The Journal of Open Source Software

Quilë: C++ genetic algorithms scientific library

Tomasz Tarkowski [1]

1 Chair of Complex Systems Modelling, Institute of Theoretical Physics, Faculty of Physics, University of Warsaw, Pasteura 5, PL-02093 Warszawa, Poland

DOI: 10.21105/joss.04902

Software

- Review
- Repository
- Archive

Editor: Vissarion Fisikopoulos

Reviewers:

- @mbarzegary
- @acrlakshman

Submitted: 18 October 2022
Published: 07 February 2023

Summary

This work discusses a general-purpose genetic algorithms (Holland, 1975) scientific header-only library named Quilë. The software is written in C++20 and has been released under the terms of the MIT license. It is available at https://github.com/ttarkowski/quile/. The name of the library come from the fictional language Neo-Quenya and means "color" (cf. origin of the word *chromosome*).

Genetic algorithms, or more broadly, *evolutionary computations*, is a field of computer science devoted to population-based, trial-and-error methods of problem solving. Evolutionary computation was invented by Alan Turing by noting that the principles of biological evolution and genetics can be applied to optimization problems (Turing, 1948, 1950). One of the first evolutionary computations performed on a computer was done by Nils A. Barricelli (Barricelli, 1962; Galloway, 2011). Evolutionary computations have found wide applications in many disciplines (Drachal & Pawłowski, 2021; Ghaheri et al., 2015; Goudos et al., 2016; Katoch et al., 2021; Kudjo et al., 2017; Lee, 2018).

The genetic algorithm is a conceptually simple procedure. The aim is to find the most optimal solution of a given optimization problem. First, one begins with some sampling of the space of potential solutions; this can be done randomly or *ad hoc*. This procedure forms the first population of the evolutionary process, i.e., its first iteration. Each candidate solution from the population is evaluated in terms of its "fitness". A "fitter" candidate solution has a greater probability of becoming a parent and entering the next population of the evolution's subsequent iteration. However, less "fit" individuals still can propagate, albeit with lower probability. This trait of evolutionary computations helps preventing premature optimization to the wrong local optimum. Iterations of the evolution continue as long as the termination condition is not met. One example of a termination condition is reaching the fitness function *plateau*.

Overview

Nature of problem: Floating-point, integer, binary, or permutation single-objective constrained finite-dimensional optimization problems of arbitrary nature, i.e., maximization of $f\colon G \to \mathbb{R}$, where $G \subseteq \prod_{i=0}^{c-1} X_i$, where X_i is equal to set of logical values $\mathbb{B}$ (i.e. false and true) or is bounded subset of set of real numbers $\mathbb{R}$ or integer numbers $\mathbb{Z}$.

Solution method: Single-objective constrained genetic algorithm with pure floating-point, integer, binary, and permutation representation with set of exchangeable components (e.g., variation operators, selection probability functions, selection mechanisms, termination conditions).

Unique features: Header-only and easy-to-deploy genetic algorithms C++20 library tailored for problems with computationally expensive fitness functions.

Functionality:

- Floating-point, integer, binary, and permutation genotype representations.

- Automatic compile-time representation/variation compatibility checks.
- Mutation operators: Gaussian, self-adaptive, swap, random-reset, and bit-flipping.
- Recombination operators: arithmetic, single arithmetic, one-point crossover, and cut-and-crossfill.
- Canonical composition of mutation and recombination is available.
- Variations can be applied stochastically.
- Selection mechanisms: stochastic universal sampling, roulette wheel algorithm, and generational selection.
- Selection probability functions: fitness proportionate selection with windowing procedure, linear and exponential pressure ranking selection.
- Termination conditions: reaching the fitness function *plateau*, reaching the maximum number of iterations, reaching the given fitness function value, reaching some user defined threshold.
- Conjunction and disjunction of termination conditions are supported.
- Possibility of addition of new variation operators, selection mechanisms, selection probability functions, and termination conditions at client-side code.

Limitations:

- Client-side program compilation time is comparatively long due to *header-only* nature of the library and the use of templates.
- Set of variation operators is limited. In case of floating-point representation alone, the popular BLX recombination (Eshelman & Schaffer, 1993) is not implemented.
- Very popular mechanisms of selection to the next generation, e.g., $(\mu + \lambda)$-selection and (μ, λ)-selection, are not implemented.
- Recombination with number of parents different than 2 is not supported.

Note: For information about API documentation, tutorial, installation instructions, test suite, code statistics, reporting problems with the library, support inquiries, and feature requests, please see the *README* file in the software archive.

