Evaluating the Connection-Set Algebra for the neural simulator NEST

Jochen Martin Eppler[1], Håkon Enger[2], Thomas Heiberg[2], Birgit Kriener[2], Hans Ekkehard Plesser[2], Markus Diesmann[1,3], Mikael Djurfeldt[4,5]

[1] Institute of Neuroscience and Medicine (INM-6), Computational and Systems Neuroscience, Research Center Jülich, Jülich, Germany
[2] Department of Mathematical Sciences and Technology, Norwegian University of Life Sciences, Ås, Norway
[3] Computational Neurophysics, RIKEN Brain Science Institute & Computational Science Research Program, Wako City, Japan
[4] PDC Center for High Performance Computing, School of Computer Science and Communication, KTH, Stockholm, Sweden
[5] INCF Secretariat, Stockholm, Sweden

Contact: j.eppler@fz-juelich.de

Introduction

We present a direct interface to the Connection-Set Algebra (CSA, [1]) for the neural simulator NEST [2]. The goals for this interface are:

- Reduce the complexity of NEST's native connection routines by basing them on CSA notation.
- Base the topology module of NEST on CSA in order to obtain better scaling, especially for very large networks with three-dimensional topologies and on HPC facilities.
- Define and develop a standard interface that can be implemented by other connection generation libraries and used by other simulators.
- Create a C++ implementation of CSA (libcsa) that can be used by neural simulators to describe network connectivity.

Using an expressive notation like CSA to describe network connectivity lowers the risk of errors in the model description and allows the investigation of relevant model sizes. In addition, a standard notation fosters model re-use by others.

NEST

NEST is a simulator for large heterogeneous networks of point neurons or neurons with few electrical compartments [2].

It is suited for a broad range of neuronal network modeling approaches and computer architectures from standard desktop computer to computer clusters or HPC facilities such as BlueGene.

The topology module [3] allows a concise specification of large, spatially structured branch neuronal networks and organizes neuronal networks in 2D sheets or 3D space. The connections in the network are represented using probability kernels and masks. The topology module is publicly available as part of the NEST package.

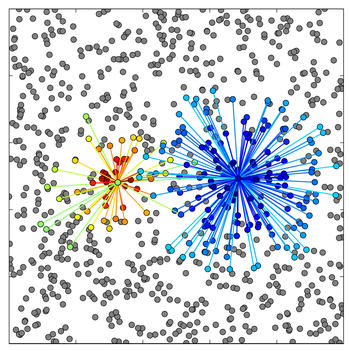
A LiveCD to try NEST without installation and the current release of the source code are available on the website of the NEST Initiative (`http://www.nest-initative.org`), as is a list of neuroscientific publications that use NEST

Connection-Set Algebra

The connection-set algebra (CSA, [1]) is a novel and general formalism for the description of connectivity in neuronal network models, from its small-scale to its large-scale structure.

It provides operators to form more complex sets of connections from simpler ones and also provides parameterization of such sets. A prototype of the CSA is available at `http://software.incf.org/software/csa`.

Example



A Connection-set with weights (g) and delays (l)

$$C_e = \langle \bar{\rho} V, g, l \rangle$$
$$V = \phi(\sigma_d, c)d$$
$$g = g_d V + \rho \mathsf{N}(0, \sigma_g)$$
$$l = r + d/v$$

A : all cells
E : excitatory cells
I : inhibitory cells

$$\{i_e, i_i\} \times \mathbb{N}_0 \cap C$$
$$C = E \times A \cap C_e \uplus I \times A \cap C_i$$

Corresponding Python code

```
from csa import *
from csa import plot
e = ival(0, 19)
i = ival(20, 29)
a = e + i
g = random2d(900)
d = euclidMetric2d(g)
g_e = gaussian(0.1, 0.3) * d
g_i = gaussian(0.2, 0.3) * d
c_e = cset(random * g_e, g_e)
c_i = cset(random * g_i, -g_i)
c = cross(e, a) * c_e + cross(i, a) * c_i
sources = [g.inverse(0.33, 0.5, e),
           g.inverse(0.67, 0.5, i)]
plot.gplotsel2d(g, c, sources,
                value=0, range=[-1, 1])
```

The C++ version of the library is currently under development. It will also provide a Python interface to allow rapid prototyping of models.

Benchmark simulations and scaling behavior

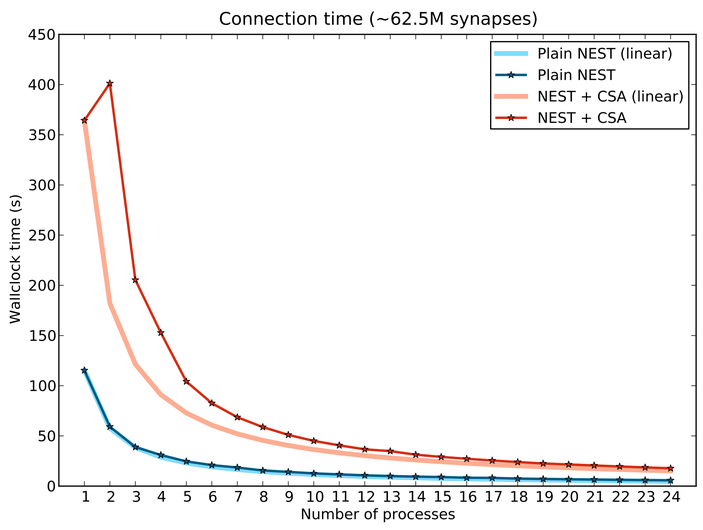
The following listing shows the part of the benchmark script which sets up connectivity:

```
ee_mask=csa.random(fanIn=CE)*csa.cross((0,NE-1),(0,NE-1))
nest.Connect(ee_mask,subnet_ex,subnet_ex,{},"excitatory")
ei_mask=csa.random(fanIn=CE)*csa.cross((0,NE-1),(0,NI-1))
nest.Connect(ei_mask,subnet_ex,subnet_in,{},"excitatory")

ie_mask=csa.random(fanIn=CI)*csa.cross((0,NI-1),(0,NE-1))
nest.Connect(ie_mask,subnet_in,subnet_ex,{},"inhibitory")
ii_mask=csa.random(fanIn=CI)*csa.cross((0,NI-1),(0,NI-1))
nest.Connect(ii_mask,subnet_in,subnet_in,{},"inhibitory")
```

An equivalent description using native NEST commands:

```
nest.RCC(nodes_ex,subnet_ex,CE,model="excitatory")
nest.RCC(nodes_ex,subnet_in,CE,model="excitatory")

nest.RCC(nodes_in,subnet_ex,CI,model="inhibitory")
nest.RCC(nodes_in,subnet_in,CI,model="inhibitory")
```

Please note that the benchmarks compare the Python prototype of CSA with the native C++ routines in NEST. This means that an absolute difference in performance is not meaningful here.



Benchmark simulation with a random balanced network [4]. The network contains 20,000 excitatory and 5,000 inhibitory neurons with 2,500 incoming synapses each. In total, the network contains ~62.5 million synapses.

The ConnectionGenerator interface

Instead of creating a custom interface for using the CSA in NEST, we decided to design and use an intermediate interface (called `ConnectionGenerator`) that abstracts from the implementation details of CSA.

This interface is completely generic and we encourage its use also for coupling other libraries for connection generation (e.g. Ivan Raikov's graph library for NineML, [5]) to NEST, as well as its use for other simulators.

The interface is designed as an abstract base class in C++ and consists of the following functions:

`int arity()`: Return the number of values associated with this iterator. Values can be parameters like weight, delay, time constants, or others.

`int size()`: Return the number of connections represented by this iterator.

`void setMask(Mask& mask)`: Inform the generator about which source and target indices exist. A mask represents a subset of the nodes in the network.

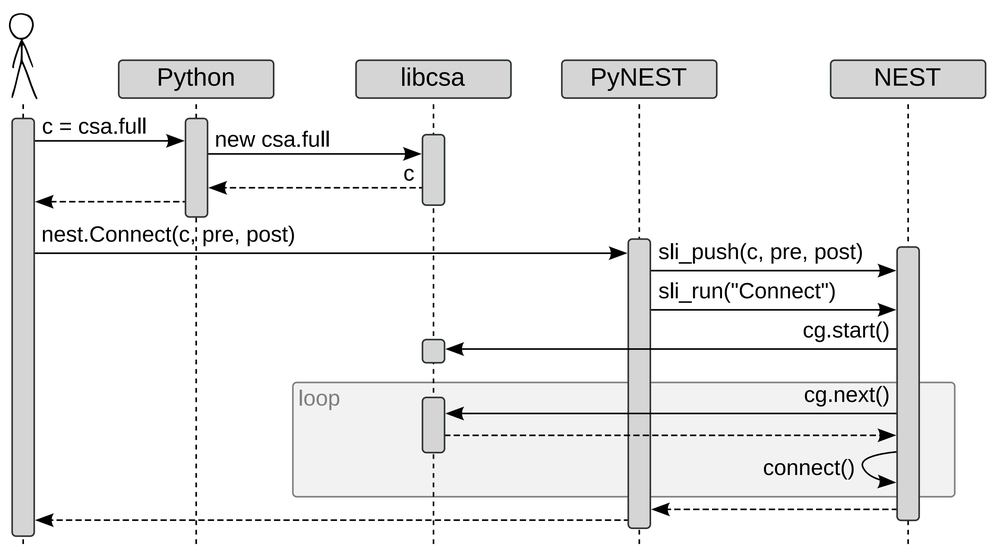
`void setMask(std::vector<Mask>& masks, int local)`: Some connection generators need to know the masks for all ranks in the parallel case.

`void start()`: Start an iteration. This function must be called before the first call to `next()`.

`bool next(int& source, int& target, double* value)`: Advance to the next connection or return false, if no more connections are available within the iterator.

The NEST–CSA interface

Using the `ConnectionGenerator` interface, coupling NEST to the CSA is almost trivial. However, as the C++ implementation of `libcsa` is not yet available, we use the Python interface to CSA in connection with PyNEST, the Python interface for NEST [6].



References

[1] Djurfeldt (2011) *CNS*2011, BMC Neuroscience.* doi:10.1186/1471-2202-12-S1-P80.
[2] Gewaltig & Diesmann (2007) NEST Neural Simulation Tool. *Scholarpedia* 2(4), 1430.
[3] Plesser & Austvoll (2009) *BMC Neuroscience*, doi:10.1186/1471-2202-10-S1-P56.
[4] Brunel (2000) *J. Comput. Neurosci.*, doi:10.1023/A:1008925309027.
[5] Raikov & De Schutter (2010) FENS Abstr., vol.5, 059.26.
[6] Eppler et al (2009) *Front. Neuroinform.*, doi:10.3389/neuro.11.012.2008.

Acknowledgments: Partially funded by "The Next-Generation Integrated Simulation of Living Matter" project, part of the Development and Use of the Next-Generation Supercomputer Project of the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan, the Helmholtz Alliance on Systems Biology, the Research Council of Norway, the International Neuroinformatics Coordinating Facility, and by a grant from the European Union (FACETS project, FP6-2004-IST-FETPI-015879).

mindzoo.de/conf/incf2011.html