Development system DEBOA for rapid prototyping of evolutionary applications

Jiří Očenášek*
ocenasek@fit.vutbr.cz
Josef Schwarz*
schwarz@fit.vutbr.cz

Abstract: This paper deals with the design of the modular development system DEBOA for the rapid prototyping of optimization applications on the basis of Bayesian Optimization Algorithm [1]. The general requirements on the development system are identified including modularity and visualization of results. The design of the development system itself is realized on the platform of Java language, which ensures great portability. DEBOA system can be easily extended to support new types of fitness functions as well as new types of visualizations.

Key Words: Evolutionary algorithm, Bayesian optimization algorithm, decision graphs, optimization problems, rapid prototyping, Java platform

1 Introduction and Motivation

In recent years evolutionary algorithms have become more and more popular tool for solving different optimization problems. The creation of a well-performing classical evolutionary algorithms [2] highly depends on problem encoding, genetic operators and control parameters.

In recent years, several researchers have concentrated on using probabilistic models in genetic algorithms. These Estimation of Distribution Algorithms (EDA) incorporate methods for automated learning of linkage between chromozome genes in population. The process of sampling new population from probabilistic model respects these mutual dependencies among genes such that disruption of important building blocks is avoided, unlike of classical recombination operators. The most advanced EDA algorithm is the Bayesian Optimization Algorithm (BOA) [1], which uses Bayesian network as the general probabilistic model for optimization problems with discrete parameters. The newest version of BOA utilizes the set of decision graphs instead of Bayesian network to further refine the description of gene dependencies.

In the field of classical EA there are many development environments for fast prototyping, but in the field of EDAs these tools are missing. We investigated the last version of the BOA [3] and design the DEBOA development system for fast prototyping of optimization tasks.

In the next chapter an overview of several typical development systems based on classical evolutionary algorithms is discussed. In chapter 3 the common features of existing systems and the most important requirements for modern development system is specified. In chapters 4 and 5 the design and implementation details of the DEBOA development system, including the most important parts from the user guide is design.

* Faculty of Information Technology, Department of Computer Systems, Božetěchova 2, CZ-612 66 Brno

2 Particular EA Tools

We present a few EA tools that we have applied personally on typical evolutionary optimization problems.

2.1 GOD

The Genetic Object Designer (GOD) [4] is a Windows based genetic algorithm shell using EOS (Evolutionary Object System) library. It facilitates the exploration of many variants of the traditional GA when applying GAs to a specific problem. GOD comes with an interpreter of EPL built in language (Evolutionary Programming Language) for purposes of specifying the domain specific parts of a genetic algorithm. The final GA can also be exported to application written in C++ using the "Generate C++ Code" dialog.

2.2 GADESIGN

The GADesign toolkit [5] was created at Brno University of Technology by Libor Kriz as a part of his master theses. Its modular concept is based on the platform of object-oriented Pascal language in the frame of the Delphi system. This development tool allows to design a proper genetic algorithm without detailed knowledge of genetic theory using only an intuitive process of specification of the genetic classes, functors and parameters. The user code can be loaded from DLL or interpreted using built-in interpreter of Pascal-like language. To the creation of the stand alone application is possible by using the DLL library containing GADesign evolutionary engine.

2.3 GALIB

Galib [6] a source code library of genetic algorithm objects. It allows for using genetic algorithms to do optimization in any C++ program. The library includes basic types of genetic algorithms, chromozome encoding and genetic operators.

2.4 BOA

The utilizing of C++ source code of Bayesian Optimization Algorithm is described in [3]. This program package allows for solving simple binary optimization problems.

We have a long-time experience with the application and improving BOA. That is why we decided to adopt the latest version of BOA [3] based on decision graphs for DEBOA development system with following properties:

- redesign of BOA core
- reusability of BOA core for variety of applications
- visualisation of BOA evolution process
- interactive parameter settings

3 Demands on Development System

3.1 Visualization

By the visualization we mean the ability of EA development system to provide as many information about the evolution as possible. The common visualization can be divided into two classes:

- visualization of EA evolution process
- visualization of current population distribution in solution space

3.2 Extensibility

Modern EA toolkits should be implemented in a modular way, such that a new operator, visualization or fitness function can be added. After building a final source code for solved optimization task two extensibility modes can be used:

- Interpreted code
- Binary code using DLL files

The interpreted mode uses built-in interpreter of pseudo-programming language in which the user code can be written. The binary one is faster, because the precompiled binary code is dynamically loaded and executed (for example from .DLL file or Java .class file).

3.3 Reusability

Most of EA toolkits support only prototyping, but some of them can also export settings and operators into final application such that the optimization engine needs not to be programmed again. For example the core of reusable EA toolkit is contained in .DLL file and its API functions are called from the source code generated according to optimal settings found.

