if your machine uses `ranlib`, use `sun` Makefile; if not, use `sgi`. If

---

[2] Which basically means that it must be able to understand templates. I think that nowadays most C++ compilers are able to understand them.

[3] If you haven't got PERL, you will not be able to run the application generator, unluckily. Perl is not a compiled language, although PERL scripts can be "compiled" in some machines, like Suns. If you need a compiled version of the script for Suns, please tell me, and I'll make it available. Otherwise, I strongly encourage you to get PERL. PERL FAQ (frequently asked questions) is available at `ftp://convex.com/pub/perl/info/faq`, also at `ftp://rtfm.mit.edu/pub/usenet-by-groups/comp.lang.perl/perl-faq`.

you work in any other machine, and run into problems, please tell me. This will compile the library, creating a file called `libGAGS.a`. Then, by typing

```
unix_prompt% make install
```

files will be copied to their proper places (as indicated by the directories set in the `Makefile`). After running it, you are ready to run the application generator, called `gags.pl`, which will be in the $BINDIR directory. If you want to have it handy from then on, include that directory into your path. In any case, if you want to test if this works, run

```
unix_prompt% make demo
```

which will get into the `demos` directory and run the application generator on the examples.

There are 4 demo fitness function files, which show different possibilities. If you want to know about the fitness function file format, skip to the next section. These files are:

*`minsqu.ga` and `minsqu2.ga`: both fitness functions try to reach the center by minimizing the root square distance to all the 2-dimensional points contained in the `nube` (spanish for *cloud*) file. The chromosome for this problem will contain only 2 genes, corresponding to the coordinates of the central point.

*`1pt.ga`: this file contains the fitness function for the minimization of the distance to the point 0.5, 0.5. It does not use a training file.

*`smplfile.ga`: this function tries to minimize the absolute value distance to 3 points 0, 1 and 2, contained in the file `smplfile.tra` (tra stands for training).

Running the demos

If you just want to test whether GAGS runs or not in your machine, just press enter to every question that has a default. Then, a `gnuplot` window should pop up in your machine, showing the evolution of the fitness, and lots of numbers, representing the best genome and and fitness, should scroll up your console. For instance, if you are running the apps generator on `smplfile.ga`, the numbers should be about 0.5 (for fitness, which is 1/2, being 2 the distance of 1 to 0 and 2) and 1 (for best genome), or a close number. Bear in mind that you are using 2 bytes (the default), and finding the exact real number may not be possible within the discreteness of the genome representation. If your machine is a real machine (like the Silicon Graphics) you'll probably just see a window flash on the screen, and the prompt will return. Change the number of generations accordingly, to, for instance, 300, and you will at least see the window for a while. Of course, this large numberis not needed to run the GA properly, only 2 or 3 generations will allow it to find the optimum.

If instead of doing this, your screen goes blank, your favourite cat gets measles, the program dumps a 200 Meg core, or just plain does not run, you can complain by writing `make bug-report`. This will get you into the `mail` program, and you can then write a detailed report of what has happened, or at least complain loudly. I will be very grateful for it, and will try to fix it as soon as possible.

3. Running the application generator

As it has already been said by somebody, the application generator is called `gags.pl`. It queries the user about the characteristics of the genetic algorithm she wants to run; some of these characteristics (as few as possible) will be hardwired into the program, and some of them put into a file, that can be edited and altered from one run to the next. You do not need to know very much about GAs to run it, but at least you should be familiar with the basic concepts.

If you have installed headers and libraries (i.e., what you defined as directories LIBDIR and INCDIR) to the default places (like `/usr/local/lib, /usr/lib` or the like), you do not need to define any environment variable. But if you have not, for instance, if you have installed them in some test directory hanging from your home directory, two environment variables should be defined: GAGSLIB and GAGSINC. You can do this by writing, in the cshell:

```
unix_prompt% setenv GAGSLIB ~/tmp/GAGS/lib
unix_prompt% setenv GAGSINC ~/tmp/GAGS/inc
```

or whatever directories you previously defined in the Makefile. Besides, the directory in which `gags.pl` is contained must be somewhere in your path; otherwise you will have to use its absolute path to run it. Of course, if it is in your current directory, just type

```
unix_prompt% ./gags.pl
```

`gags.pl` generates a program that runs a genetic algorithm on the fitness function you supply, optionally using a training file, which you may also supply. Let's see what we are talking about. For GAGS, chromosomes are bit strings composed of *genes*. All genes have the same length in bytes, and its length is an integer amount of bytes. When fitness is evaluated, each gene is decoded to a number of a certain type (in the programming language sense), and with a certain range. GAGS asks you about some of these characteristics (range and number of bytes) and deduces others (like type and number of genes).

