SANDIA REPORT

SAND2001-3789
Unlimited Release
Printed December 2001

SGOPT User Manual Version 2.0

William E. Hart

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2001–3789
Unlimited Release
Printed December 2001

SGOPT User Manual
Version 2.0

William E. Hart
Optimization and Uncertainty Estimation Dept
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM
`http://www.cs.sandia.gov/~wehart`
`wehart@sandia.gov`

Abstract

This document provides a user manual for the SGOPT software library. SGOPT is a C++ class library for nonlinear optimization. This library uses an object-oriented design that allows the software to be extended to new problem domains. Furthermore, this library was designed so that the interface is straightforward while providing flexibility to allow new algorithms to be easily added to this library. The SGOPT library has been used by several software projects at Sandia, and it is integrated into the DAKOTA design and analysis toolkit. This report provides a high-level description of the optimization algorithms provided by SGOPT and describes the C++ class heirarchy in which they are implemented. Finally, installation instructions are included.

Contents

1 Introduction

The SGOPT library contains a variety of global optimization algorithms, with an emphasis on stochastic methods. SGOPT currently includes the following global optimization methods: genetic algorithms (PGAreal,PGAint), evolutionary pattern search algorithms (EPSA), simulated annealing (SAreal), tabu search (TSreal), multistart local search (MSreal) and stratified Monte Carlo (sMCreal). Additionally, SGOPT includes several local search algorithms such as Solis-Wets (SWOpt) and pattern search (PatternSearch).

SGOPT stands for Stochastic Global OPTimization and for expensive optimization problems its global optimizers are best suited for identifying promising regions in the global design space. In multimodal design spaces, the combination of global identification (from SGOPT) with efficient local convergence (from a gradient-based algorithm) can be highly effective. The SGOPT methods are not gradient-based, which makes them appropriate for discrete problems as well as problems for which gradient information is unavailable or is of questionable accuracy due to numerical noise, etc. No SGOPT methods currently support general linear and nonlinear constraints directly, although penalty function formulations for nonlinear constraints have been employed with success [21].

This document describes the algorithms that are implemented within SGOPT at a high level. Presently SGOPT does not include software to provide a standardized interface for initializing the optimizers. Instead, the optimization library is linked into an executable defined by the user. Many of the optimizers defined by SGOPT are currently included in the interface to the DAKOTA Iterator Toolkit [8].

Historically, this software evolved as a library of algorithms used for research purposes. Consequently, there are many places in this library where the software could be further polished, and some of the libraries are more stable than others. Indications are made at the end of the sections describing each of the optimizers in SGOPT concerning the stability of each optimizer.

1.1 Generic Optimizers

The majority of optimizers in SGOPT are designed to perform optimization over R^n. However, some optimizers like genetic algorithms can be readily adapted to perform optimization over a wide variety of search domains. Two approaches are taken in SGOPT to accomodate the optimization of a generic search domain.

1.1.0.1 Generic Point Optimization A facility for optimizating a user-defined search domain has been constructed for Monte Carlo search and genetic algorithms. This facility uses the class GenericPointBase to define the basic operations that are needed to define these optimizers. The user simply defines a subclass of GenericPointBase and instantiates one of the generic optimizers to perform optimization. Examples of the use of this facility are included in the subdirectory `examples/generics`. This facility is easy to use, but the user's ability to customize the optimizer to a particular search domain is limited to the user's choice of the methods used to generate new trial points in the search domain. {*Note*: I expect to templatize much of this code in the near future, which may remove the needed for this capability.}

1.1.0.2 Object-Oriented Design When possible, an object oriented design has been adopted which allows a user to easily define a new C++ class to perform optimization over a novel search domain. The simplest example of this design is the class structure for Monte Carlo search. This algorithm simply generates random samples from a search domain. The current class heirarchy for Monte Carlo search is

```
        sMC
        / \
       /   \
sMCreal   sMCgen
```

The class sMC defines the main loop of a Monte Carlo search algorithm. This loop utilizes a variety of abstract functions that are defined by the descendents of sMC. These functions define how randomization is performed as well as some IO. This type of object oriented design allows for significant code reuse when designing a Monte Carlo search algorithm for a new search domain. The other example of this type of object oriented design is the class heirarchy used for genetic algorithms (see Section **Evolutionary Algorithms** (p. 3)). Instantiating a new optimizer is considerably more complex than using the generic-point facility, but this option does allow the user to make algorithmic modifications that tailor the optimizers in SGOPT for a particular search domain.

