$\rho\mu$: A Java library of randomization enhancements and other math utilities

Vincent A. Cicirello ⓘ [1]

1 Computer Science, School of Business, Stockton University, Galloway, NJ 08205, USA

Summary

The $\rho\mu$ library is a Java library of Randomization enHancements and Other Math Utilities. It originated as a core dependency for our other open source research software libraries, such as JavaPermutationTools (Cicirello, 2018b) and Chips-n-Salsa (Cicirello, 2020), providing these libraries with efficient randomization related algorithms. For example, Chips-n-Salsa is a library of stochastic local search and evolutionary algorithms, which requires fast generation of random indexes into arrays and other structures, as well as fast generation of non-uniform random numbers to implement Gaussian mutation, Cauchy mutation, etc. The $\rho\mu$ library includes implementations of efficient algorithms for randomly sampling indexes into arrays and other sequential structures, randomly sampling pairs and triples of distinct indexes, randomly sampling k indexes, among others, and also includes efficient random number generation from non-uniform distributions, such as Gaussian, Cauchy, and Binomial. It also includes math functions required by the randomization utilities. The documentation is hosted on the web (https://rho-mu.cicirello.org/), and source code on GitHub (https://github.com/cicirello/rho-mu), which includes a directory of example programs illustrating $\rho\mu$ usage.

Functionality

The randomization enhancements provided by $\rho\mu$ include:

- Faster generation of random integers subject to a bound or bound and origin, using the algorithm of Lemire (2019), as well as faster generation of streams of bounded integers. A sample program demonstrates the speed advantage, where on average $\rho\mu$ used 53.3% less CPU time than Java's `RandomGenerator.nextInt(bound)`.
- Faster generation of Gaussian distributed random doubles, with a Java port of the GNU Scientific Library's C implementation (Voss, 2014) of the Ziggurat algorithm (Leong et al., 2005; Marsaglia & Tsang, 2000).
- Additional distributions available beyond what is supported by the Java API's `RandomGenerator` classes, such as Binomial and Cauchy random variables.
- Ultrafast, but biased, `nextBiasedInt` methods that exclude rejection sampling to trade-off uniformity for speed, based on Lemire (2019), as well as streams of such biased integers. A sample program shows the substantial speed advantage offered for cases where strict uniformity is not required, where on average $\rho\mu$'s `nextBiasedInt(bound)` used 99.2% less CPU time than Java's `RandomGenerator.nextInt(bound)`.
- Methods for generating random pairs and triples of integers without replacement.
- Methods for generating random samples of k integers without replacement from a range of n integers, including three alternative algorithms, reservoir sampling (Vitter, 1985), pool sampling (Ernvall & Nevalainen, 1982), and insertion sampling (Cicirello, 2022), as well as a method that chooses among these based on n and k.
- Streams from binomial, Cauchy, exponential, and Gaussian distributions.

Comparisons used OpenJDK 17 and Windows 10 on an AMD A10-5700 3.4GHz CPU with 8GB DDR3 RAM. The GitHub repository includes the data and t-Test results demonstrating the speed enhancements at extremely statistically significant levels.