These chromosomes form a population, and GAGS asks about the number of individuals that compose it, although it can be varied from run to run. The genetic algorithm is run on the population for a number of generations.

3.1 Fitness function setup

Each generation, every individual is evaluated. The user (that means *you*) must write a function to evaluate the chromosome that will return the fitness for each chromosome, and save it in a file that is conventionally designed with the extension `.ga`. This file and function must follow the following guidelines.

*The file must be self-contained; that means that everything needed to run the function must be in the same file. This is not a good C programming practice, I know, but I'll have to give it a little bit more of thought. Next version will include file analysis, so that the objects or sources corresponding to the `#included` files will be also compiled. Honest, next version, no later.

*The function must be called `fitness`, and declared in this way

```
<returnType> fitness () {
      <function body>
}
```

where <returnType> must be substituted by some scalar type, like float or double.

*Each chromosome is evaluated by means of some function, or by reference to some (usually called *training)* file. Each element of the chromosome will be referenced by the variable `__gen[<index>]` (double underscore has been included to avoid that most undesirable effect called namespace pollution). The training file elements will be referenced by the variable `__sample[<fileIndex>][<vectorIndex>]`. For instance, if there is a file with 40 elements, each one of them a 2-element vector, and the chromosome includes 2 genes, it will be referenced in this way:

```
float dist;
for ( i = 0; i < FILESIZE; i ++ )
     dist += fabs(__sample[i][0]* __gen[0]) +
          fabs( __sample[i][1] + __gen[1] )
return dist;
```

FILESIZE will take the value of the number of vectors in the file, when the program is parsed by `gags.pl`.

*This function file is parsed to compute the sample vector size and the chromosome size. This is computed in two different ways: first, by looking at the biggest actual index of the `__gen` variable, and second, by looking at the biggest value a `for` loop variable can take, if `__gen` and `__sample` are referenced inside the loop. That means that you must use `__gen` and `__sample` in one of the two ways. I you want to cheat the procedure, you will probably succeed, so please don't try to do it. If you, dear user, have big troubles with this procedure, tell me, and I will try to make it smarter. The parsing procedure knows also about `#defined` constants, so that you can also indulge in its use, but not about `const` constants. Loop ending condition must only include 1 condition, with a < or <= sign (this last option is understood, but discouraged by The Wise Men That Know C++)

*If you have some doubt, take a look at the demos included in the release, inside the `demos` directory.

*If `__sample` is not used, GAGS understands you are not going to use a file; you will not be asked about it later, and the generated program will forget about it.

*If you define more functions inside the file, they may be written in C or C++. There will be *lots* of trouble if you use Kernigan & Ritchie (pre-ANSI) style, but this will arise later, when the program is compiled. Bear in mind that you shoul not name any other function `fitness`, although you can use this word inside any other construction (if it does not look much like a function declaration).

From this file, a program source file called `gags.cc` is created, that includes the `main()` function and all the functions necessary to run the genetic algorithm. This program will be then compiled to an object file named `gags` and executed (but only if you wanna).

3.2 Running the application generator.

GAGS looks for any file with extension `.ga` in the current directory, if it does not find any, or finds more than one, asks about the file that is going to be used; if there is exactly one, it takes it as the fitness function file.

After that, GAGS asks about the input file, if it has been referenced in the fitness function file. This file has the following format

```
comp1,1 comp1,2...... comp1,n
comp2,1 comp2,2...... comp2,n
..
```

that is, vectors separated by newline with its components separated by whitespace.

Then some questions about the genetic algorithm are asked. Almost all of them have a fixed default value, which does not mean that it will work well for *your* problem. Some of them will be saved to a file with an `.ini` extension, and some of them will be hardwired into the program. This file, written in plain ASCII, can be edited, and used in subsequent runs of the program.

‣**Number of bytes per parameter:** bytes used in the chromosome to encode each parameter. Goes to the `ini` file, under the name `locusSize`. The number of bytes must be strictly less than the number of bytes used to represent the `long` type in C. That means that the absolute maximum is about 7 for workstations and 3 for PCs.

▸**Parameter range for each gene:** that is, the interval to which the gene will be decoded at evaluation time. It varies for each gene; one parameter range will be asked for each gene. `ini` name: `startRange` and `endRange`. Bear in mind that the format in the `ini` file is different to the format you are queried.

▸**Mutation rate**: rate of mutation that each chromosome undergoes at reproduction time. Crossover rate is not asked, as 2-point crossover is performed every time 2 chromosomes mate, with the two points randomly selected.[4] `mutRate` inside the `ini`.

▸**Population size**: number of chromosomes in the population. If your genetic algorithm does not converge well, just throw more chromosomes at it™. `popSize` inside the `ini`.

