VERY FAST SIMULATED REANNEALING (VFSR) ©

Lester Ingber
Lester Ingber Research
P.O. Box 857, McLean, VA 22101
ingber@alumni.caltech.edu

and

Bruce Rosen
Division of Mathematics, Computer Science and Statistics
The University of Texas at San Antonio, San Antonio, TX 78249
rosen@ringer.cs.utsa.edu

1. License and Availability

1.1. GNU Copyleft License

This Very Fast Simulated Reannealing (VFSR) code is being made available under a GNU COPYING-LIB "copyleft" license, and is owned jointly by Lester Ingber and Bruce Rosen[1]. Please read the copy of this license contained in this directory. Our intent is to make this code available in the public domain to the widest audience while maintaining the integrity of the basic algorithm.

1.2. NETLIB Electronic Availability of VFSR

You can obtain our code from NETLIB or STATLIB. This can be done interactively or by electronic mail request.

1.2.1. Interactive from NETLIB

From your local machine login to research.att.com:

```
local% ftp research.att.com
Name (research.att.com: your_login_name): netlib
Password: [type in your_login_name or anything]
ftp> cd opt
ftp> binary
ftp> get vfsr.Z
ftp> quit
```

After 'uncompress vfsr.Z' read the header of vfsr for simple directions on obtaining your source files. For example, on most machines, after 'sh vfsr' they will reside in a VFSR.DIR directory.

1.2.2. Electronic Mail Request from NETLIB

Send the following one-line electronic mail request

```
send vfsr from opt
[For general NETLIB info, just use: send index]
```

to one of the NETLIB sites:

netlib@research.att.com	(AT&T Bell Labs, NJ, USA)
netlib@ornl.gov	(Oak Ridge Natl Lab, TN, USA)
netlib@ukc.ac.uk	(U Kent, UK)
netlib@nac.no	(Oslo, Norway)
netlib@cs.uow.edu.au	(U Wollongong, NSW, Australia)

1.2.3. Retrieve from STATLIB

Similarly, you may retrieve the file vfsr from STATLIB by sending the message

```
send vfsr from general
```

to statlib@lib.stat.cmu.edu. For interactive retrieval, ftp to lib.stat.cmu.edu, login as user statlib, cd to the directory general, and retrieve the file vfsr.

1.2.4. Updates

If you wish to be placed on our electronic mailing list to receive major updates between public announcements of new versions, please send email stating this request to ingber@alumni.caltech.edu.

2. Background and Context

VFSR was developed in 1987 to deal with the necessity of performing adaptive global optimization on multivariate nonlinear stochastic systems[2]. VFSR was recoded and applied to several complex systems, in combat analysis[3], finance[4], and neuroscience[5]. A comparison has shown VFSR to be superior to a standard genetic algorithm simulation on a suite of standard test problems[6], and VFSR has been examined in the context of a review of methods of simulated annealing[7]. A project comparing standard Boltzmann annealing with "fast" Cauchy annealing with VFSR has concluded that VFSR is a superior algorithm[8]. A paper has indicated how this technique can be enhanced by combining it with some other powerful algorithms[9].

2.1. Efficiency Versus Necessity

VFSR is not necessarily an "efficient" code. For example, if you know that your cost function to be optimized is something close to a parabola, then a simple gradient Newton search method most likely would be faster than VFSR. VFSR is believed to be faster and more robust than other simulated annealing techniques for *most* complex problems with multiple local optima; again, be careful to note that some problems are best treated by other algorithms. If you do not know much about the structure of your system, and especially if it has complex constraints, and you need to search for a global optimum, then we heartily recommend our VFSR code to you.

Since VFSR has not been applied extensively to many systems, it is certain that there is much research to be done on determining optimal or even reasonable VFSR parameters, e.g., the set of DEFINE_OPTIONS, for different classes of systems, especially in higher dimensional spaces of user parameters. A major purpose of placing this code in the public domain is to motivate more of this research, and thus make the code more useful to a wider audience.

2.2. Outline of Use

Set up the VFSR interface: Your program should be divided into two basic modules. (1) The user calling procedure, containing the cost function to be minimized (or its negative if you require a global maximum), here is contained in user.c and user.h. (2) The VFSR optimization procedure, here is contained in vfsr.c and vfsr.h. Furthermore, there are some options to explore in the Makefile. We assume there will be no confusion over the standard uses of the term "parameter" in different contexts, e.g., as an element passed by a subroutine or as a physical coefficient in a cost function. We have indented all C files using GNU indent as 'indent -kr -sob'.

We will make every reasonable effort to maintain only the vfsr module, to permit it to compile without "error," not necessarily without compiler "warnings." The user module is offered only as a guide to accessing the vfsr module. The NOTES file will contain updates for some standard machines.