Architecture

Figure 1 provides a UML diagram illustrating the architecture of the library. Java 17 introduced a RandomGenerator interface, and five nested subinterfaces for special types of random number generator. The $\rho\mu$ library provides a hierarchy of wrapper classes corresponding to Java 17's RandomGenerator interface hierarchy. This enables using $\rho\mu$'s EnhancedRandomGenerator, and its subclasses, as drop-in replacements in existing applications.
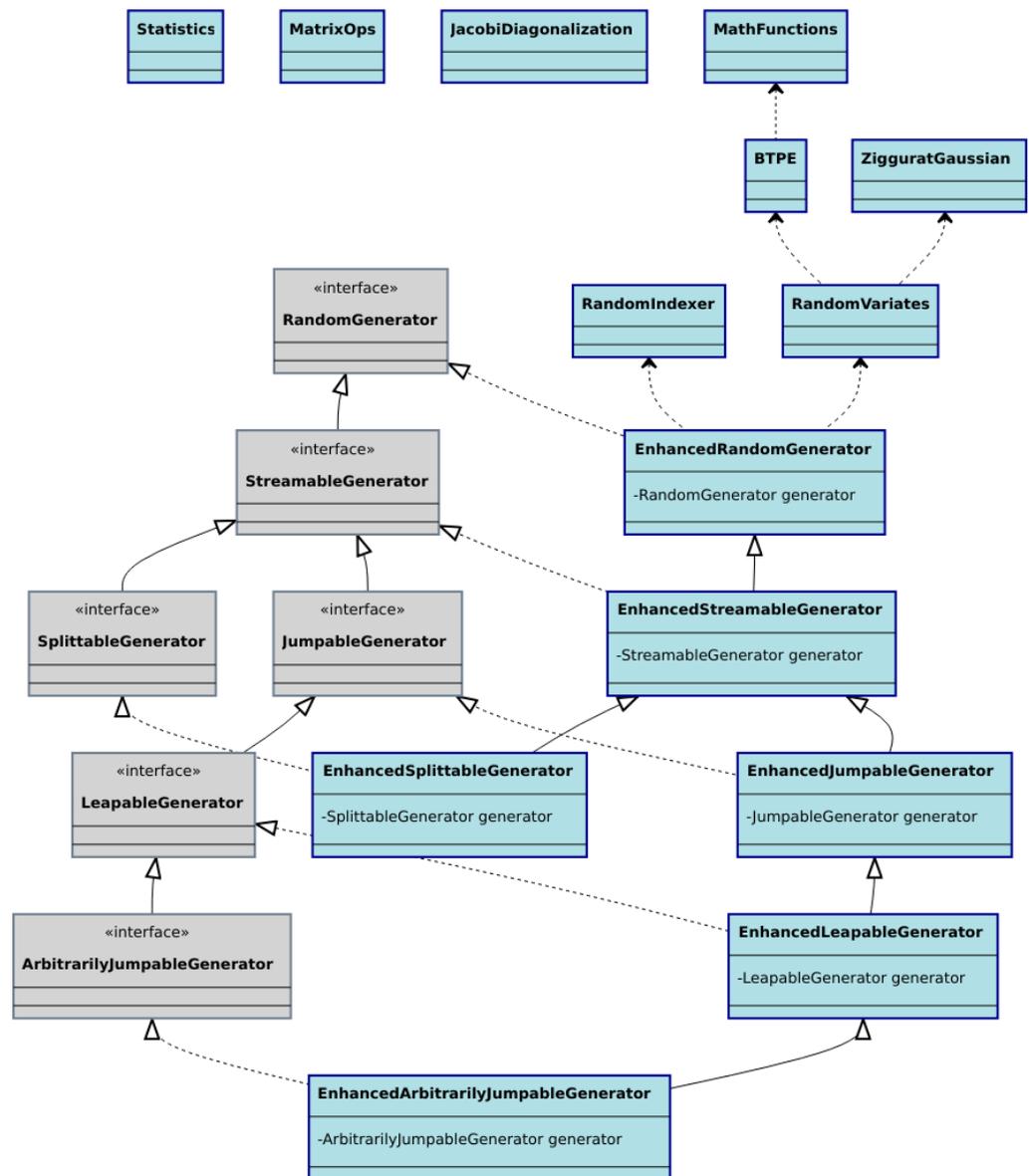

Figure 1: UML diagram of $\rho\mu$. Java API interfaces in gray, and $\rho\mu$ classes in blue.

Cicirello. (2022). $\rho\mu$: A Java library of randomization enhancements and other math utilities. *Journal of Open Source Software*, *7*(76), 4663. https://doi.org/10.21105/joss.04663.

Statement of Need

The original motivation for the $\rho\mu$ library is to provide efficient algorithms for randomization operations commonly needed by stochastic local search (Hoos & Stützle, 2018) and evolutionary algorithms (Doerr & Neumann, 2019; Petrowski & Ben-Hamida, 2017). Such algorithms rely heavily on randomized behavior, including the need to generate very many random indexes into linear structures such as vectors of bits. Sometimes independent random indexes are required, while other times (such as for a k-point crossover) a random sample without replacement is necessary. For real-valued optimization problems, evolutionary algorithms require random numbers from non-uniform distributions such as for Gaussian mutation (Hinterding, 1995) or Cauchy mutation (Szu & Hartley, 1987). Our prior research showed that evolutionary algorithms are so reliant upon random number generation that one can achieve as much as a 25% speed up by optimizing choice of random number algorithms alone (Cicirello, 2018a).

Efficient randomization has important research applications in a variety of other areas as well, such as in modeling and simulation (Greasley & Edwards, 2021; Rabe et al., 2020; Zhang et al., 2019), domain randomization (Tobin et al., 2017) for sim-to-real transfer learning (Zhao et al., 2020), among others.

Java 17 introduced a substantial overhaul of its random number support, adding a hierarchy of interfaces, where no common interface previously existed; as well as adding several pseudo-random number generators (PRNG) along with a factory class. Java libraries exist, such as Apache Commons (The Apache Software Foundation, 2021), PRNGine (Jenetics, 2022), and Uncommons Maths (Dyer, 2014), that provide additional PRNGs and in some cases specialized algorithms for sampling, additional distributions, and other randomization operations. But, I believe that $\rho\mu$ is the first library to be designed around Java 17's RandomGenerator interface hierarchy, providing a convenient mechanism to add support for additional specialized randomization operations to any of Java 17's PRNGs or even those from other libraries like Apache Commons. $\rho\mu$'s architecture provides a drop-in replacement approach enabling easily upgrading randomization functionality of existing PRNGs in existing applications.