3.4 Portability

By the portability we mean the transfer of EA toolkit from one platform to another. Usually the commercial toolkits contain platform-specific binaries whereas the open-source toolkits are easily portable.

4 Design and Implementation of DEBOA System [8]

4.1 Java Language

We adopted the original BOA package in C++ [3] and proposed new object oriented modular design based on Java language.

Java contains several features that argue for it. It is widely distributed and has become one of the major programming languages. The development kit, including compiler and debugger, is freely available on a number of different computer platforms. The core libraries contain many functions which can be used directly and need not be adopted from external libraries, which is not the case in C++ for instance. Another important point for future work is the built-in support for multithreading and distributed computing.

Java is a strictly object-oriented language. By combining the object-oriented desing with the other Java features, our project has several unique advantages:

- in the Java applet mode it is suitable for WWW presentations
- pre-compiled modules with new visualisations and new fitness computation can be dynamically loaded during run-time as new classes
- the class files are easily portable, no need to distribute the source code
- the class containing optimization core is reusable, it can be easily incorporated into the final project

4.2 Modular Design

For object oriented design the “design patterns” technique was used. The simplified scheme of our system is shown in the following UML diagram.

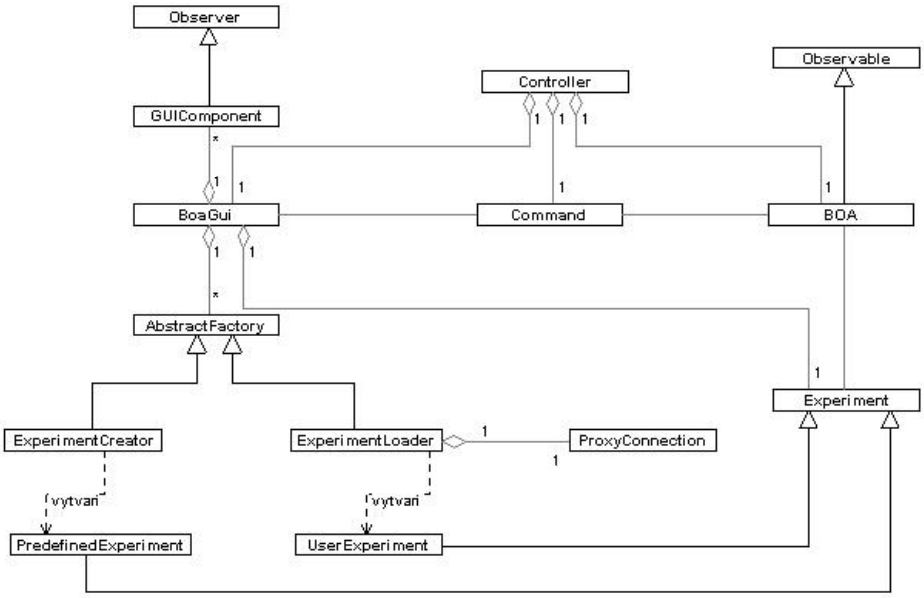


Fig. 1: The UML diagram of DEBOA

4.3 Visualisation of Characteristics

We implemented the following common visualisations:

- Statistics – basic statistics of optimization process
- Fitness – graph of minimal, average and maximal fitness function values
- Fitness Hist – fitness distribution histogram
- Similarity – minimal, average and maximal Hamming distance between individual genotype and global optimum genotype if known
- Similarity Hist – similarity distribution histogram