1.2 Overview

It will often be convenient to describe the methods and information in optimization classes in five categories:

- **General** Information: definitions of generic methods and data
- **Configuration** Controls: definitions of methods and data that are used to parameterize the operation of an optimizer
- **Termination** Controls: definitions of methods and data that are used to determine when an optimizer terminates
- **Debugging** Controls: definitions of methods and data that are used to print debugging information
- **Iteration** Controls: defintions of methods and data that are used to define the optimizers main operations in each iteration

2 Evolutionary Algorithms

2.1 Overview

Evolutionary search is an adaptive random search that maintains a collection of solutions that are ranked by their performance and uses a competition between these solutions to select solutions for further processing. Research on evolutionary search algorithms incorporates elements of both biological evolution and global optimization. These algorithms are inspired by biological evolutionary mechanisms and are often used to perform global optimization.

The exemplars of evolutionary search algorithms are genetic algorithms, evolutionary strategie and evolutionary programming [3, 10, 12]. The design and motivation for these algorithms are different, but they incorporate the same basic adaptive components [2, 15]. These methods use a collection of solutions (population of individuals) that are updated iteratively using selection mechanisms and genetic operators. The general process of each iteration (generation) is described in Figure 1.

```
Initialize population (with uniformly generated solutions)
Repeat
  Evaluate solutions in the population
  Perform competitive selection
  Apply genetic operators
  Perform local search (optional)
Until convergence criteria satisfied
```

Figure 1: Pseudo-algorithm for a genetic algorithm.

The selection mechanism performs a competition to select a subset of the solutions for further processing. The genetic operators are used to generate deviates from the selected individuals. Two types of genetic operators are commonly employed: mutation and recombination. Mutation uses a single individual to generate a deviate that is located in the local neighborhood of the individual. Recombination uses two individuals to generate another individual that is typically located in the smallest hypercube that contains them both. Local search is another genetic operator that is sometimes employed with evolutionary algorithms to refine solutions in their local neighborhood.

Using these genetic operators, evolutionary search algorithms perform a global search. Global convergence is not guaranteed for all evolutionary algorithms [22], but experiments with these algorithms indicate that they often converge to regions of the search space that contain near-optimal solutions.

2.2 Genetic Algorithms

For historical reasons, the development of evolutionary algorithms within SGOPT has focused on Genetic Algorithms (GAs). The GA was initially described using populations of binary strings in $\{0,1\}^n$, which are evaluated by the objective function (fitness function) [16, 12, 18]. When searching spaces other than $\{0,1\}^n$, the objective function decodes the binary string and performs the function evaluation.

Holland [16] proposed a selection mechanism that stochastically selects individuals with probability

$$p_i = \frac{f(x_i)}{\sum_i f(x_i)}$$

This selection mechanism is called *proportional selection*, since the number of copies of an individual will be in proportion to the its fraction of the population's total fitness. This method assumes the GA is minimizing $f(x)$ and that the global minimum is greater than or equal to zero, but it can be easily modified to perform selection when maximizing a function, or when the global minimum is negative.

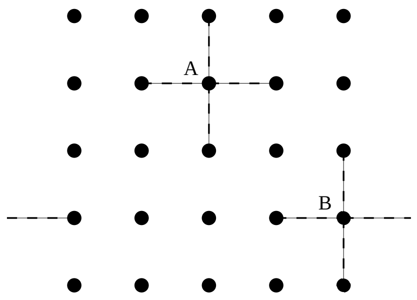

Figure 2: The two dimensional grid used by GSGAs to define population subsets.