References

Cicirello, V. A. (2018a). Impact of random number generation on parallel genetic algorithms. *Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference*, 2–7.

Cicirello, V. A. (2018b). JavaPermutationTools: A java library of permutation distance metrics. *Journal of Open Source Software*, *3*(31), 950. https://doi.org/10.21105/joss.00950

Cicirello, V. A. (2020). Chips-n-salsa: A java library of customizable, hybridizable, iterative, parallel, stochastic, and self-adaptive local search algorithms. *Journal of Open Source Software*, *5*(52), 2448. https://doi.org/10.21105/joss.02448

Cicirello, V. A. (2022). Cycle mutation: Evolving permutations via cycle induction. *Applied Sciences*, *12*(11). https://doi.org/10.3390/app12115506

Doerr, B., & Neumann, F. (2019). *Theory of evolutionary computation: Recent developments in discrete optimization*. Springer. ISBN: 9783030294144

Dyer, D. W. (2014). Uncommons maths: Random number generators, probability distributions, combinatorics and statistics for java. In *GitHub Repository*. GitHub. https://github.com/dwdyer/uncommons-maths

Ernvall, J., & Nevalainen, O. (1982). An algorithm for unbiased random sampling. *The Computer Journal*, *25*(1), 45–47. https://doi.org/10.1093/comjnl/25.1.45

Greasley, A., & Edwards, J. S. (2021). Enhancing discrete-event simulation with big data

analytics: A review. *Journal of the Operational Research Society*, *72*(2), 247–267. https://doi.org/10.1080/01605682.2019.1678406

Hinterding, R. (1995). Gaussian mutation and self-adaption for numeric genetic algorithms. *IEEE International Conference on Evolutionary Computation*, *1*, 384–389. https://doi.org/10.1109/ICEC.1995.489178

Hoos, H. H., & Stützle, T. (2018). Stochastic local search. In *Handbook of approximation algorithms and metaheuristics methologies and traditional applications* (2nd ed., Vol. 1). Chapman; Hall/CRC.

Jenetics. (2022). PRNGine - pseudo random number engines for monte carlo simulations. In *GitHub repository*. GitHub. https://github.com/jenetics/prngine

Lemire, D. (2019). Fast random integer generation in an interval. *ACM Transactions on Modeling and Computer Simulation*, *29*(1). https://doi.org/10.1145/3230636

Leong, P. H. W., Zhang, G., Lee, D.-U., Luk, W., & Villasenor, J. (2005). A comment on the implementation of the ziggurat method. *Journal of Statistical Software*, *12*(7), 1–4. https://doi.org/10.18637/jss.v012.i07

Marsaglia, G., & Tsang, W. W. (2000). The ziggurat method for generating random variables. *Journal of Statistical Software*, *5*(8), 1–7. https://doi.org/10.18637/jss.v005.i08

Petrowski, A., & Ben-Hamida, S. (2017). *Evolutionary algorithms*. Wiley. https://doi.org/10.1002/9781119136378

Rabe, M., Deininger, M., & Juan, A. A. (2020). Speeding up computational times in simheuristics combining genetic algorithms with discrete-event simulation. *Simulation Modelling Practice and Theory*, *103*, 102089. https://doi.org/10.1016/j.simpat.2020.102089

Szu, H. H., & Hartley, R. L. (1987). Nonconvex optimization by fast simulated annealing. *Proceedings of the IEEE*, *75*(11), 1538–1540. https://doi.org/10.1109/PROC.1987.13916

The Apache Software Foundation. (2021). *Apache commons RNG: Random numbers generators*. Apache. https://commons.apache.org/proper/commons-rng/

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23–30. https://doi.org/10.1109/IROS.2017.8202133

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, *11*(1), 37–57. https://doi.org/10.1145/3147.3165

Voss, J. (2014). The ziggurat method for generating gaussian random numbers. In *GNU Scientific Library*. GNU. https://www.seehuhn.de/pages/ziggurat

Zhang, L., Zhou, L., Ren, L., & Laili, Y. (2019). Modeling and simulation in intelligent manufacturing. *Computers in Industry*, *112*, 103123. https://doi.org/10.1016/j.compind.2019.08.004

Zhao, W., Queralta, J. P., & Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: A survey. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 737–744. https://doi.org/10.1109/SSCI47803.2020.9308468