Scholarly publications and research projects using the software:

- research project in computational materials science (cf. Acknowledgements section)
- PhD thesis (Tarkowski, 2022b)
- 1 report and 2 articles (Tarkowski, 2022a; Tarkowski & Gonzalez Szwacki, 2022, 2023)

Statement of need

Scientific use of C++ genetic algorithms libraries seems to be dominated by four software packages: GAlib (Wall, n.d.), Evolving Objects (Keijzer et al., 2002), OpenBEAGLE (Gagné & Parizeau, 2006), and Evolutionary Computation Framework (ECF) (Jakobović, n.d.). Evolving Objects, OpenBEAGLE, and ECF are written in the C++98 standard of the language, while GAlib is written in a pre-standard version of it. Those libraries, which are written in C++98, while being comprehensive and feature-rich, also tend to be relatively hard to use for the novice user inclined toward scientific computation. The reason is the comparatively complex installation process or complexity of the library itself.

The Quilë library tries to fill the niche of easy-to-use, high-performance genetic algorithms scientific libraries by implementing the features using the modern C++20 standard of the language. The software provides an easy starting point for researchers and academic teachers who need genetic algorithms and use C++ for their work. The library is available as a header-only (one file) implementation that can be installed by simply copying its source code. The user can run the accompanying examples in matter of mere minutes. On the other hand, the library also intentionally strives to be minimal, so only a limited set of use cases is covered.

The library is implemented in generic (template metaprogramming) and partly in functional programming style with elements of concurrency. Modern elements of C++ are used, e.g., *concepts* (Sutton, 2017), alongside established constructs, e.g., substitution failure is not an error (SFINAE) (Vandevoorde & Josuttis, 2002). In order to compile programs developed with the library, a C++ compiler supporting the C++20 standard of the language is needed.

The library employs a database of already-computed fitness function values. The software is therefore well suited for optimization tasks with fitness functions calculated from simulation codes like *ab initio* (condensed matter physics) or finite element method calculations. This feature, however, does not exclude the use of inexpensive fitness functions. For the sake of convenience the database itself is available through the intermediary objects, which are responsible for database cohesion and lifetime management. The intermediary object type is implemented with the use of the smart pointer *std::shared_ptr* (Alexandrescu, 2001). Calculations of fitness function values not yet available in the database are computed concurrently in order to speed up the whole evolutionary process; thread pool design pattern is used to optimally balance the load on CPU cores (Williams, 2019).

Example usage of the library features is outlined in the *tutorial/tutorial.txt* file in the software archive. Example results obtained with the use of the library are shown in Figure 1 and in Figure 2.

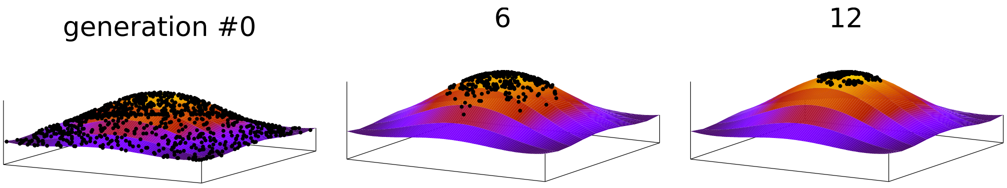

Figure 1: Genetic drift over function $f(x,y) = \cos\frac{r(x,y)}{4} + e$, where $r(x,y) = \sqrt{x^2+y^2}$, on domain $[-10,10]^2$. Three genotype generations were presented: the first one (#0), in half of the evolution (#6) and at the end (#12). Each generation has the same size. This figure can be reproduced by running *examples/example_1* example from the software archive.

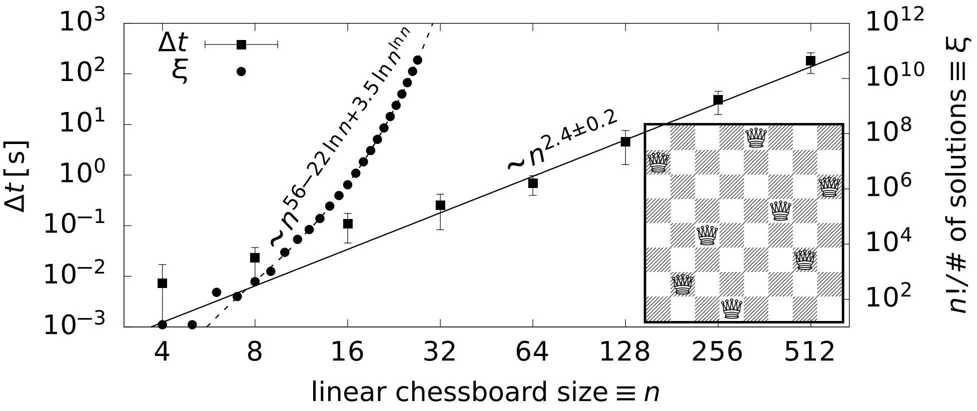

Figure 2: The n queens puzzle: time of genetic program execution Δt and problem complexity ξ. Calculations were performed on low-spec CPU (*thermal design power* 6 W) for sample size of 10 per measurement point. Number of n queens puzzle solutions can be taken from (*A000170 – Number of Ways of Placing n Nonattacking Queens on an $n \times n$ Board*, n.d.). Inset: Sample solution *4Q3/Q7/7Q/5Q2/2Q5/6Q1/1Q6/3Q4 w - - 0 0* for eight queens puzzle reached after creation of 56 unique genotypes in evolution (on average 123.61 with standard deviation 111.69 for sample size 100). This figure can be reproduced by running *examples/example_2/time* example from the software archive.

