DE-Sim: an object-oriented, discrete-event simulation tool for data-intensive modeling of complex systems in Python

Arthur P. Goldberg[1] and Jonathan R. Karr[1]

1 Icahn Institute for Data Science and Genomic Technology and Department of Genetics and Genomic Sciences, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA

Summary

Recent advances in data collection, storage, and sharing have created unprecedented opportunities to gain insights into complex systems such as the biochemical networks that generate cellular behavior. Understanding the behavior of such systems will likely require larger and more comprehensive dynamical models that are based on a combination of first principles and empirical data. These models will likely represent each component and interaction using mechanistic approximations that are derived from first principles and calibrated with data. For example, dynamical models of biochemical networks often represent the interactions among molecules as chemical reactions whose rates are determined by combining approximations of chemical kinetics and empirically-observed reaction rates. Furthermore, complex models that represent multiple types of components and their interactions will require diverse approximations and large, heterogeneous datasets. New tools are needed to build and simulate such data-intensive models.

One of the most promising methods for building and simulating data-intensive models is discrete-event simulation (DES). DES represents the dynamics of a system as a sequence of instantaneous events (Fishman, 2013). DES is used for a wide range of research, such as studying the dynamics of biochemical networks, characterizing the performance of computer networks, evaluating potential war strategies, and forecasting epidemics (Banks et al., 2009). Although multiple DES tools exist, it remains difficult to build and simulate data-intensive models. First, it is cumbersome to create complex models with the low-level languages supported by many of the existing tools. Second, most of the existing tools are siloed from the ecosystems of data science tools that are exploding around Python and R.

To address this problem, we developed DE-Sim (https://github.com/KarrLab/de_sim), an open-source, object-oriented (OO), Python-based DES tool. DE-Sim helps researchers model complex systems by enabling them to use Python's powerful OO features to manage multiple types of components and multiple types of interactions. By building upon Python, DE-Sim also makes it easy for researchers to employ Python's powerful data science tools, such as pandas (McKinney, 2010) and SciPy (Virtanen et al., 2020), to use large, heterogeneous datasets to build comprehensive and detailed models. We anticipate that DE-Sim will enable a new generation of models that capture systems with unprecedented breadth and depth. For example, we are using DE-Sim to develop WC-Sim (Arthur P. Goldberg & Karr, 2020), a multi-algorithmic simulation tool for whole-cell models (Arthur P. Goldberg et al., 2016; Arthur P. Goldberg et al., 2018; Karr et al., 2012, 2015) that predict phenotype from genotype by capturing all of the biochemical activity in a cell.

Here, we describe the need for new tools for building and simulating more comprehensive and more detailed models, and outline how DE-Sim addresses this need. In addition, we summarize

the strengths of DE-Sim over existing DES tools, and we report the simulation performance of DE-Sim. Finally, we outline our plans to increase the performance of simulations executed by DE-Sim. A tutorial that describes how to build and simulate models with DE-Sim, examples, and documentation are available online, as described in the 'Availability of DE-Sim' section below.

Statement of Need

Many scientific fields can now collect detailed data about the components of complex systems and their interactions. For example, deep sequencing has dramatically increased the availability of molecular data about biochemical networks. Combined with advances in computing, we believe that it is now possible to use this data and first principles to create comprehensive and detailed models that can provide new insights into complex systems. For example, deep sequencing and other molecular data can be used to build whole-cell models.

Achieving such comprehensive and detailed models will likely require integrating disparate principles and diverse data. While there are several DES tools, such as SimEvents (Clune et al., 2006) and SimPy (Matloff, 2008), and numerous tools for working with large, heterogeneous datasets, such as pandas and SQLAlchemy (Bayer, 2020), it is difficult to use these tools in combination. As a result, despite having most of the major ingredients, it remains difficult to build and simulate data-intensive models.

DE-Sim provides critical features for building and simulating data-intensive models

DE-Sim simplifies the construction and simulation of *discrete-event models* through several features. First, DE-Sim structures discrete-event models as OO programs (Zeigler, 1987). This structure enables researchers to use *simulation object* classes to encapsulate the complex logic required to represent *model components*, and use *event message* classes to encapsulate the logic required to describe the *interactions* among model components. With DE-Sim, users define simulation object classes by creating subclasses of DE-Sim's simulation object class. DE-Sim simulation object classes can exploit all the features of Python classes. For example, users can encode relationships between the types of components in a model into hierarchies of subclasses of simulation objects. As a concrete example, a model of the biochemistry of RNA transcription and protein translation could be implemented using a superclass that captures the behavior of polymers and three subclasses that represent the specific properties of DNAs, RNAs, and proteins. By representing model components as Python simulation objects, DE-Sim makes it easy to model complex systems that contain multiple types of components by defining multiple classes of simulation objects. Users can then model arbitrarily many instances of each type of component by creating multiple instances of the corresponding simulation object class.

