

Object Oriented Toolkit for Multiobjective Genetic Optimisation

Rajeev Kumar, N. Vinay Kumar and I. J. Nagrath
Centre for Robotics & Intelligent Systems
Birla Institute of Technology & Science
Pilani - 333 031, India
rajeevk@bits-pilani.ac.in

Abstract

Evolutionary computations are emerging as powerful tools for search and optimisation, and increasingly being used in many scientific and engineering applications. Side-by-side object-oriented computing has revolutionised, during the current decade, the style of programming and the software system design and development which is now configured around 'class' concept. In this paper, we present a general-purpose object oriented toolkit which serves as a generic utility for wide ranging applications involving optimisation of both single and multiple objectives. The toolkit supports the state of the art of genetic optimisation techniques; the design is modular, flexible and extensible inline with object oriented programming paradigm. The toolkit is currently being implemented in C++ for obvious reasons of wider support and portability across platforms. Templates and derived classes are used for elegance and re-use of the code and the library. The interfaces try to hide as much as implementation details as possible so that the programming and modification at higher level become simple. Nonetheless, defining interfaces is an iterative process so with the design and implementation of the toolkit, with each major addition and upgradation, they are constantly evolving.

1. Introduction

Evolutionary Computations (EC) are emerging as powerful tools to many real-world applications of diverse nature, and are increasingly being used in problem domains which can be (re-)defined in terms of search procedures and subsequent optimisation of objective function(s). The term EC includes all the paradigms of genetic optimisations, *e.g.*, Genetic Algorithms (GAs), Evolutionary Programming (EPs), and Evolutionary Strategies (ESs) [1]. We do not wish to discriminate among them, rather we are interested to encapsulate in EC, all the methods of 'simulating evolutions on computers' which mimic the biological processes of *survival of the fittest*. Simulating evolutions implies that there is always a need of test-beds to explore different evolutionary strategies and algorithms for different applications. With this view, one of the very first genetic utility Simple Genetic Algorithm (SGA) [2] supports restricted capabilities of a genetic simulation environment in procedural/modular programming paradigms of Pascal/C languages.

The procedural/modular programming languages have limitations of flexibility, extensibility and re-organisation, and such limitations restrict their uses in development of large software systems/simulation environments where technological changes demand constant re-use, understanding and upgradation of existing code and libraries [3]. In this context, the emergence of object oriented computing (OOC) has influenced the software system design and development. OOC paradigm assumes a design will have to evolve - that is the programming

code/system will have to be extended, ported and upgraded in a number of ways that can not all be foreseen [4]. In fact, it is realistic to assume that the requirements for the system will change with time - between the time of the initial design and the first release of the system, first release of the system and the subsequent upgradation of the system - in keeping with the demands/changes of the application technology. The direct implication is that the system must be designed to remain as simple as possible with implementation independent simple interfaces, and in wider terms, it must truly be designed and configured around the concept of 're-use'. The language of choice in object oriented programming paradigms, is obviously C++ because of wider support across platforms, rich libraries supporting scientific computations, and easy to use and familiar constructs [4-6].

One such C++ Genetic Algorithm Library, GALib [7], exists but it supports problem-solving of single objective optimisation only. In practice, many problems require the simultaneous optimisation of a number of, possibly competing, objectives - such problems fall into the domain of multiobjective genetic optimisation problem solving [8]. One of the first multiobjective genetic optimisation toolbox was developed by Chipperfield *et al* [9] at University of Sheffield; the toolbox is implemented in MATLAB environment. In general, applications embedded in MATLAB environment are good only for learning/teaching. They support only static data types and demands excessive memory at run-time resulting in slow processing. They inherit all the drawbacks of modular programming; they are not portable and do not optimise compute resources.

In this paper we present a general purpose object based toolkit which serves as a generic utility for applications involving simultaneous optimisation of multiple objectives. The toolkit supports the state of the art of genetic optimisation techniques, and the design is inline with object oriented programming paradigm [4]. We discuss the system architecture and the interfaces. The toolkit is currently being implemented in C++ and the emphasis is on elegance and software re-use. The design and the implementation both are evolving as the new features are being added; we wish to provide the finer implementation details in the final version of the paper, length of which shall be guided by the maximum page-limit.

2. Multiobjective Optimisation Strategy

Multiobjective optimisation is a superset of single objective genetic optimisation. In simple treatments, the multiple objectives are combined in an *ad hoc* manner to yield a scalar objective, typically by linear combination of the vector elements, and constraints incorporated with associated thresholds and penalty functions. Though the process is much simplified optimisation algorithm, the obtained solution is very sensitive to small changes in weight vector and penalty function. Secondly such a solution based on pure heuristics is mostly unproductive, *i.e.*, the solutions for other choices of weights and constraints are discarded. A more satisfactory approach is to use the notion of Pareto optimality first proposed by Goldberg [2] in which an optimal set of solutions prescribe some surface - the Pareto-front - in the vector space of the objectives. For a solution on the Pareto-front no objective can be improved without simultaneously degrading at least one other. Key to Pareto optimality is the concept of *domination*; there exists a family of alternative solutions that are superior to the rest of the solutions and are considered equivalent from the perspective of simultaneous optimisation of multiple and possibly competing objective functions - see [8,10] for detailed review and description.