In VFSR.DIR/TESTS we have included some user_out files from some sample runs, containing timed runs on a Sun4c/4.1.3 (SPARC-2) using compilers cc, acc and gcc, and on a Dec3100/Ultrix-4.2 using compilers cc and gcc. No attempt was made to optimize the use of any of these compilers, so that the runs do not really signify any testing of these compilers or architectures; rather they are meant to be used as a guide to determine what you might expect on your own machine.

3. Makefile

This file was generated using 'make doc'. The Makefile contains some options for formatting this file differently, including the PostScript version README.ps and the text version README. Use 'make' or 'make all' to compile and run vfsr_run, the executable we have prepared for our test function.

Examine the Makefile to determine the "clean" options available.

Since complex problems by their nature are often quite unique, it is unlikely that our default parameters are just right for your problem. However, our experience has shown that if you *a priori* do not have any reason to determine your own parameters, then you might do just fine using our defaults, and we recommend using them as a first-order guess. Our defaults can be changed simply by uncommenting lines in the Makefile or by changing structure elements in the user module. Remember to recompile the entire code every time you change any options in the Makefile. Depending on how you integrate VFSR into your own user modules, you may wish to modify this Makefile or at least use some of these options in your own compilation procedures.

Read through all the options in the Makefile. As the comments therein suggest, it may be necessary to change some of them on some systems. Here are just a couple of examples you might consider:

3.1. SMALL_FLOAT

SMALL_FLOAT is a measure of accuracy permitted in log and divide operations in vfsr, i.e., which is not precisely equivalent to a given machine's precision. There also are DEFINE_OPTIONS in the Makefile to separately set constants for minimum and maximum reals permitted by your machine. This affects checks on exponential arguments. Experts who require the very best precision can fine-tune these parameters in the code.

Such issues arise because the fat tail of VFSR, associated with high parameter temperatures, is very important for searching the breadth of the ranges especially in the initial stages of search. However, the parameter temperatures require small values at the final stages of the search to converge to the best solution, albeit this is reached very quickly given the exponential schedule proven in the referenced publications to be permissible with VFSR. Note that our test problem in user.c is a particularly nasty one, with 1E20 local minima and requiring VFSR to search over 12 orders of magnitude of the cost function before correctly finding the global minimum.

In VFSR.DIR/TESTS We have included vfsr_out files comparing results using SMALL_FLOAT=1.0E-12, SMALL_FLOAT=1.0E-18 (the default), and SMALL_FLOAT=1.0E-20. The intermediate values of all differ, and the final results of SMALL_FLOAT-1.0E-12 differ from the others. The SMALL_FLOAT=1.0E-20 calculation on a Convex-C120 crashed when required to handle the greater precision.

3.2. HAVE_ANSI

As another example, setting HAVE_ANSI=FALSE will permit you to use an older K&R C compiler. This option can be used if you do not have an ANSI compiler, overriding the default HAVE_ANSI=TRUE.

4. User Module

We have set up this module as user.c and user.h. You may wish to combine them into one file, or you may wish to use our VFSR module as one component of a library required for a large project.

4.1. int main(int argc, char **argv)

In main, set up your initializations and calling statements to vfsr. In the files user.c and user.h, we have provided a sample program, as well as a sample cost function for your convenience. If you do not intend to pass parameters into main, then you can just declare it as main() without the argc and argv arguments.

4.2. void initialize_parameters()

Before calling vfsr, the user must allocate storage and initialize some of the passed parameters. A sample procedure is provided as a template. In this procedure the user should allocate storage for the passed arrays and define the minimum and maximum values. Below, we detail all the parameters which must be initialized. If your arrays are of size 1, still use them as arrays as described in user.c.

4.3. double user_cost_function(
double *x,
double *parameter_minimum,
double *parameter_maximum,
int *cost_flag,
DEFINES *OPTIONS)

You can give any name to user_cost_function as long as you pass this name to vfsr. x (or whatever name you pass to vfsr) is an array of doubles representing a set of parameters to evaluate, and cost_flag (or whatever name you pass to vfsr) is the address of an integer.

OPTIONS is described further below under the section describing parameters passed to vfsr(). If you need to change DEFINE_OPTIONS adaptively during recursive calls to vfsr() from user_cost_function(), OPTIONS will be the address of a structure containing the DEFINE_OPTIONS with typedef DEFINES. Otherwise, it simply will be the address of any dummy integer.

In user_cost_function, *cost_flag should be set to FALSE (0) if the parameters violate a set of user defined constraints (e.g., as defined by a set of boundary conditions) or TRUE (1) if the parameters represent a valid state. If *cost_flag is FALSE, no acceptance test will be attempted, and a new set of trial parameters will be generated. The function returns a real value which VFSR will minimize.