Second, by building on top of Python, DE-Sim enables researchers to conveniently use Python's extensive suite of data science tools to build models from heterogeneous, multidimensional datasets. For example, researchers can use tools such as ObjTables (Karr et al., 2020), H5py, requests, SQLAlchemy, and pandas to access diverse data in spreadsheets, HDF5 files, REST APIs, databases, and other sources; use tools such as NumPy (Oliphant, 2015) to integrate this data into a unified model; and use tools such as SciPy and NumPy to perform calculations during simulations of models. DE-Sim also facilitates the use of Python tools to analyze simulation results.

In addition, DE-Sim provides several features to help users execute, analyze, and debug simulations:

- **Stop conditions:** DE-Sim makes it easy to terminate simulations when specific criteria are reached. Researchers can specify stop conditions as functions that return true when a simulation should conclude.
- **Results checkpointing:** The results of a simulation can be conveniently recorded by configuring periodic checkpoints of specified parts of the simulation's state.
- **Reproducible simulations:** To help researchers debug simulations, repeated executions of the same simulation with the same configuration and same random number generator seed produce the same results.
- **Space-time visualizations:** DE-Sim generates space-time visualizations of simulation trajectories (Figure 1). These diagrams can help researchers understand and debug simulations.
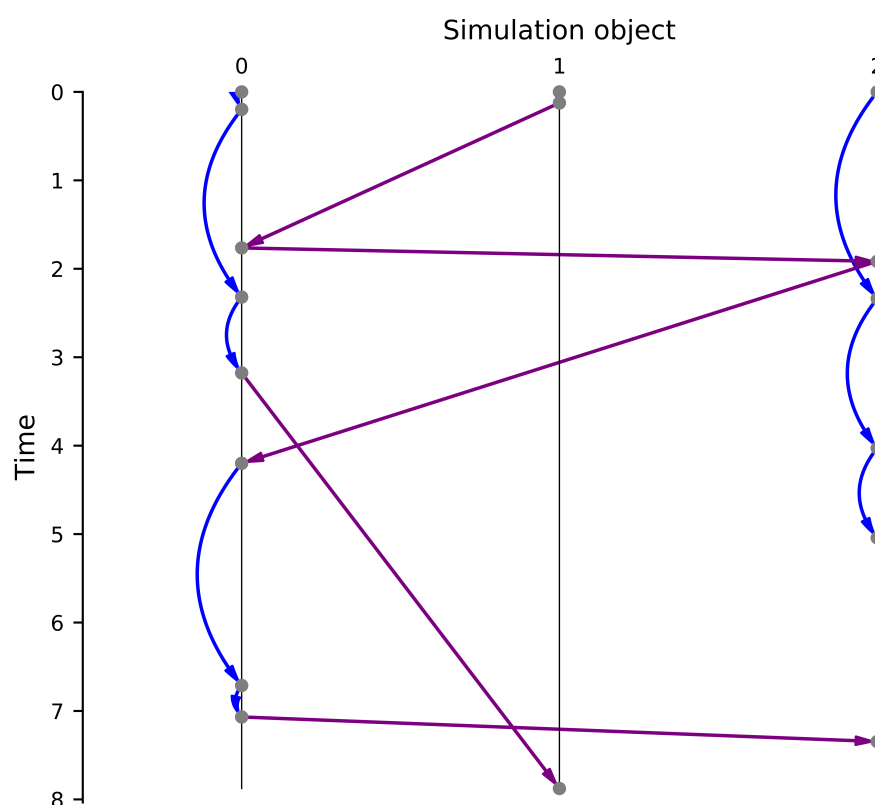
Figure 1: DE-Sim can generate space-time visualizations of simulation trajectories. This figure illustrates a space-time visualization of all of the events and messages in a simulation of the parallel hold (PHOLD) DES benchmark model (Fujimoto, 1990) with three simulation objects. The timeline (black line) for each object shows its events (grey dots). The blue and purple arrows illustrate events scheduled by simulation objects for themselves and other objects, respectively. The code for this simulation is available in the DE-Sim Git repository.

We believe that these features can simplify and accelerate the development of complex, data-intensive models.

Comparison of DE-Sim with existing discrete-event simulation tools

Although multiple DES tools already exist, we believe that DE-Sim uniquely facilitates data-intensive modeling through a novel combination of OO modeling and support for numerous

3

high-level data science tools. Figure 2 compares the features and characteristics of DE-Sim with some of the most popular DES tools.

