Missouri Estimation of Distribution Algorithms Laboratory

Generator and Interface for Random Decomposable Problems in C

Martin Pelikan, Kumara Sastry, Martin V. Butz, and David E. Goldberg

MEDAL Report No. 2006003

March 2006

Abstract

This technical report describes how to download, compile and use the C source code of the generator of random additively decomposable functions and the functions that enable the use of the generated instances in any standard optimization algorithm.

Keywords

Problem generator, additively decomposable functions, decomposable problems, performance analysis, scalability, optimization.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

Generator and Interface for Random Decomposable Problems in C

Martin Pelikan
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
pelikan@cs.umsl.edu

Kumara Sastry
Illinois Genetic Algorithms Laboratory, 107 TB
University of Illinois at Urbana-Champaign
104 S. Mathews Ave., Urbana, IL 61801
kumara@illigal.ge.uiuc.edu

Martin V. Butz
Department of Cognitive Psychology
University of Würzburg
Röntgenring 11, 97070 Würzburg, Germany
mbutz@psychologie.uni-wuerzburg.de

David E. Goldberg
Illinois Genetic Algorithms Laboratory, 107 TB
University of Illinois at Urbana-Champaign
104 S. Mathews Ave., Urbana, IL 61801
deg@illigal.ge.uiuc.edu

Abstract

This technical report describes how to download, compile and use the C source code of the generator of random additively decomposable functions and the functions that enable the use of the generated instances in any standard optimization algorithm. For more information about the considered class of test problems, please see Pelikan, Sastry, Butz, and Goldberg (2006), which is also available for download at http://medal.cs.umsl.edu/.

Keywords: Problem generator, additively decomposable functions, decomposable problems, performance analysis, scalability, optimization.

1 Downloading the Code

The code can be downloaded from the following web address:

`http://medal.cs.umsl.edu/files/decomposable-problems.tar.gz`

After downloading the file, move the file into the directory where you want the generator to reside. Then, use your favorite compression package to unpack the downloaded archive. In UNIX systems, you can use gunzip and tar from the directory with the downloaded archive as follows:

```
gunzip decomposable-problems.tar.gz
tar xvf decomposable-problems.tar.gz
```

On most UNIX systems, you can also use a single command,

```
tar xvzf decomposable-problems.tar.gz
```

All files will extract into a directory `decomposable-problems`. After examining the content of this directory, you will notice the following 2 files and 3 subdirectories:

```
README
howto.pdf
generator/
instances/
solver/
```

`README` contains basic information about the code. `howto.pdf` contains the text of this document. The three subdirectories contain the code and example instances: `generator` contains the code of the problem generator, `instances` contains several example generated instances, and `solver` contains the interface functions for testing optimization algorithms with the generated problem instances and a simple hill climber that provides an example use of the interface functions.

2 Compiling the Code

The source code is written according to ANSI C standard and it should thus compile with most available C compilers.

To compile the code for the generator of random problem instances, go to `generator` subdirectory first. The provided `Makefile` can be used on most UNIX systems to compile the code using GNU make, and can be modified appropriately to compile in other systems as well. To compile, run `make`. The result will be an executable named `generator`.

Analogically, to compile the functions for incorporating the generated problem instances into your own code, go to `solver` subdirectory first and run `make` after appropriate modifications of `Makefile`. The result will be an executable named `solver`.

3 Source Code Files

The source code of the generator consists of 3 files:

- `main.c`
  Defines most high-level functions, including the problem generator and the main function that processes arguments and controls the generation of new problem instances.
- `random.c`
  Defines functions for generating random numbers.
- `random.h`
  The header for `random.c`.

The source code of the solver consists of 7 files:

- `main.c`
  Defines the main function, which processes arguments, loads the problem instance, calls the solver, and frees up the memory in the end. The file can be used to gain understanding of the provided interface for testing optimization algorithms on the generated problem instances.
- `decomposable-problem.c`
  Provides functions for loading and evaluating the generated problem instances as well as testing candidate solutions for optimality. This is the file (together with the corresponding header file) that enables the generated problem instances to be used for testing various optimization algorithms. Examples of using these functions are provided in `hc.c` and `main.c`.
- `decomposable-problem.h`
  The header for `decomposable-problem.c`.
- `hc.c`
  Provides a simple hill climber based on bit-flip mutation, which serves as an example optimization algorithm that uses provided functions for evaluating the generated problem instances and testing candidate solutions for optimality in the generated problems.
- `hc.h`
  The header for `hc.c`.
- `random.c`
  Defines functions for generating random numbers.
- `random.h`
  The header for `random.c`.

4 Running the Code

To generate new problem instances, run "generator" without any arguments, which will provide a short help on using the generator. The algorithm takes 5 integers as arguments: N, n, k, *overlap*, and *seed*. N is the number of problem instances to generate; n is the total number of bits in the generated problems; k is the size of the subproblems (order of decomposition); *overlap* is the amount of overlap and should be between 0 and $k/2$; finally, *seed* denotes the random seed (unsigned integer).

For example, to generate 100 random problems of 10 subproblems of 4 bits each with *overlap*=1 where the overall number of bits is 4+3*9=31, we would run the following (1234 is the random seed):

```
./generator 100 31 4 1 1234
```

The new instances are named pn_k_o_i, where n is the total number of bits, k is the size of subproblems, o is the size of the overlap (from 0 to $k/2$), and i is the number of the instance (to distinguish *different* instances with the same values of n, k, and o). In the above example, instances will match the mask `p31_4_1_*` where the star will be replaced by the id number of the instance (from `0` to `99`).

The solver can be run either with no arguments (using the default file name of "p20_4_0_0", which is included), but one can specify an argument for the file name of the input problem instance. For example, we can copy the solver into the subdirectory `instances`, and then run the solver on one of those instances as follows:

```
./solver p31_4_1_5
```

The solver then runs a simple hill climber on that instance and does not terminate until the global optimum has been found. This can sometimes take long time, of course. However, the code is not to provide an efficient solver but to show how one can incorporate generated problems into one's code.

All functions required for incorporating the generated problem instances into one's code are located in `decomposable_problem.c` and are defined in `decomposable_problem.h`. For examples of using these functions in optimization algorithms, consult `hc.c` and `main.c`.

5 Final Comments

The code is distributed for academic purposes with absolutely no warranty of any kind, either expressed or implied, to the extent permitted by applicable state law. We are not responsible for any damage from its proper or improper use.

Feel free to use, modify and distribute the code with an appropriate acknowledgment of the source, but in all resulting publications please include a citation to Pelikan, Sastry, Butz, and Goldberg (2006).

The code contains README files, which provide basic information, and it also contains HTML documentation generated by doxygen (in `html` subdirectories of the directories with the code). The code is written according to ANSI C standard and is commented.

Acknowledgments

This work was supported by the Research Award and the Research Board at the University of Missouri, the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, the National Science Foundation under NSF CAREER grant ECS-0547013 and ITR grant DMR-03-25939 at Materials Computation Center, UIUC, and ITR grant DMR-0121695 (at CPSD), and the Dept. of Energy under grant DEFG02-91ER45439 (at Fredrick Seitz MRL).

The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Most experiments were completed at the Beowulf cluster at the University of Missouri–St. Louis.

References

Pelikan, M., Sastry, K., Butz, M. V., & Goldberg, D. E. (2006). *Hierarchical BOA on random decomposable problems* (MEDAL Report No. 2006001). Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO.