libRoadRunner: A High Performance SBML Compliant Simulator

A. Somogyi [1,*], M. T. Karlsson [2], M. Swat [1] and H. M Sauro [3*]

[1,3]Biocomplexity Institute, Indiana University, Simon Hall MSB1, 043D, 212 S. Hawthorne Drive, Bloomington, IN 47405-7003
[2]Dune Scientific, 10522 Lake City Way NE, #302 Seattle WA
[3]Department of Bioengineering, University of Washington, Seattle, WA, 98195

Associate Editor: XXXXX

ABSTRACT

Summary: Here we describe libRoadRunner, a high performance SBML compliant simulator written in C/C++. Simulation speed is achieved by creating a LLVM backend of a model at runtime based on the loaded SBML. The primary motivation for developing libRoadRunner was to create a portable that would have superior performance to any other existing simulator, provide an extensible architecture, and a wide variety of functionality. In addition it would have superior support for event handling in addition to the normal features such as conservation analysis.

Availability and Implementation: The core of libRoadRunner is written in C++ but with an additional thin layer that supports a pure C API. Users can link the library to other C++ applications or systems that can only link to C functions. The library uses libSBML for SBML support, CVODE for differential equation implementation and event handling, NLEQ2 for solving nonlinear equations and LAPACK for stoichiometric analysis. The library is cross-platform and runs on Windows, Mac OS X and Linux. We also provide extensive bindings for Python. The library is open source and licensed under the Apache License, Version 2.0.

Contact: hsauro@u.washington.edu

Supplementary information: Online documentation, build instructions and source code are available at http://www.libroadrunner.org

1 INTRODUCTION

There has been a long history (ref) of researchers developing (Bag *et al.*, 2001) biochemical network simulators. Most are monolithic making it very difficult to reuse the code for other applications. Recently there has been more interest in development reusable libraries. Most notable of these are COPASI, libSBMLSim, roadRunner, SOSLib and the Systems Biology Simulation Core Library (SBSCL). Each has strengths and weakness, for example COPASI has excellent support for optimization but its API can be challenging to use (see https://code.google.com/p/copasi-simple-api/), libSBMLSim passes all the tests in the SBML test suite but its API and functionality is somewhat limited, roadRunner is written in C# which is excellent for .NET developers but less so for others, SOSLib is no longer supported but it has a reasonable API but

does not support conservation analysis and finally SBSCL is written in Java which makes it an excellent choice for Java developers but less so for others and also doesn't support conservation analysis. All the libraries have performance issues and stability concerns (see benchmarking), most have problems tracking events except perhaps roadRunner and all have a non-extensible architectures.

Here we wish to describe libRoadRunner, a rewrite of the C# roadRunner. libRoadRunner is written in C++ with an additional thin C API. The API supports over 130 methods than provide a wide range of functionality and rapid access to the core to improve performance in realtime interactive environments. libRoadRunner supports a plugin extension mechanism making it possible to easily write extensions to the core (See Plugins). One of the key and novel features of libRoadRunner is that superior runtime performance is achieved through the use of LLVM (formerly Low Level Virtual Machine) which is designed for real-time optimization and dynamic compilation of application software. This has allowed us to develop the fastest biochemical simulator currently available. This is particularly important for large simulation applications such as ComuCell3D where 10,000 of cells are simulated simultaneously and performance is critical. Figure 1 shows the architectural overview of libRoadRunner.
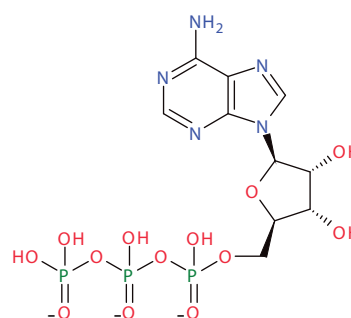

Fig. 1. Architectural Overview of libRoadRunner.

---

*to whom correspondence should be addressed

2 METHODS

2.1 Features

List of features here

2.2 LLVM Backend

For those systems such as embedded and more specialist situations where a C++ compiler might be unavailable, roadRunner also provide san option to generate C code and to compile and link the C code into roadRunner using the small portable tcc C compiler.

2.3 Extensibility

Extensibility was an importance consideration in the design of the roadRunner library. Two methods are available to add extensions. The easiest is to add new extension by writing Python scripts, this is simple to do but with the disadvantage of poorer performance. The alternative is to write native code plugins that can be loaded on demand by the roadRunner core. A plugin API is available as a pure C API so that plugins can be written in almost any language. Under Windows, a plugin is just a standard DLL. Plugins can be loaded and unloaded on demand and can be executed in a separate thread. Plugins have full access to the roadRunner API which includes the ability to create new roadRunner instances. A primary advantage of plugins is that additional functionality can be developed without disturbing any of core roadRunner code. In addition, the plugin exposes a set of methods to allow generic access any exported plugin functionality. This means that scripting languages such as Python can easily access a plugin without having to implement a new API for every plugin. Plugins also support a degree of runtime type information that allows graphical user interface applications to create user interfaces ar runtime when the plugin is loaded. The roadRunner distribution comes with two plugins that illustrate how plugins can be built, these include a a plugin that implements the Levenberg-Marquardt algorithm (ref) that allows models to be fitted to experimental data and a smaller plugin that can be used to generate synthetic experimental data for testing the Levenberg-Marquardt plugin.

2.4 Python API

Two Python APIs were developed, a ctypes based library and customized SWIG binding. The SWIG binding is more comprehensive and Pythonic in its design and is the recommended interface to use on a daily basis. The ctypes Python interface is used to specifically test the C API. Here we briefly describe the SWIG bindings.

2.5 Performance

We compared three other simulator libraries with libRoadRunner (ref), these included libSBMLSim (ref), COPASI (ref) and Systems Biology Simulation Core Library (ref). We tested the ability pass the SBML test suite, the ability to simulate very large models and the ability to simulate systems with large numbers of events. Table 1 lists the times recorded from the four different libraries based on three different types of model. The first model is a moderate sized model that tests the general performance in integration and evaluating the differential equations. The second model tests the ability of the libraries to scale to very large models and the third library tests the ability of the library to deal with large number of discrete events. The timings also include the amount of time requite to load the SBML model and compile it to executable code.

Implementation Details

The dedicated web site, libroadrunner.org includes all the necessary information concerning libRoadRunner including documentation for the C++ and C API, and documentation for the Python SWIG binding. Full build instructions are provided for Windows and Mac OS X. Build instructions for specific Linux disruptions can be obtained upon request particularly XYZ and ABC. All source code is stored on Github, documentation is generated via Doxygen and sphinc for Python. The build system is based around CMake, Linux

Table 1. All times are in milliseconds. Test models are available at libroadrunner.org

Application	Simulation	Large Models	Event Handling
libRoadRunner	23	45	67
COPASI	200	400	fail
libSBMLSim	450	fail	fail
SBSCL	500	800	fail

binaries are generated using GCC, Mac binaries via CLang Bofelli *et al.*, 2000 and Windows binaries via Visual Studio 2010. Binaries are provided for Windows, Mac OS X, including binaries for Python 2.7 on Windows and Mac OS X.

3 DISCUSSION

In this application note we describe a new high performance SBML compliant simulator. The reason for developing libRoadRunner was to satisfy a number of specific applications that require high performance, good SBML compatibility and broad functionality. These applications include interactive desktop simulation, multicellular simulations requiring 10,000 of simultaneous simulations, and multi-session needs for web based interactive simulation. For the future we are planning a number of additional features, these include new plugins, particularly alternative optimizers, high performance distributed stochastic simulation support and additional efforts to help make distributed simulation easier to achieve. Currently used by: CompuCell3D, JDesigner, Tellurium, sysb.io

ACKNOWLEDGEMENT

REFERENCES

Bofelli,F., Name2, Name3 (2003) Article title, *Journal Name*, **199**, 133-154.

Bag,M., Name2, Name3 (2001) Article title, *Journal Name*, **99**, 33-54.

Yoo,M.S. *et al.* (2003) Oxidative stress regulated genes in nigral dopaminergic neurnol cell: correlation with the known pathology in Parkinson's disease. *Brain Res. Mol. Brain Res.*, **110**(Suppl. 1), 76–84.

Lehmann,E.L. (1986) Chapter title. *Book Title*. Vol. 1, 2nd edn. Springer-Verlag, New York.

Crenshaw, B.,III, and Jones, W.B.,Jr (2003) The future of clinical cancer management: one tumor, one chip. *Bioinformatics*, doi:10.1093/bioinformatics/btn000.

Auhtor,A.B. *et al.* (2000) Chapter title. In Smith, A.C. (ed.), *Book Title*, 2nd edn. Publisher, Location, Vol. 1, pp. ???–???.

Bardet, G. (1920) Sur un syndrome d'obesite infantile avec polydactylie et retinite pigmentaire (contribution a l'etude des formes cliniques de l'obesite hypophysaire). PhD Thesis, name of institution, Paris, France.