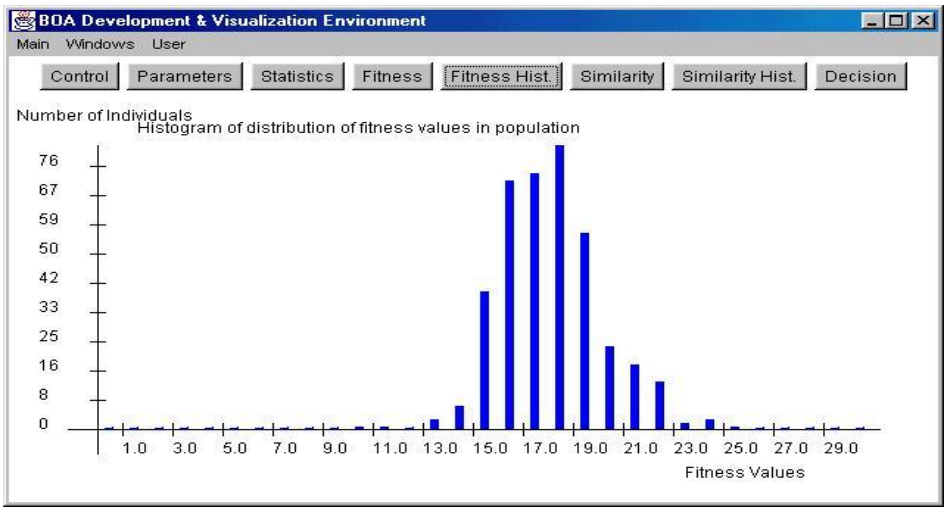
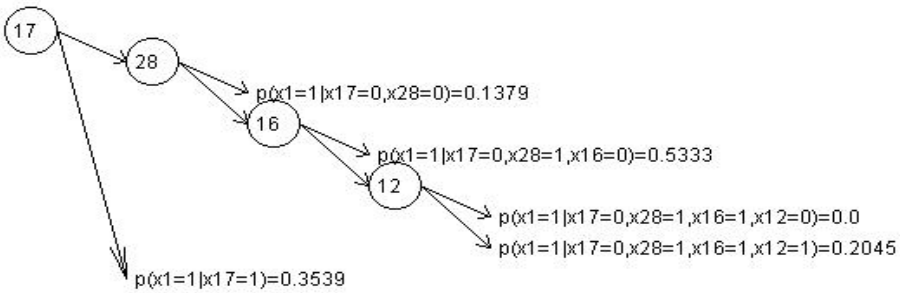


Fig. 2: Fitness distribution histogram

Moreover, we implemented the visualisation of BOA probabilistic model:

- Decision – decision trees for each variable/gene of the chromosome



x12	x16	x17	x28	P
*	*	0	0	0.13793103
*	*	1	*	0.3539823
*	0	0	1	0.53333336
0	1	0	1	0.0
1	1	0	1	0.20454545

Fig. 3: Binary decision diagram

4.4 Tool for Rapid Prototyping

Due to modular architecture the system can be easily extended. New experiments are inherited from the class *develop.user.Experiment.* Both types of extensibility are supported. We created the class *ExperimentCreator* for loading of built-in experiments and class *ExperimentLoader* for dynamical loading of experiments during run-time. Thanks to the class *ProxyConnection* the dynamical loading is transparent and can be done from local filesystem as well as from URL.

To create new experiments the users can use these steps :

1) Create new file NewExperiment.java
2) Import interface *develop.boa* using the directive *import*, if methods working with populations or other data types defined in this package should be redefined
3) Create new class by extending already existing class *develop.user.Experiment* using keyword *extends*
4) Declare new variables and data types for new experiment
5) Define the method Init() that sets up the parameters of the experiment (name, menu items names, names of user graphs, etc.) and the flag *isBestDefined* if the best individual is defined
6) If required, redefine methods *FitnessFunction(), isOptimal(), howSimilarToOptimal(), Init(), updateStatistics(), alterPopulation(), LoadData(), SaveData(), DrawUserGraph1(), DrawUserGraph2()*
7) Compile new experiment using command: *javac NewExperiment.java*

Each experiment class provides support for common visualisations. For example the method *howSimilarToOptimal()* computes the Hamming distance between individual and global optimum, which is needed for Similarity graph. Moreover, the base class *develop.user.Experiment* declares methods *DrawUserGraph1(java.awt.Graphics g)* and *DrawUserGraph2(java.awt.Graphics g)*. The addition of application-specific visualisations can be implemented by redefining them.

4.5 Built-in Benchmarks

We have implemented the following implicit benchmarks:

- Graph partitioning
- Knapsack problem
- Optimization of artifical functions (OneMax function, 3-deceptive function, …)

5 User Guide

5.1. DEBOA Versions

The development system DEBOA can be used in two versions :

- Java Applet – this version is designed for the presentation of the system in WWW browsers, which includes built-in Java interpreter.
- Java Aplication – the full version allows to design new application.

To compile and run DEBOA it is necessary to use JDK 1.1.8 or higher (Java Development Kit), that includes Java interpreter, compiler and AppletViewer for running applets, and operating system with GUI, because the system runs only in graphical mode.

In case of Java applet the www browser is usable. But it is important to set access rights for unsigned Java Applets in the browser:

- read / write access to user files
- dialogs opening

If a browser is not capable of setting access rights for Java, it is still possible to run system with following constraints :

- System is not allowed to generate output files of BOA nor the output files defined by the user parameters „Evolution history file“ and „Data file for experiment to save“. Thus, in window „Parameters“ these settings must be set blank.
- System is also not allowed to read user defined classes with experiments nor user data files, thus also parameters „Data file for experiment to load“ and „Experiment class filename“ in the same window must be set blank. Only predefined experiments are accessible.