Mathematically, a general multiobjective optimisation problem containing number of objectives to be maximised/minimised along with (optional) constraints for satisfaction of achievable goal vectors can be written as:

$$\begin{aligned} & \text{Minimise / Maximise Objective } f_m(\mathbf{X}) & m = 1, 2, \dots, M \\ & \text{subject to Constraint } g_k(\mathbf{X}) \leq c_k & k = 1, 2, \dots, K \end{aligned}$$

where $\mathbf{X} = \{x_n : n = 1, 2, \dots, N\}$ is a N – tuple vector of variables;
and $\mathbf{F} = \{f_m : m = 1, 2, \dots, M\}$ is a M – tuple vector of objectives.

Goldberg's [2] condition of Pareto-optimality is stated as: in an minimisation problem, *if* an individual objective vector \mathbf{F}_i is partially less than another individual objective vector \mathbf{F}_j (symbolically $\mathbf{F}_i < \mathbf{F}_j$) *iff*

$$(\mathbf{F}_i < \mathbf{F}_j) \triangleq (\forall_m)(f_{mi} \leq f_{mj}) \wedge (\exists_m)(f_{mi} < f_{mj})$$

then the individual \mathbf{F}_i dominates the individual \mathbf{F}_j . If an individual is not dominated by any other in the population, it is said to be *nondominated*.

From computational point of view, the multiobjective genetic optimisation needs the use of Pareto-ranking/nondominance, niching and mating restrictions in variable/objective space, and a different notion for convergence. For a single objective GA gauging convergence is almost trivial. For multiple objectives, there is no simple matter. A systematic approach based on histograms of rank is proposed for assessing convergence to the Pareto-front which, by definition, is unknown in most of the real-life search problems [11]. This approach requires the generation and comparison of *intra*-tribal rank histograms for successive epochs of evolutions within a tribe, and optionally the *inter*-tribal rank histograms for comparing and merging nondominated solutions among tribes. The toolkit under development support such strategies, and can be configured to a user-chosen one.

3. Toolkit Design and Architecture

The toolkit architecture and interfaces are shown in Figure 1. Chromosomes can be represented as binary strings or real-valued numbers. They can be of fixed or variable length. They can be built from any C++ data types - bit strings, arrays, lists or tress - using the types built-in to the library. Random number generator routines [6] are provided for initialisation and

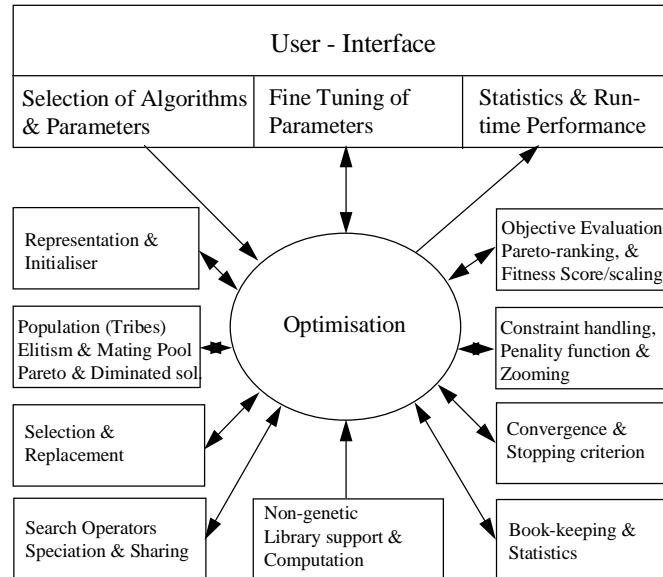


Figure 1: System architecture and interfaces

generating random points at various stages of evolution. Both generational and steady-state populations are supported. Elitism is optional for generational type of GAs so that a fraction of the best-fit individuals are copied into the mating pool. A bit-tag is attached to each individual of the population which indicates its non-dominating or dominating position on the Pareto front. Selection and replacement strategies are provided. Search operators include a variety of cross-over and mutation operators which also include the constraints for variable length representations.

A user can code his objective functions and opt for the genetic strategy. The population can be ranked on Pareto-front and fitness score calculated. For constraint optimisation, the toolkit design imposes constraints on variables, and supports penalising the objectives and zooming-out the Pareto-front in a region of interest. A stopping criterion can be defined - to run the GA for some fixed number of iterations, to terminate the optimisation when some fraction of the population has become non-dominated, or assessing the convergence of Pareto-front using rank-histograms.