The binary GA proposed by Holland uses mutation and crossover operators. With binary strings, the mutation operator changes a single bit on a string, and it is typically used with low frequency. The crossover operator picks two points on the the binary representation and generates the new sample by taking all of the bits between these points from one parent and the remaining bits from the other parent. For example, if $n = 10$ and the chosen points are $p_1 = 2$ and $p_2 = 6$:

```
Parent(1): 1111111111    Parent(2): 0000000000    Sample: 0011110000
```

Crossover is typically used with high frequency, so most of the individuals in each generation are generated using crossover.

The manner in which the parameters of the objective function are encoded on each string does not affect the mechanisms of the GA, though it can affect the GA's search dynamics. In particular, much research has been done examining how crossover composes and disrupts patterns in binary strings, based on their contribution to the total fitness of the individual [11, 24, 25, 30]. This research has motivated the use of modified crossover operators that restrict the distribution of crossover points. For example, if the binary string is decoded into a vector of integers or floating point values, then crossover is often applied only between the integer or floating point values on the binary string [6].

2.3 Panmictic and Geographically Structured Genetic Algorithms

GAs can be distinguished by the manner in which the selection mechanism and genetic operators are applied to the population. *Panmictic* GAs employ selection mechanisms (like proportional selection) that use global information about the entire population to perform a global selection. In proportional selection the population's total fitness is used to perform selection. Panmictic GAs apply the crossover operator to pairs of individuals randomly taken from individuals selected from the entire population.

Geographically structured genetic algorithms (GSGAs) perform a structured selection in which individuals compete against a fixed subset of the population, and the genetic operators are applied to individuals selected from these subsets. The most common way of structuring the selection mechanism uses a toroidal two dimensional grid like the one in Figure 2 [1, 4, 17, 26]. Every element of the population is assigned to a location on the grid. The grid locations are not necessarily related to the individuals' solutions. They are often arbitrary designations used to perform selection. Thus, there are distinct notions of locality with respect to the population grid and the search space. When local search is performed with GSGAs, it is performed in the search space. When local selection is performed, it is performed in the population grid.

Two general methods of local selection have been used to perform selection in GSGAs: (1) fixed size neighborhoods have been used to define the set of neighboring individuals [5, 13], and (2) random walks have been used to stochastically sample the locations of neighboring individuals [4, 17]. Figure 2 illustrates the fixed size neighborhoods that could be used to perform selection. Proportional selection is applied to the solutions in each of these neighborhoods. Since one individual is assigned to each grid location, the selection procedure is used to select only as many individuals as are necessary to use the genetic operators. For example, two individuals will be selected if crossover is used. The new individual generated from a genetic operator is assigned to the grid location at which selection is performed.

The early motivation for GSGAs came from SIMD designs for GAs. McInerney [17] describes a SIMD GSGA and analyzes the effect of different methods of local selection. He shows how local selection encourages local regions of the 2D grid to form *demes* of very similar individuals, and argues that inter-deme competition enables GSGAs to perform search while maintaining diversity in the population. He observes that selection using random walks gave very good results in his experiments. He notes that this method enabled good solutions to diffuse through the population, while strongly encouraging the formation of demes.

Gordon and Whitley [13] argue that the algorithmic nature of GSGAs may be of interest, independent from their implementation on a particular architecture. They experimentally compare GSGAs to panmictic GAs and observe that the GSGAs provide superior performance. This philosophy is echoed by Davidor, Yamada and Nakano [5] in their motivation for the ECO framework. The ECO framework provides a serial design for implementing a geographically structured GA.