If another algorithm suggests a way of incorporating constraints into the cost function, then this modified cost function can be used as well by VFSR, or that algorithm might best be used a front-end to VFSR.

Passing the parameter bounds in the cost function permits some additional adaptive features during the search. For example, setting the lower bound equal to the upper bound will remove a parameter from consideration.

4.4. double user_random_generator()

A random number generator function must be passed next. It may be as simple as one of the UNIX random number generators (e.g. drand48), or may be user defined, but it should return a real value within [0,1) and not take any parameters. We have provided a good random number generator, randflt, and its auxiliary routines with the code in the file user module.

4.5. void initialize_rng()

Most random number generators should be "warmed-up" by calling a set of dummy random numbers.

4.6. void print_time(char *message)

As a convenience, we have included this subroutine, and its auxiliary routine aux_print_time, to keep track of the time spent during optimization. This can give some problems on some platforms, and may be bypassed using one of the DEFINE_OPTIONS. It takes as its only parameter a string which will be printed. We have given an example in user_cost_function to illustrate how print_time may be called periodically every set number of calls by defining PRINT_FREQUENCY in user.h.

An independent set of timing routines has been placed in the vfsr module, to permit you to independently alter this information from the vfsr and user modules.

4.7. final_cost = vfsr(
user_cost_function,
user_random_generator,
number_parameters,
parameter_type,
parameter_initial_final,
parameter_minimum,
parameter_maximum,
tangents,
curvature,
&exit_status,
OPTIONS);

This is the form of the call to vfsr from user.c.

4.8. double vfsr(
double (*user_cost_function) (),
double (*user_random_generator) (),
int number_parameters,
int *parameter_type,
double *parameter_initial_final,
double *parameter_minimum,
double *parameter_maximum,
double *tangents,
double *curvature,
int *exit_status,
DEFINES *OPTIONS)

This is how vfsr is defined in the VFSR module, contained in vfsr.c and vfsr.h. Each parameter is described below as it must be passed to this module from the user module.

4.8.1. double final_cost;

This double, defined in vfsr.c, is returned to the calling program as whatever it is named by the user. It will be the minimum cost value found by vfsr.

4.8.2. double (*user_cost_function) ()

The parameter (*user_cost_function*) () is a pointer to the cost function that you defined in your user module.

4.8.3. double (*user_random_generator) ()

As discussed above, a pointer to the random number generator function, defined in the user module, must be passed next.

4.8.4. int number_parameters

An integer containing the dimensionality of the state space is passed next. Each of the arrays that follow are to be of the size number_parameters.

4.8.5. int *parameter_type

This integer array is passed next. Each element of this array (each flag) is either REAL_TYPE (0) (indicating the parameter is a real value) or INTEGER_TYPE (1) (indicating the parameter can take on only integer values).

4.8.6. double *parameter_initial_final

Next, an array of doubles is passed. Initially, this array holds the set of starting parameters which should satisfy any constraints or boundary conditions. Upon return from the VFSR procedure, the array will contain the best set of parameters found by vfsr to minimize the user's cost function. Experience shows that any guesses within the acceptable ranges should suffice, since initially the system is at high annealing temperature and VFSR samples the breadth of the ranges. The default is to have vfsr generate a set of initial parameters satisfying the user's constraints. This can be overridden in the Makefile, using USER_INITIAL_PARAMETERS=TRUE, to have the user's initial guess be the first generated set of parameters.

4.8.7. double *parameter_minimum

4.8.8. double *parameter_maximum

These two arrays of doubles are passed. Since VFSR works only on bounded search spaces, these arrays should contain the minimum and maximum values each parameter can attain. If you aren't sure, try a factor of 10 or 100 times any reasonable values. The exponential temperature annealing schedule should quickly sharpen the search down to the most important region.

4.8.9. double *tangents

This array of doubles is passed. On return from vfsr, for real parameters, this contains the first derivatives of the cost function with respect to its parameters. These can be useful for determining the value of your fit. In this implementation of VFSR, the tangents are used to determine the relative reannealing among parameters.

A "hook" has been placed in vfsr.c, which is turned on by an option in the Makefile, to permit you to individually initialize the set of parameter temperatures in this array (passed as *cost_tangents in user.c). This can be useful in constrained problems, where greater efficiency can be achieved in focussing the search than might be permitted just by setting upper and lower bounds.

4.8.10. double *curvature

This array of doubles is passed next. On return from vfsr, for real parameters, this contains the second derivatives of the cost function with respect to its parameters. These also can be useful for determining the value of your fit.

A "hook" has been placed in vfsr.c, which is turned on by an option in the Makefile, to permit you to initialize the cost temperature in the first element of this array, curvature[0] (passed as cost_curvature[0] in user.c). This can be useful in problems where you want to start the search at a specific scale.