▸**Gnuplot initialization string**: GAGS library includes a class that wraps around `gnuplot`, and pipes output to it (thanks, gnuplot authors). This way, I saved a lot of cross platform GUI development, with the cost of not having graphics output under MSDOS. Well, anyways, this string should include things like range setting, terminal setting, and so on. It won't do any harm if you write in the Silicon Graphics

```
set term iris4d 24
```

because the default term option is rather buggy, as well as ugly.

▸**Number of generations**. Ditto. `Generations` inside the `ini`.

▸**Selection method** Three selection methods have been included in the library, *roulette wheel* selection, *elitist* selection and *tournament selection*. Roulette wheel works as featured in Goldberg's *Genetic Algorithms*. Elitist selection eliminates a percentage of individuals with lowest fitness, and substitutes them by the offspring of those with highest fitness, mated with another random member of the population; this type of

[4] This does not mean it is the only option available in the GAGS library. Uniform crossover is also available; but the application generator does not know about it. Just check the library functions in the Programmer's Manual, available Real Soon Now.

selection is sometimes called *steady-state*. Tournament selection picks up a number of individuals at random, and selects the one with the highest fitness. As you see, some of these selection methods accept a parameter (tournament and elitist), and some not. This parameters will be asked, if you are running the apps generator, or must be supplied in the `ini` file, if you edit it. In this file, you must include the line

```
repType {ELITE|ROULWHEEL|TOURNAMENT} [repArg]
```

where `repArg` is a real number in the (0..0,5) open interval for elitist reproduction, and an integer number (often called *magic*) which ranges about 2-10 in TOURNAMENT selection, being 7 a good choice. ROULWHEEL does not require a parameter.

▸**Output interval** is the number of generations that passes between information update in the files. It is also hardwired, after all, you are not very likely to change opinion about this after some generations.

▸**Save best vector or fitness**: fitness refers to the fitness as evaluated by the function you have defined; the best vector is the decodification of the chromosome with this best fitness. Always, the best fitness is piped to `gnuplot`.

▸**Output file name**: if you want to save the results to show them later to your grandsons, give the file a name (no default); if not, it will be printed onto the screen.

And that is all. After all that, the file `gags.cc` is generated, compiled and linked to the `GAGS` library file previously created, and if everything works fine, you can run in this way

```
unix_prompt% gags filename.ini.
```

If `gags.pl` does not work on your particular fitness function, please mail it to me and I'll do what I can to fix it.

After this first and succesful run, the configuration file can be edited, to test different values. Have a look at the generated file, `gags.cc`, and see how the library functions are called, which objects are around and so on.

4. Coda

Well, I hope this li'l user's manual has been useful to you. Bear in mind that the library has much more than meets the eye, and its full potential can only be extracted by programming in C++ and using its functions. All objects and functions are fully documented in english, so that you can get a grasp of its work just by looking at the source file.

I hope also to hear from you. I have had always in mind the end user, you, and tried not to release it until it was pretty well finished. That's why I have christened it version 0.9, which basically means it does not coredumps every other while, and when it is Friday, but that I don't know if it completely meets the user needs.

Anyways, you must understand that this is still a beta product, as it has not been tested by many other people besides me. But if you find any bug/you have some special need/etc, etc, just drop me an (e-mail) line `(jmerelo@kal-el.ugr.es)`, or phone or fax me. If there are enough users around (which I strongly doubt), I'll set up and maintain a mailing list. If you have tried to compile GAGS for a different platform, and succeeded or not, tell me, and, if possible, send me patches and/or Makefile. It should compile OK on MSDOS, but graphics support will be lost. There is also a version of PERL available for MS-DOS, so that would pose no problem. Under request, I can make binary versions of the library, along with include files available. I also accept all kinds of suggestions about this manual.

The library is quite complete, but, from my point of view, lacks some GUI support (maybe *InterViews?*), as well as neural networks support. I have written some neural network classes, and I will make them available under request. It has been also tested in the MS-DOS environment, and it seems to run well, except for gnuplot. I will also make a GAGS*lib Programmer's Manual* available RSN.

If you want to offer some help or collaboration, I will not disdain it, quite the other way round.

Finally, I had quite a look at the user's manual of GAUCSD, by Nici Schraudolph, and the similitude of the *modus operandi* is more than casual. Thanks, Nici, for making it available to the net.

5. Acknowledgements

I am very grateful to my research group mate, Ignacio Rojas (`ignacio@casip.ugr.es`) who crash-tested the first version of the apps generator. I am also very grateful to M. Srinivar (`msrini@turing.miel.mot.com`), which pointed out some problems when gnuplot is not present. Thanks also to Silicon Graphics Spain which is lending me the programming environment in which I developed GAGS.