Finally, we note that our definition of GSGAs includes GAs which structure the selection at a fine granularity. A number of GAs have been proposed whose competitive selection is intermediate between GSGAs and panmictic GAs. Mühlenbein [20] makes a similar distinction and describes a GA which uses a set of independent subpopulations and structures the inter-population communication with a ladder structure. These subpopulations are typically small, so they perform a localized search of the function. For example, Whitely [31] illustrates how a small population can perform a locallized search in the context of neural network optimization problems. Inter-population communication enables populations to combine disparate solutions and enables them to perform a global search.

3 Pattern Search Methods

3.1 Overview

Pattern search methods are a class of direct search optimizers that have recently received a lot of attention because of new convergence proofs that guarantee weak first order stationary point convergence [7, 27, 29, 28].

The GPSOpt class provides a generic framework for defining generalized pattern search algorithms. This class implements the framework used by Torczon [29] to define generalized pattern search methods. The class PatternSearch provides a specific implementation of several types of pattern search methods that perform a rather local search about the best current iterate. This includes randomized variants of these methods, which randomize the order of the steps taken in the deterministic algorithm; these methods have the same convergence properties as their deterministic counterparts.

The PatternSearch class defines a variety of pattern search algorithms. Although these algorithms could have been defined with seperate classes, it has proved easier to maintain these algorithms within a single class, using a switch to select which algorithm is executed.

For each of these pattern search algorithms, a single expansion and contraction factor is used. The default expansion factor is 2.0 and the default contraction factor is 0.5. None of these methods explicitly maintains a pattern matrix, so the `UpdateMatrix` method is not defined. The variable `em_case` is used to select amongst the following pattern search algorithms:

- **Standard PS**: **(em_case=0)** This PS method checks for improvement in each dimension iteratively, examining dimensions from 1 to n in order. If an improvement is detected, this algorithm keeps that improvement and continues checking the remaining dimensions, using the improved point as the starting point from which new offsets are examined. The order of the dimensions may be shuffled. This is the default PS method.
- **Best Offset First**: **(em_case=1)** This PS method checks for improvement in each of the $2n$ possible offsets iteratively. This algorithm terminates as soon as any improving point is found. In PatternSearch each dimension from 1 to n is examined in order. The order of the offsets may be shuffled.
- **Best Dimension First**: **(em_case=2)** This PS method checks for improvement in each dimension iteratively, examining dimensions from 1 to n in order. This algorithm terminates as soon as any improving point is found. The order of the dimensions may be shuffled.
- **Asynchronous**: **(em_case=3)** This PS method spans off asynchronous function evaluations that compute all $2n$ offsets simultaneously. It then synchronizes the calculation of these offests and keeps the offset which provides the best improvement.
- **Biased Best Dimension First**: **(em_case=4)** This PS method uses a bias to guide the algorithm in a direction where improving points have previously been found. If improving points are not found with the bias, the bias is halved and then zeroed. In each of these phases, the algorithm acts like the **Best Dimension First** method. The order of the dimensions may be shuffled.

3.2 Current Status

The code in GPSOpt is rather stable. The code defining the various pattern search methods continues to be refined.

4 Solis-Wets Local Search

4.1 Overview

The class SWOpt provides a framework for implementing the stochastic direct search algorithms described by Solis and Wets [23]. Figure 3 provides pseudo-code for the main loop of SWOpt. These algorithms generate a new iterate using coordinate-wise steps. If the new iterate is better than the current iterate then it is accepted and the algorithm repeats. Otherwise the algorithm considers a step in the opposite direction. If this new point is also worse that then current iterate then a new iterate is again generated in the neighborhood of the current iterate. SWOpt also defines mechanisms for expanding and contracting the step size of the offsets used to generate the new iterate.

```
best = INFTY
bias = $\overline{0}$
n_succ = 0
n_fail = 0

while (rho > rho_lower_bound)
   generate dx
   curr = f(x + dx + bias)
   if (curr < best)
      bias = 0.2 * bias + 0.4 * (dx + bias)
      x += dx + bias
      n_succ++
      n_fail=0
   else
      curr = f(x - dx - bias)
      if (curr < best)
         bias = bias - 0.4 * (dx + bias)
         x -= dx + bias
         n_succ++
         n_fail=0
      else
         bias = bias/2
         n_fail++
         n_succ=0
      endif
   endif

   if (n_succ $\geq$ max_succ)
      n_succ=0
      rho = ex_factor * rho
   endif
   if (n_fail $\geq$ max_fail)
      n_fail=0
      rho = ct_factor * rho
   endif
endwhile
```