Acknowledgements

This work is a result of projects funded by the National Science Centre of Poland (Twardowskiego 16, PL-30312 Kraków, Poland, http://www.ncn.gov.pl/) under grant number UMO-2016/23/B/ST3/03575.

References

A000170 – number of ways of placing n nonattacking queens on an $n \times n$ board. (n.d.). https://oeis.org/A000170

Alexandrescu, A. (2001). *Modern C++ design: Generic programming and design patterns applied*. Addison-Wesley.

Barricelli, N. A. (1962). Numerical testing of evolution theories. *Acta Biotheoretica*, *16*, 69–98. https://doi.org/10.1007/BF01556771

Drachal, K., & Pawłowski, M. (2021). A review of the applications of genetic algorithms to forecasting prices of commodities. *Economies*, *9*(1). https://doi.org/10.3390/economies9010006

Eshelman, L. J., & Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schemata. In *Foundations of genetic algorithms* (Vol. 2, pp. 187–202). Elsevier. https://doi.org/10.1016/B978-0-08-094832-4.50018-0

Gagné, C., & Parizeau, M. (2006). Genericity in evolutionary computation software tools: Principles and case study. *International Journal on Artificial Intelligence Tools*, *15*(02), 173–194. https://doi.org/10.1142/S021821300600262X

Galloway, A. R. (2011). Creative evolution. *Cabinet. A Quarterly of Art and Culture*, *42*, 45–50.

Ghaheri, A., Shoar, S., Naderan, M., & Hoseini, S. S. (2015). The applications of genetic algorithms in medicine. *Oman Medical Journal*, *30*(6), 406–416. https://doi.org/10.5001/omj.2015.82

Goudos, S. K., Kalialakis, C., & Mittra, R. (2016). Evolutionary algorithms applied to antennas and propagation: A review of state of the art. *International Journal of Antennas and Propagation*, *2016*, 1010459. https://doi.org/10.1155/2016/1010459

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press.

Jakobović, D. (n.d.). *ECF – evolutionary computation framework*. http://ecf.zemris.fer.hr/

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, *80*, 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

Keijzer, M., Merelo, J. J., Romero, G., & Schoenauer, M. (2002). Evolving Objects: A general purpose evolutionary computation library. *Artificial Evolution – 5th International Conference*, 231–242. https://doi.org/10.1007/3-540-46033-0_19

Kudjo, P. K., Ocquaye, E. N. N., & Ametepe, W. (2017). Review of genetic algorithm and application in software testing. *International Journal of Computer Applications*, *160*(2), 1–6. https://doi.org/10.5120/ijca2017912965

Lee, C. K. H. (2018). A review of applications of genetic algorithms in operations management. *Engineering Applications of Artificial Intelligence*, *76*, 1–12. https://doi.org/10.1016/j.engappai.2018.08.011

Sutton, A. (2017). *Wording paper, C++ extensions for Concepts* (No. P0734R0). International Organization for Standardization.

Tarkowski, T. (2022a). *Genetic algorithm formulation and tuning with use of test functions*. arXiv. https://doi.org/10.48550/ARXIV.2210.03217

Tarkowski, T. (2022b). *Przewidywanie struktury krystalicznej nanodrutów z użyciem obliczeń ewolucyjnych (crystal structure prediction of nanowires using evolutionary computations)* [PhD thesis, Faculty of Physics, University of Warsaw]. https://depotuw.ceon.pl/bitstream/handle/item/4452/0000-DR-95841-praca.pdf?sequence=1

Tarkowski, T., & Gonzalez Szwacki, N. (2022). *The structure of thin boron nanowires predicted using evolutionary computations*. arXiv. https://doi.org/10.48550/ARXIV.2211.11901

Tarkowski, T., & Gonzalez Szwacki, N. (2023). Boron nanotube structure explored by evolutionary computations. *Crystals*, *13*(1). https://doi.org/10.3390/cryst13010019

Turing, A. M. (1948). *Intelligent machinery*. National Physical Laboratory.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–460.

Vandevoorde, D., & Josuttis, N. M. (2002). *C++ templates: The complete guide*. Addison-Wesley.

Wall, M. (n.d.). *GAlib – a C++ library of genetic algorithm components*. http://lancet.mit.edu/ga/

Williams, A. (2019). *C++ concurrency in action* (Second edition). Manning Publications.