The run-time statistics of population evolutions and objective function evaluations can be recorded by setting which statistics should be recorded and how often they should be. Most commonly used statistics are maximum, minimum, mean, standard deviation, movement of Pareto-front, age-distribution of the nondominated solutions and the rank-histograms. GA parameters can be configured from file, code and/or user-interface. User can also fine-tune the parameters while viewing the run-time performance and analysing the generational statistics.

4. Implementation

We aim to deliver the toolkit so that it can be used on multiple platforms. However, at the moment we are developing the back-end on Egcs/Gnu C++ compiler, which has the support for standard ANSI C++, and a rich template library, on a Unix/Linux machine. We are using templates extensively for optimising the code and having the best of 're-use'. We are trying to make interfaces as simple as they can be by hiding as much implementation details as possible. The major thrust is on arriving at an elegant design. We have also minimised allocating and deallocating time of small objects and avoided fragmentation of memory by pre-allocating 'chunks' of objects and link them together to reduce allocation and deallocation to simple linked list operations.

The toolkit makes extensive use of STL library [5]. Although it is possible to directly use all of STL containers for representing genomes, an intermediate layer is added to enhance the data types. This layer provides the abstraction of the data types. Because of this abstraction layer, the other components, such as genetic operator, *e.g.*, crossover and mutation need not to know about the actual data structures. The intermediate layer provides all the operations, which are most commonly used in GAs. All the operators work on all kinds of genomes, in an efficient and seamless manner.

The Graphical User Interface (GUI) is currently being developed using Troll Tech's Qt library which is portable across Linux and Windows. The GUI is designed for the user to select the algorithms, input the parameters and configure the application. The GUI also prompts the user with the run-time statistics. In order to simplify the development of GUI based GA applications, wizards are also provided. This takes away the burden of coding the GUI, and letting the user concentrate only on his objective functions. The user interface library is separate from the backend. The backend libraries can also be used without a GUI. This feature allows the systems to run on applications embedded within other applications.

Our implementation has a Genetic Algorithm Object on the top. The GA Object requests the genetic operators and interacts with both type of genetic optimisation - single objective and

multiple. In multiple optimisation, it gets additional information regarding dominance on pareto-front. The convergence that is a trivial process for single optimisation, is monitored by the tail of the rank-histograms in successive evolution. The multi-objective optimisation yields a set of solutions, which are not inferior but optimal on the pareto-front. The required solutions are handpicked by prioritising the objectives.

5. Conclusions and Ongoing Work

We have provided a general-purpose object oriented toolkit, which serves as a generic utility for applications involving simultaneous optimisation of multiple objectives. The toolkit can be used for both - fast development of prototypes for experimentation as well for developing applications. The toolkit design supports the state of the art of genetic optimisation techniques. We have discussed the system architecture and interfaces. The toolkit is currently being implemented in C++ and the emphasis is on elegance and software re-use. The design and the implementation both are evolving as the new features are added. At the moment the framework is complete and the toolkit has been validated on simple multiobjective problems as proof of demonstration. The current version is implemented on Egcs/Gnu C++ compiler but we aim to make it portable across platforms. We are currently adding a window-based interface for ease of use and gathering run-time statistics. At a later stage, we plan to provide application-generators so that the user has to code only the specialised objective functions needed for a particular application.

6. References

- [1] T. Bäck, D. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, New York: Oxford University Press, 1997.
- [2] D. E. Goldberg, *Genetic Algorithms in Search, Optimisation and Machine Learning*, Reading, MA: Addison-Wesley, 1989.
- [3] B. Stroustrup, *Design and Evolution of C++*, Reading, MA: Addison-Wesley, 1994.
- [4] G. Booch, *Object Oriented Analysis and Design with Applications*, 2nd Ed., Benjamin Cummings, 1994.
- [5] B. Stroustrup, *The C++ Programming Language*, 3rd Eds., Reading, MA: Addison-Wesley, 1997.
- [6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd Ed., Cambridge, UK: Cambridge University Press, 1992.
- [7] GALib: A C++ Genetic Algorithm Library, URL: <http://lancet.mit.edu/ga/>, Cambridge, MA: Massachusetts Institute of Technology, 1996.
- [8] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimisation", *Evolutionary Computation*, vol. 3, no. 1, pp. 1-16, 1995.
- [9] A. Chipperfield, P. J. Fleming, H. Pohlheim, and C. M. Fonseca, "GA toolbox user's guide", Research Report 512, Dept. Automatic Control & Systems Engineering, University of Sheffield, UK, July 1994.
- [10] R. Kumar, *Feature Selection, Representation & Classification in Vision*, Ph. D. Thesis, Dept. Electronic & Electrical Engineering, University of Sheffield, UK, March 1997.
- [11] R. Kumar and P. I. Rockett, "Assessing the convergence of rank-based multiobjective genetic algorithms", in *Proc. IEE/IEEE Second Int. Conf. Genetic Algorithms in Engineering Systems: Innovations & Applications (GALESIA 97)*, IEE Conference Publication No. 446, 1997, pp. 19-23.