Figure 3: Pseudo-code for the Solis-Wets algorithms.

Classes SWOpt1 and SWOpt2 differ in their definition of the private method `gen_new_point`, which is used to generate a new iterate. SWOpt1 generates a iterate using normally distributed deviates with standard deviation `rho`. SWOpt2 generates a new iterate using uniformly distributed deviates from the range [− `rho` , `rho`].

4.2 Current Status

These classes are stable.

5 Stratified Monte Carlo

5.1 Overview

The class sMC provides an abstract base class for stratified Monte Carlo sampling (sMC). Stratified Monte Carlo sampling partitions the search domain into a finite number of disjoint regions, each of which is sampled independently. In each iteration sMC samples a point from each region (from a fixed distribution), and the best of these points is reported. sMC reduces to standard Monte Carlo sampling algorithm (MC) in the case where there is a single region, the entire search domain. Ermakov, Zhigyavskii and Kondratovich [9] and Hart [14] provide formal descriptions and theoretical analyses of sMC. It is particularly interesting to note that for a given number of samples from the total search domain, the probability that sMC samples an ϵ-close point is greater than or equal to the probability that MC samples an ϵ-close point.

Classes sMCreal and sMCint provide a common interface for performing sMC on R^n and Z^n. The method `set_lsopt` in these classes defines a local search optimizer and specifies the frequency with which local search should be applied to the randomly generated points. When `freq` = 1.0, this converts sMCreal to stratified multistart local search (see Morris and Wong [32, 19] and Hart [14]).

5.2 Current Status

sMC and sMCreal are stable. sMCint needs to be re-implemented. The termination rules described in Hart [14]) need to be incorporated into sMC.

6 Installation

6.1 Downloading

The SGOPT software can be downloaded either as a compressed tar file or directly from the SGOPT Concurrent Version System (CVS) repository. The latest release of SGOPT is available at

```
http://www.cs.sandia.gov/~wehart/SGOPT
```

and earlier versions are available in the same directory.

The CVS repository for SGOPT can be accessed by executing

```
cvs -d :ext:GEUutili@gaston.cs.sandia.gov:/usr/local/cvs/cvsroot checkout sgopt
```

The password for this repository is 'anonymous'. The developer's password for this repository is restricted; please contact Bill Hart at `wehart@sandia.gov` to request the password to commit changes to this repository. If you are accessing this repository throught a firewall (e.g. Sandia's SRN firewall), or you expect to checkout updates frequently, then the script `cvs-s` can be used to encapsulate the access to the CVS repository. The `cvs-s` script can be downloaded at

```
ftp://ftp.cs.sandia.gov/pub/papers/wehart/src/cvs-shells.tar
```

Note that this script uses the `ssh` command, version 1.x.

6.2 Installation on Unix

Installation of SGOPT on UNIX systems is performed by the following steps:

1. Unpack the archive, unless you have already done that

   ```
   gunzip sgopt-$VERSION.tar.gz    # uncompress the archive
   tar xf sgopt-$VERSION.tar       # unpack it
   ```

2. Move into the `sgopt` directory and run the configure script.

   ```
   ./configure
   ```

   The `configure` script automates much of the setup activity associated with building large suites of programs like SGOPT on various hardware platforms. This includes

   (a) making symbolic links so that files used for configuration can be accessed from one location
   (b) generating Makefiles so that objects, libraries, executables and other 'targets' can be created for specific and unique hardware platforms
   (c) calling itself recursively so that sub-directories can also be configured