Simulation tool	Application domain	Modeling language	Object-oriented models	GUI model builder	Open-source	Latest update
DE-Sim	Complex, data-intensive models	Python	✓		✓	2020
SimEvents	Communications networks and process flows	MATLAB	✓	✓		2020
SimPy	General purpose process-based framework	Python			✓	2020
SIMSCRIPT III	Object-oriented simulation of engineered systems	SIMSCRIPT III	✓			2020
SIMUL8	Business processes	Visual Logic		✓		2020
SystemC	Digital hardware	C++	✓		✓	2018

Figure 2: **Comparison of DE-Sim with some of the most popular DES tools.** DE-Sim is the only open-source, OO DES tool based on Python. This combination of features makes DE-Sim uniquely suitable for creating and simulating complex, data-intensive models.

SimPy is an open-source DES tool that enables users to write functions that describe simulation processes (SimPy's analog to DE-Sim's simulation objects). As another Python-based tool, SymPy also makes it easy for researchers to leverage the Python ecosystem to build models. However, we believe that DE-Sim makes it easier for researchers to build complex models by enabling them to implement models as collections of classes rather than collections of functions. DE-Sim thereby enables modelers to use a Python object to encapsulate the state of a model component together with operations on the state, and use inheritance to share state and operations among related types of model components. In addition, we believe that DE-Sim is simpler to use because DE-Sim supports a uniform approach for scheduling events, whereas SimPy simulation processes must use two different approaches: one to schedule events for themselves, and another to schedule events for other processes.

SimEvents is a library for DES within the MATLAB/Simulink environment. While SimEvents' graphical interface makes it easy to create simple models, we believe that DE-Sim makes it easier to implement more complex models. By facilitating use of the many Python-based data science tools, DE-Sim makes it easier to use data to create models than SimEvents, which builds on a smaller ecosystem of data science tools.

SystemC is a `C++`-based OO DES tool that is frequently used to model digital systems (Mueller et al., 2001). While SystemC provides many of the same core features as DE-Sim, we believe that DE-Sim is more accessible to researchers than SystemC because DE-Sim builds upon Python, which is more popular than `C++` in many fields of research.

SIMSCRIPT III (Rice et al., 2005) and SIMUL8 (Concannon et al., 2003) are commercial DES tools that define proprietary languages which researchers can use to implement models. SIMSCRIPT III is a general-purpose simulation language designed for modeling decision support systems in domains such as war-gaming, transportation, and manufacturing. We believe that DE-Sim is more powerful than SIMSCRIPT III for most scientific and engineering problems because it leverages Python's robust data science ecosystem.

SIMUL8 models business processes as workflows. It provides a powerful GUI for describing the flow of *work items* through a network of queues and servers, and includes tools to analyze

and visualize the potentially stochastic behavior of a process. DE-Sim is more suitable than SIMUL8 for modeling scientific or engineering systems because modelers can use DE-Sim to describe processes that cannot be easily structured as workflows.

Performance of DE-Sim

Figure 3 illustrates the performance of DE-Sim simulating a model of a cyclic messaging network over a range of network sizes. A messaging network consists of a ring of nodes. When a node handles an event, it schedules the same type of event for its forward neighbor with a one time-unit delay. Each simulation is initialized by sending a message to each node at the first time-unit. The code for this performance test is available in the DE-Sim Git repository, and in a Jupyter notebook at https://sandbox.karrlab.org/tree/de_sim.

		Run time (s)		Event rate (events s^{-1})	
Nodes	**Events**	**Mean**	**Std dev**	**Mean**	**Std dev**
4	400	0.05	0.01	8,930	1,930
16	1,600	0.10	0.02	16,531	3,052
64	6,400	0.31	0.04	20,986	2,734
256	25,600	1.22	0.16	21,139	2,597
1,024	102,400	5.34	0.96	19,573	3,465
4,096	409,600	20.06	1.53	20,491	1,506

Figure 3: **Performance of DE-Sim simulating a cyclic messaging network over a range of sizes.** Each simulation was executed for 100 time-units. Each statistic represents the average of three simulation runs in a Docker container running on a 2.9 GHz Intel Core i5 processor.

Conclusion

In summary, DE-Sim is an open-source, object-oriented, discrete-event simulation tool implemented in Python that makes it easier for modelers to create and simulate complex, data-intensive models. First, DE-Sim enables researchers to conveniently use Python's OO features to manage multiple types of model components and interactions among them. Second, DE-Sim enables researchers to directly use Python data science tools, such as pandas and SciPy, and large, heterogeneous datasets to construct models. Together, we anticipate that DE-Sim will accelerate the construction and simulation of unprecedented models of complex systems, leading to new scientific discoveries and engineering innovations.

To further advance the simulation of data-intensive models, we aim to improve the simulation performance of DE-Sim. One potential direction is to use DE-Sim as a specification language for a parallel DES system such as ROSS (Carothers et al., 2000). This combination of DE-Sim and ROSS would enable modelers to both create models with DE-Sim's high-level model specification semantics and quickly simulate models with ROSS.