5.2. Running the System

After compiling and launching the program (applet or application version) the main window opens. There are two equivalent ways to switch between windows of the program :

- user menu
- button panel located below the menu bar

The "Main" window contains controls button for loading experiments and for loading or saving user data. The user can run, step and stop the optimization process thanks to control panel of the application that is accessible in the window „Control". The menu „Windows" gives the user access to all windows displaying the visualized characteristics. The menu „User" gives the user access to user defined graphs. The user can define two user output graphs altogether with their names that will appear in the menu „User". These items can differ from experiment to experiment, see section 4.4 .

5.3. Loading User Defined Experiments

Lets suppose that we have created the new experiment called MyExperiment. After its compilation we get file MyExperiment.class. Furthermore we suppose that this experiment will load the data from file C:\my.dat and write the data to file C:\my.out.

1) Launch DEBOA
2) Switch to window „Parameters"
3) Set the parameter „Data file for experiment to load :" to „C:\my.dat"
4) Set the parameter „Data file for experiment to load :" to „C:\my.out"
5) Set the parameter „Enter Experiment class filename or press button to locate it" to „$path\MyExperiment.class", where $path is the path to file MyExperiment.class. In case that we want to run the experiment placed on the WWW we can do that by specifying the URL of the file instead of path, for example:

   „http://www.experiments.com/MyExperiment.class"
6) Switch to window „Control" and run the experiment

5.4. Description of Output Files

Our DEBOA system produces the output files:

- *.fitness - the history of min./max./avg. fitness
- *.log – more detailed history, including best individuals
- *.model – the history of model evolution, a set of decision graphs for each generation

Note: * is the name of output file specified in „Parameters" window

6. Conclusion

We have implemented the evolutionary design system DEBOA for rapid prototyping of optimization tasks. It includes all the important features of evolutionary toolkits – visualisation, extensibility, reusability and portability. Due to using of Java language our system fulfills all the requirements for modern development system. One of its unique properties is the ability to be executed as java applet for WWW presentations. As far as we know, DEBOA is the only development environment based on Bayesian optimization algorithm for efficient solution of complex optimization problems with high epistasis.

The future work will be focused on an extension of the development system to parallel version

.

6.1 Future work - design and simulation of parallel BOA

One of our main goals is to provide a toolkit, which supports the design of parallel BOA algorithm. The classical parallel GAs use the island model where the global population is split into several local populations and some migration rules are used to exchange the individuals. Unfortunately, this approach is not reasonable for BOA, because the local models created from the local populations would be far more inaccurate and useless for creation of new individuals than the global model created from the whole aggregated population.

In paper [9] we proposed the original parallel BOA based on the parallel construction of probability model – each processor creates its part of probability model and then the whole model is collected from those parts using the collective message passing communication. To ensure the BN acyclicity, the algorithm for parallel construction of Bayesian network defines the topological ordering of BN nodes in advance, such that only one direction of edge is allowed between each two nodes.

In our toolkit the future work lies in the implementation of this parallel BN construction using multithreaded features of Java language. This multithreaded parallel algorithm will speed up the execution time nearly *m*-times, where *m* is the number of processors used. The advantage of this approach is that it is not based on the island model, so the experimentation with migration rules is not required. Reusing of our toolkit in the final project will result in parallel application.

This research has been carried out under the financial support of the Research intention no. CEZ: J22/98: 262200012 - "Research in information and control systems" (Ministry of Education, CZ) and the research grant GA 102/02/0503 "Parallel system performance prediction and tuning" (Grant Agency of Czech Republic).

References

1. Pelikan M., Goldberg D. E. , Sastry K. : Bayesian Optimization Algorithm, Decision Graphs, and Ocams' Razor, Illigal Report No. 2000020
2. Goldberg, E.D.: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison – Wesley Publishing Company, Inc., 1989
3. Pelikan M.: A C++ Implementation of Bayesian Optimization Algorithm (BOA) with Decision Graphs, Illigal Report No. 2000025.
4. Genetic Object Designer – Documentation, Man Machine Interfaces, 1994
5. Kříž L.: Development Tools for a Fast Genetic Algorithms Design, master thesis, Faculty of Informaton Technology, VUT Brno, Brno, 2000
6. Wall, M. : GAlib C++ Library of Genetic Algorithm Components, version 2.4, Documentation Revision B, August 1996
7. Corcoran, Wainwright : LibGA a user-friendly workbench for order-based genetic algorithm research, ACM Press, New York 1993
8. Kostka L.: The development system for the class of Bayesian Optimization Algorithm, Faculty of Informaton Technology, VUT Brno, Brno, 2001
9. Očenášek, J., Schwarz, J.: The Distributed Bayesian Optimization Algorithm, Proceeedings of the Eurogen 2001 - Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems, The National Technical University of Athens, Greece, 2001