   By default, the configure script does not assume that SGOPT relies on any other software libraries. There are a number of configuration options that can be used to customize the installation. The full parameter list for the `configure` script is:

```
configure hosttype [--target=target] [--srcdir=dir] [--rm]
                   [--site=site] [--prefix=dir] [--exec-prefix=dir]
                   [--program-prefix=string] [--tmpdir=dir]
                   [--with-package[=yes/no]] [--without-package]
                   [--enable-feature[=yes/no]] [--disable-feature]
                   [--norecursion] [--nfp] [-s] [-v] [-V | --version]
                   [--help]
```

Many of these options are not necessary since system information can be often acquired from your local machine. Refer to the Cygnus `configure` documentation for complete information. The following options are either commonly used or specific to SGOPT (examples of arguments are provided):

[–with-compiler=<gcc,CC>]	Sets up a specific compiler; The native compiler is the default.
[–target=<solaris>]	Optional flag to specify the target machine that you are cross-compiling for.
[–site=<snl980>]	Specifies the site-specific locations for MPI, etc.
[–with-debugging]	Turns on the DEBUGGING macro and sets the OPTIMIZATION macro to <flag> (code is compiled with -g by default).
[–with-mpi]	Turns on the use of the MPI package.
[–with-mpe]	Turns on the use of the MPE package.
[–with-swig]	Enables the use of swig to wrap SGOPT for use with the Python scripting language.
[–with-static]	Enables the compilation of statically linked libraries (the default).
[–with-insure]	Enables the compilation with the insure++ debugging tool.
[–with-shared]	Enables the compilation of dynamically linked libraries, which can be shared.
[–with-optimization=<level>]	Sets the optimization level used when compiling the source files. This is overridden by the –with-debugging flag.
[–with-ansi]	Sets up the compiler to use ANSI standard constructs for C++. (the default)
[–with-ansiheaders]	Creates flags that force the use of ANSI standard C++ header conventions. (the dfault)

The configure script creates Makefiles from `Makefile.in` template files, which outline the basic 'targets' that need to get built. Variables that are package, site or hardware dependent are stored in individual 'fragment' files. These 'fragment' files are added to the custom created Makefiles when users and code developers (recursively) configure this repository with specific host, target, package and/or site parameters.

Running `configure` takes a while, so be patient. Verbose output will always be displayed unless the user/developer wishes to silence it by specifying the parameter, '–silent'. If you wish to configure only one level/directory, remember to use the option '–norecursion'. All generated "config.status" files include this parameter as a default for easy makefile re-generation; after editing a Makefile.in file, you can construct the associate Makefile file by typing `config.status`.

After the `configure` command is completed, three files will be generated in each configured directory (specified by the file, 'configure.in').

(a) Makefile-${target}
The suffix, ${target}, will depend on the target specified. Native builds have identical host and target values.

(b) Makefile
This will be a symbolic link to the file mentioned above. A user or developer will simply type `make` and the last generated Makefile-${target} will then be referenced.

(c) config.status
A 'recording' of the configuration process (i.e., what commands were executed to generate the makefile). It can be used by the custom makefile to re-generate itself with a command such as this

```
make Makefile.
```

Fragment files exist so that `configure` can support multi-platform environments. SGOPT can be configured for code development and execution on the following platforms :

```
SPARC-SUN-SOLARIS2.5.1    (Sun ULTRAsparc)
MIPS-SGI-IRIX6.4          (SGI Octane)
HPPA1.1-HP-HPUX9.05       (HP 9000/700 series)
PENTIUM-INTEL-COUGAR      (Intel TFLOP supercomputer at SNL)
i686-UNKNOWN-LINUX        (Red Hat 7.1)
```