Availability of DE-Sim

DE-Sim is freely and openly available under the MIT license at the locations below.

- Source code repository: GitHub: KarrLab/de_sim

- Jupyter notebook tutorials: https://sandbox.karrlab.org/tree/de_sim
- Documentation: docs.karrlab.org

DE-Sim requires Python 3.7 or higher and pip. This article discusses version 1.0.5 of DE-Sim.

Acknowledgements

References

Banks, J., Carson II, J., Nelson, B., & Nicol, D. (2009). *Discrete-event system simulation*. Pearson. ISBN: 978-0136062127

Bayer, M. (2020). *SQLAchemy-the database toolkit for Python*. https://www.sqlalchemy.org/

Carothers, C. D., Bauer, D., & Pearce, S. (2000). ROSS: A high-performance, low memory, modular Time Warp system. *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation*, *62*, 53–60. https://doi.org/10.1109/PADS.2000.847144

Clune, M. I., Mosterman, P. J., & Cassandras, C. G. (2006). Discrete Event and Hybrid System Simulation with SimEvents. *Proceedings of the 8th International Workshop on Discrete Event Systems*, 386–387. https://doi.org/10.1109/wodes.2006.382398

Concannon, K. H., Hunter, K. I., & Tremble, J. M. (2003). *Dynamic scheduling II: SIMUL8-planner simulation-based planning and scheduling*. 1488–1493. https://doi.org/10.1109/WSC.2003.1261534

Fishman, G. S. (2013). *Discrete-event simulation: Modeling, programming, and analysis*. Springer Science & Business Media. ISBN: 978-0-387-95160-7

Fujimoto, R. M. (1990). Performance of Time Warp under synthetic workloads. *Proceedings of the SCS Multiconference on Distributed Simulations*, *22*, 23–28. https://gdo149.llnl.gov/attachments/20776356/24674621.pdf

Goldberg, Arthur P., Chew, Y. H., & Karr, J. R. (2016). Toward scalable whole-cell modeling of human cells. *Proceedings of the ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 259–262. https://doi.org/10.1145/2901378.2901402

Goldberg, Arthur P., & Karr, J. R. (2020). *WC-Sim: A multi-algorithmic simulator for whole-cell models*. https://github.com/KarrLab/wc_sim

Goldberg, Arthur P., Szigeti, B., Chew, Y. H., Sekar, J. A., Roth, Y. D., & Karr, J. R. (2018). Emerging whole-cell modeling principles and methods. *Current Opinion in Biotechnology*, *51*, 97–102. https://doi.org/10.1016/j.copbio.2017.12.013

Karr, J. R., Liebermeister, W., Goldberg, A. P., Sekar, J. A. P., & Shaikh, B. (2020). ObjTables: Structured supplementary spreadsheets that promote data quality, reuse, and integration. *arXiv*. https://arxiv.org/abs/2005.05227

Karr, J. R., Sanghvi, J. C., Macklin, D. N., Gutschow, M. V., Jacobs, J. M., Bolival Jr, B., Assad-Garcia, N., Glass, J. I., & Covert, M. W. (2012). A whole-cell computational model predicts phenotype from genotype. *Cell*, *150*(2), 389–401. https://doi.org/10.1016/j.cell.2012.05.044

Karr, J. R., Takahashi, K., & Funahashi, A. (2015). The principles of whole-cell modeling. *Current Opinion in Microbiology*, *27*, 18–24. https://doi.org/10.1016/j.mib.2015.06.004

Matloff, N. (2008). *Introduction to discrete-event simulation and the SimPy language*. https://web.cs.ucdavis.edu/~matloff/matloff/public_html/156/PLN/DESimIntro.pdf

McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, *445*, 51–56. https://doi.org/10.25080/majora-92bf1922-00a

Mueller, W., Ruf, J., Hoffmann, D., Gerlach, J., Kropf, T., & Rosenstiehl, W. (2001). The simulation semantics of SystemC. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 64–70. https://doi.org/10.1109/DATE.2001.915002

Oliphant, T. E. (2015). *A guide to NumPy*. CreateSpace Independent Publishing Platform. ISBN: 978-1517300074

Rice, S. V., Marjanski, A., Markowitz, H. M., & Bailey, S. M. (2005). The SIMSCRIPT III programming language for modular object-oriented simulation. *Proceedings of the Winter Simulation Conference, 2005.*, 10–pp. https://doi.org/10.1109/wsc.2005.1574302

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., & others. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Zeigler, B. P. (1987). Hierarchical, modular discrete-event modelling in an object-oriented environment. *Simulation*, *49*(5), 219–230. https://doi.org/10.1177/003754978704900506