4.8.11. int *exit_status

The address of this integer is passed to vfsr. On return it contains the code for the reason vfsr exited.

NORMAL_EXIT = 0. Given the criteria set largely by the DEFINE_OPTIONS in Makefile, the search has run its normal course.

P_TEMP_TOO_SMALL = 1. A parameter temperature was too small using the set criteria. Often this is an acceptable status code.

C_TEMP_TOO_SMALL = 2. The cost temperature was too small using the set criteria. Often this is an acceptable status code.

COST_REPEATING = 3. The cost function value repeated a number of times using the set criteria. Often this is an acceptable status code.

TOO_MANY_INVALID_STATES = 4. Too many repetitive generated states were invalid using the set criteria. This is helpful when using *cost_flag, as discussed above, to include constraints.

Note that just relying on such a simple summary given by *exit_status can be extremely deceiving, especially in highly nonlinear problems. We *strongly* suggest that the user set VFSR_PRINT=TRUE before any production runs. An examination of some periodic output of VFSR can be essential to its proper use.

4.8.12. DEFINES *OPTIONS

OPTIONS only can be passed if ADAPTIVE_OPTIONS=TRUE in the Makefile. When ADAPTIVE_OPTIONS=FALSE (the default), all changes in default Program Options DEFINE_OPTIONS are defined in the Makefile. (Compiler Options and Printing Options always are to be set in the Makefile.) When ADAPTIVE_OPTIONS=TRUE, all Program Options are defined in the structure OPTIONS, permitting adaptive changes during repeated searches or at different recursive levels of VFSR optimization.

ADAPTIVE_OPTIONS=TRUE can also be used to read in the options from a separate data file, permitting efficient tuning/debugging of these parameters without having to recompile the code. This option has been added to the user module.

As illustrated in the user module, if ADAPTIVE_OPTIONS=FALSE, only "typedef int DEFINES" must be specified to use this format; e.g., the address of any dummy integer variable may be passed. IF ADAPTIVE_OPTIONS=TRUE, the structure address OPTIONS of typedef DEFINES must be passed. (In user.c, we have named this structure USER_DEFINES *USER_OPTIONS.)

5. Additional Documentation

CHANGES is a terse record of major changes made in the VFSR code. NOTES is a collection of those recommended fixes, modifications, comments, caveats, etc., not reasonable to place in this file, but that might be of interest to a wider audience.

6. Bug Reports

While we do not have time to help you solve your own applications, we do want VFSR to be helpful to a large community. Therefore, we welcome your bug reports and constructive critiques regarding our code. It likely would help if you would enclose relevant portions your VFSR_OUT file, or a uuencoded compressed copy if it is very long. “Flames” will be rapidly quenched.

References

1. L. Ingber and B. Rosen, “Very Fast Simulated Reannealing (VFSR),” [ftp research.att.com: /netlib/opt/vfsr.Z] or [ftp lib.stat.cmu.edu: /general/vfsr], AT&T Bell Labs, Murray Hill, NJ (1992).
2. L. Ingber, “Very fast simulated re-annealing,” *Mathl. Comput. Modelling,* 12, pp. 967-973 (1989).
3. L. Ingber, H. Fujio, and M.F. Wehner, “Mathematical comparison of combat computer models to exercise data,” *Mathl. Comput. Modelling,* 15, pp. 65-90 (1991).
4. L. Ingber, “Statistical mechanical aids to calculating term structure models,” *Phys. Rev. A,* 42, pp. 7057-7064 (1990).
5. L. Ingber, “Statistical mechanics of neocortical interactions: A scaling paradigm applied to electroencephalography,” *Phys. Rev. A,* 44, pp. 4017-4060 (1991).
6. L. Ingber and B. Rosen, “Genetic algorithms and very fast simulated reannealing: A comparison,” *Mathl. Comput. Modelling,* 16, pp. 87-100 (1992).
7. L. Ingber, “Simulated annealing: Practice versus theory,” *Statistics Comput.,* p. (to be published) (1993).
8. B. Rosen, “Function optimization based on advanced simulated annealing,” [ftp ringer.cs.utsa.edu: /pub/rosen/rosen.advsim.ps.Z], University of Texas, San Antonio, TX (1992).
9. L. Ingber, “Generic mesoscopic neural networks based on statistical mechanics of neocortical interactions,” *Phys. Rev. A,* 45, pp. R2183-R2186 (1992).

[*] Some (p)reprints and current versions of VFSR can be obtained via anonymous ftp: from ftp.umiacs.umd.edu in the /pub/ingber directory, and from ringer.cs.utsa.edu in the /pub/rosen

directory.

$Id: README.ps,v 7.3 1993/02/04 18:33:42 ingber Exp ingber $