The fragment files for these platforms and for the packages that SGOPT relies on are located in the `sgopt/config` directory. There are five types of files in this directory:

```
mf-<host>-<target>-<site>
Automatically generated by the configure scripts.

mh-<host>
Fragments that define the utilities provided by the host (e.g. the
definition of MAKE.

mp-<target>-<site>
Fragments that define information for the packages that are used by
SGOPT (e.g. MPI).

ms-<site>
Fragments that define the site-specific general configuration
information. If this does not exist for a given site, then the
default ms-default fragment is used.

mt-<target>
Fragments needed to specfy how to compile code for a target
architecture (e.g. compiler name/location).
```

3. Compile the program by running make.

```
make
```

Note that the makefiles in SGOPT may not be portable to all `make` commands. However, they do work with the GNU `gmake` command. The latest file `Makefile-${target}` generated by `configure` will be referenced by this command. The target directory for the library is created for the particular target platform as a subdirectory of sgopt/lib.

Prior to making object files header files are linked into the directory `sgopt/include`.

4. Optional: Generate the html library documentation.

   ```
   make html
   ```

   This requires the `doxygen` utility.

5. Optional: Generate the postscript version of the user manual.

   ```
   make ps
   ```

   This requires the `doxygen`, `latex`, and `dvips`.

6. Optional: Generate the PDF version of the user manual.

   ```
   make pdf
   ```

   This requires the `doxygen`, `latex`, `dvips` and ghostscript packages.

6.3 Installation on Windows

SGOPT was originally developed under UNIX, but it has been ported to Windows NT using Microsoft's Visual C++ (version 6.0). A MSVC++ project is provided in `sgopt/src/vcpp`. This project defines a DLL that will be compiled for SGOPT, and it can be easily included in a user's workspace. The project file relies on the environmental variable 'SGOPT', which is defined from the MS Windows Control Panel under `System/Environment`. This variable should be set to the path of the `sgopt` directory. Note: this project file is out of date.

7 Acknowledgements

The genesis of the SGOPT library is the BBUMS library developed by Bill Hart and Brian Bartell while graduate students at U.C. San Diego. Many of the optimizers in SGOPT were initially developed in BBUMS. The BBUMS library was subsequently reorganized and divided into SGOPT and UTILIB; UTILIB provides generic C++ templates and support for portability.

I would like to thank Keith Hunter, Boyd Schimel and Mike Eldred for their input on this software. Each of them has identified numerous bugs, and refinements in the configuration process are largely due to the demands that their uses of SGOPT have made.

This document was prepared using the Doxygen software documentation tool, developed by Dimitri van Heesch, copyright 1997-2001.

This work was supported in part by the Mathematics, Information and Computational Science program, U.S. Department of Energy, Office of Energy Research. This work was performed at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

References

[1] D. H. Ackley. A case for Lamarackian evolution. In *To appear in Proc. of the Third Conf. on Artificial Life*, 1993.

[2] T. Bäck and F. Hoffmeister. Extended selection mechanisms in genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Intl. Conf. on Genetic Algorithms*, pages 92–99, San Mateo, CA, 1991. Morgan-Kaufmann.

[3] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Intl. Conf. on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1991. Morgan-Kaufmann.

[4] R. J. Collins and D. R. Jefferson. Selection in massively parallel genetic algorithms. In *Proc of the fourth Intl Conf on Genetic Algorithms*, pages 249–256, 1991.

[5] Y. Davidor, T. Yamada, and R. Nakano. The ECOlogical framework II: Improving GA performance at virtually zero cost. In S. Forrest, editor, *Proc. of the Fifth Intl. Conf. on Genetic Algorithms*, pages 171–176, San Mateo, CA, 1993. Morgan-Kaufmann.

[6] L. Davis, editor. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[7] J. E. Dennis and V. Torczon. Derivative-free pattern search methods for multidisciplinary design problems. In *Proc 5th AIAA/NASA/ISSMO Symp Multidisciplinary Analysis and Optimization*, pages 922–932, 1994. AIAA Paper 94-4349.

[8] M. S. Eldred, W. E. Hart, W. J. Bohnhoff, V. J. Romero, S. A. Hutchinson, and A. G. Salinger. Utilizing object-oriented design to build advanced optimization strategies with generic implementation. In *Proc Sixth AIAA/USAF/NASA/ISSMO Symp on Multidisciplinary Analysis and Optimization*, pages 1568–1582, 1996.

[9] S. Ermakov, A. Zhigyavskii, and M. Kondratovich. Comparison of some random search procedures for a global extremum. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 29(1):112–117, 1989.

[10] D. B. Fogel. *Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1995.

[11] D. E. Goldberg. Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, 3:129–152, 1989.

[12] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Co., Inc., 1989.

[13] V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proc of the Fifth Intl Conf on Genetic Algorithms*, pages 177–183, San Mateo, CA, 1993. Morgan-Kaufmann.

[14] W. E. Hart. *Adaptive Global Optimization with Local Search.* PhD thesis, University of California, San Diego, May 1994. http://www.cs.sandia.gov/ wehart/papers.html.

[15] F. Hoffmeister and T. Bäck. Genetic algorithms and evolutionary strategies: Similarities and differences. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 455–469. Springer-Verlag, 1990.

[16] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1976.

[17] J. M. McInerney. *Biologically Influenced Algorithms and Parallelism in Non-Linear Optimization.* PhD thesis, University of California, San Diego, 1992.

[18] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 3rd edition, 1996.

[19] R. J. Morris and W. S. Wong. Systematic choice of initial points in local search: Extensions and application to neural networks. *Information Processing Letters*, 39:213–217, 1991.

[20] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan-Kauffmann, San Mateo, CA, 1991.

[21] E. R. Ponslet and M. S. Eldred. Discrete optimization of isolator locations for vibration isolation systems: An analysis and experimental investigation. In *Proc 6th AIAA/USAF/NASA/ISSMO Symp on Multidisciplinary Analysis and Optimization*, pages 1703–1716, 1996.

[22] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Trans Neural Networks*, 5(1):96–101, 1994.

[23] F. Solis and R.-B. Wets. Minimization by random search techniques. *Mathematical Operations Research*, 6:19–30, 1981.

[24] W. M. Spears and K. A. De Jong. An analysis of multi-point crossover. In G. J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 301–315. Morgan-Kauffmann, San Mateo, CA, 1991.

[25] W. M. Spears and K. A. De Jong. On the virtues of parametrized uniform crossover. In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Intl. Conf. on Genetic Algorithms*, pages 230–236, San Mateo, CA, 1991. Morgan-Kaufmann.

[26] P. Spiessens and B. Manderick. A massively parallel genetic algorithm: Implementation and first analysis. In *Proc of the Fouth Intl Conf on Genetic Algorithms*, pages 279–285, 1991.

[27] V. Torczon. Multi-directional search: A direct search algorithm for parallel machines. Technical Report TR90-7, Rice University, May 1989.

[28] V. Torczon. On the convergence of the multidirectional search algorithm. *SIAM J. Optimization*, 1:123–145, 1991.

[29] V. Torczon. On the convergence of pattern search methods. *SIAM J Optimization*, 7(1):1–25, Feb 1997.

[30] M. D. Vose and G. E. Liepens. Schema disruption. In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Intl. Conf. on Genetic Algorithms*, pages 237–242, San Mateo, CA, 1991. Morgan-Kaufmann.

[31] D. Whitley, S. Dominic, and R. Das. Genetic reinforcement learning with multilayer neural networks. In R. K. Belew and L. B. Booker, editors, *Proc. of the Fourth Intl. Conf. on Genetic Algorithms*, pages 562–569, San Mateo, CA, 1991. Morgan-Kaufmann.

[32] W. S. Wong and R. J. Morris. A new approach to choosing initial points in local search. *Information Processing Letters*, 30:67–72, 1989.